Weakly Relational Numerical Abstract Domains

Antoine Miné

THÈSE

présentée à

l'ÉCOLE POLYTECHNIQUE

pour l'obtention du titre de

DOCTEUR DE L'ÉCOLE POLYTECHNIQUE EN INFORMATIQUE

Antoine MINÉ

6 décembre 2004

Domaines numériques abstraits faiblement relationnels

Weakly Relational Numerical Abstract Domains

Président:	Chris HANKIN *Professeur, Imperial College, Londres*
Rapporteurs:	Roberto GIACOBAZZI *Professeur, Università degli Studi di Verona*
	Helmut SEIDL *Professeur, Technische Universität München*
Directeur de thèse:	Patrick COUSOT *Professeur, École Normale Supérieure, Paris*

École Normale Supérieure
Département d'Informatique

Cette recherche a été conduite à l'École Normale Supérieure (Paris) durant un contrat d'allocation couplée (normalien) de l'Université Paris IX Dauphine. Cette recherche a été financée en partie par les projets Daedalus (projet européen IST-1999-20527 du programme FP5) et Astrée (projet français RNTL).

Résumé

Le sujet de cette thèse est le développement de méthodes pour l'analyse automatique des programmes informatiques. Une des applications majeures est la conception d'outils pour prévoir les *erreurs de programmation* avant qu'elles ne se produisent, ce qui est crucial à l'heure où des tâches critiques et complexes sont confiées à des ordinateurs. Nous nous plaçons dans le cadre de l'*interprétation abstraite*, qui est une théorie de l'approximation sûre des sémantiques de programmes, et nous nous intéressons en particulier aux *domaines numérique abstraits*, spécialisés dans la découverte automatique des propriétés des variables numériques d'un programme.

Dans cette thèse, nous introduisons plusieurs nouveaux domaines numériques abstraits et, en particulier, le domaine des zones (permettant de découvrir des invariants de la forme $X - Y \leq c$), des zones de congruence ($X \equiv Y + a\,[b]$) et des octogones ($\pm X \pm Y \leq c$). Ces domaines sont basés sur les concepts existants de graphe de potentiel, de matrice de différences bornées et sur l'algorithmique des plus courts chemins. Ils sont intermédiaires, en terme de précision et de coût, entre les domaines non relationnels (tel celui des intervalles), très peu précis, et les domaines relationnels classiques (tel celui des polyèdres), très coûteux. Nous les nommons « faiblement relationnels ». Nous présentons également des méthodes permettant d'appliquer les domaines relationnels à l'analyse de nombres à virgule flottante, jusqu'à présent uniquement réalisable par des domaines non relationnels, donc peu précis. Enfin, nous présentons des méthodes symboliques génériques dites de « linéarisation » et de « propagation de constantes symboliques » permettant d'améliorer la précision de tout domaine numérique, pour un surcoût réduit.

Les méthodes introduites dans cette thèse ont été intégrées à Astrée, un analyseur spécialisé dans la vérification de logiciels avioniques embarqués critiques, et se sont révélées indispensables pour prouver l'absence d'erreurs à l'exécution dans certains logiciels de commande de vol électrique des avions Airbus A340 et A380. Ces résultats expérimentaux viennent justifier l'intérêt de nos méthodes pour des cadres d'applications réelles.

Abstract

The goal of this thesis is to design techniques related to the automatic analysis of computer programs. One major application is the creation of tools to discover *bugs* before they actually happen, an important goal in a time when critical yet complex tasks are performed by computers. We will work in the *Abstract Interpretation* framework, a theory of sound approximations of program semantics. We will focus, in particular, on *numerical abstract domains* that specialise in the automatic discovery of properties of the numerical variables of programs.

In this thesis, we introduce new numerical abstract domains: the zone abstract domain (that can discover invariants of the form $X - Y \leq c$), the zone congruence domain ($X \equiv Y + a\,[b]$), and the octagon domain ($\pm X \pm Y \leq c$), among others. These domains rely on the classical notions of potential graphs, difference bound matrices, and algorithms for the shortest-path closure computation. They are in-between, in terms of cost and precision, between non-relational domains (such as the interval domain), that are very imprecise, and classical relational domains (such as the polyhedron domain), that are very costly. We will call them "weakly relational". We also introduce some methods to apply relational domains to the analysis of floating-point numbers, which was previously only possible using imprecise, non-relational, domains. Finally, we introduce the so-called "linearisation" and "symbolic constant propagation" generic symbolic methods to enhance the precision of any numerical domain, for only a slight increase in cost.

The techniques presented in this thesis have been integrated within Astrée, an analyser for critical embedded avionic software, and were instrumental in proving the absence of run-time errors in fly-by-wire softwares used in Airbus A340 and A380 planes. Experimental results show the usability of our methods in the context of real-life applications.

Acknowledgments

First of all, I would like to thank deeply my Professor and Ph.D. adviser, Patrick Cousot. He introduced me to the field of program semantics through the rewarding and enlightening way of Abstract Interpretation, and then, allowed me to join his team in the search for better abstractions. He managed to protect my autonomy while supporting my work. Patrick has always had the most respect for his students, considering them as his peer researchers. ◦ I would like to thank Chris Hankin, Roberto Giacobazzi, and Helmut Seidl for accepting to be in my jury and for their helpful comments on my work. ◦ During my thesis, I had the rare opportunity to work on a thrilling group project. ASTRÉE was at the same time a practical experiment and a theory-stressing challenge. It funnelled my passion for programming in a way supplementing my theoretical research. Its astounding and repeated successes strengthened my faith in times of disillusionment. I would like to thank all the members of the "magic team" for sharing this experience with me: Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, David Monniaux, and Xavier Rival. Additional acknowledgments go to Bruno and Patrick for their proof-reading of this thesis. ◦ The ASTRÉE project would not have been possible without the support and trust from Famantanantsoa Randimbivololona and Jean Souyris at Airbus. I am glad they actually did buy us the promised dinner. ◦ I also thank the bakers at the "Boulange V$^{\text{ème}}$", the official sandwich supplier for the "magic team". ◦ I shall not forget to thank my estimated co-workers and friends from Radhia's "sister team": Charles Hymans, Francesco Logozzo, Damien Massé, and Élodie-Jane Sims. ◦ A well-balanced life cannot include only work and I found also much support during my "off-line" time. This is why I would like to thank my parents and all my sisters: Judith, Garance, and Manuèle.

Contents

Chapter 1

Introduction

1.1 Motivation

Since the birth of computer science, writing *correct* programs has always been considered a great challenge. Generally, much more time and effort is needed to hunt down and eliminate bugs, that is, unintentional programming errors, than to actually write programs. As we rely more and more on software, the consequences of a bug are more and more dramatic, causing great financial and even human losses. Moreover, software complexity seems to follow Moore's law and grow exponentially with time, making them harder and harder to debug. Two extreme examples are the overflow bug that caused the failure of the Ariane 5 launcher in 1996 [ea96] and the cumulated imprecision errors in a Patriot missile defense that caused it to miss its Scud missile target, resulting in 28 people being killed in 1992 [Ske92].

In order to ensure the correctness of programs, one massively used technique is *testing*. As only a few sample program behaviors can actually be observed, testing methods easily leave bugs. Also, testing does not seem to catch up with the software complexity explosion: it becomes more and more costly while giving worse and worse results. Formal methods, on the other hand, try to address these problems by providing mathematically sound techniques that guarantee a full coverage of all program behaviors while relying on symbolic — as opposed to explicit — representations to achieve efficiency. In this thesis, we wish to contribute to the field of formal methods used in the verification of the correctness of programs.

1.2 Key Concepts

Program Semantics. *Semantics* is the branch of computer science devoted to associating a mathematical meaning to computer programs, thus allowing formal reasoning about

programs and their properties. *Formal methods* grounded on program semantics should be distinguished from purely empirical or statistical methods.

Static Analysis. *Static Analysis* is devoted to the conception of analysers, that is, programs able to discover properties of programs at compile-time. Unlike testing, debugging, and profiling, static analysers do not need to actually run the analysed program; they can reason about infinite sets of unbounded computations in arbitrary contexts at once. Moreover, a static analyser should preferably always terminate and its computation time should be more or less predictable. By Rice's theorem [Ric53], any property on the outcome of a program that is not always true for all programs or false for all programs — and this includes every single interesting property — is undecidable, so, a perfect static analyser cannot exist. All static analysers will thus make approximations, one way or another. A *sound* static analyser is one for which approximations do not compromise the truth of its result: it can output either a definite "yes" — meaning that the property is indeed true regardless of approximations — or "I don't know" — meaning that the approximations prevent the analyser from issuing a definite answer.

Abstract Interpretation. *Abstract Interpretation* [CC77, CC92b] is a general theory of the approximations of program semantics. It allows, among other applications, designing static analyses that are sound by construction. A core concept is that of *abstract domain*, that is, a class of properties together with a set of operators to manipulate them allowing, in conjunction with the abstract interpretation framework, the design of a static analyser for this class of properties. Each abstract domain embeds some sort of approximation. There does not exist a single, all-purpose, abstract domain: abstract domains must be chosen depending on the properties that need to be inferred, but also the language constructs that are used in the analysed program and the amount of computing resources available for the static analysis. However, an abstract domain is not tied to one particular program but to a whole class of programs as it generally embeds an *infinite* set of properties.

Advantages Over Other Formal Methods. Unlike other formal methods for reasoning on programs, once an abstract domain is designed, the static analysis performs fully automatically and directly on the source code. This is to be compared to *model-checking* that requires the user to construct by hand a — generally finite — model for each new program to be analysed and *theorem-proving* that often requires much user assistance during the proof generation.

Numerical Properties. The focus of this thesis is the *numerical properties* of the variables of a program. Such properties allow answering questions of the form: "Can there be a division by zero ?", "Can this computation exceed the digit capacity of the computer ?",

"Can this array index exceed the array bounds ?". They have quite a lot of applications in program verification and optimisation. All numerical properties, except the simplest ones, are undecidable.

Weakly Relational Numerical Abstract Domains. *Numerical* abstract domains focus on the properties of the numerical program variables. *Relational* domains are able to express relationships between variables, that is, arithmetic properties involving several variables at a time, such as "X=Y+Z". Classical relational domains are more precise but also much more costly than non-relational ones. The subject of this thesis is the introduction of new relational numerical abstract domains that are in-between, in terms of precision and cost, between existing non-relational and relational domains. As each introduced domain can be seen as the restriction, with respect to the expressed properties, of an existing relational domain, we call them *weakly relational*. The algorithms underlying these new domains are, however, quite different and allow a reduced time and memory consumption.

1.3 Overview of the Thesis

This thesis is organised as follows. Chap. 2 recalls the formal framework of Abstract Interpretation and its application to the design of static analyses for the numerical properties of programs. The construction of a generic static analyser, parametrised by a numerical abstract domain, for a simple programming language called Simple is presented. It will serve throughout the thesis to illustrate our newly introduced abstract domains. Chap. 3 then introduces our first weakly relational abstract domain: the domain of *zones* that allows discovering invariants of the form $\pm X \leq c$ and $X - Y \leq c$, where X and Y are program variables and c is any constant in $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$. Then, Chap. 4 presents the *octagon* abstract domain that extends the zone abstract domain to discover invariants of the form $\pm X \pm Y \leq c$. Chap. 5 proposes another generalisation of the zone abstract domain that allows representing invariants of the form $X - Y \in S$ where S lives in a non-relational abstraction. We will present several instances of this abstract domain family. One of particular interest is the *zone congruence* domain that can infer invariants of the form $X \equiv Y + c\,[d]$. Another example is the *strict zone* domain that can infer invariants of the form $X - Y < c$. Chap. 6 presents two generic techniques that can be applied to improve the precision of all numerical abstract domains: *linearisation* widens their scope by easily allowing a support for non-linear expression manipulations while *symbolic constant propagation* improves their robustness against simple program transformations. The last two chapters focus on the static analysis of real-life programming languages. Chap. 7 explains how to adapt numerical abstract domains to the analysis of machine-integers and floating-point numbers, which behave quite differently from the perfect mathematical numbers in $\mathbb{Z}$ and $\mathbb{R}$ that we use in our Simple language for simplicity. It also explains how to implement

abstract domains efficiently using machine-integers and floating-point numbers. Chap. 8 presents our experiments with the octagon abstract domain of Chap. 4 and the techniques of Chaps. 6 and 7 in the Astrée static analyser for real-life embedded critical avionic software.

1.4 Our Contribution

Our main goal during this research was to provide static analysis tools that are both firmly grounded mathematically and of practical interest. More precisely, our contribution can be conceived in three layered levels:

- Firstly, we give theoretical results: mathematical definitions, algorithms, and theorems, presented with their proof.
- Secondly, most of these results have been implemented in small academic analysers and tested on small program fragments written in small illustrative languages in the spirit of our Simple language. Throughout this thesis, we will illustrate the introduced techniques by such sample analyses. Moreover, an academic analyser for the octagon domain of Chap. 4 is available on-line [Minb] — see also Fig. 4.5 at the end of Chap. 4.
- Finally, a part of the theoretical work has been implemented in an industrial-strength, freely available, library [Mina] and incorporated into the Astrée static analyser [Asta]. The Astrée analyser, developed at the École Normale Supérieure, is used daily by industrial end-users to prove the absence of run-time errors in real-life critical embedded softwares for Airbus [Air] planes, written in the `C` programming language and featuring much numerical computation.

Some of the results described in Chaps. 3 to 7 have been the subject of publications in workshops and symposiums [Min01a, Min01b, Min02, Min04] and are presented here with many extensions, as well as complete proofs. We also co-wrote two papers [BCC+02, BCC+03] discussing the implementation of and experimentation with Astrée. Only the part of the work on the Astrée static analyser relevant to this thesis is described in Chap. 8.

Chapter 2

Abstract Interpretation of Numerical Properties

L'interprétation abstraite est une théorie générale de la construction et la preuve de correction d'approximations de sémantiques de programmes. Nous rappelons ses principes de base et son application à la conception d'analyseurs statiques pour l'inférence des propriétés des variables numériques d'un programme.
Ensuite, nous présentons la syntaxe et la sémantique formelle d'un petit langage, nommé Simple, qui illustre la conception par interprétation abstraite d'un analyseur statique, paramétré par un domaine abstrait numérique. Quelques exemples de domaines numériques abstraits classiques sont également rappelés. Ce langage et cet analyseur seront utilisés dans les chapitres suivants pour illustrer les propriétés des nouveaux domaines numériques abstraits introduits par cette thèse.

Abstract Interpretation is a general framework for the construction and soundness proof of approximated semantics of programs. We recall here its core features and its applications to the design of static analysers for the automatic inference of the properties of the numerical variables of a program.
We then present a very small programming language, called Simple, together with its syntax and formal semantics, and exemplify the abstract interpretation framework by designing a small static analyser parametrised by a numerical abstract domain. We also recall some classical numerical abstract domains. This language and this analyser will be used throughout the following chapters to exemplify the new numerical abstract domains introduced in this thesis.

2.1 General Definitions

Definitions, Properties. We will use the symbols $\stackrel{\text{def}}{=}$ and $\stackrel{\text{def}}{\iff}$ to introduce new objects: the left object is defined to be equal or equivalent to the formula on the right. These should not be confused with the $=$ and $\iff$ symbols that describe equality or equivalence properties on already defined objects.

Orders. A *partially ordered set* (or *poset*) $(\mathcal{D}, \sqsubseteq)$ is a non-empty set $\mathcal{D}$ together with a partial order $\sqsubseteq$, that is, a binary relation that is reflexive, transitive, and anti-symmetric. A reflexive, transitive, but not necessarily anti-symmetric relation is called a *preorder* and can be turned into a poset by identifying elements in $\mathcal{D}$ such that both $a \sqsubseteq b$ and $b \sqsubseteq a$. When they exist, the *greatest lower bound* (or *glb*) of a set $D \subseteq \mathcal{D}$ will be denoted by $\bigsqcap D$, and its *least upper bound* (or *lub*) will be denoted by $\bigsqcup D$. We will denote by $\bot$ the *least element* and $\top$ the *greatest element*, if they exist. A poset with a least element will be called a *pointed poset*, and denoted by $(\mathcal{D}, \sqsubseteq, \bot)$. Note that any poset can be transformed into a pointed poset by adding a new element that is smaller than everyone. A *cpo* is a poset that is *complete*, that is, every increasing chain of elements $(X_i)_{i \in I}$, $i \leq j \Longrightarrow X_i \sqsubseteq X_j$ (where the ordinal I may be greater than ω) has a least upper bound, $\bigsqcup_{i \in I} X_i$, which is called the *limit* of the chain. Note that a cpo is always pointed as the least element can be defined by $\bot \stackrel{\text{def}}{=} \bigsqcup \emptyset$.

Lattices. A *lattice* $(\mathcal{D}, \sqsubseteq, \sqcup, \sqcap)$ is a poset where each pair of elements $a, b \in \mathcal{D}$ has a least upper bound, denoted by $a \sqcup b$, and a greatest lower bound, denoted by $a \sqcap b$. A lattice is said to be *complete* if any set $D \subseteq \mathcal{D}$ has a least upper bound. A complete lattice is always a cpo; it has both a least element $\bot \stackrel{\text{def}}{=} \bigsqcup \emptyset$ and a greatest element $\top \stackrel{\text{def}}{=} \bigsqcup \mathcal{D}$; also, each set $D \subseteq \mathcal{D}$ has a greatest lower bound $\bigsqcap D \stackrel{\text{def}}{=} \bigsqcup \{ X \in \mathcal{D} \mid \forall Y \in D,\, X \sqsubseteq Y \}$. In this case, it is denoted by $(\mathcal{D}, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$. An important example of complete lattice is the *power-set* $(\mathcal{P}(S), \subseteq, \cup, \cap, \emptyset, S)$ of any set S.

Structure Lifting. If $(\mathcal{D}, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$ is a complete lattice (resp. poset, cpo, lattice) and S is a set, then $(S \to \mathcal{D}, \dot{\sqsubseteq}, \dot{\sqcup}, \dot{\sqcap}, \dot{\bot}, \dot{\top})$ is also a complete lattice (resp. poset, cpo, lattice) if we define the dotted operators by *point-wise lifting*:

Definition 2.1.1. Point-wise lifting.

$$
\begin{array}{lcll}
(S \to \mathcal{D},\, \dot{\sqsubseteq},\, \dot{\sqcup},\, \dot{\sqcap},\, \dot{\bot},\, \dot{\top}) & & & \\
X \mathrel{\dot{\sqsubseteq}} Y & \stackrel{\text{def}}{\iff} & \forall s \in S,\ X(s) \sqsubseteq Y(s) & \\
(\dot{\bigsqcup} \mathcal{X})(s) & \stackrel{\text{def}}{=} & \bigsqcup \{ X(s) \mid X \in \mathcal{X} \} & \qquad \dot{\bot}(s) \stackrel{\text{def}}{=} \bot \\
(\dot{\bigsqcap} \mathcal{X})(s) & \stackrel{\text{def}}{=} & \bigsqcap \{ X(s) \mid X \in \mathcal{X} \} & \qquad \dot{\top}(s) \stackrel{\text{def}}{=} \top
\end{array}
$$

●

Applications. We will sometimes use the *lambda notation* for applications: $\lambda X.expr$ is the function that maps X to the value of the expression $expr$ where X is a free variable. We will sometimes use the explicit notation $[X_1 \mapsto expr_1, \ldots, X_n \mapsto expr_n]$ to denote the application that associates the value of $expr_i$ to X_i. We will also denote by $f[X \mapsto expr]$ the application equal to f, except that it maps X to the value of $expr$ instead of $f(X)$. Finally, we denote by Id the identity application: $Id \stackrel{\text{def}}{=} \lambda X.X$. An application $F \in \mathcal{D}_1 \to \mathcal{D}_2$ between two posets $(\mathcal{D}_1, \sqsubseteq_1)$ and $(\mathcal{D}_2, \sqsubseteq_2)$ is said to be *monotonic* if $X \sqsubseteq_1 Y \Longrightarrow F(X) \sqsubseteq_2 F(Y)$. It is said to be *strict* if $F(\bot_1) = \bot_2$. An *operator* $F \in \mathcal{D} \to \mathcal{D}$, that is, a function from a poset $\mathcal{D}$ to the same poset, is said to be *extensive* if $\forall X,\ X \sqsubseteq F(X)$. A monotonic application $F \in \mathcal{D}_1 \to \mathcal{D}_2$ that preserves the existing limits of increasing chains $(X_i)_{i\in I}$, *i.e.*, $F\left(\bigsqcup_1 \{ X_i \,|\, i \in I \}\right) = \bigsqcup_2 \{ F(X_i) \,|\, i \in I \}$ whenever $\bigsqcup_1 \{ X_i \,|\, i \in I \}$ exists, is said to be *continuous*. If F preserves existing least upper bounds (resp. greatest lower bounds), *i.e.*, $F(\bigsqcup_1 D) = \bigsqcup_2 \{ F(X) \mid X \in D \}$, it is said to be a *complete* $\sqcup-$*morphism* (resp. $\sqcap-$*morphism*). Note that complete $\sqcup-$morphisms and complete $\sqcap-$morphisms are always monotonic. If F is an operator and i is an ordinal, we will denote by $F^i(X)$ F's $i-$th iterate on X which is defined by transfinite induction as $F^0(X) \stackrel{\text{def}}{=} X$, $F^{\alpha+1}(X) \stackrel{\text{def}}{=} F(F^\alpha(X))$, and $F^\beta(X) \stackrel{\text{def}}{=} \bigsqcup_{\alpha<\beta} F^\alpha(X)$ when β is a limit ordinal.

Fixpoints. A *fixpoint* of an operator F is an element X such that $F(X) = X$. We denote by $\mathrm{lfp}_X F$ the least fixpoint of F that is greater than X, if it exists, and by $\mathrm{gfp}_X F$ the greatest fixpoint of F smaller than X. We also denote by $\mathrm{lfp}\, F$ and $\mathrm{gfp}\, F$ the least and greatest fixpoints of F, if they exist: $\mathrm{lfp}\, F \stackrel{\text{def}}{=} \mathrm{lfp}_\bot F$ and $\mathrm{gfp}\, F \stackrel{\text{def}}{=} \mathrm{gfp}_\top F$. A *pre-fixpoint* X is such that $F(X) \sqsupseteq X$, and a *post-fixpoint* X is such that $F(X) \sqsubseteq X$. In particular, $\bot$ is a pre-fixpoint for all operators, and $\top$ is a post-fixpoint for all operators. We now recall two fundamental theorems about fixpoints of operators in ordered structures:

Theorem 2.1.1. Tarskian fixpoints.

The set of fixpoints of a monotonic operator F in a complete lattice is a complete lattice.
Moreover, $\mathrm{lfp}_X F = \bigsqcap \{$ *F's post-fixpoints larger than X* $\}$.
Dually, $\mathrm{gfp}_X F = \bigsqcup \{$ *F's pre-fixpoints smaller than X* $\}$.

●

Proof. By Tarski in [Tar55]. ○

Theorem 2.1.2. Kleenian fixpoints.

1. *If F is a monotonic operator in a cpo and X is a pre-fixpoint for F, then $F^i(X)$ is stationary at some ordinal ϵ and* $\mathrm{lfp}_X F = F^\epsilon(X)$.
2. *If, moreover, F is continuous, then* $\mathrm{lfp}_X F = F^\omega(X)$.

●

Proof. Found in [Cou78, § 2.5.2.0.2, § 2.7.0.1]. ○

2.2 Abstract Interpretation Primer

A core principle in the Abstract Interpretation theory is that all kinds of semantics can be expressed as fixpoints of monotonic operators in partially ordered structures, would it be operational, denotational, rule-based, axiomatic, based on rewriting systems, transition systems, abstract machines, etc. Having all semantics formalised in a unified framework allows comparing them more easily. Beside comparing already existing semantics, Abstract Interpretation allows building new semantics by applying *abstractions* to existing ones. Abstractions are themselves first-class citizens that can be manipulated and composed at will.

A key property of the semantics designed by abstraction is that they are guaranteed to be sound, by construction. Thus, a sound and fully automatic static analyser can be designed by starting from the non-computable formal semantics of a programming language, and composing abstractions until the resulting semantics is computable.

2.2.1 Galois Connection Based Abstract Interpretation

Galois Connections. Let $\mathcal{D}^\flat$ and $\mathcal{D}^\sharp$ be two posets[1] used as semantic domains. A *Galois connection*, as introduced by Cousot and Cousot in [CC77], between $\mathcal{D}^\flat$ and $\mathcal{D}^\sharp$ is a function pair (α, γ) such that:

Definition 2.2.1. Galois connection.

1. $\alpha : \mathcal{D}^\flat \to \mathcal{D}^\sharp$,
2. $\gamma : \mathcal{D}^\sharp \to \mathcal{D}^\flat$,
3. $\forall X^\flat, X^\sharp,\ \alpha(X^\flat) \sqsubseteq^\sharp X^\sharp \iff X^\flat \sqsubseteq^\flat \gamma(X^\sharp)$.

●

This is often pictured as follows:

$$\mathcal{D}^\flat \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{D}^\sharp \ .$$

An important consequence of Def. 2.2.1 is that *both α and γ are monotonic* [CC92a, § 4.2.2]. Also, we have $(\alpha \circ \gamma)(X^\sharp) \sqsubseteq^\sharp X^\sharp$ and $X^\flat \sqsubseteq^\flat (\gamma \circ \alpha)(X^\flat)$. The fact that $\alpha(X^\flat) \sqsubseteq^\sharp X^\sharp$,

[1] From now on, a poset $(\mathcal{D}^x, \sqsubseteq^x)$ will only be refereed to as $\mathcal{D}^x$ when there is no ambiguity, that is, when there is only one partial order of interest on $\mathcal{D}^x$. The same also holds for cpo, lattices, and complete lattice: the same superscript x as the one of the set $\mathcal{D}^x$ is used when talking about its order $\sqsubseteq^x$, lub $\sqcup^x$, glb $\sqcap^x$, least element $\bot^x$, and greatest element $\top^x$, when they exist.

or equivalently that $X^\flat \sqsubseteq^\flat \gamma(X^\sharp)$, will formalise the fact that $X^\sharp$ is a sound approximation (or sound *abstraction*) of $X^\flat$. $\mathcal{D}^\flat$ will be called the *concrete domain*; $\mathcal{D}^\sharp$ will be called the *abstract domain*; α will be called the *abstraction function*; γ will be called the *concretisation function*. Moreover, $\alpha(X^\flat)$ will be the *best abstraction* for $X^\flat$.

Galois Insertions. If the concretisation γ is one-to-one or, equivalently, α is onto, or $\alpha \circ \gamma = Id$, then (α, γ) is called a *Galois insertion* and pictured as follows:

$$D^\flat \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{D}^\sharp \ .$$

Designing an abstract domain linked to the concrete one through a Galois insertion corresponds to choosing, as abstract elements, a subset of the concrete ones and, as the abstract order, the order induced by the concrete domain. If we impose only the existence of a general Galois connection, then one concrete element can be represented by several, possibly incomparable, abstract elements.

Canonical Abstractions and Concretisations. Sometimes, it is not necessary to define both the concretisation and the abstraction functions in a Galois connection as the missing function can be synthesised in a canonical way. We can use the following theorem:

Theorem 2.2.1. Canonical α, γ.

1. *If $\mathcal{D}^\flat$ has lubs for arbitrary sets and $\alpha : \mathcal{D}^\flat \to \mathcal{D}^\sharp$ is a complete $\sqcup-$morphism, then there exists a unique concretisation γ that forms a Galois connection $\mathcal{D}^\flat \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{D}^\sharp$. This γ is defined as:*
$$\gamma(X) \stackrel{\text{def}}{=} \bigsqcup^\flat \{ \ Y \mid \alpha(Y) \sqsubseteq^\sharp X \ \} \ .$$
2. *Likewise, if $\mathcal{D}^\sharp$ has glbs for arbitrary sets and $\gamma : \mathcal{D}^\sharp \to \mathcal{D}^\flat$ is a complete $\sqcap-$morphism, then there exists a unique α that forms a Galois connection $\mathcal{D}^\flat \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{D}^\sharp$. It is defined as:*
$$\alpha(X) \stackrel{\text{def}}{=} \bigsqcap^\sharp \{ \ Y \mid X \sqsubseteq^\flat \gamma(Y) \ \} \ .$$

●

Proof. See [CC92a, § 4.2.2]. ○

Operator Abstractions. Let $F^\flat$ be an operator on a concrete domain. An operator $F^\sharp$ on the abstract domain will be said to be a *sound abstraction* for $F^\flat$ with respect to a Galois connection $\mathcal{D}^\flat \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{D}^\sharp$ if and only if $\forall X,\ F^\sharp(X) \sqsupseteq^\sharp (\alpha \circ F^\flat \circ \gamma)(X)$ or, equivalently, $\forall X,\ (\gamma \circ F^\sharp)(X) \sqsupseteq^\flat (F^\flat \circ \gamma)(X)$. Given $F^\flat$, the *best abstraction* $F^\sharp$ for $F^\flat$ is defined to

be exactly $F^\sharp \stackrel{\text{def}}{=} \alpha \circ F^\flat \circ \gamma$. When $\gamma \circ F^\sharp = F^\flat \circ \gamma$, the operator $F^\sharp$ is said to be an *exact abstraction* for $F^\flat$. Exact abstractions seldom exist because it is quite rare that the result of a concrete operator $F^\flat$ is exactly representable in the abstract domain, even when its argument is. However, if the Galois connection is a Galois insertion and an exact abstraction for $F^\flat$ exists, then it is unique and it corresponds to the best abstraction for $F^\flat$.

Note that all these definitions and properties apply to $n-$ary operators as well.

Given a *monotonic* concrete operator $F_1^\flat$ and a concrete operator $F_2^\flat$, with corresponding sound abstractions $F_1^\sharp$ and $F_2^\sharp$, the composition $F_1^\sharp \circ F_2^\sharp$ is a sound abstraction of the concrete composition $F_1^\flat \circ F_2^\flat$. Also, the composition of exact abstractions of — not necessarily monotonic — operators is an exact abstraction of the composition. However, the composition of the best abstractions of two operators is seldom the best abstraction of the composition of the operators. So, when designing the abstraction of a complex function by composing abstractions of small functions, the chosen "granularity" can greatly affect the quality of an analysis.

Given two Galois connections $\mathcal{D}^\flat \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \mathcal{D}_1^\sharp$ and $\mathcal{D}^\flat \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \mathcal{D}_2^\sharp$ from the same concrete domain $\mathcal{D}^\flat$, we will say that $\mathcal{D}_1^\sharp$ is a more precise abstraction than $\mathcal{D}_2^\sharp$ if and only if $\gamma_1(\mathcal{D}_1^\sharp) \supseteq \gamma_2(\mathcal{D}_2^\sharp)$, that is, if $\mathcal{D}_1^\sharp$ can represent exactly more concrete elements than $\mathcal{D}_2^\sharp$. When this is the case, every best abstraction in $\mathcal{D}_1^\sharp$ of a monotonic operator $F^\flat$ is more precise than the corresponding best abstraction in $\mathcal{D}_2^\sharp$, which is stated as follows:

Theorem 2.2.2. Relative precision of abstract domains.

If $\gamma_1(\mathcal{D}_1^\sharp) \supseteq \gamma_2(\mathcal{D}_2^\sharp)$ *and* $F^\flat : \mathcal{D}^\flat \to \mathcal{D}^\flat$ *is monotonic, then:*

$$\gamma_1(X_1^\sharp) \sqsubseteq^\flat \gamma_2(X_2^\sharp) \Longrightarrow \gamma_1((\alpha_1 \circ F^\flat \circ \gamma_1)(X_1^\sharp)) \sqsubseteq^\flat \gamma_2((\alpha_2 \circ F^\flat \circ \gamma_2)(X_2^\sharp)) \ .$$

●

Proof.

We first prove that $X_1^\flat \sqsubseteq^\flat X_2^\flat \Longrightarrow (\gamma_1 \circ \alpha_1)(X_1^\flat) \sqsubseteq^\flat (\gamma_2 \circ \alpha_2)(X_2^\flat)$ (1). By Galois connection properties, $(\gamma_2 \circ \alpha_2)(X_2^\flat) \sqsupseteq^\flat X_2^\flat$. Using the hypothesis, this gives $(\gamma_2 \circ \alpha_2)(X_2^\flat) \sqsupseteq^\flat X_1^\flat$. Now, by applying the monotonic $\gamma_1 \circ \alpha_1$, we get $(\gamma_1 \circ \alpha_1 \circ \gamma_2 \circ \alpha_2)(X_2^\flat) \sqsupseteq^\flat (\gamma_1 \circ \alpha_1)(X_1^\flat)$. Because $(\gamma_2 \circ \alpha_2)(X_2^\flat) \in \gamma_2(\mathcal{D}_2^\sharp)$, we also have $(\gamma_2 \circ \alpha_2)(X_2^\flat) \in \gamma_1(\mathcal{D}_1^\sharp)$, so there exists some $Y_1^\sharp$ such that $(\gamma_2 \circ \alpha_2)(X_2^\flat) = \gamma_1(Y_1^\sharp)$. By property of Galois connections $\gamma_1 \circ \alpha_1 \circ \gamma_1 = \gamma_1$, so $(\gamma_1 \circ \alpha_1 \circ \gamma_2 \circ \alpha_2)(X_2^\flat) = (\gamma_1 \circ \alpha_1 \circ \gamma_1)(Y_1^\sharp) = \gamma_1(Y_1^\sharp) = (\gamma_2 \circ \alpha_2)(X_2^\flat)$ and we have actually proved that $(\gamma_2 \circ \alpha_2)(X_2^\flat) \sqsupseteq^\flat (\gamma_1 \circ \alpha_1)(X_1^\flat)$.

If we now define $X_1^\flat \stackrel{\text{def}}{=} (F^\flat \circ \gamma_1)(X_1^\sharp)$ and $X_2^\flat \stackrel{\text{def}}{=} (F^\flat \circ \gamma_2)(X_2^\sharp)$, then the hypothesis and the monotonicity of $F^\flat$ imply that $X_1^\flat \sqsubseteq^\flat X_2^\flat$ and we can apply (1) to get the desired result.

○

Thus, a Galois connection is sufficient to fully characterise the *maximal* precision of

an abstract domain: it hints on how much precision is lost, at least, by computing in the abstract world $\mathcal{D}^\sharp$ instead of the concrete world $\mathcal{D}^\flat$. More precision may be lost when choosing operator abstractions that are not best ones.

The Category of Galois Connections. A main property of Galois connections is that they can be composed. Given three posets, $\mathcal{D}^\flat$, $\mathcal{D}^\natural$, and $\mathcal{D}^\sharp$ linked by Galois connections as follows:

$$\mathcal{D}^\flat \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \mathcal{D}^\natural \qquad\qquad \mathcal{D}^\natural \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \mathcal{D}^\sharp$$

then, $(\alpha_2 \circ \alpha_1, \gamma_1 \circ \gamma_2)$ is also a Galois connection:

$$\mathcal{D}^\flat \underset{\alpha_2\circ\alpha_1}{\overset{\gamma_1\circ\gamma_2}{\leftrightarrows}} \mathcal{D}^\sharp \ .$$

Hence, posets linked by Galois connections form a category. Moreover, the notion of sound, best, and exact abstractions of operators is transitive with respect to chained Galois connections, which allows a design based on successive abstractions.

We have seen — and we will see more shortly — properties of operator abstractions that are only true when they abstract *monotonic* concrete operators. In fact, as Galois connections compose, it is sufficient to relate abstract operators to monotonic concrete ones through a chain of Galois connections. Generally, all the semantic operators defined on the most concrete domain are monotonic while their successive abstractions are not.

2.2.2 Concretisation-Based Abstract Interpretation

Imposing the existence of a Galois connection between the concrete and the abstract domains is sometimes too strong a requirement and we will see several abstraction examples for which no α function exists (see Sects. 2.4.6 and 2.4.7). In [CC92b], Cousot and Cousot explain how to relax the Galois connection framework in order to work only with a concretisation operator γ — or, dually, only an abstraction operator α; however, concretisation-based abstract interpretation is much more used in practice.

Concretisations. Let $\mathcal{D}^\flat$ and $\mathcal{D}^\sharp$ be two posets. A concretisation is simply a monotonic function $\gamma : \mathcal{D}^\sharp \to D^\flat$. $X^\sharp$ is said to be an *abstraction* for $X^\flat$ if $\gamma(X^\sharp) \sqsupseteq^\flat X^\flat$. Note that, if there is no simple way to compare two abstract elements, we can always fall back to the "coarse" partial order defined by $X^\sharp \sqsubseteq^\sharp Y^\sharp \iff^{\text{def}} X^\sharp = Y^\sharp$ which makes every γ function automatically monotonic.

Operator Abstractions Revisited. The abstract operator $F^\sharp$ is said to be a *sound abstraction* for the concrete operator $F^\flat$ if and only if $\forall X^\sharp$, $(\gamma \circ F^\sharp)(X^\sharp) \sqsupseteq^\flat (F^\flat \circ \gamma)(X^\sharp)$. It

is an *exact abstraction* if, moreover, $\gamma \circ F^\sharp = F^\flat \circ \gamma$. Note that sound and exact abstractions compose as in the Galois connection case.

Because there is no α function, there is no notion of a best abstraction for a concrete element and we cannot define the best abstraction for an operator as we did in the Galois connection case. If the concrete domain $\mathcal{D}^\flat$ has arbitrary glbs, an alternate definition could be that $F^\sharp$ is an *optimal abstraction* of $F^\flat$ if and only if $(\gamma \circ F^\sharp)(X^\sharp) = \bigsqcap^\flat \{ \gamma(Y^\sharp) \mid \gamma(Y^\sharp) \sqsupseteq^\flat (F^\flat \circ \gamma)(X^\sharp) \}$. Such an $F^\sharp$ does not always exist and, when it exists, might not be unique. Also, this is only a characterisation while, in the Galois connection case, $F^\sharp$ could be *derived automatically* from the definitions of α, γ, and $F^\flat$ as $\alpha \circ F^\flat \circ \gamma$. Another consequence is that there is no easy way of telling whether one abstraction is more precise than another just by comparing the set of concrete elements that can be exactly represented in the abstract posets.

2.2.3 Partial Galois Connections

There exists concretisation-based abstractions that "almost" enjoy a Galois connection property and for which we would like to have a unequivocal definition of the best abstraction of an operator. One solution, proposed by Cousot and Cousot in [CC92b], is to enrich the abstract domain with new elements or, on the contrary, to deprive it of some. We propose here a new alternative, which consists in requiring best abstractions only for a set of concrete transfer functions.

Galois Connections Revisited. Let $\mathcal{D}^\flat$ and $\mathcal{D}^\sharp$ be two posets. Let $\mathcal{F}$ be a set of *concrete* operators of arbitrary arity. We define the notion of *$\mathcal{F}-$partial Galois connection* as follows:

Definition 2.2.2. Partial Galois connection.

The pair (α, γ) is an $\mathcal{F}-$partial Galois connection if and only if:

1. $\alpha : \mathcal{D}^\flat \rightharpoonup \mathcal{D}^\sharp$ *is a monotonic* partial *function, and*
2. $\gamma : \mathcal{D}^\sharp \to \mathcal{D}^\flat$ *is a monotonic total function, and*
3. $\forall F^\flat \in \mathcal{F}, X_1^\sharp \ldots, X_n^\sharp \in \mathcal{D}^\sharp$, *where n is the arity of $F^\flat$, $\alpha(F^\flat(\gamma(X_1^\sharp), \ldots, \gamma(X_n^\sharp)))$ exists, and*
4. $\forall X^\flat, X^\sharp$ *such that $\alpha(X^\flat)$ is defined, $\alpha(X^\flat) \sqsubseteq^\sharp X^\sharp \iff X^\flat \sqsubseteq^\flat \gamma(X^\sharp)$.*

●

We will use the same notation $\mathcal{D}^\flat \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{D}^\sharp$ for partial Galois connections as for regular Galois connections and precise separately the concrete operator set $\mathcal{F}$.

The concretisation-based framework can be retrieved by choosing $\mathcal{F} \stackrel{\text{def}}{=} \emptyset$. In order to retrieve a full Galois connection, it is sufficient to put every constant function in $\mathcal{F}$. Another interesting case is when $Id \in \mathcal{F}$. It means that $\alpha \circ \gamma$ is always defined, that is, every abstract element has a best — *i.e.*, smallest for $\sqsubseteq^\sharp$ — abstract representation. In particular, when $Id \in \mathcal{F}$ and $\alpha \circ \gamma = Id$, we have a *partial Galois insertion.*

Operator Abstractions Revisited. Due to our definition, each operator $F^\flat \in \mathcal{F}$ has a best abstraction $F^\sharp \stackrel{\text{def}}{=} \alpha \circ F^\flat \circ \gamma$. As the partial Galois connection framework is strictly stronger than the concretisation one, we can still use the definition of a sound abstraction for *any* operator $F^\flat$ as being an operator $F^\sharp$ such that $\forall X, (\gamma \circ F^\sharp)(X) \sqsupseteq^\flat (F^\flat \circ \gamma)(X)$, and an exact abstraction if $\gamma \circ F^\sharp = F^\flat \circ \gamma$. As in the Galois connection case, abstractions and exact abstractions compose, but not best abstractions.

Finally, given two $\mathcal{F}-$partial Galois connection $\mathcal{D}^\flat \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \mathcal{D}_1^\sharp$ and $\mathcal{D}^\flat \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \mathcal{D}_2^\sharp$ from the same concrete domain $\mathcal{D}^\flat$, the precision of the two abstract domains with respect to all functions in $\mathcal{F}$ can be compared by merely comparing $\gamma_1(\mathcal{D}_1^\sharp)$ and $\gamma_2(\mathcal{D}_2^\sharp)$: Thm. 2.2.2 applies for every function $F^\flat \in \mathcal{F}$.

Canonical Abstractions α. Recall that Thm. 2.2.1 states that when $\mathcal{D}^\sharp$ has glbs for arbitrary sets and γ is a complete $\sqcap-$morphism, there exists a canonical α. In order to derive from γ a canonical partial α such that $\mathcal{D}^\flat \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{D}^\sharp$ forms a partial Galois connection, it is only necessary for *a few glbs* to exist and be preserved by γ:

Theorem 2.2.3. Canonical partial α.

If for each $F^\flat \in \mathcal{F}$ and $X_1^\sharp, \ldots, X_n^\sharp \in \mathcal{D}^\sharp$ (where n is the arity of F)

$$X^\sharp \stackrel{\text{def}}{=} \bigsqcap^\sharp \{ Y^\sharp \mid F^\flat(\gamma(X_1^\sharp), \ldots, \gamma(X_n^\sharp)) \sqsubseteq^\flat \gamma(Y^\sharp) \}$$

is well-defined and moreover

$$\gamma(X^\sharp) = \bigsqcap^\flat \{ \gamma(Y^\sharp) \mid F^\flat(\gamma(X_1^\sharp), \ldots, \gamma(X_n^\sharp)) \sqsubseteq^\flat \gamma(Y^\sharp) \}$$

then the partial function defined by:

$$\alpha(X^\flat) \stackrel{\text{def}}{=} \bigsqcap^\sharp \{ Y^\sharp \mid X^\flat \sqsubseteq^\flat \gamma(Y^\sharp) \}$$

forms with γ a $\mathcal{F}-$partial Galois connection.

●

Proof. Almost identical to that of Thm. 2.2.1. ○

An interesting remark is that, if α is defined this way, then the best abstraction for every $F^\flat \in \mathcal{F}$ will be $F^\sharp(X^\sharp) \stackrel{\text{def}}{=} (\alpha \circ F^\flat \circ \gamma)(X^\sharp) = \bigsqcap^\sharp \{ Y^\sharp \mid (F^\flat \circ \gamma)(X^\sharp) \sqsubseteq^\flat \gamma(Y^\sharp) \}$

and so we have $(\gamma \circ F^\sharp)(X^\sharp) = \bigsqcap^\flat \{ \gamma(Y^\sharp) \mid (F^\flat \circ \gamma)(X^\sharp) \sqsubseteq^\flat \gamma(Y^\sharp) \}$ which was precisely our loose characterisation of an optimal abstraction in the concretisation-based framework. However, $F^\sharp$ is now uniquely defined because a choice of an abstract element among others with the same concretisation is done by α.

Composing Partial Galois Connections. Let us assume that we have a $\mathcal{F}_1-$partial Galois connection $\mathcal{D}^\flat \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \mathcal{D}^\natural$ and a $\mathcal{F}_2-$partial Galois connection $\mathcal{D}^\natural \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \mathcal{D}^\sharp$. In order for $\mathcal{D}^\flat \underset{\alpha_2 \circ \alpha_1}{\overset{\gamma_1 \circ \gamma_2}{\leftrightarrows}} \mathcal{D}^\sharp$ to still be a $\mathcal{F}_1-$partial Galois connection it is sufficient to have $\mathcal{F}_2 \supseteq \{ \alpha_1 \circ F^\flat \circ \gamma_1 \mid F^\flat \in \mathcal{F}_1 \}$.

2.2.4 Fixpoint Computation

Fixpoint Transfer. Let us first consider the case of an abstract and a concrete domain linked by a Galois connection. Given a monotonic $F^\flat$ on the concrete domain and a monotonic abstraction $F^\sharp$ of $F^\flat$, we can compare the fixpoints of these operators using one of the following two theorems:

Theorem 2.2.4. Tarskian fixpoint transfer.

If $\mathcal{D}^\flat$ and $\mathcal{D}^\sharp$ are complete lattices, then $\mathrm{lfp}_{\gamma(X)} F^\flat \sqsubseteq^\flat \gamma(\mathrm{lfp}_X F^\sharp)$.

●

Proof. See Thm. 2 in [Cou02]. ○

Theorem 2.2.5. Kleenian fixpoint transfer.

1. *If $\mathcal{D}^\flat$ and $\mathcal{D}^\sharp$ are cpos and γ is continuous, then* $\mathrm{lfp}_{\gamma(X)} F^\flat \sqsubseteq^\flat \gamma(\mathrm{lfp}_X F^\sharp)$.
2. *If, moreover, $F^\sharp$ is exact, then* $\mathrm{lfp}_{\gamma(X)} F^\flat = \gamma(\mathrm{lfp}_X F^\sharp)$.

●

Proof. This is similar to Thms. 1 and 3 in [Cou02]. ○

Exact Fixpoint Computation. We now suppose that we have designed an abstract domain where all elements are computer-representable and all the operator abstractions we need are computable. We still need a practical way to compute fixpoints in the abstract domain.

When the abstract domain does not have any infinite chain of elements in strictly increasing order, then Kleene's theorem (Thm. 2.1.2) gives a constructive way to compute the least fixpoint abstraction:

Theorem 2.2.6. Kleenian iterations in domains with no infinite increasing chain.

If $F^\sharp$ is monotonic and $\mathcal{D}^\sharp$ has no infinite strictly increasing chain, then the Kleene iterations $X^\sharp_{i+1} \stackrel{\text{def}}{=} F^\sharp(X^\sharp_i)$ converge in finite time *towards* $\text{lfp}_{X^\sharp_0} F^\sharp$.

●

Proof. Consequence of Thm. 2.1.2 (the ordinal ϵ being finite). ○

The simplest case where Thm. 2.2.6 applies is when the abstract domain $\mathcal{D}^\sharp$ is finite, such as the *sign domain* — see Fig. 2.9 and [CC76]. A more complex case is when $\mathcal{D}^\sharp$ is infinite but satisfies the so-called *ascending chain condition*, such as the *constant domain* — see Fig. 2.9 and [Kil73].

When dealing with domains containing infinite increasing chains, one cannot guarantee the termination of the iterations except in rare cases. Two notable exceptions are presented in [SW04] and [RCK04b]: the authors introduce non-standard iteration schemes as well as sufficient conditions on the abstract functions for the iterates to converge in finite time within, respectively, the interval domain and the domain of varieties.

Approximate Fixpoint Computation. A more general way of dealing with domains with infinite increasing chains is the so-called *widening* $\triangledown^\sharp$ operator introduced by Cousot and Cousot in [CC76].

Definition 2.2.3. Widening.

An abstract binary operator $\triangledown^\sharp$ is a widening if and only if:

1. $\forall X^\sharp, Y^\sharp \in \mathcal{D}^\sharp,\ (X^\sharp \triangledown^\sharp Y^\sharp) \sqsupseteq^\sharp X^\sharp, Y^\sharp$, *and*
2. *for every chain $(X^\sharp_i)_{i\in\mathbb{N}}$, the increasing chain $(Y^\sharp_i)_{i\in\mathbb{N}}$ defined by*

$$\begin{cases} Y^\sharp_0 & \stackrel{\text{def}}{=} & X^\sharp_0 \\ Y^\sharp_{i+1} & \stackrel{\text{def}}{=} & Y^\sharp_i \triangledown^\sharp X^\sharp_{i+1} \end{cases}$$

is stable after a finite time, i.e., $\exists i < \omega,\ Y^\sharp_{i+1} = Y^\sharp_i$.

●

The widening operator allows computing, in finite time, an over-approximation of a Kleenian fixpoint, thanks to the following theorem:

Theorem 2.2.7. Fixpoint approximation with widening.

If $F^\flat$ is a monotonic operator in a complete lattice, $F^\sharp$ is an abstraction of $F^\flat$, and $\gamma(X^\sharp)$ is a pre-fixpoint for $F^\flat$, then the chain $(Y^\sharp_i)_{i\in\mathbb{N}}$ defined by:

$$\begin{cases} Y_0^\sharp & \stackrel{\text{def}}{=} & X^\sharp \\ Y_{i+1}^\sharp & \stackrel{\text{def}}{=} & Y_i^\sharp \ \triangledown^\sharp \ F^\sharp(Y_i^\sharp) \end{cases}$$

reaches an iterate $Y_n^\sharp$ *such that* $F^\sharp(Y_n^\sharp) \ \sqsubseteq^\sharp \ Y_n^\sharp$ *in finite time; moreover, we have* $\gamma(Y_n^\sharp) \ \sqsupseteq^\flat \ \mathrm{lfp}_{\gamma(X^\sharp)} F^\flat$.

●

Proof. See, for instance, Prop. 33 in [CC92c]. ○

Generally, one iterates from some $X^\sharp$ such that $\gamma(X^\sharp) = \bot^\flat$ to get an abstraction of $\mathrm{lfp}\, F^\flat$ (as $\bot^\flat$ is always a pre-fixpoint for $F^\flat$).

Note that unlike $F^\flat$, $F^\sharp$ need not be monotonic. On the other hand, the output of the iterations with widening is generally not a monotonic function of $X^\sharp$, even when $F^\sharp$ is monotonic, while $\mathrm{lfp}_{X^\flat} F^\flat$ is monotonic in $X^\flat$. It is perfectly safe to approximate nested limits of monotonic operators by nested iterations with widening. Also, unlike Thms. 2.2.4–2.2.6, we do not require to have a Galois connection: iterations with widening work in the partial Galois connection and the concretisation-based frameworks. When using a widening, there is no longer a way to ensure that we compute the best possible abstraction of a fixpoint.

It is always possible to refine the fixpoint over-approximation $Y^\sharp$ by applying the $F^\sharp$ operator some more without widening: while $\gamma((F^\sharp)^k(Y^\sharp)) \sqsubseteq^\flat \gamma(Y^\sharp)$, $(F^\sharp)^k(Y^\sharp)$ is a better fixpoint over-approximation than $Y^\sharp$. An even better idea would be to iterate a sound counterpart of $F^\flat(\gamma(Y^\sharp)) \sqcap^\flat \gamma(Y^\sharp)$ which decreases more rapidly while still being a sound approximation of the same least fixpoint. Unfortunately, none of these techniques are guaranteed to terminate as $\mathcal{D}^\sharp$ can have infinite strictly *decreasing* chains. In [CC76], Cousot and Cousot propose a so-called *narrowing* operator $\triangle^\sharp$ to drive decreasing iterations to a stable point in finite time:

Definition 2.2.4. Narrowing.

An abstract binary operator $\triangle^\sharp$ *is a narrowing if and only if:*

1. $\forall X^\sharp, Y^\sharp \in \mathcal{D}^\sharp$, $(X^\sharp \sqcap^\sharp Y^\sharp) \sqsubseteq^\sharp (X^\sharp \ \triangle^\sharp \ Y^\sharp) \sqsubseteq^\sharp X^\sharp$, *and*
2. *for every chain* $(X_i^\sharp)_{i\in\mathbb{N}}$, *the chain* $(Y_i^\sharp)_{i\in\mathbb{N}}$ *defined by:*

$$\begin{cases} Y_0^\sharp & \stackrel{\text{def}}{=} & X_0^\sharp \\ Y_{i+1}^\sharp & \stackrel{\text{def}}{=} & Y_i^\sharp \ \triangle^\sharp \ X_{i+1}^\sharp \end{cases}$$

is ultimately stationary after a finite time, i.e., $\exists i < \omega$, $Y_{i+1}^\sharp = Y_i^\sharp$.

●

Thanks to the following theorem, one can refine an approximate fixpoint using decreasing iterations with narrowing that are guaranteed to terminate, even when the domain has infinite decreasing chains:

Theorem 2.2.8. Fixpoint refinement with narrowing.

If $F^\flat$ is a monotonic operator on a complete lattice, $F^\sharp$ is an abstraction of $F^\flat$, and $Y^\sharp$ is such that $\gamma(Y^\sharp) \sqsupseteq^\flat \mathrm{lfp}_{\gamma(X^\sharp)} F^\flat$, then the chain $(Z_i^\sharp)_{i\in\mathbb{N}}$ defined by:

$$\begin{cases} Z_0^\sharp & \stackrel{\text{def}}{=} & Y^\sharp \\ Z_{i+1}^\sharp & \stackrel{\text{def}}{=} & Z_i^\sharp \mathbin{\triangle^\sharp} F^\sharp(Z_i^\sharp) \end{cases}$$

reaches in finite time a stable iterate $Z_n^\sharp$, that is $Z_{n+1}^\sharp = Z_n^\sharp$; moreover, we have $\gamma(Z_n^\sharp) \sqsupseteq^\flat \mathrm{lfp}_{\gamma(X^\sharp)} F^\flat$.

●

Proof. See, for instance, Prop. 34 in [CC92c]. ○

Informally, widenings and narrowings are responsible for the inductive part of the semantics computation. Hence, they embed a great deal of the analysis intelligence. Widenings and narrowings can still be useful in domains with no strictly infinite ascending or descending chains; they can be designed to trade precision for efficiency and compute a fixpoint abstraction with much fewer steps than the classical iteration *à la Kleene* presented in Def. 2.2.6. As discovered by Cousot and Cousot in [CC92c], more precision can always be obtained by using ad-hoc widening and narrowing operators on an abstract domain with infinite chains than by further abstracting this domain into a domain verifying the ascending chain condition. However, the widening and narrowing operators must be tailored toward the inductive properties one wishes to discover.

2.2.5 Chaotic Iterations

Frequently, the concrete domain is the point-wise lifting $\mathcal{L} \to \mathcal{D}^\flat$ of an ordered structure $\mathcal{D}^\flat$ to a finite set of *labels* $\mathcal{L}$ — such as, the set of syntactic program points — and the semantic function $F^\flat$ can be decomposed as several monotonic components $F_1^\flat, \ldots, F_n^\flat \in (\mathcal{L} \to \mathcal{D}^\flat) \to \mathcal{D}^\flat$, with $n = |\mathcal{L}|$. The least fixpoint of $F^\flat$ can be seen as the least solution of the following equation system:

$$\begin{cases} X_1^\flat & = & F_1^\flat(X_1^\flat, \ldots, X_n^\flat) \\ & & \vdots \\ X_n^\flat & = & F_n^\flat(X_1^\flat, \ldots, X_n^\flat) \end{cases}$$

Given a sound counterpart $F_i^\sharp \in (\mathcal{L} \to \mathcal{D}^\sharp) \to \mathcal{D}^\sharp$ for each $F_i^\flat$, we wish to compute an abstraction of the solution of the concrete equations. *Chaotic iterations with widening* is a technique proposed in [Cou78, § 4.1.2] that extends the iterations with widening seen in Thm. 2.2.7. The main idea is to choose a set of *widening points* $W \subseteq \mathcal{L}$ and an unbounded sequence of components $(L_i)_{i\in\mathbb{N}}$, $L_i \in \mathcal{L}$ that is fair — *i.e.*, each element from $\mathcal{L}$ appears infinitely often in the sequence — and compute the following sequence of abstract vectors:

Definition 2.2.5. Chaotic iterations with widening.

$$X_k^{\sharp\, i+1} \stackrel{\text{def}}{=} \begin{cases} X_k^{\sharp\, i} & \text{if } k \neq L_i \text{ or } F_k^\sharp(X_1^{\sharp\, i}, \ldots, X_n^{\sharp\, i}) \sqsubseteq^\sharp X_k^{\sharp\, i} \\ X_k^{\sharp\, i} \mathbin{\triangledown^\sharp} F_k^\sharp(X_1^{\sharp\, i}, \ldots, X_n^{\sharp\, i}) & \text{otherwise if } k \in W \\ X_k^{\sharp\, i} \cup^\sharp F_k^\sharp(X_1^{\sharp\, i}, \ldots, X_n^{\sharp\, i}) & \text{otherwise} \end{cases}$$

●

An important notion is that of *dependency graph*, which is a directed graph with nodes in $\mathcal{L}$ and an arc between nodes i and j whenever $F_j^\sharp$ actually depends upon $X_i^\sharp$. Cycles in the dependency graph incur feed-back, and so, special conditions must be ensured to get the convergence of the abstract computation, *i.e.*, each cycle must be broken by a widening application $\triangledown^\sharp$. On other nodes, it is safe to use any abstraction $\cup^\sharp$ of the concrete lub operator $\sqcup^\flat$ instead. We have the following theorem:

Theorem 2.2.9. Chaotic iterations with widening.

If each dependency cycle between the $F_i^\sharp$ has a node in W, $(L_i)_{i\in\mathbb{N}}$ is a fair sequence, and $(\gamma(X_0^{\sharp\, 0}), \ldots, \gamma(X_n^{\sharp\, 0}))$ is a pre-fixpoint for $F^\flat$, then the chaotic iterations converge in finite time towards an over-approximation of the least solution of the concrete equation system larger than $(\gamma(X_0^{\sharp\, 0}), \ldots, \gamma(X_n^{\sharp\, 0}))$.

●

Proof. See [Cou78, § 4.1.2.0.6]. ○

The precise choice of W and $(L_i)_{i\in\mathbb{N}}$ is a parameter of the analysis that influences both the convergence speed and the precision of the obtained result. Note that the chaotic iteration scheme is much more flexible than plain iterations with widening presented in Thm. 2.2.7 that would require the use of a global widening on $\mathcal{L} \to \mathcal{D}^\flat$ as well as the complete recomputation of all components of $F^\flat$ at each iteration step. The choice of the W and $(L_i)_{i\in\mathbb{N}}$ parameters has been extensively studied by Bourdoncle in [Bou93b].

It is also possible to perform decreasing iterations with narrowing to refine the computed abstract solution. As for the widening sequence, the narrowing operator $\triangle^\sharp$ needs only to be

applied at selected points that break equation dependencies and, elsewhere, we can safely use any abstraction $\cap^\sharp$ of the concrete glb $\sqcap^\flat$ instead.

Definition 2.2.6. Chaotic iterations with narrowing.

$$Y_k^{\sharp\, i+1} \stackrel{\text{def}}{=} \begin{cases} Y_k^{\sharp\, i} & \text{if } k \neq L_i \\ Y_k^{\sharp\, i} \mathbin{\triangle^\sharp} F_k^\sharp(Y_1^{\sharp\, i}, \ldots, Y_n^{\sharp\, i}) & \text{otherwise if } k \in W \\ Y_k^{\sharp\, i} \cap^\sharp F_k^\sharp(Y_1^{\sharp\, i}, \ldots, Y_n^{\sharp\, i}) & \text{otherwise} \end{cases}$$

●

Theorem 2.2.10. Chaotic iterations with narrowing.

If each dependency cycle between the $F_i^\sharp$ has a node in W, $(L_i)_{i\in\mathbb{N}}$ is a fair sequence, and $(Y_0^{\sharp\, 0}, \ldots, Y_n^{\sharp\, 0})$ is an abstraction of a solution of the concrete equation system, then the chaotic iterations converge in finite time towards a better abstraction of the same concrete solution.

●

Proof. See [Cou78, § 4.1.2.0.17]. ○

2.2.6 Reduced Product

Given a concrete domain $\mathcal{D}^\flat$ and two abstract domains $\mathcal{D}_1^\sharp$ and $\mathcal{D}_2^\sharp$ linked to $\mathcal{D}^\flat$ through monotonic concretisations $\gamma_1 : \mathcal{D}_1^\sharp \to \mathcal{D}^\flat$ and $\gamma_2 : \mathcal{D}_2^\sharp \to \mathcal{D}^\flat$, we can consider the *product domain* $\mathcal{D}_1^\sharp \times \mathcal{D}_2^\sharp$, ordered by the component-wise comparison $\sqsubseteq_{1\times 2}^\sharp$ defined by:

$$(X_1^\sharp, X_2^\sharp) \sqsubseteq_{1\times 2}^\sharp (Y_1^\sharp, Y_2^\sharp) \iff^{\text{def}} X_1^\sharp \sqsubseteq_1^\sharp Y_1^\sharp \text{ and } X_2^\sharp \sqsubseteq_2^\sharp Y_2^\sharp$$

and with concretisation $\gamma_{1\times 2}$ defined by:

$$\gamma_{1\times 2}(X_1^\sharp, X_2^\sharp) \stackrel{\text{def}}{=} \gamma_1(X_1^\sharp) \sqcap^\flat \gamma_2(X_2^\sharp)$$

which is obviously monotonic for $\sqsubseteq_{1\times 2}^\sharp$.

Abstracting Operators. If $F_1^\sharp$ and $F_2^\sharp$ are sound abstractions in $\mathcal{D}_1^\sharp$ and $\mathcal{D}_2^\sharp$ for an operator $F^\flat$ in $\mathcal{D}^\flat$, then the *product* operator $F_{1\times 2}^\sharp$ defined component-wise by:

$$F_{1\times 2}^\sharp(X_1^\sharp, X_2^\sharp) \stackrel{\text{def}}{=} (F_1^\sharp(X_1^\sharp), F_2^\sharp(X_2^\sharp))$$

is a sound abstraction for $F^\flat$ in $\mathcal{D}_1^\sharp \times \mathcal{D}_2^\sharp$ and, if both $F_1^\sharp$ and $F_2^\sharp$ are exact, then it is also exact.

We now suppose that we have (partial) Galois connections $\mathcal{D}^\flat \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \mathcal{D}_1^\sharp$ and $\mathcal{D}^\flat \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \mathcal{D}_2^\sharp$ that allow defining a concept of *best* abstraction. Then the function $\alpha_{1\times 2}$ defined component-wise by:

$$\alpha_{1\times 2}(X) \overset{\text{def}}{=} (\alpha_1(X), \alpha_2(X))$$

allows forming a Galois connection $\mathcal{D}^\flat \underset{\alpha_{1\times 2}}{\overset{\gamma_{1\times 2}}{\leftrightarrows}} \mathcal{D}_{1\times 2}^\sharp$. The function $\rho \overset{\text{def}}{=} \alpha_{1\times 2} \circ \gamma_{1\times 2}$ that computes the best representation of an abstract pair is of particular interest as it allows propagating information between the two abstract domains to refine both abstract elements at once. If $(Y_1^\sharp, Y_2^\sharp) = \rho(X_1^\sharp, X_2^\sharp)$, we can have $\gamma_1(Y_1^\sharp) \sqsubset^\flat \gamma_1(X_1^\sharp)$ or $\gamma_2(Y_2^\sharp) \sqsubset^\flat \gamma_2(X_2^\sharp)$ or even both. We call ρ a *reduction*; it is a lower closure operator in $\mathcal{D}_{1\times 2}^\sharp$. An application of the reduction is to replace $F_{1\times 2}^\sharp$ with $G_{1\times 2}^\sharp \overset{\text{def}}{=} \rho \circ F_{1\times 2}^\sharp \circ \rho$. $G_{1\times 2}^\sharp$ may not be as precise as the best abstraction for F in $\mathcal{D}_{1\times 2}^\sharp$ — even if $F_1^\sharp$ and $F_2^\sharp$ are best abstractions — but it is more precise than $F_{1\times 2}^\sharp$.

Partial Reductions. Sometimes, ρ is not easily computable, or does not exist due to the lack of proper abstraction functions. Still, we can use the idea of applying some reduction ρ' after each abstract operator application $F_{1\times 2}^\sharp$ in $\mathcal{D}_1^\sharp \times \mathcal{D}_2^\sharp$: $F_{1\times 2}^\sharp$ operates independently on its two components while ρ' propagates information between them. Any ρ' such that if $(Y_1^\sharp, Y_2^\sharp) = \rho'(X_1^\sharp, X_2^\sharp)$ then $Y_1^\sharp \sqsubseteq_1^\sharp X_1^\sharp$, $Y_2^\sharp \sqsubseteq_2^\sharp X_2^\sharp$ but $\gamma_1(Y_1^\sharp) \sqcap^\flat \gamma_2(Y_2^\sharp) = \gamma_1(X_1^\sharp) \sqcap^\flat \gamma_2(X_2^\sharp)$ will do. Such an operator will be called a *partial reduction* because it may not transfer all available information between the two abstract domains, but is a compromise between precision and cost. The benefit of the partially reduced product of abstract domains is that no abstract transfer function needs to be redefined: all existing transfer functions on the base domains can be reused and it is sufficient to define a single partial reduction function ρ' to be assured to get more precise results than separate analyses on $\mathcal{D}_1^\sharp$ and $\mathcal{D}_2^\sharp$.

Extrapolation Operators. Suppose that one — or both — base abstract domain has strictly increasing infinite chains, then so does the product domain $\mathcal{D}_1^\sharp \times \mathcal{D}_2^\sharp$ and we need to define a widening. An acceptable widening $\triangledown_{1\times 2}$ can be constructed by applying, component-wise, $\triangledown_1$ on the $\mathcal{D}_1^\sharp$ component and $\triangledown_2$ on the $\mathcal{D}_2^\sharp$ component — if $\mathcal{D}_1^\sharp$ or $\mathcal{D}_2^\sharp$ has no strictly infinite increasing chain and no widening, we can use any union abstraction $\cup_i^\sharp$ instead. An important remark is that it is dangerous to apply our partial reduction ρ' to the iterates: only the sequence $X^{i+1} \overset{\text{def}}{=} X^i \triangledown_{1\times 2} Y^i$ is guaranteed to converge and the sequence $X^{i+1} \overset{\text{def}}{=} \rho'(X^i) \triangledown_{1\times 2} Y^i$ may not. One solution is not to apply any reduction at all on the widened iterates X^i. Another solution is to define a new widening directly on $\mathcal{D}_1^\sharp \times \mathcal{D}_2^\sharp$; we are then free to introduce any reduction technique internally as long as we prove that widened sequences converge. A narrowing on $\mathcal{D}_1^\sharp \times \mathcal{D}_2^\sharp$ can also be defined the same way.

2.2.7 Related Work in Abstract Interpretation Theory

Other Presentations of Abstract Interpretation. In this thesis, we have chosen a presentation of Abstract Interpretation based on (partial) Galois connections, but other equivalent presentations exist that are based on Moore families or upper closure operators. Also, in some cases, it is possible to define a *computational order* for fixpoint iterations that is distinct from the abstract partial order $\sqsubseteq^\sharp$ used in the Galois connection (so-called *approximation order*). Finally, some frameworks are weak enough to accommodate for the existence of preorders instead of partial orders in the concrete and abstract domains. A comprehensive description of these frameworks, from the weakest to the strongest, can be found in [CC92b]. However, no framework is presented to allow the automatic derivation — or even the existence — of best operator abstractions unless there is a Galois connection, which motivated the introduction of partial Galois connections in this thesis.

Other Static Analysis Frameworks. There exists many different frameworks for static analysis, such as, data-flow analysis [MJ81], model-checking [CGP00], predicate transformers [Man92], typing [Car97], set-constraint-based analysis [Rey69], to cite only a few. It seems that all of them can be formalised in terms of Abstract Interpretation, meaning that the concepts presented in this thesis might be applicable to these frameworks, or, at least, be used in combination with analyses developed in these frameworks.

2.3 The Simple Language

We now present a small language, called Simple, that will be used to exemplify our work in the following four chapters. It is a simple procedure-less, pointer-less sequential language with only statically allocated variables and no recursion. There is also only one data-type: scalar numbers in $\mathbb{I}$, where $\mathbb{I}$ may be chosen to be the set of integers $\mathbb{Z}$, rationals $\mathbb{Q}$, or reals $\mathbb{R}$. As $\mathbb{I}$ is at least countable and the language contains loops, it is Turing-complete, and so, no interesting property is decidable by Rice's theorem [Ric53].

2.3.1 Language Syntax

The syntax of our language is depicted in Fig 2.1. A program is simply an instruction block, that is, a semi-colon `;` separated list of instructions. Each instruction in the block is preceded and followed by a unique syntactic label denoted by a circled number $①\cdots⑩ \in \mathcal{L}$. The instruction set comprises assignments $\leftarrow$, `if` conditionals, and `while` loops. Note that loops have an extra syntactic label $❶\cdots❿ \in \mathcal{L}$ between the `while` keyword and the boolean condition; it is reached upon entering the loop and after each loop iteration, just before the boolean condition is tested. To lighten the notations in our code examples, we will not always put all syntactic labels, but only those of interest.

$$
\begin{array}{lcll}
\mathit{expr} & ::= & X & X \in \mathcal{V} \\
& | & [a, b] & a \in \mathbb{I} \cup \{-\infty\},\ b \in \mathbb{I} \cup \{+\infty\},\ a \leq b \\
& | & - \ \mathit{expr} & \\
& | & \mathit{expr} \diamond \mathit{expr} & \diamond \in \{+, -, \times, /\} \\
\\
\mathit{test} & ::= & \mathit{expr} \bowtie \mathit{expr} & \bowtie \in \ \{=, \neq, <, \leq\} \\
& | & \texttt{not}\ \mathit{test} & \\
& | & \mathit{test}\ \texttt{and}\ \mathit{test} & \\
& | & \mathit{test}\ \texttt{or}\ \mathit{test} & \\
\\
\mathit{inst} & ::= & X \leftarrow \mathit{expr} & X \in \mathcal{V} \\
& | & \texttt{if}\ \mathit{test}\ \texttt{\{}\ \mathit{block}\ \texttt{\}}\ \texttt{else}\ \texttt{\{}\ \mathit{block}\ \texttt{\}} & \\
& | & \texttt{while}\ ❶\ \mathit{test}\ \texttt{\{}\ \mathit{block}\ \texttt{\}} & ❶ \in \mathcal{L} \\
\\
\mathit{block} & ::= & ①\ \mathit{inst}\ ;\ ②\ \cdots\ \mathit{inst}\ \ ⑩ & ① \cdots ⑩ \in \mathcal{L}
\end{array}
$$

$\mathcal{L}$ is a finite set of labels.
$\mathcal{V}$ is a finite set of variables.
$\mathbb{I}$ is either $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$.

Figure 2.1: Syntax of a Simple program.

Numerical expressions *expr* contain variables X, constants $[a, b]$, and the classical arithmetic operators $+$, $-$, $\times$, and $/$. Note that constants are in fact intervals with constant — possibly infinite — bounds and denote a new random choice of a number in the interval each time the expression is evaluated. This provides a useful notion of non-determinism without much notation burden; it will be most helpful in Chap. 7 to model rounding errors when considering programs that compute using floating-point numbers. Classical constants are a special case of interval and we will often denote by c the interval $[c, c]$, $c \in \mathbb{I}$. All intervals are non-strict: each bound is included in the interval, except when the bound is infinite. We will say that an expression is *interval linear* if it has the form $[a, b] + ([a_1, b_1] \times X_1) + \cdots + ([a_m, b_m] \times X_m)$ and *linear* if it is interval linear and all coefficients are singletons: $a = b$ and, for every i, $a_i = b_i$. Finally, it will be said that the expression is *quasi-linear* if all but the constant coefficient $[a, b]$ are singletons, that is, for every i, $a_i = b_i$. Note that, due to the associativity of the semantics of the $+$ operator, we do not need to explicitly specify an evaluation order using extra parentheses.

Conditionals and loops use boolean expressions *test* that can be constructed by comparing two numerical expressions and using the boolean `not`, `and`, and `or` operators. Note that

$$[\![expr]\!] : (\mathcal{V} \to \mathbb{I}) \to \mathcal{P}(\mathbb{I})$$

$$\begin{array}{lcl}
[\![X]\!]\rho & \stackrel{\text{def}}{=} & \{\ \rho(X)\ \} \\
[\![[a,b]]\!]\rho & \stackrel{\text{def}}{=} & \{\ x \in \mathbb{I} \mid a \leq x \leq b\ \} \\
[\![\mathbf{-}expr]\!]\rho & \stackrel{\text{def}}{=} & \{\ -x \mid x \in [\![expr]\!]\rho\ \} \\
[\![expr_1 \diamond expr_2]\!]\rho & \stackrel{\text{def}}{=} & \{\ x \diamond y \mid x \in [\![expr_1]\!]\rho,\ y \in [\![expr_2]\!]\rho\ \} \qquad \diamond \in \{\mathbf{+}, \mathbf{-}, \mathbf{\times}\} \\
[\![expr_1 \mathbf{/} expr_2]\!]\rho & \stackrel{\text{def}}{=} & \{\ adj(x/y) \mid x \in [\![expr_1]\!]\rho,\ y \in [\![expr_2]\!]\rho,\ y \neq 0\ \}
\end{array}$$

where $adj : \mathbb{R} \to \mathbb{I}$ is defined as follows:

$$adj(x) \stackrel{\text{def}}{=} \begin{cases} \max\{\ y \in \mathbb{Z} \mid y \leq x\ \} & \text{if } \mathbb{I} = \mathbb{Z} \text{ and } x \geq 0 \\ \min\{\ y \in \mathbb{Z} \mid y \geq x\ \} & \text{if } \mathbb{I} = \mathbb{Z} \text{ and } x \leq 0 \\ x & \text{if } \mathbb{I} \neq \mathbb{Z} \end{cases}$$

Figure 2.2: Semantics of numerical expressions.

the *greater than* operators $\mathbf{>}$ and $\mathbf{\geq}$ would be redundant, and so, they are not present in the syntax. Also note that all syntactic arithmetic and comparison operators are denoted by bold symbols to differentiate them from the corresponding mathematical operators on numbers.

For a given program, we denote by $\mathcal{L}$ the finite set of its labels and by $\mathcal{V}$ the finite set of its variables.

2.3.2 Concrete Semantics

The *concrete semantics* of a program will be the most precise mathematical expression of its behavior. All the other semantics we will derive for the same program, including the result of computable static analyses, will be abstractions of this most concrete semantics, linked to it by a succession of partial Galois connections. This means, in particular, that all these semantics will be correct only with respect to the concrete semantics presented here.

Numerical Expression Semantics. We call *concrete environment* any function $\rho : \mathcal{V} \to \mathbb{I}$ mapping each program variable to its actual value. The semantics of a numerical expression *expr* is a function $[\![expr]\!]$ that maps an environment ρ to the set, in $\mathcal{P}(\mathbb{I})$, of all possible values computed by the expression in the given environment. It is presented in Fig. 2.2. This set may contain several elements because of the non-determinism embedded in expressions. Note that divisions by zero are undefined, that is, return no result. When the divisor or the dividend is negative, the result of the division is obtained using the

$$\llbracket\, test \,\rrbracket : (\mathcal{V} \to \mathbb{I}) \to \mathcal{P}(\{\mathbf{T}, \mathbf{F}\})$$

$$\begin{array}{lcl}
\llbracket\, expr_1 \bowtie expr_2 \,\rrbracket\rho & \stackrel{\text{def}}{=} & \{\ \mathbf{T} \text{ if } \exists v_1 \in \llbracket\, expr_1 \,\rrbracket\rho,\ v_2 \in \llbracket\, expr_2 \,\rrbracket\rho,\ v_1 \bowtie v_2\ \} \cup \\
& & \{\ \mathbf{F} \text{ if } \exists v_1 \in \llbracket\, expr_1 \,\rrbracket\rho,\ v_2 \in \llbracket\, expr_2 \,\rrbracket\rho,\ v_1 \not\bowtie v_2\ \} \\
& & \bowtie \in\ \{=, \neq, <, \leq\}
\end{array}$$

$$\begin{array}{lcl}
\llbracket\, test_1 \texttt{ and } test_2 \,\rrbracket\rho & \stackrel{\text{def}}{=} & \{\ t_1 \wedge t_2 \mid t_1 \in \llbracket\, test_1 \,\rrbracket\rho,\ t_2 \in \llbracket\, test_2 \,\rrbracket\rho\ \} \\
\llbracket\, test_1 \texttt{ or } test_2 \,\rrbracket\rho & \stackrel{\text{def}}{=} & \{\ t_1 \vee t_2 \mid t_1 \in \llbracket\, test_1 \,\rrbracket\rho,\ t_2 \in \llbracket\, test_2 \,\rrbracket\rho\ \} \\
\llbracket\, \texttt{not } test \,\rrbracket\rho & \stackrel{\text{def}}{=} & \{\ \neg t \mid t \in \llbracket\, test \,\rrbracket\rho\ \}
\end{array}$$

Figure 2.3: Semantics of boolean expressions.

$$\{\!|\ \cdot\ |\!\} : \mathcal{P}(\mathcal{V} \to \mathbb{I}) \to \mathcal{P}(\mathcal{V} \to \mathbb{I})$$

$$\begin{array}{lcl}
\{\!|\, X \leftarrow expr \,|\!\}R & \stackrel{\text{def}}{=} & \{\ \rho[X \mapsto v] \mid \rho \in R,\ v \in \llbracket\, expr \,\rrbracket\rho\ \} \\
\{\!|\, X \rightarrow expr \,|\!\}R & \stackrel{\text{def}}{=} & \{\ \rho \mid \exists v \in \llbracket\, expr \,\rrbracket\rho,\ \rho[X \mapsto v] \in R\ \} \\
\{\!|\, test\ ? \,|\!\}R & \stackrel{\text{def}}{=} & \{\ \rho \mid \rho \in R,\ \mathbf{T} \in \llbracket\, test \,\rrbracket\rho\ \}
\end{array}$$

Figure 2.4: Transfer functions.

"rule of signs": $(-a)/b = a/(-b) = -(a/b)$ and $(-a)/(-b) = a/b$. Also, when $\mathbb{I} = \mathbb{Z}$, the *adj* function rounds the possibly non-integer result of the division towards an integer by *truncation*, as it is common in computer languages — such as the C programming language. Because of undefined results, $\llbracket\, expr \,\rrbracket\rho$ may be empty.

Boolean Expression Semantics. The semantics of tests and boolean expressions is similar to the semantics of numerical expressions, except that instead of a set of numbers, it outputs a set of boolean values in $\mathbb{B} \stackrel{\text{def}}{=} \{\mathbf{T}, \mathbf{F}\}$. It is described in Fig. 2.3. In $\mathbb{B}$, $\mathbf{T}$ means "true", $\mathbf{F}$ means "false", and we define the operators $\wedge$, $\vee$, and $\neg$ to be respectively the boolean "and", "or", and "not" operators.

Transfer Functions. The two basic instructions in our programming language are assignments and tests. To this, we add a *backward assignment* that is not used directly in our concrete semantics but can be quite useful once abstracted: one classical application is to refine a static analysis by performing combined forward and backward passes [CC92a, § 6]; another one is to backtrack from a user-specified program behavior to its origin, such

as in Bourdoncle's *abstract debugging* [Bou93a]. The semantics of our three instructions is defined by *transfer functions* that map sets of environments to sets of environments. They are presented in Fig. 2.4: assignment transfer functions $\{\!\!\{ X \leftarrow expr \}\!\!\}$ return environments where one variable has changed its value, backward assignments transfer functions $\{\!\!\{ X \rightarrow expr \}\!\!\}$ return the set of environments that *can* lead to the specified set of environments by an assignment, and test transfer functions $\{\!\!\{ test\ ? \}\!\!\}$ filter out environments that do not verify a boolean expression. These definitions deserve a few remarks:

- $\{\!\!\{ \texttt{not}\ (test)\ ? \}\!\!\} R \subseteq R \setminus \{\!\!\{ test\ ? \}\!\!\} R$ but these sets may not be equal because the former does not contain environments that lead to undefined expressions, while the later may contain some of them.

- The transfer function for a backward assignment is simply the *inverse relation* of that of the corresponding forward assignment. We will say that an assignment $X \leftarrow expr$ is *invertible* if and only if there exists some expression $expr'$ such that $\{\!\!\{ X \rightarrow expr' \}\!\!\} = \{\!\!\{ X \leftarrow expr \}\!\!\}$, which intuitively means that the value of X is not irremediably lost by the assignment. For instance, $X \leftarrow X + 1$ is invertible while $X \leftarrow Y$ is not.

- These transfer functions are complete $\cup-$morphisms (because the expression semantics $[\![\cdot]\!]$ is), and so, are monotonic.

Transition System. We present the semantics of our programming language by the mean of a *transition system*, that is, a relation $\rightarrow$ between *states*, as it is customary in the operational semantics world. A state of the program is a pair *(label,environment)* in $\mathcal{S} \stackrel{\text{def}}{=} \mathcal{L} \times (\mathcal{V} \rightarrow \mathbb{I})$. The transition system $\rightarrow \in \mathcal{S} \times \mathcal{S}$ is defined using our transfer functions by the rule system pictured in Fig. 2.5. It embeds the effect of elementary program actions on states. In effect, this definition simply adds control flow information, while the semantics of environments is fully embedded within transfer functions. Also, we define the *initial states* I to be the states with arbitrary environment, but as label the first label $① \in \mathcal{L}$ of the program: $I \stackrel{\text{def}}{=} \{\ (①, \rho) \mid \rho \in (\mathcal{V} \rightarrow \mathbb{I})\ \}$.

Reachability Semantics. Given a program, we seek to find out the possible values of all variables at each program point and disregard other information. This is a *reachability problem* with respect to the initial state set I and the transition relation $\rightarrow$. Our concrete semantic domain is the complete lattice of the powerset of states: $\mathcal{D}^c \stackrel{\text{def}}{=} (\mathcal{P}(\mathcal{L} \times (\mathcal{V} \rightarrow \mathbb{I})), \subseteq, \cup, \cap, \emptyset, \mathcal{L} \times (\mathcal{V} \rightarrow \mathbb{I}))$. The concrete semantics is defined by:

$$\text{lfp}\, F^c \quad \text{where} \quad F^c(S) \stackrel{\text{def}}{=} I \cup \{\ s' \mid \exists s \in S,\ s \rightarrow s'\ \}\ .$$

$$\frac{① \; X \leftarrow expr \; ② \quad \rho' \in \{\!|\, X \leftarrow expr \,|\!\}\{\rho\}}{(①, \rho) \rightarrow (②, \rho')}$$

$$\frac{① \; \texttt{if} \; test \; \{ \; ② \; \cdots \; ③ \; \} \; \texttt{else} \; \{ \; ④ \; \cdots \; ⑤ \; \} \; ⑥ \quad \rho' \in \{\!|\, test \; ? \,|\!\}\{\rho\}}{(①, \rho) \rightarrow (②, \rho')}$$

$$\frac{① \; \texttt{if} \; test \; \{ \; ② \; \cdots \; ③ \; \} \; \texttt{else} \; \{ \; ④ \; \cdots \; ⑤ \; \} \; ⑥ \quad \rho' \in \{\!|\, (\texttt{not} \; test) \; ? \,|\!\}\{\rho\}}{(①, \rho) \rightarrow (④, \rho')}$$

$$\frac{① \; \texttt{if} \; test \; \{ \; ② \; \cdots \; ③ \; \} \; \texttt{else} \; \{ \; ④ \; \cdots \; ⑤ \; \} \; ⑥}{(③, \rho) \rightarrow (⑥, \rho) \qquad (⑤, \rho) \rightarrow (⑥, \rho)}$$

$$\frac{① \; \texttt{while} \; ❷ \; test \; \{ \; ③ \; \cdots \; ④ \; \} \; ⑤}{(①, \rho) \rightarrow (❷, \rho) \qquad (④, \rho) \rightarrow (❷, \rho)}$$

$$\frac{① \; \texttt{while} \; ❷ \; test \; \{ \; ③ \; \cdots \; ④ \; \} \; ⑤ \quad \rho' \in \{\!|\, test \; ? \,|\!\}\{\rho\}}{(❷, \rho) \rightarrow (③, \rho')}$$

$$\frac{① \; \texttt{while} \; ❷ \; test \; \{ \; ③ \; \cdots \; ④ \; \} \; ⑤ \quad \rho' \in \{\!|\, (\texttt{not} \; test) \; ? \,|\!\}\{\rho\}}{(❷, \rho) \rightarrow (⑤, \rho')}$$

Figure 2.5: Small-step transition system $\rightarrow$ of a Simple program.

Equation System. Note that $\mathcal{D}^c$ is isomorphic to the point-wise lifting to $\mathcal{L}$ of the powerset of environments: $\mathcal{D}^c \approx \mathcal{D}^{c\prime} \stackrel{\text{def}}{=} (\mathcal{L} \rightarrow \mathcal{P}(\mathcal{V} \rightarrow \mathbb{I}), \dot{\cup}, \dot{\cap}, \lambda l.\emptyset, \lambda l.(\mathcal{V} \rightarrow \mathbb{I}))$. In this form, it becomes apparent that our concrete semantics is a *flow-sensitive invariant semantics*: to each program point it associates the strongest property of states that is verified each time the control flow of the program reaches this particular program point. Also, it allows rewriting the semantics $\operatorname{lfp} F^c$ as an equation system in $(X_l)_{l \in \mathcal{L}}$, derived from the syntax of a program as presented in Fig. 2.6. If we consider the vector $(X_l^0)_{l \in \mathcal{L}}$ such that $X_①^0 = \mathcal{P}(\mathcal{V} \rightarrow \mathbb{I})$ for the first label ① of the program and $X_l^0 = \emptyset$ for all other labels $l \in \mathcal{L}$, then the system has a unique least solution greater than $(X_l^0)_{l \in \mathcal{L}}$ and it corresponds exactly to $\operatorname{lfp} F^c$: each X_l in the solution is the strongest invariant holding at program point l. Note that the equation system is cyclic due to the `while` construct. We will now work on the equation system and compute solution abstractions using the chaotic iteration scheme of Sect. 2.2.5.

$$\frac{① \; V \leftarrow expr \; ②}{X_② = \{\!|\, V \leftarrow expr \,|\!\} X_①}$$

$$\frac{① \; \texttt{if} \; (test) \; \{ \; ② \; \cdots \; ③ \; \} \; \texttt{else} \; \{ \; ④ \; \cdots \; ⑤ \; \} \; ⑥}{\begin{array}{rcl} X_② & = & \{\!|\, test \; ? \,|\!\} X_① \\ X_④ & = & \{\!|\, (\texttt{not} \; test) \; ? \,|\!\} X_① \\ X_⑥ & = & X_③ \cup X_⑤ \end{array}}$$

$$\frac{① \; \texttt{while} \; ❷ \; (test) \; \{ \; ③ \; \cdots \; ④ \; \} \; ⑤}{\begin{array}{rcl} X_③ & = & \{\!|\, test \; ? \,|\!\} X_❷ \\ X_⑤ & = & \{\!|\, (\texttt{not} \; test) \; ? \,|\!\} X_❷ \\ X_❷ & = & X_① \cup X_④ \end{array}}$$

Figure 2.6: Equation system equivalent to the semantics of Fig. 2.5.

A Note About Errors. Note that a state may have no successor by $\rightarrow$, meaning that the program halts. This is the case, for instance, if the program reaches its last label, in which case the program successfully terminates. However, a statement involving a division by zero also halts the program. For the sake of simplicity, we have not introduced any *error state* so there is no distinction between correct program termination and run-time errors.

Discussion. Many kinds of semantics have been designed to discuss about program properties, some of which are more precise than the reachability semantics proposed here. For instance, in [Cou02], Cousot starts from a *trace semantics* and constructs the reachability semantics by abstracting the trace semantics. A trace semantics would allow us to discuss about many other properties, such as liveness properties, but it is not our purpose here. One can always imagine even more precise semantics (that allow discussing about the timing, energy consumption, or security aspects of a program, for instance). A reachability semantics is sufficient to describe invariants accurately; also, it is concise and intuitive enough to be proposed directly without the bother of deriving it from a lower-level semantics.

2.3.3 A Note on Missing Features

Our Simple language has been carefully chosen to demonstrate fully the analysis of numerical properties of variables and to compare easily existing and new numerical abstract domains, while simplifying the presentation by not considering other aspects of programming languages. Most of the extra features needed for the analysis of real-life programming languages, such as procedures, recursion, dynamic memory allocation, complex data-

structures, and parallelism are quite orthogonal to the way we analyse numerical properties of variables. They will not be discussed in this thesis. However, one important feature of real-life programs that will be discussed in much details is that they do not manipulate perfect integers, reals, and rationals, but imperfect machine-integers and floating-point numbers with limited range and precision. From now on, and up to Chap. 6, included, we will discuss the abstraction of perfect numbers while Chap. 7 will be entirely devoted to adapting our techniques to machine-integers and floating-point numbers.

2.4 Discovering Properties of Numerical Variables

We now turn to the design of a static analyser for the automatic discovery of the numerical properties of program variables in our Simple language. As this problem is not decidable, we seek decidable approximations that are sound with respect to the concrete semantics proposed earlier, using the Abstract Interpretation framework.

2.4.1 Numerical Abstract Domains

In order to compute semantic over-approximations, we need a computer-representable abstract version of the concrete domain $\mathcal{D}^c$ and computable abstractions of the semantic functions we used in the previous section. A *numerical abstract domain* will be defined by a choice, for every finite variable set $\mathcal{V}$, of:

Definition 2.4.1. Numerical abstract domain.

1. *a set* $\mathcal{D}^\sharp$ *whose elements are computer-representable,*
2. *a partial order* $\sqsubseteq^\sharp$ *on* $\mathcal{D}^\sharp$ *together with an effective algorithm to compare abstract elements,*
3. *a partial Galois connection* $\mathcal{P}(\mathcal{V} \mapsto \mathbb{I}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{D}^\sharp$,
4. *a smallest* $\bot^\sharp$ *and greatest elements* $\top^\sharp$ *such that* $\gamma(\bot^\sharp) = \emptyset$ *and* $\gamma(\top^\sharp) = (\mathcal{V} \mapsto \mathbb{I})$,
5. *effective algorithms to compute sound abstractions* $\llparenthesis X \leftarrow expr \rrparenthesis^\sharp$, $\llparenthesis X \rightarrow expr \rrparenthesis^\sharp$, *and* $\llparenthesis expr \bowtie expr \ ? \rrparenthesis^\sharp$ *of our transfer functions,*
6. *effective algorithms to compute sound abstractions* $\cup^\sharp$ *and* $\cap^\sharp$ *of* $\cup$ *and* $\cap$,
7. *effective widening algorithms* $\triangledown^\sharp$, *if* $\mathcal{D}^\sharp$ *has strictly increasing infinite chains,*
8. *effective narrowing algorithms* $\triangle^\sharp$, *if* $\mathcal{D}^\sharp$ *has strictly decreasing infinite chains.*

●

Whenever possible and practicable, we would like the operator abstractions to be best abstractions, that is, when best abstractions exist, are computable, and also not too costly.

Note that whenever $\mathcal{D}^\sharp$ has a lattice structure, we can choose $\cup^\sharp$ to be $\sqcup^\sharp$, but this is not mandatory. For instance, if $\sqcup^\sharp$ is costly to compute, it is worth defining a custom $\cup^\sharp$ operator with an adequate cost versus precision trade-off. The converse may also be true: we will see in the next chapter an abstract domain featuring a $\cup^\sharp$ operator that is more costly but also more precise than $\sqcup^\sharp$ to abstract the union $\cup$. Often γ is a — possibly non-complete — $\sqcap-$morphism; in that case, we can define $\cap^\sharp$ to be $\sqcap^\sharp$ and this intersection abstraction is *exact*.

2.4.2 Abstract Interpretor

An effective static analyser is obtained by applying the chaotic iterations of Sect. 2.2.5. The concrete transfer functions and $\cup$ operators are simply replaced with their abstract versions, and we start the chaotic iteration from the abstract environment $(X_l^{\sharp\,0})_{l\in\mathcal{L}}$ such that $X_{①}^{\sharp\,0} = \top^\sharp$ for the first label ① of the program and $X_l^{\sharp\,0} = \bot^\sharp$ for all other labels. The only loops in the dependency graph of the equation system of Fig. 2.6 are due to the `while` operators. As set of widening points W to break the dependency cycles, we take all the special labels ❶ $\cdots$ ❿ $\in \mathcal{L}$ introduced between each `while` keyword and the subsequent loop condition. It is a sufficient set of widening points as, if we remove the dependency edges ④ $\rightarrow$ ❷ — using the loop labelling of Fig. 2.6 — for all loops, we get a directed acyclic sub-graph. Following Bourdoncle in [Bou93b], we choose, as iteration ordering $(L_i)_{i\in\mathbb{N}}$, any *topological order* in this acyclic dependency sub-graph and try to stabilise the innermost loops first, in a "recursive" way.

This construction is reminiscent from the abstract interpretor presented by Cousot and Cousot as early as in 1976 [CC76], except that the interval domain is replaced with an arbitrary numerical abstract domain. A modern, detailed, and modular presentation of such an abstract interpretor can be found in [Cou99].

Non-Atomic Tests. In the design of numerical abstract domains we only require an abstract transfer function for *atomic* tests, that is, tests of the form ($expr \bowtie expr$?), $\bowtie \in \{=, \neq, <, \leq\}$, however, the analysed program may use more complex tests. We now fill this gap by proposing an abstract domain independent way to synthesise such transfer functions. A first step is to transform our test into an equivalent test that does not use the negation `not` operator. This is done by "pushing" the `not` operators into the `and` and `or` operators using the De-Morgan laws and reversing the comparison operators:

$$\begin{array}{lcl}
\texttt{not}(test_1\ \texttt{and}\ test_2) & \rightarrow & (\texttt{not}\ test_1)\ \texttt{or}\ (\texttt{not}\ test_2) \\
\texttt{not}(test_1\ \texttt{or}\ test_2) & \rightarrow & (\texttt{not}\ test_1)\ \texttt{and}\ (\texttt{not}\ test_2) \\
\texttt{not}(\texttt{not}\ test) & \rightarrow & test \\
\texttt{not}(expr_1 \leq expr_2) & \rightarrow & expr_2 < expr_1 \\
\texttt{not}(expr_1 < expr_2) & \rightarrow & expr_2 \leq expr_1 \\
\texttt{not}(expr_1 = expr_2) & \rightarrow & expr_1 \neq expr_2 \\
\texttt{not}(expr_1 \neq expr_2) & \rightarrow & expr_1 = expr_2
\end{array}$$

Then, the transfer function for this `not`-free test is computed by structural induction using the already available $\cup^\sharp$ and $\cap^\sharp$ operators, and $\{\!|\ expr \bowtie expr\ ?\ |\!\}^\sharp$ for the base case:

$$\begin{array}{lcl}
\{\!|\ (test_1\ \texttt{and}\ test_2)\ ?\ |\!\}^\sharp X^\sharp & \stackrel{\text{def}}{=} & \{\!|\ test_1\ ?\ |\!\}^\sharp X^\sharp\ \cap^\sharp\ \{\!|\ test_2\ ?\ |\!\}^\sharp X^\sharp \\
\{\!|\ (test_1\ \texttt{or}\ test_2)\ ?\ |\!\}^\sharp X^\sharp & \stackrel{\text{def}}{=} & \{\!|\ test_1\ ?\ |\!\}^\sharp X^\sharp\ \cup^\sharp\ \{\!|\ test_2\ ?\ |\!\}^\sharp X^\sharp
\end{array}$$

Backward Assignment Transfer Function Usage. For the sake of conciseness, we do not detail here how the backward assignment transfer functions can be used in an abstract interpretor to refine its result or focus the analysis on user-supplied properties, and instead refer the reader to [CC92a, Bou93a]. Yet, we will take care to include a backward assignment abstract transfer function with the numerical abstract domains introduced in this thesis so that the interested reader can plug them into backward-aware abstract interpretors.

2.4.3 Fall-Back Transfer Functions

Sometimes, too few transfer functions are provided for an abstract domain with respect to the analysed language constructs. For instance, the polyhedron domain presented in [CH78], including its recent implementations [Jea, PPL], only focuses on transfer functions for linear expressions with singleton coefficients, while our language allows non-linear expressions and interval coefficients. Provided an abstract *forget* operator is defined, it is always possible to complete the definition using the generic fall-back transfer functions and operators presented here. These are very coarse, but also quite fast.

Abstracting Assignments. Let us define the additional *forget transfer function* the following way:

$$\{\!|\ X \leftarrow ?\ |\!\} R \stackrel{\text{def}}{=} \{\ \rho[X \mapsto v] \mid \rho \in R,\ v \in \mathbb{I}\ \}\ .$$

It can be equivalently defined in a "backwards" way as:

$$\{\!|\ X \leftarrow ?\ |\!\} R \stackrel{\text{def}}{=} \{\ \rho \mid \exists v \in \mathbb{I},\ \rho[X \mapsto v] \in R\ \}\ .$$

If a sound counterpart $\{\!|\ X \leftarrow ?\ |\!\}^\sharp$ for $\{\!|\ X \leftarrow ?\ |\!\}$ is given, then we can define coarse generic assignment and backward assignment transfer functions as follows:

$$\begin{array}{lcl}
\{\!|\ X \leftarrow expr\ |\!\}^\sharp & \stackrel{\text{def}}{=} & \{\!|\ X \leftarrow ?\ |\!\}^\sharp \\
\{\!|\ X \rightarrow expr\ |\!\}^\sharp & \stackrel{\text{def}}{=} & \{\!|\ X \leftarrow ?\ |\!\}^\sharp
\end{array}$$

Note that, in very last resort, $\{\!\!\{ X \leftarrow ? \}\!\!\}^\sharp X^\sharp$ can be defined to be $\top^\sharp$.

Abstracting Tests. As $\{\!\!\{ test\ ? \}\!\!\} R \subseteq R$, the identity $\{\!\!\{ test\ ? \}\!\!\}^\sharp X^\sharp \stackrel{\text{def}}{=} X^\sharp$ is always a sound abstraction for tests.

Abstracting Unions and Intersections. Finally, a sound choice for $\cup^\sharp$ is to always return $\top^\sharp$ while a sound choice for $\cap^\sharp$ is to return either argument.

Another solution is to perform the missing abstract transfer function or operator in an alternate abstract domain which implements it. In this case, we also need sound *conversion operators* — possibly incurring a loss of precision — between the two abstract domains.

2.4.4 Non-Relational Abstract Domains

A *non-relational domain* is an abstract domain that abstracts each variable independently, and so, is not able to discover relationships between variables. We present here a systematic way to construct non-relational numerical abstract domains from smaller bricks that is greatly inspired from Cousot's design of a generic abstract interpretor in [Cou99]. Not only will it simplify the presentation of subsequent non-relational domains — such as the interval domain of Sect. 2.4.6 — but it will also pave the way towards a similar parametric construct for a new class of *relational* domains, introduced in Chap. 5. The basic building brick is a *non-relational basis* that expresses how to abstract *one* variable and has abstract counterparts for arithmetic operators instead of abstract counterparts for complex transfer functions.

Definition 2.4.2. Non-relational basis.

1. *A* basis *is a poset* $(\mathcal{B}, \sqsubseteq^\sharp_\mathcal{B})$ *together with*
2. *a partial Galois connection* $(\mathcal{P}(\mathbb{I}), \subseteq) \underset{\alpha_\mathcal{B}}{\overset{\gamma_\mathcal{B}}{\leftrightarrows}} (\mathcal{B}, \sqsubseteq^\sharp_\mathcal{B})$*, and*
3. *a least* $\bot^\sharp_\mathcal{B}$ *and greatest element* $\top^\sharp_\mathcal{B}$ *such that* $\gamma_B(\bot^\sharp_\mathcal{B}) = \emptyset$ *and* $\gamma_B(\top^\sharp_\mathcal{B}) = \mathbb{I}$*, and*
4. *algorithms to compute sound abstractions of* $\cup^\sharp_\mathcal{B}$ *and* $\cap^\sharp_\mathcal{B}$ *of* $\cup$ *and* $\cap$*, and*
5. *algorithms to compute sound (forward) abstractions of all the arithmetics operators:*

   $[a,b]^\sharp \in \mathcal{B}$ *such that* $\gamma_\mathcal{B}([a,b]^\sharp) \supseteq [a,b]$

   $-^\sharp : \mathcal{B} \to \mathcal{B}$ *such that* $\gamma_\mathcal{B}(-^\sharp X^\sharp) \supseteq \{\ -x \mid x \in \gamma(X^\sharp)\ \}$

   $\diamond^\sharp : \mathcal{B}^2 \to \mathcal{B}$ *such that* $\gamma_\mathcal{B}(X^\sharp \diamond^\sharp Y^\sharp) \supseteq \{\ x \diamond y \mid x \in \gamma(X^\sharp),\ y \in \gamma(Y^\sharp)\ \}$
   $\diamond \in \{+, -, \times, /\}$

6. *algorithms to compute sound* backward *abstractions of all the arithmetics operators that, given some arguments and a target result, try to refine all the arguments while keeping the coverage of the result:*

   $\overleftarrow{-}^\sharp : \mathcal{B}^2 \to \mathcal{B}$ *such that*
   $\gamma_{\mathcal{B}}(X^\sharp) \supseteq \gamma_{\mathcal{B}}(\overleftarrow{-}^\sharp(X^\sharp, R^\sharp)) \supseteq \{\ x \in \gamma_{\mathcal{B}}(X^\sharp) \mid -x \in \gamma_{\mathcal{B}}(R^\sharp)\ \}$

   $\overleftarrow{\diamond}^\sharp : \mathcal{B}^3 \to \mathcal{B}^2$ *such that*
   $(U^\sharp, V^\sharp) = \overleftarrow{\diamond}^\sharp(X^\sharp, Y^\sharp, R^\sharp) \quad \Longrightarrow$
   $\gamma_{\mathcal{B}}(X^\sharp) \supseteq \gamma_{\mathcal{B}}(U^\sharp) \supseteq \{\ x \in \gamma_{\mathcal{B}}(X^\sharp) \mid \exists y \in \gamma_{\mathcal{B}}(Y^\sharp),\ x \diamond y \in \gamma_{\mathcal{B}}(R^\sharp)\ \}$
   $\gamma_{\mathcal{B}}(Y^\sharp) \supseteq \gamma_{\mathcal{B}}(V^\sharp) \supseteq \{\ y \in \gamma_{\mathcal{B}}(Y^\sharp) \mid \exists x \in \gamma_{\mathcal{B}}(X^\sharp),\ x \diamond y \in \gamma_{\mathcal{B}}(R^\sharp)\ \}$
   $\diamond \in \{+, -, \times, /\}$

7. *algorithms to compute sound* backward *abstractions for the comparison operators that, given two arguments, refine their value by supposing that the comparison is true:*

   $\overleftarrow{\bowtie}^\sharp : \mathcal{B}^2 \to \mathcal{B}^2$ *such that*
   $(U^\sharp, V^\sharp) = \overleftarrow{\bowtie}^\sharp(X^\sharp, Y^\sharp) \quad \Longrightarrow$
   $\gamma_{\mathcal{B}}(X^\sharp) \supseteq \gamma_{\mathcal{B}}(U^\sharp) \supseteq \{\ x \in \gamma_{\mathcal{B}}(X^\sharp) \mid \exists y \in \gamma_{\mathcal{B}}(Y^\sharp),\ x \bowtie y\ \}$
   $\gamma_{\mathcal{B}}(Y^\sharp) \supseteq \gamma_{\mathcal{B}}(V^\sharp) \supseteq \{\ y \in \gamma_{\mathcal{B}}(Y^\sharp) \mid \exists x \in \gamma_{\mathcal{B}}(X^\sharp),\ x \bowtie y\ \}$
   $\bowtie \in \{=, \neq, <, \leq\}$

8. *algorithms to compute widenings* $\nabla^\sharp_{\mathcal{B}}$ *(resp. narrowings* $\Delta^\sharp_{\mathcal{B}}$*) if* $\mathcal{B}$ *contains strictly increasing (resp. decreasing) infinite chains.*

●

Thanks to a (partial) Galois connection, it may be possible to derive best forward and backward abstractions mechanically — for this matter, each backward abstraction of a binary operator may be considered as two distinct abstract operators in $\mathcal{B}$, each output being optimised independently from the other one.

Domain Lifting. The non-relational domain $\mathcal{D}^\sharp$ derived from the basis $\mathcal{B}$ is the pointwise lifting of $\mathcal{B}$ to $\mathcal{V}$: $\mathcal{D}^\sharp \stackrel{\text{def}}{=} \mathcal{V} \to \mathcal{B}$. Thus, if $\mathcal{B}$ is a cpo, lattice, or complete lattice, so is $\mathcal{D}^\sharp$. The partial Galois connection $\mathcal{P}(\mathcal{V} \to \mathbb{I}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{D}^\sharp$ is derived from the basis partial Galois connection as follows:

$$\begin{array}{lcl} \gamma(R^\sharp) & \stackrel{\text{def}}{=} & \{\ \rho \in (\mathcal{V} \to \mathbb{I}) \mid \forall X \in \mathcal{V},\ \rho(X) \in \gamma_{\mathcal{B}}(R^\sharp(X))\ \} \\ \alpha(R) & \stackrel{\text{def}}{=} & \lambda X.\alpha_{\mathcal{B}}(\{\ \rho(X) \mid \rho \in R\ \}) \end{array}$$

Fall-Back Operators. Any forward operator can be soundly — but poorly — abstracted by an operator always returning $\top^\sharp_{\mathcal{B}}$, if we are not interested in the precise analysis of expressions using this operator. Likewise, backward operators can be soundly abstracted

$$\llbracket\, expr \,\rrbracket^\sharp : (\mathcal{V} \to \mathcal{B}) \to \mathcal{B}$$

$$\begin{array}{lcll}
\llbracket\, V \,\rrbracket^\sharp X^\sharp & \stackrel{\text{def}}{=} & X^\sharp(V) & V \in \mathcal{V} \\
\llbracket\, [a,b] \,\rrbracket^\sharp X^\sharp & \stackrel{\text{def}}{=} & [a,b]^\sharp & \\
\llbracket\, -expr \,\rrbracket^\sharp X^\sharp & \stackrel{\text{def}}{=} & -^\sharp(\llbracket\, expr \,\rrbracket^\sharp X^\sharp) & \\
\llbracket\, expr_1 \diamond expr_2 \,\rrbracket^\sharp X^\sharp & \stackrel{\text{def}}{=} & (\llbracket\, expr_1 \,\rrbracket^\sharp X^\sharp) \diamond^\sharp (\llbracket\, expr_2 \,\rrbracket^\sharp X^\sharp) & \diamond \in \{+, -, \times, /\}
\end{array}$$

Figure 2.7: Non-relational abstract semantics of numerical expressions.

by returning their arguments unchanged, that is, performing no refinement at all. Finally, $\cup^\sharp_{\mathcal{B}}$ can be abstracted by $\top^\sharp_{\mathcal{B}}$, while $\cap^\sharp_{\mathcal{B}}$ can be abstracted by returning any of its arguments. There is, however, a simple way to derive rather precise backward arithmetic operators from the forward ones, as follows:

- $\overleftarrow{-}^\sharp(X^\sharp, R^\sharp) \stackrel{\text{def}}{=} X^\sharp \cap^\sharp_{\mathcal{B}} (-^\sharp R^\sharp)$
- $\overleftarrow{+}^\sharp(X^\sharp, Y^\sharp, R^\sharp) \stackrel{\text{def}}{=} (X^\sharp \cap^\sharp_{\mathcal{B}} (R^\sharp -^\sharp Y^\sharp),\ Y^\sharp \cap^\sharp_{\mathcal{B}} (R^\sharp -^\sharp X^\sharp))$
- $\overleftarrow{-}^\sharp(X^\sharp, Y^\sharp, R^\sharp) \stackrel{\text{def}}{=} (X^\sharp \cap^\sharp_{\mathcal{B}} (Y^\sharp +^\sharp R^\sharp),\ Y^\sharp \cap^\sharp_{\mathcal{B}} (X^\sharp -^\sharp R^\sharp))$
- $\overleftarrow{\times}^\sharp(X^\sharp, Y^\sharp, R^\sharp) \stackrel{\text{def}}{=} (X^\sharp \cap^\sharp_{\mathcal{B}} (R^\sharp /^\sharp Y^\sharp),\ Y^\sharp \cap^\sharp_{\mathcal{B}} (R^\sharp /^\sharp X^\sharp))$
- $\overleftarrow{/}^\sharp(X^\sharp, Y^\sharp, R^\sharp) \stackrel{\text{def}}{=} (X^\sharp \cap^\sharp_{\mathcal{B}} (Y^\sharp \times^\sharp adj^\sharp(R^\sharp)),\ Y^\sharp \cap^\sharp_{\mathcal{B}} ((X^\sharp /^\sharp adj^\sharp(R^\sharp)) \cup^\sharp_{\mathcal{B}} [0,0]^\sharp))$

  where $adj^\sharp(X^\sharp) \stackrel{\text{def}}{=} \begin{cases} X^\sharp +^\sharp [-1,1]^\sharp & \text{if } \mathbb{I} = \mathbb{Z} \\ X^\sharp & \text{if } \mathbb{I} \neq \mathbb{Z} \end{cases}$

The division is a bit complex. First, as divisions by zero are silently ignored, the refinement of the dividend must contain 0 whenever $Y^\sharp$ contains it. Then, because of the truncation adj when $\mathbb{I} = \mathbb{Z}$, $\rho(R) \in \llbracket\, X / Y \,\rrbracket\rho$ does not implies $\rho(X) \in \llbracket\, R \times Y \,\rrbracket\rho$. However, we do have $\rho(X) \in \llbracket\, (R + [-1,1]) \times Y \,\rrbracket\rho$, and so, we use the $adj^\sharp$ function to add a ± 1 correction term to $R^\sharp$ in the abstract.

Abstract Operators and Transfer Functions on $\mathcal{D}^\sharp$. On $\mathcal{D}^\sharp$, the definitions of $\sqsubseteq^\sharp$, $\cap^\sharp$, $\cup^\sharp$, $\nabla^\sharp$, $\triangle^\sharp$, $\bot^\sharp$, and $\top^\sharp$ are derived by lifting point-wisely those on $\mathcal{B}$, which also provides us with effective algorithms.

Using the forward abstract arithmetic operators, we can derive the abstract semantics $\llbracket\, expr \,\rrbracket^\sharp X^\sharp$ of an expression $expr$ by structural induction, as presented in Fig. 2.7. It bears a striking resemblance to the concrete semantics of expressions of Fig. 2.2. The forward abstract assignment is then simply defined by: $\{\!\!\{\, V \leftarrow expr \,\}\!\!\}^\sharp X^\sharp \stackrel{\text{def}}{=} X^\sharp\,[\, V \mapsto \llbracket\, expr \,\rrbracket^\sharp X^\sharp\,]$.

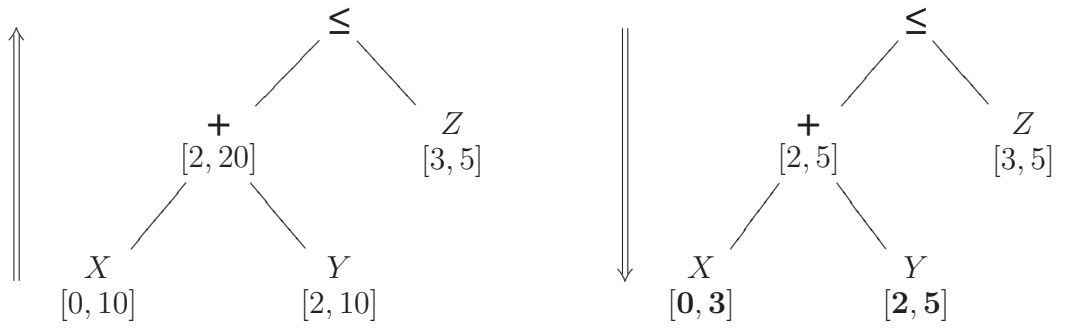


Figure 2.8: Non-relational abstraction for the test $X + Y \leq Z$, exemplified on the interval abstract domain (see Sect. 2.4.6). The intervals for X and Y, respectively $[0, 10]$ and $[2, 10]$, are refined to $[0, 3]$ and $[2, 5]$.

Abstracting a test $(expr_1 \bowtie expr_2\ ?)$, $\bowtie \in \{=, \neq, <, \leq\}$, is a little more complex. We must first evaluate $expr_1$ and $expr_2$ by induction, as for the forward assignment, but also remember for each expression node the evaluation of the corresponding sub-expression. When fed with the abstract values $[\![expr_1]\!]^\sharp X^\sharp$ and $[\![expr_2]\!]^\sharp X^\sharp$, the backward test $\overleftarrow{\bowtie}^\sharp$ operator returns refined values by taking test filtering into account. These new values should then be propagated back, in a top-down way. At each node, we feed the operator arguments computed in the first, bottom-up, pass and the refined result obtained by the current top-down pass to the backward arithmetics operator to obtain refined arguments, enabling the refinement of deeper sub-expressions. At the end of this process, the leaves contain over-approximations of the values of the expression variables after the test. This is exemplified in Fig. 2.8 using the interval abstract domain that will be recalled in Sect. 2.4.6. To gain more precision, Granger proposes to perform *local iterations* [Gra92], that is, to perform the bottom-up and top-down passes several times, using the refined environment of one pass as the input of the next pass. This allows propagating information between expression leaves, which is especially useful when a variable appears several times in the test. Finally, when abstracting complex tests by combining our atomic test abstractions with the $\cup^\sharp$ and $\cap^\sharp$ operators, as presented in Sect. 2.4.2, it is worth iterating over the whole test abstraction instead of iterating separately on each atomic test, so that information can flow from one atomic test to the other.[2]

Abstracting a backward assignment $V \rightarrow expr$ can be done using the same principle, but with subtle differences. First, in the bottom-up forward evaluation pass, we need the

[2]The local iteration technique is not limited to non-relational abstract domains; it can indeed be reformulated as follows: $\{\!|\ test\ ?\ |\!\}^\sharp$ is replaced with $(\{\!|\ test\ ?\ |\!\}^\sharp)^n$ for some n. Alternatively, a fixpoint iteration with narrowing $\triangle^\sharp$ can be used.

values of the variables *before* the assignment while we are provided with an environment $X^\sharp$ valid *after* the assignment. In most static analyses, backward assignments are only used to refine the result of a previous forward analysis meaning that an over-approximation $Y^\sharp$ of the environment before the assignment is generally available. If it is not available, we can always perform the bottom-up evaluation using $Y^\sharp \stackrel{\text{def}}{=} X^\sharp[V \mapsto \top^\sharp_{\mathcal{B}}]$; it is a valid over-approximation of the environments before the assignment as only the value of V may have been altered by the assignment. Then, to initiate the top-down pass, we refine the root abstract element by intersecting it with $X^\sharp(V)$. The output of one pass is the environment $Y^\sharp$ refined with the information at the leaves. Finally, if we are to perform local iterations, we can use the output $Y^\sharp$ of one refinement pass to perform the evaluation in the next pass, but we should always intersect the root with the same value $X^\sharp(V)$.

An often overlooked fact is that backward versions $\overleftarrow{\diamond}^\sharp$ of the arithmetic operators are not only needed to design backward assignment transfer functions, but also precise test abstract functions.

Precision of the Transfer Functions. Our transfer functions are not able to exploit the *aliasing* of variables in an expression, that is, the fact that one variable may appear at several places. One crude example is the abstract evaluation of the expression $V - V$ leading to $\{\, x-y \mid x,y \in \gamma^\sharp_{\mathcal{B}}(X^\sharp(V)) \,\}$ which is strictly larger than the correct answer $\{0\}$ whenever $\gamma^\sharp_{\mathcal{B}}(X^\sharp(V))$ is not a singleton. Likewise, a non-relational abstract domain may not be able to discover that the expression $X \neq X$ is always false. Sometimes, local iterations can help to propagate information between two expression leaves containing the same variable but it is not always sufficient; for instance, it does not help in the two preceding examples. As a conclusion, the constructed transfer functions are not optimal even when the basic forward and backward operators on $\mathcal{B}$ are. We will present in Chap. 6 a so-called *linearisation* technique to help with this precision problem.

Coalescent Domain Lifting. Note that we can replace the point-wise lifting with a *coalescent* point-wise lifting $\mathcal{D}^\sharp \stackrel{\text{def}}{=} (\mathcal{V} \to \mathcal{B} \setminus \{\bot^\sharp_{\mathcal{B}}\}) \cup \{\mathbb{\bot}^\sharp\}$ that merges all function having a $\bot^\sharp_{\mathcal{B}}$ component into a single $\mathbb{\bot}^\sharp$ element, as they all abstract the empty environment $\emptyset$. If $\gamma_{\mathcal{B}}$ is one-to-one, this ensures that the γ concretisation derived on $\mathcal{D}^\sharp$ is also one-to-one. In practice, this shortcuts abstract computations over non-reachable execution paths, and so, improves both the efficiency and the precision of the analyser. As this alternate construction is conceptually similar but adds much notation burden, we do not expose it here in more details.

Cartesian Galois Connection. A way to characterise the inherent loss of precision due to the choice of being non-relational is to present the *Cartesian Galois connection* that

constructs the most precise non-relational abstraction of sets of environments:

Definition 2.4.3. Cartesian galois connection.

The function pair $(\alpha^{Cart}, \gamma^{Cart})$ *defined as follows:*

$$\begin{array}{lcl} \alpha^{Cart}(R) & \stackrel{\text{def}}{=} & \lambda X.\{\ x \in \mathbb{I} \mid \exists \rho \in R,\ x = \rho(X)\ \} \\ \gamma^{Cart}(R^\sharp) & \stackrel{\text{def}}{=} & \{\ \rho \in (\mathcal{V} \to \mathbb{I}) \mid \forall X \in \mathcal{V},\ \rho(X) \in R^\sharp(X)\ \} \end{array}$$

forms a Galois connection: $\mathcal{P}(\mathcal{V} \to \mathbb{I}) \underset{\alpha^{Cart}}{\overset{\gamma^{Cart}}{\leftrightarrows}} (\mathcal{V} \to \mathcal{P}(\mathbb{I}))$.

●

Any non-relational numerical abstract domain defined by a partial Galois connection $\mathcal{P}(\mathcal{V} \to \mathbb{I}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{D}^\sharp$ is such that $\alpha = \alpha \circ \gamma^{Cart} \circ \alpha^{Cart}$, meaning that we can never retrieve more precise results than those of the Cartesian abstraction.

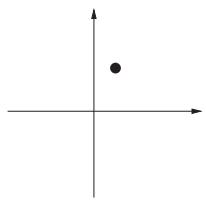
2.4.5 Overview of Existing Numerical Abstract Domains

There exists a wide variety of numerical abstract domains which differ in expressive power and computational complexity. Fig. 2.9 presents a comprehensive "zoo" of classical numerical abstract domains. These domains can be classified according to several characteristics:

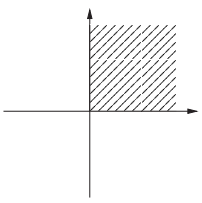
- the numerical expressions allowed: single variables, linear, modulo, or multiplicative expressions; there can also be a restriction on the allowed constant coefficients;
- how many variables can appear at once in a relation: only one (non-relational), or any number (relational);
- whether they use equality or inequality predicates — most domains that can represent non-strict inequalities also have a version that can represent strict inequalities.

The following two sub-sections present in more details the interval and the polyhedron domains that are quite important for our purpose. They represent two extremes in the cost versus precision trade-off spectrum: the non-relational and fast interval domain at one end, and the fully relational but costly polyhedron domain at the other end. Until the late 90's, there was no other domain able to discuss about numerical inequalities. We wish, in this thesis, to fill the gap between these two domains and allow a finer cost versus precision trade-off, so, the interval and polyhedron domains are a natural choice for comparison. Moreover, the interval transfer functions will be used internally in the relational abstract domains introduced in the following chapters.

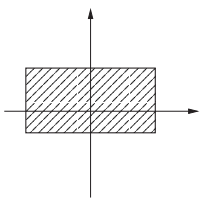
Fig. 2.10 presents three numerical abstract domains that are introduced in this thesis: zones (Chap. 3), octagons (Chap. 4), and zone congruences (Chap. 5). Note that they are



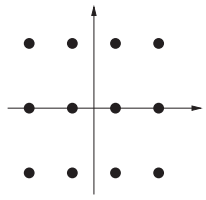
Constant Propagation
$X_i = c_i$
[Kil73]



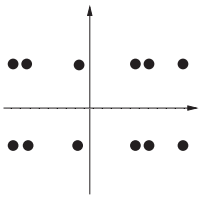
Signs
$X_i \geq 0,\ X_i \leq 0$
[CC76]



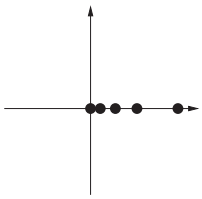
Intervals
$X_i \in [a_i, b_i]$
[CC76]



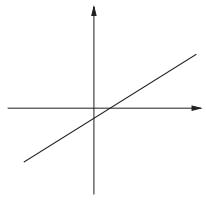
Simple Congruences
$X_i \equiv a_i\,[b_i]$
[Gra89, Gra97]



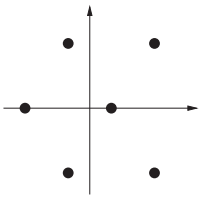
Interval Congruences
$X_i \in \alpha_i[a_i, b_i]$
[Mas93]



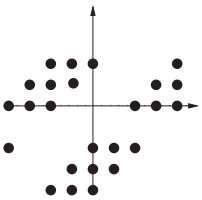
Power Analysis
$X_i \in \alpha_i^{a_i\mathbb{Z}+b_i}, \alpha_i^{[a_i,b_i]}$, etc.
[Mas01]



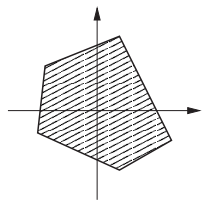
Linear Equalities
$\sum_i \alpha_{ij} X_i = \beta_j$
[Kar76]



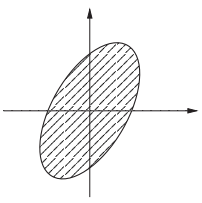
Linear Congruences
$\sum_i \alpha_{ij} X_i \equiv \beta_j\,[\gamma_j]$
[Gra91]



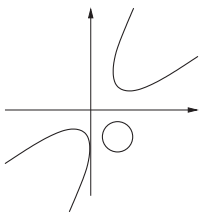
Trapezoidal Congruences
$X_i = \sum_j \lambda_j \alpha_{ij} + \beta_j$
[Mas92]



Polyhedra
$\sum_i \alpha_{ij} X_i \leq \beta_j$
[CH78]

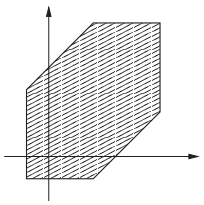


Ellipsoids
$\alpha X^2 + \beta Y^2 + \gamma XY \leq \delta$
[Fer04b]

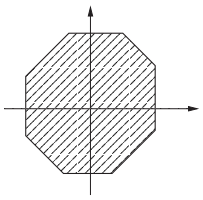


Varieties
$P_i(\vec{X}) = 0,\ P_i \in \mathbb{R}[\mathcal{V}]$
[RCK04a]

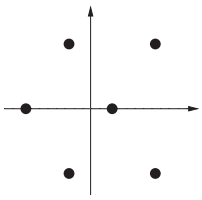
Figure 2.9: The "zoo" of existing numerical abstract domains. The first two rows present non-relational domains while the last two rows present relational domains.



Zones
$X_i - X_j \leq c_{ij}$
[Min01a]

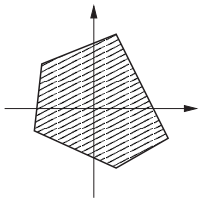


Octagons
$\pm X_i \pm X_j \leq c_{ij}$
[Min01b]

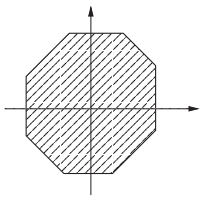


Zone Congruences
$X_i \equiv X_j + \alpha_{ij}\,[\beta_{ij}]$
[Min02]

Figure 2.10: Numerical abstract domains introduced in this thesis.



TVPLI
$\alpha_{ij} X_i + \beta_{ij} X_j \leq c_{ij}$
[SKH02]



Octahedra
$\sum_i \epsilon_{ij} X_i \leq \beta_j,\ \epsilon_{ij} \in \{-1, 0, 1\},\ X_i \geq 0$
[CC04]

Figure 2.11: Other recent numerical abstract domains.

restrictions of the polyhedron and trapezoidal congruence relational abstract domains, and so, we will call them *weakly* relational domains. In particular, the polyhedron domain is strictly more expressive than the octagon domain, which is strictly more expressive than the zone domain, which is strictly more expressive than the interval domain: Fig. 2.12 presents a comparison of the sets obtained by abstracting the same concrete set using these four domains. Finally, our work on the octagon domain has inspired two orthogonal extensions depicted in Fig. 2.11: the Two Variables Per Inequalities (TVPLI) abstract domain proposed by Simon, King, and Howe [SKH02] and the octahedra domain proposed by Clarisó and Cortadella [CC04] — note that the difference between TVPLI and polyhedra, and between octagons and octahedra, are only visible when there are at least three variables, which explains why the planar drawings in Figs. 2.9–2.11 seem similar.

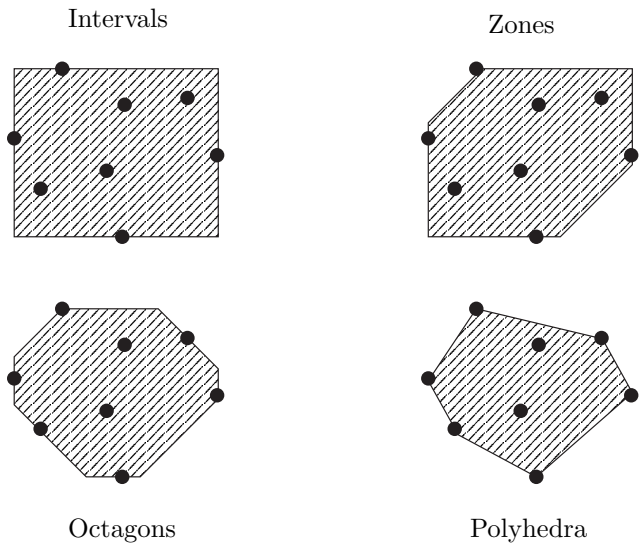


Figure 2.12: Comparing non-relational, weakly relational, and fully relational domains.

2.4.6 The Interval Abstract Domain

The interval abstract domain $\mathcal{D}^{Int}$ is based on the classical concept of *interval arithmetics*, introduced in the 60's by R. E. Moore [Moo66], and since then widely used in scientific computing. It was adapted to the needs of Abstract Interpretation by Cousot and Cousot in [CC76]. As we restate the interval abstract domain within the generic non-relational construction of Sect. 2.4.4, it is only necessary to present here the *interval basis* $\mathcal{B}^{Int}$.

Interval Basis $\mathcal{B}^{Int}$. Let $\mathcal{B}^{Int}$ be the set of empty and non-empty intervals with bounds in $\mathbb{I}$, where $\mathbb{I} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$: $\mathcal{B}^{Int} \stackrel{\text{def}}{=} \{\bot^{Int}_{\mathcal{B}}\} \cup \{\ [a,b] \mid a \in \mathbb{I} \cup \{-\infty\},\ b \in \mathbb{I} \cup \{+\infty\},\ a \leq b\ \}$. Finite bounds are inclusive but infinite bounds are strict. In particular, $\mathcal{B}^{Int}$ contains $[-\infty, +\infty]$ which denotes the whole set $\mathbb{I}$, and the singletons $[a,a]$ when $a \in \mathbb{I}$. The empty interval is denoted by $\bot^{Int}_{\mathcal{B}}$.

Interval Basis Structure. The partial order $\sqsubseteq^{Int}_{\mathcal{B}}$ is defined as $[a,b] \sqsubseteq^{Int}_{\mathcal{B}} [a',b'] \iff^{\text{def}} a \geq a'$ and $b \leq b'$, and $\bot^{Int}_{\mathcal{B}}$ is the smallest element. The basis concretisation $\gamma^{Int}_{\mathcal{B}}$ is defined as $\gamma^{Int}_{\mathcal{B}}([a,b]) \stackrel{\text{def}}{=} \{\ x \in \mathbb{I} \mid a \leq x \leq b\ \}$ and $\gamma^{Int}_{\mathcal{B}}(\bot^{Int}_{\mathcal{B}}) \stackrel{\text{def}}{=} \emptyset$. Note that $\gamma^{Int}_{\mathcal{B}}$ is a one-to-one $\sqcap$−complete morphism. When $\mathbb{I}$ is $\mathbb{Z}$ or $\mathbb{R}$, $\mathcal{B}^{Int}$ is a complete lattice and, by Thm. 2.2.1, a canonical $\alpha^{Int}_{\mathcal{B}}$ exists. In all cases, we define $\alpha^{Int}_{\mathcal{B}}$ as: $\alpha^{Int}_{\mathcal{B}}(\emptyset) = \bot^{Int}_{\mathcal{B}}$ and $\alpha^{Int}_{\mathcal{B}}(S) \stackrel{\text{def}}{=} [\min S, \max S]$. When $\mathbb{I} = \mathbb{Q}$ the lattice is not complete and sets such as $\{\ x \mid x^2 < 2\ \}$ are not defined for $\alpha^{Int}_{\mathcal{B}}$.

Interval Basis Operators. We define all the required operators for a basis the following way:

- The best $\cup_{\mathcal{B}}^{Int}$ and $\cap_{\mathcal{B}}^{Int}$ operators are exactly $\sqcup_{\mathcal{B}}^{Int}$ and $\sqcap_{\mathcal{B}}^{Int}$:

$$X^\sharp \cup_{\mathcal{B}}^{Int} Y^\sharp \quad \stackrel{\text{def}}{=} \quad \begin{cases} [\min(a,a'),\, \max(b,b')] & \text{if } X^\sharp = [a,b] \text{ and } Y^\sharp = [a',b'] \\ X^\sharp & \text{if } Y^\sharp = \bot_{\mathcal{B}}^{Int} \\ Y^\sharp & \text{if } X^\sharp = \bot_{\mathcal{B}}^{Int} \end{cases}$$

$$X^\sharp \cap_{\mathcal{B}}^{Int} Y^\sharp \quad \stackrel{\text{def}}{=} \quad \begin{cases} [\max(a,a'),\, \min(b,b')] & \text{if } X^\sharp = [a,b],\, Y^\sharp = [a',b'], \\ & \text{and } \max(a,a') \leq \min(b,b') \\ \bot_{\mathcal{B}}^{Int} & \text{otherwise} \end{cases}$$

  Note that $\sqcap_{\mathcal{B}}^{Int}$ is exact, but $\sqcup_{\mathcal{B}}^{Int}$ is not.

- The best forward operators are the classical interval arithmetic ones. Whenever one argument is $\bot_{\mathcal{B}}^{Int}$, the result is $\bot_{\mathcal{B}}^{Int}$; elsewhere we get:

$$[a,b]^{Int} \quad \stackrel{\text{def}}{=} \quad [a,b]$$

$$[a,b] \mathbin{+^{Int}} [a',b'] \quad \stackrel{\text{def}}{=} \quad [a+a',\, b+b']$$

$$[a,b] \mathbin{-^{Int}} [a',b'] \quad \stackrel{\text{def}}{=} \quad [a-b',\, b-a']$$

$$[a,b] \mathbin{\times^{Int}} [a',b'] \quad \stackrel{\text{def}}{=} \quad \begin{array}{l} [\min(a\times a',\, a\times b',\, b\times a',\, b\times b'), \\ \max(a\times a',\, a\times b',\, b\times a',\, b\times b')] \end{array}$$

$$[a,b] \mathbin{/^{Int}} [a',b'] \quad \stackrel{\text{def}}{=} \quad \begin{array}{ll} [adj(\min(a/a',\, a/b',\, b/a',\, b/b')), & \text{when } 0 \leq a' \\ adj(\max(a/a',\, a/b',\, b/a',\, b/b'))] & \end{array}$$

$$X \mathbin{/^{Int}} Y \quad \stackrel{\text{def}}{=} \quad \begin{array}{ll} (X \mathbin{/^{Int}} (Y \cap_{\mathcal{B}}^{Int} [0,+\infty])) \;\cup_{\mathcal{B}}^{Int} & \text{otherwise} \\ ((\mathbin{-^{Int}} X) \mathbin{/^{Int}} ((\mathbin{-^{Int}} Y) \cap_{\mathcal{B}}^{Int} [0,+\infty])) & \end{array}$$

  $+$ and $-$ should be extended to $+\infty$ and $-\infty$ the standard way — as the left bound is never $+\infty$ and the right bound is never $-\infty$, there is no undefined case.

  $\times$ is extended to $+\infty$ and $-\infty$ the usual way, except that we consider that $+\infty \times 0 \stackrel{\text{def}}{=} -\infty \times 0 \stackrel{\text{def}}{=} 0$.

  $/$ is extended to $+\infty$ and $-\infty$ by stating that $\forall x > 0,\, x/0 \stackrel{\text{def}}{=} +\infty$, $\forall x < 0,\, x/0 \stackrel{\text{def}}{=} -\infty$, and $\forall x,\, x/+\infty \stackrel{\text{def}}{=} 0$, but also $0/0 \stackrel{\text{def}}{=} 0$, so that $x/y = x \times (1/y)$ in all

cases. The abstract division is quite complex. First, we must split the dividend into a positive part and a negative part to correctly handle dividend intervals that cross zero. Then, we must use the *adj* truncation function introduced in Fig. 2.2 to get sound integer bounds when $\mathbb{I} = \mathbb{Z}$. The function *adj* is straightforwardly extended to $\pm\infty$ by stating that $adj(+\infty) \stackrel{\text{def}}{=} +\infty$ and $adj(-\infty) \stackrel{\text{def}}{=} -\infty$.

By density, these operators are exact when $\mathbb{I} = \mathbb{Q}$ or $\mathbb{I} = \mathbb{R}$, except when dividing by an interval containing 0 as the resulting concrete set is not convex. The situation is similar when $\mathbb{I} = \mathbb{Z}$ except that $\boldsymbol{\times}^{Int}$ is no longer exact: indeed, the multiplication enforces congruence properties that cannot be exactly represented by intervals.

- As backward arithmetic operators, we choose the generic ones derived, as in Sect. 2.4.4, from the forward ones.
- We now present the backward tests. Whenever one argument is $\bot^{Int}_{\mathcal{B}}$, the result is $(\bot^{Int}_{\mathcal{B}}, \bot^{Int}_{\mathcal{B}})$; elsewhere we get:

$$\boldsymbol{=}^{Int}([a,b],[a',b']) \stackrel{\text{def}}{=} ([a,b] \cap^{Int}_{\mathcal{B}} [a',b'],\ [a,b] \cap^{Int}_{\mathcal{B}} [a',b'])$$

$$\boldsymbol{\leq}^{Int}([a,b],[a',b']) \stackrel{\text{def}}{=} ([a,b] \cap^{Int}_{\mathcal{B}} [-\infty,b'],\ [a',b'] \cap^{Int}_{\mathcal{B}} [a,+\infty])$$

$$\boldsymbol{<}^{Int}([a,b],[a',b']) \stackrel{\text{def}}{=} \begin{cases} \boldsymbol{\leq}^{Int}([a,b],[a',b']) & \text{when } \mathbb{I} \in \{\mathbb{Q},\mathbb{R}\} \\ ([a,b] \cap^{Int}_{\mathcal{B}} [-\infty,b'-1],\ [a',b'] \cap^{Int}_{\mathcal{B}} [a+1,+\infty]) & \text{when } \mathbb{I} = \mathbb{Z} \end{cases}$$

$$\boldsymbol{\neq}^{Int}([a,b],[a',b']) \stackrel{\text{def}}{=} \boldsymbol{<}^{Int}([a,b],[a',b']) \cup^{Int}_{\mathcal{B}^2} \boldsymbol{<}^{Int}([a',b'],[a,b])$$

where $\cup^{Int}_{\mathcal{B}^2}$ preforms a component-wise union $\cup^{Int}_{\mathcal{B}}$ on two pairs of elements in $\mathcal{B}$.

- As widening and narrowing, we recall the original ones proposed by Cousot and Cousot in [CC76]:

$$[a,b] \triangledown^{Int}_{\mathcal{B}} [a',b'] \stackrel{\text{def}}{=} \left[\begin{cases} a & \text{if } a \leq a' \\ -\infty & \text{otherwise} \end{cases}, \begin{cases} b & \text{if } b \geq b' \\ +\infty & \text{otherwise} \end{cases} \right]$$

$$[a,b] \triangle^{Int}_{\mathcal{B}} [a',b'] \stackrel{\text{def}}{=} \left[\begin{cases} a' & \text{if } a = -\infty \\ a & \text{otherwise} \end{cases}, \begin{cases} b' & \text{if } b = +\infty \\ b & \text{otherwise} \end{cases} \right]$$

$$\bot^{Int}_{\mathcal{B}} \triangledown^{Int}_{\mathcal{B}} X \stackrel{\text{def}}{=} X \triangledown^{Int}_{\mathcal{B}} \bot^{Int}_{\mathcal{B}} \stackrel{\text{def}}{=} X$$

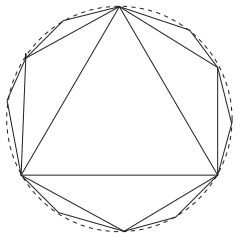


Figure 2.13: A strictly increasing infinite set of polyhedra whose limit is a disk.

$$\bot^{Int}_{\mathcal{B}} \vartriangle^{Int}_{\mathcal{B}} X \stackrel{\text{def}}{=} X \vartriangle^{Int}_{\mathcal{B}} \bot^{Int}_{\mathcal{B}} \stackrel{\text{def}}{=} \bot^{Int}_{\mathcal{B}}$$

2.4.7 The Polyhedron Abstract Domain

The polyhedron domain, introduced in [CH78] by Cousot and Halbwachs, allows manipulating conjunctions of linear inequalities or, equivalently, bounded and unbounded closed polyhedra in $\mathbb{I}^n$.

Representations. Internally, two dual representations can be used:

- a matrix / vector couple of constraint coefficients $\mathbf{M} \in \mathbb{I}^{m \times n}$, $\vec{B} \in \mathbb{I}^m$ with concretisation:
$$\gamma^{Poly}(< \mathbf{M}, \vec{B} >) \stackrel{\text{def}}{=} \{ \vec{X} \in \mathbb{I}^n \mid \mathbf{M}\vec{X} \geq \vec{B} \}$$
- a *frame*, that is, a set of vertices $\mathbf{V} = (\vec{V_1}, \ldots, \vec{V_k})$ and a set of rays $\mathbf{R} = (\vec{R_1}, \ldots, \vec{R_l})$ in $\mathbb{I}^n$ with concretisation:
$$\gamma^{Poly}(< \mathbf{V}, \mathbf{R} >) \stackrel{\text{def}}{=} \left\{ \sum_{i=1}^{k} \lambda_i \vec{V_i} + \sum_{i=1}^{l} \mu_i \vec{R_i} \mid \lambda_i \geq 0,\ \mu_i \geq 0,\ \sum_{i=1}^{k} \lambda_i = 1 \right\} .$$

There is no bound on the size of either representation, but experiments show that the polyhedron abstract domain has a time and memory cost exponential in the number of program variables in practice.

Lattice. There may be several different constraint or frame representations for a polyhedron; we will thus consider the set $\mathcal{D}^{Poly}$ of all polyhedra by not distinguishing these representations. The intersection of two polyhedra can be exactly represented as a polyhedron, while the union must be over-approximated by the *convex hull*. These operators embed $\mathcal{D}^{Poly}$ with a lattice structure. Unlike $\mathcal{D}^{Int}$, the $\mathcal{D}^{Poly}$ lattice is not complete. Consider, for instance, the infinite sequence of planar polygons with an increasing number of

edges in Fig. 2.13: each polygon includes all the preceding ones, and the limit is a disc that cannot be represented with a finite set of edges.

Transfer Functions and Operators. Transfer functions are provided for assignments, backward assignments, and tests of linear forms only, and these are exact. It is fairly easy to extend them to the case of *quasi*-linear forms, that is, linear forms where the constant coefficient can be an interval. However, no work seems devoted to the design of transfer functions for generic expressions, or even generic interval linear forms; we will present a partial solution to this problem in Chap. 6. Some transfer functions and operators — such as linear tests and intersections — are straightforward on the constraint representation while others — such as linear assignments and convex hulls — are more easily described using the frame representation. An algorithm, due originally to Chernikova and enhanced by Le Verge in [LV92], allows passing from one representation to the other one while minimising the number of constraints or vertices and rays. It has a worst-case exponential time and memory cost, simply because the size of a minimised representation is, at worse, exponential in the size of the minimised dual representation. Modern implementations such as the NewPolka library [Jea] and the Parma Polyhedra Library [PPL] dynamically choose the most useful representation and rely on Chernikova's algorithm to switch representations when needed. We do not present here these algorithms as they are quite complex but refer instead the reader to these libraries' documentation for more information.

Case $\mathbb{I} = \mathbb{Z}$. All algorithms used in the polyhedron domain rely on $\mathbb{I}$ being a field, so they only work when $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$. Problems related to the integer solutions of linear inequalities are known to be much more difficult than in $\mathbb{Q}$ and $\mathbb{R}$ — the simple satisfiability problem "jumps" from polynomial to NP-complete complexity. Operations that are quite simple and linear in $\mathbb{Q}$ and $\mathbb{R}$ are no longer linear on $\mathbb{Z}$; for instance, expressing the projection on the variable X of the set $S = \{\ (X, Y) \in \mathbb{Z}^2 \mid X = 2 \times Y\ \}$ require the use of a congruence relation: $X \equiv 0\ [2]$.

A customary solution in abstract interpretation is to use rational polyhedra to represent sets of points with integer coordinates. We keep all algorithms as if we had $\mathbb{I} = \mathbb{Q}$ and only change the concretisation of polyhedron P into: $\gamma^{Poly}(P) \cap \mathbb{Z}^n$. All transfer functions and operators are still sound, but no longer exact. For instance, the projection on X of the preceding set S would be $\mathbb{Z}$. In certain cases, we even lose optimality. This actually amounts to abstracting integers into rationals by forgetting their "integerness".

2.5 The Need for Relational Domains

We present here a few programs that cannot be analysed precisely enough with non-relational domains and motivate the search for low-cost relational domains.

Obviously, when the desired property is not representable in a non-relational domain, we need a more powerful domain:

Example 2.5.1. Property not representable in a non-relational domain.

In the following code:

```
①  X ← [0, 10];
②  if X ≤ Y { Y ← X }
③
```

The polyhedron domain is able to prove that, at ③, $Y \leq X$ but the interval domain is only able to prove that $Y \leq 10$.

●

Even when the property is exactly representable in a non-relational domain, some intermediate computations may not:

Example 2.5.2. Computation not possible in a non-relational domain.

Consider the following code:

```
①  X ← [0, 10];
②  Y ← [0, 10];
③  S ← X − Y;
④  if 2 ≤ S { ⑤ Y ← Y + 2 }
⑥
```

We always have $Y \in [0, 10]$ at ⑥. The polyhedron domain is able to discover this invariant because it is able to combine the information $S = X - Y$ and $S \geq 2$ at ⑤ to infer that $Y \leq X - 2$, that is $Y \leq 8$. As the interval domain "forgets" that $S = X - Y$, its upper bound for Y at ⑤ is 10 instead of 8, so, it infers that $Y \in [0, 12]$ at ⑥.

●

Recall that, for each loop, we have added a syntactic label ❶ ⋯ ❿ $\in \mathcal{L}$ between the `while` keyword and the subsequent loop condition. Any invariant discovered at this label will be called a *loop invariant* as it is true before each loop iteration. In order to discover a given invariant after the execution of the loop, it is generally necessary to find a loop invariant of a more complex form:

Example 2.5.3. Loop invariant not representable in a non-relational domain.
Suppose that we want to infer that $Y = 10$ at the end of the execution of the following program:

```
①  X ← 10;
②  Y ← 0;
③  while ❹ X ≠ 0 { X ← X − 1; Y ← Y + 1 }
⑤
```

Even though the required invariant has an interval form, the interval domain will fail to find it and discover the imprecise result $Y \geq 0$ instead. In order to discover that $Y = 10$ at ⑤, we must first infer the loop invariant $X + Y = 10$ at ❹. Combining this information with the fact that the loop ends when $X = 0$, we can infer that $Y = 10$ at ⑤. Relational domains such as the polyhedron domain, or the octagon domain presented in Chap. 4, are able of such a reasoning.

●

Our last example motivating the use of relational abstract domains is the analysis of programs where some variables are left unspecified:

Example 2.5.4. Symbolic invariant not representable in a non-relational domain.
Consider the following code:

```
①  X ← 0;
②  while X ≤ N {
③    if X < 0 or N < X { ④ fail };
⑤    X ← X + 1
⑥  }
```

Our goal is to prove that the control point ④ that triggers the failure of the program is never reached. The interval domain will be able to prove this assertion provided that N is a constant.[3] A relational domain such as the polyhedron, or the zone domain of Chap. 3, will be able to prove this assertion without knowing the exact value of N.

●

This ability of relational domains to discover *symbolic* invariants allows them to analyse programs in a modular way. For instance, a procedure such as Ex. 2.5.4 may be analysed out of its context. Moreover, a function can be abstracted as an invariant relating its output value to the values of its arguments. As our Simple language does not provide procedures, we will not insist on this aspect in this thesis and refer the reader to a survey [CC02] by Cousot and Cousot on the topic of modular static analysis.

[3]The discovery of the invariant $0 \leq X \leq N$, when N is a constant, in the interval domain requires the use of a narrowing to refine the invariant $0 \leq X$ discovered by iterations with widening solely.

2.6 Other Applications of Numerical Abstract Domains

Even though we focus in this thesis on the discovery of properties of numerical program variables, numerical abstract domains can be applied to other kinds of analysis.

Many non-uniform static analyses start by designing a non-standard instrumented semantics that introduces numerical quantities (or, equivalently, perform an automatic program re-writing that introduces numerical instrumentation variables) and then perform a numerical analysis on them. More complex analyses can perform reductions between several abstract domains, some of which are numerical. In some complex analyses, the set of numerical instrumentation variables is created dynamically by some kind of structural abstract domain that *drives* the numerical analysis.

In all these cases, the used numerical abstract domain is only a parameter of the analysis. The cost versus precision trade-off of the analysis is directly affected by the choice of the plugged numerical abstract domain. Interesting non-uniform properties can only be obtained if the chosen numerical abstract domain is relational. Thus, such analyses will directly benefit from the new domains introduced in the present thesis.

Pointer Analysis. There exists quite a lot of static analyses devoted to the analysis of pointers but few allow distinguishing the different instances of pointers in recursive data-structures. Such *non-uniform* analyses often *tag* these instances using a numerical scheme. In his seminal paper, [Deu94], Deutsch proposes a non-uniform pointer aliasing analysis that abstracts sets of access paths using counters. His framework is parametric in the numerical abstract domain chosen to abstract counter values. This allows discovering, for instance, properties of the form $x \rightarrow next^i = y \rightarrow next^j \Longrightarrow i \geq 0 \wedge i = j + 1$, using the set of counters $\{i, j\}$, meaning that y aliases the tail of the linked list x.

In order to analyse correctly loop-based manipulations of recursive structures or arrays, Venet proposes in [Ven02] a time-stamp scheme that relates each memory block to the value of the loop counters at its creation date. His framework is also parametric in a numerical abstract domain.

Quantitative Shape Analysis. Another noteworthy work is that of Rugina who proposes, in [Rug04], an analysis proving the correctness of the re-balancing algorithm used after an insertion in an AVL-tree. His method involves abstracting non-local numerical heap properties, such as the difference between the depth of the two sub-trees of a node. Even though his example only requires relational invariants of the simple form $x = y + c$, the methodology may be applied to more complex problems requiring richer numerical constraints.

String Cleanness. In the C programming language, character strings are represented by pointers to zero-terminated character buffers. Because strings live in user-allocated buffers the size of which is not explicitly available, most string manipulation functions in the C library and user code are not safe. Static string cleanness checking, introduced by Dor in [DRS01], aims at proving statically that a program does not perform erroneous string manipulation. In order to do this, strings are abstracted using a few numerical values encoding the size of the allocated buffers, the position of the terminating zero, and pointer offsets. Hence string cleanness is reduced to a purely numerical analysis problem.

$\pi-$calculus Analysis. In [Fer04a], Feret applies a technique similar to Venet's pointer time-stamping [Ven02] to discover properties of mobile programs written in the $\pi-$calculus. In particular, different instances of the same process can be distinguished using numerical values whose relationship are analysed using relational numerical abstract domains to infer non-uniform properties.

Parametric Predicate Abstraction. The idea of predicate abstraction is to consider, as abstract domain, a small set of logical predicates that focus on a complex property to be proved. In [Cou03], Cousot proposes to merge this idea with the use of numerical abstract domains to construct *infinite* families of predicates. As an example, the predicate $\forall (V_1, \ldots, V_n, X, Y, Z, T) \in \gamma(X^\sharp),\ \forall i \in [X, Y],\ \forall j \in [Z, T],\ a[i] \leq a[j]$ parametric in the numerical abstract element $X^\sharp$ allows proving that the bubble sort indeed sorts an array, provided the chosen numerical abstract domain for $X^\sharp$ is sufficiently precise — this works for the octagon abstract domain presented in Chap. 4. More examples are also proposed in [Ce03].

Termination Analysis. Abstract Interpretation is not limited to safety properties and reachability analyses. For instance, termination is a liveness property that can be proved using a numerical abstract domain by synthesising a *ranking function*, that is, a positive function that strictly decreases at each program step. Colón and Sipma propose in [CS01] a method based on the polyhedron abstract domain to synthesise non-trivial linear ranking functions.

Chapter 3

The Zone Abstract Domain

Ce chapitre présente le domaine abstrait des zones. Ce domaine permet de représenter et de manipuler des invariants de la forme $X - Y \leq c$ et $\pm X \leq c$. Il repose sur une structure de données déjà connue, les matrices de différences bornées (DBMs), étendue considérablement pour les besoins de l'interprétation abstraite. En particulier, nous proposons de nombreuses fonctions de transfert ainsi que des opérateurs d'élargissement et de rétrécissement. Grâce à une représentation compacte à base de matrices et une algorithmique basée sur des calculs de plus-court chemins, nous obtenons un coût $\mathcal{O}(n^2)$ en mémoire et $O(n^3)$ (au pire) en temps.

We introduce the zone abstract domain that is able to represent and manipulate invariants of the form $X - Y \leq c$ and $\pm X \leq c$. We use previous works on a data-structure called Difference Bound Matrices (DBMs) but considerably extend it to cope with Abstract Interpretation needs. In particular, we introduce many transfer functions, as well as widening and narrowing operators. Thanks to algorithms based on shortest-path closure and a compact matrix representation, we achieve a $\mathcal{O}(n^2)$ memory cost and a $O(n^3)$ worse-case time cost.

3.1 Introduction

We are interested in this chapter in the automatic discovery of constraints of the form $X - Y \leq c$ and $\pm X \leq c$, where X and Y are program variables and c is a constant. Such constraints are quite important as they frequently appear in loop invariants — see Fig. 3.1. Also, this type of constraints enjoys specific graph-based algorithms that are much more efficient than full linear programming, and so, we can build an abstract domain that is between, in terms of both cost and precision, the interval and the polyhedron domains.

```
X ← 1;
while ❶ X ≤ N {
  ⋮
  X ← X + 1
}
```

Figure 3.1: Typical loop example. The strongest loop invariant at ❶ is $1 \leq X \leq N + 1$, which is a zone constraint.

Previous Works on Difference Bound Matrices. The Difference Bound Matrix (or DBM) data-structure used in this chapter is borrowed from the work on model checking of timed automata [Yov98] and timed petri nets [MB83]. The algorithms we present to compute a canonical representation of DBMs, using the notion of shortest-path closure, as well as the intersection, inclusion, and equality tests have already been used for a long time; they are described, for instance, in [Yov98]. However, most algorithms needed by [Yov98] and [MB83] are not useful for Abstract Interpretation while many — such as: union abstractions, general assignment and test transfer functions, widenings, etc. — are missing. As the classical approach to model-checking is to first abstract by hand a problem into a computable model, and then compute on the model exactly, previous works on DBMs only focus on designing *exact* operators. Our approach is different as we only require *soundness* for our operators: each abstract operator application can result in some loss of precision. This is necessary if we are to abstract complex operators and transfer functions that have no exact counterpart on DBMs. It also gives us the flexibility to design several abstract versions of the same operator with different cost versus precision trade-offs.

Difference Bound Matrices Applied to Abstract Interpretation. The idea of using Difference Bound Matrices to design an abstract domain is not new. The possibility is suggested in the Ph. D. works of Bagnara [Bag97, Chap. 5] and Jeannet [Jea00, § 2.4.3]; a widening is even suggested. An actual use of DBMs in an analysis based on abstract interpretation is due to Shaman, Kolodner, and Sagiv, in [SKS00]. The authors only add to the classical DBM operators a widening and an approximated union. Compared to these previous works, ours is much more comprehensive. We propose abstract transfer functions for *all* assignments, tests, and backward assignments, not only those that can be abstracted exactly. We also examine carefully the interaction between the shortest-path closure algorithm on DBMs and the design of operators and transfer functions. It allows us to *prove* new best approximation results, but also expose flaws in previous works — such as the incorrect use of shortest-path closure on iterations with widening.

3.2 Constraints and Their Representation

Let $\mathcal{V} \stackrel{\text{def}}{=} \{V_1, \cdots, V_n\}$ be the finite set of program variables and let $\mathbb{I}$ be the numeric set they live in, $\mathbb{I} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$. We will often assimilate an environment $\rho \in (\mathcal{V} \to \mathbb{I})$ to a point in $\mathbb{I}^n$.

3.2.1 Constraints

Potential Constraints. A *potential constraint* is a constraint of the form $V_i - V_j \leq c$, with $c \in \mathbb{I}$. The term *potential* comes from the fact that solutions of conjunctions of potential constraints are defined up to a constant:

Theorem 3.2.1. Solutions of potential constraints conjunctions.

If $(v_1, \ldots, v_n) \in \mathbb{I}^n$ is a solution of a conjunction of potential constraints, then $\forall c \in \mathbb{I}$, $(v_1 + c, \ldots, v_n + c)$ is also a solution of the same conjunction of potential constraints.

●

Proof. Easy. ○

The set of points in $\mathbb{I}^n$ that satisfy a given conjunction of potential constraints will be called a *potential set*. We will denote by *Pot* the set of all potential sets for a fixed dimension n.

Zone Constraints. An interesting consequence of Thm. 3.2.1 is that if a conjunction of potential constraints is satisfiable, then it has a solution for which $V_1 = 0$. This allows the reduction of the satisfiability problem of a conjunction of constraints of the form $V_i - V_j \leq c$, $V_i \leq c$, and $V_i \geq c$ to the satisfiability of a conjunction of potential constraints. One simply adds a new synthetic variable V_0 and encode the constraints $V_i \leq c$ and $V_i \geq c$ respectively as $V_i - V_0 \leq c$ and $V_0 - V_i \leq -c$. Constraints of the form $V_i - V_j \leq c$, $V_i \leq c$, or $V_i \geq c$, that include both potential and interval constraints, will be called *zone constraints*. The set of points in $\mathbb{I}^n$ that satisfy a conjunction of zone constraints will be called a *zone*, and we will denote by *Zone* the set of zones.

3.2.2 Representations

Potential Graphs. A conjunction of potential constraints can be represented by a directed, weighted graph $\mathcal{G}$ with nodes in $\mathcal{V}$ and weights in $\mathbb{I}$, which is called a *potential graph*. For each ordered pair of variables $(V_i, V_j) \in \mathcal{V}^2$, there will be an arc from V_j to V_i with weight c if the constraint $V_i - V_j \leq c$ is in the constraint conjunction. We can assume,

without loss of generality, that there is at most one arc from any given node to any other given node as if two constraints with the same left member appear in the conjunction, the one with the largest right coefficient is obviously redundant.

We will call *path* in $\mathcal{G}$ a sequence of nodes, denoted by $\langle V_{i_1}, \ldots, V_{i_m} \rangle$, such that there is an arc from each V_{i_k} to $V_{i_{k+1}}$. The path is said to be *simple* if its *internal nodes* $V_{i_2}, \ldots, V_{i_{m-1}}$ are pairwise distinct, and different from V_{i_1} and V_{i_m}. A *cycle* is a path $\langle V_{i_1}, \ldots, V_{i_m} \rangle$ such that $V_{i_m} = V_{i_1}$. Note that a simple path may be a cycle, in which case it is called a *simple cycle*.

Difference Bound Matrices. Let $\overline{\mathbb{I}} \stackrel{\text{def}}{=} \mathbb{I} \cup \{+\infty\}$ be the extension of $\mathbb{I}$ to $+\infty$. The order $\leq$ is extended by stating that $\forall c \in \mathbb{I},\ c \leq +\infty$ — the extension of other operators to $\overline{\mathbb{I}}$ will be presented when needed. An equivalent representation of potential constraint conjunctions is by the mean of a *Difference Bound Matrix*, or *DBM* for short. A DBM $\mathbf{m}$ is a $n \times n$ square matrix with elements in $\overline{\mathbb{I}}$. The element at line i, column j, where $1 \leq i \leq n$, $1 \leq j \leq n$, denoted by $\mathbf{m}_{ij}$, equals c if there is a constraint of the form $V_j - V_i \leq c$ in our constraint conjunction, and $+\infty$ otherwise. We will denote by DBM the set of all DBMs. The potential set described by a DBM $\mathbf{m}$ is given by the following concretisation function:

Definition 3.2.1. Potential set concretisation γ^{Pot} of a DBM.

$$\gamma^{Pot}(\mathbf{m}) \stackrel{\text{def}}{=} \{\ (v_1, \ldots, v_n) \in \mathbb{I}^n \mid \forall i, j,\ v_j - v_i \leq \mathbf{m}_{ij}\ \}\ .$$

$$Pot \stackrel{\text{def}}{=} \{\ \gamma^{Pot}(\mathbf{m}) \mid \mathbf{m} \in \mathsf{DBM}\ \}\ .$$

●

Each DBM $\mathbf{m}$ can be seen as the adjacency matrix of a potential graph. It will be denoted in the following by $\mathcal{G}(\mathbf{m})$. Indeed, DBMs and potential graphs are just two different notations for the same objects. These notations are complementary: some theorems and algorithms will be best described using the matrix notation, while others will use graph-related terms, such as paths and cycles. The choice of a pointer-based hollow graph representation or a full matrix representation for actual implementation is completely orthogonal; it will be discussed in Sect. 3.8.2. In the following, we will often present examples using the graph notation even when the corresponding algorithms are presented using the matrix notation, as constraints graphs are much easier to read.

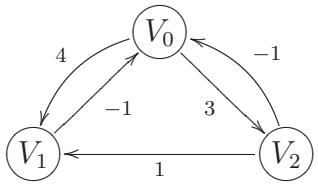
Zone Constraints Representation. Using the encoding in terms of potential constraints presented in the previous section, a conjunction of zone constraints will be represented as either a potential graph with an extra vertex V_0 that denotes the constant 0 or, equivalently, as a $(n+1) \times (n+1)$ DBM where the first line and column encode interval constraints. We will number lines and columns of DBMs representing zones from 0 to n, so

(a)
$$\begin{cases} V_1 \leq 4 \\ -V_1 \leq -1 \\ V_2 \leq 3 \\ -V_2 \leq -1 \\ V_1 - V_2 \leq 1 \end{cases}$$

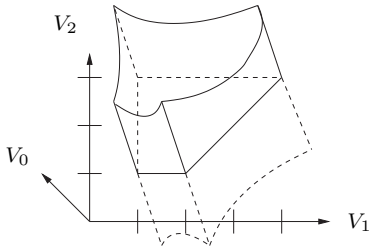
(b)

i \ j	V_0	V_1	V_2
V_0	$+\infty$	4	3
V_1	-1	$+\infty$	$+\infty$
V_2	-1	1	$+\infty$

(c)



(d)



(e)

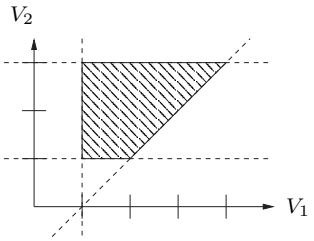


Figure 3.2: A zone constraint conjunction (a); the corresponding DBM $\mathbf{m}$ (b) and potential graph $\mathcal{G}(\mathbf{m})$ (c); the potential set concretisation $\gamma^{Pot}(\mathbf{m})$ (d) and zone concretisation $\gamma^{Zone}(\mathbf{m})$ (e).

that line or column i still corresponds to variable V_i, possibly V_0. The *zone concretisation* of a DBM, that is, the zone it represents, is defined by:

Definition 3.2.2. Zone concretisation γ^{Zone} of a DBM.

$$\gamma^{Zone}(\mathbf{m}) \stackrel{\text{def}}{=} \{\ (v_1, \ldots, v_n) \in \mathbb{I}^n \mid (0, v_1, \ldots, v_n) \in \gamma^{Pot}(\mathbf{m})\ \}\ .$$

$$\mathit{Zone} \stackrel{\text{def}}{=} \{\ \gamma^{Zone}(\mathbf{m}) \mid \mathbf{m} \in \mathsf{DBM}\ \}\ .$$

●

It is the intersection of the concretisation $\gamma^{Pot}(\mathbf{m})$ of the DBM as a potential set with the hyperplane $V_0 = 0$. Fig. 3.2 gives an example of a zone, its graph representation, and the corresponding DBM.

In this chapter, we are interested in manipulating zones and not potential sets which are less expressive. As there is an isomorphism between the set of zones in $\mathbb{I}^n$ and the set of potential sets in $\mathbb{I}^{n+1}$, most properties are uniformly true on *Zone* and *Pot*, but they

are easier to state and prove on potential sets which are more orthogonal. Also, we will see, in the next chapters, other domains constructed from DBMs for which properties also stem from properties of potential sets. This explains why we will often present properties on potential sets first, and later extend them to zones.

3.2.3 Lattice Structure

Matrix Ordering. Let us consider the total order $\leq$ on $\mathbb{I}$, extended to $\bar{\mathbb{I}}$ by $\forall x,\ x \leq +\infty$. Its point-wise extension to matrices gives a partial order on DBM denoted by $\sqsubseteq^{\mathsf{DBM}}$. Intuitively, $\mathbf{m} \sqsubseteq^{\mathsf{DBM}} \mathbf{n}$ means that each constraint in $\mathbf{m}$ is *tighter* than the corresponding constraint in $\mathbf{n}$. The set DBM has a natural greatest element $\top^{\mathsf{DBM}}$, defined as $\forall i, j,\ \top^{\mathsf{DBM}}_{ij} \stackrel{\text{def}}{=} +\infty$. However it has no smallest element, thus, we enrich DBM with a new smallest element denoted by $\bot^{\mathsf{DBM}}$. From now on, DBM will refer to this pointed domain and *bold* letters will refer to matrix elements in DBM, that is, any element but $\bot^{\mathsf{DBM}}$. DBM now forms a lattice:

Theorem 3.2.2. DBM lattice.

The set $(\mathsf{DBM}, \sqsubseteq^{\mathsf{DBM}}, \sqcup^{\mathsf{DBM}}, \sqcap^{\mathsf{DBM}}, \bot^{\mathsf{DBM}}, \top^{\mathsf{DBM}})$ *where:*

$$\begin{array}{ll}
\mathbf{m} \sqsubseteq^{\mathsf{DBM}} \mathbf{n} \iff \forall i, j,\ \mathbf{m}_{ij} \leq \mathbf{n}_{ij} & \bot^{\mathsf{DBM}} \sqsubseteq^{\mathsf{DBM}} X^\sharp \\
(\mathbf{m} \sqcup^{\mathsf{DBM}} \mathbf{n})_{ij} \stackrel{\text{def}}{=} \max(\mathbf{m}_{ij}, \mathbf{n}_{ij}) & \bot^{\mathsf{DBM}} \sqcup^{\mathsf{DBM}} X^\sharp \stackrel{\text{def}}{=} X^\sharp \sqcup^{\mathsf{DBM}} \bot^{\mathsf{DBM}} \stackrel{\text{def}}{=} X^\sharp \\
(\mathbf{m} \sqcap^{\mathsf{DBM}} \mathbf{n})_{ij} \stackrel{\text{def}}{=} \min(\mathbf{m}_{ij}, \mathbf{n}_{ij}) & \bot^{\mathsf{DBM}} \sqcap^{\mathsf{DBM}} X^\sharp \stackrel{\text{def}}{=} X^\sharp \sqcap^{\mathsf{DBM}} \bot^{\mathsf{DBM}} \stackrel{\text{def}}{=} \bot^{\mathsf{DBM}} \\
(\top^{\mathsf{DBM}})_{ij} \stackrel{\text{def}}{=} +\infty &
\end{array}$$

is a lattice. Moreover, if $\mathbb{I} \in \{\mathbb{Z}, \mathbb{R}\}$*, then this lattice is complete.*

●

Partial Galois Connection. If we extend γ^{Pot} and γ^{Zone} naturally by stating that $\gamma^{Pot}(\bot^{\mathsf{DBM}}) \stackrel{\text{def}}{=} \gamma^{Zone}(\bot^{\mathsf{DBM}}) \stackrel{\text{def}}{=} \emptyset$, we obtain complete $\sqcap-$morphisms — which are, thus, monotonic. Applying Thm. 2.2.3, we can deduce abstraction functions:

Definition 3.2.3. Zone and potential set abstractions.

- $\alpha^{Pot}(R) \stackrel{\text{def}}{=} \begin{cases} \bot^{\mathsf{DBM}} & \text{if } R = \emptyset \\ \mathbf{m} \text{ where } \mathbf{m}_{ij} \stackrel{\text{def}}{=} \min \{\ \rho(V_j) - \rho(V_i) \mid \rho \in R\ \} & \text{if } R \neq \emptyset \end{cases}$

(a)

i \ j	V_1	V_2	V_3
V_1	$+\infty$	4	3
V_2	-1	$+\infty$	$+\infty$
V_3	-1	1	$+\infty$

(b)

i \ j	V_1	V_2	V_3
V_1	0	5	3
V_2	-1	$+\infty$	$+\infty$
V_3	-1	1	$+\infty$

(c)

i \ j	V_1	V_2	V_3
V_1	0	4	3
V_2	-1	0	$+\infty$
V_3	-1	1	0

Figure 3.3: Three different DBMs with the same potential set concretisation as in Fig. 3.2. Remark that (a) and (b) are not even comparable with respect to $\sqsubseteq^{\mathsf{DBM}}$. Their closure is presented in (c).

- $\alpha^{Zone}(R) \stackrel{\text{def}}{=} \begin{cases} \bot^{\mathsf{DBM}} & \textit{if } R = \emptyset \\ \mathbf{m} \textit{ where } \mathbf{m}_{ij} \stackrel{\text{def}}{=} \begin{cases} 0 & \textit{when } i = j = 0 \\ \min \{ \rho(V_j) \mid \rho \in R \} & \textit{when } i = 0 \\ \min \{ -\rho(V_i) \mid \rho \in R \} & \textit{when } j = 0 \\ \min \{ \rho(V_j) - \rho(V_i) \mid \rho \in R \} & \textit{when } i, j \neq 0 \end{cases} & \\ & \textit{if } R \neq \emptyset \end{cases}$

●

$\mathcal{P}(\mathcal{V} \to \mathbb{I}) \underset{\alpha^{Pot}}{\overset{\gamma^{Pot}}{\leftrightarrows}} \mathsf{DBM}$ and $\mathcal{P}(\mathcal{V} \to \mathbb{I}) \underset{\alpha^{Zone}}{\overset{\gamma^{Zone}}{\leftrightarrows}} \mathsf{DBM}$ form partial Galois connections. We will see, in Sect. 3.5.2, that α^{Zone} is defined for any — bounded or unbounded — convex polyhedron, which ensures that the Galois connections are partial with respect to all transfer functions involving interval linear forms, at least. If $\mathbb{I} \in \{\mathbb{Z}, \mathbb{R}\}$, then α^{Pot} and α^{Zone} are total and we obtain regular Galois connections. If $\mathbb{I} = \mathbb{Q}$, then some abstract transfer functions involving non-linear expressions, such as ($X \times X \leq 2$?), have no best abstractions.

3.3 Canonical Representation

One must not confuse a set in *Zone* (resp. *Pot*) with a representation of such a set in DBM. In particular, γ^{Zone} is not injective, meaning that a zone can have several distinct DBM representations. These representations need not even be comparable in the abstract, as exemplified by Fig 3.3.

Thankfully, $(\alpha^{Zone}, \gamma^{Zone})$ is a $Id-$partial Galois connection, meaning that each zone has a canonical abstract representation that is minimal for $\sqsubseteq^{\mathsf{DBM}}$.

3.3.1 Emptiness Testing

We first consider the simpler case of determining whether $\gamma^{Zone}(\mathbf{m})$ (resp. $\gamma^{Pot}(\mathbf{m})$) is empty, that is, whether a conjunction of constraints in matrix form is satisfiable.

A core result that dates back to [Bel58] is that the satisfiability of a conjunction of potential constraints — and hence of zone constraints — can be tested by examining the simple cycles of the associated potential graph:

Theorem 3.3.1. Satisfiability of a conjunction of constraints.

1. $\gamma^{Pot}(\mathbf{m}) = \emptyset \iff \mathcal{G}(\mathbf{m})$ *has a cycle with a strictly negative total weight.*
2. *A graph has a cycle with a strictly negative total weight if and only if it has a* simple *cycle with a strictly negative total weight.*
3. $\gamma^{Zone}(\mathbf{m}) = \emptyset \iff \gamma^{Pot}(\mathbf{m}) = \emptyset$.

●

Proof. See, for instance, [CLR90, § 25.5] and [Pra77]. ○

The problem of checking for the weight of simple cycles can be reduced to the problem of *shortest path from an node*, that is, the weight of the path with minimal weight, originating from a given node. Indeed, the weight of the shortest path from a node belonging to a simple cycle with a strictly negative weight is unbounded and tends to $-\infty$ as it is always possible to prepend this cycle to a path once more to decrease its total weight. The idea is to add yet another node, V_∞, in the graph, as well as an arc with weight 0 from V_∞ to every other node in the graph. This makes every node accessible from V_∞, so, we only need to check the shortest path from V_∞ to decide whether the original graph has a simple cycle with a strictly negative weight.

The problem of determining the shortest path from a node has been well-studied and classical textbooks — such as [CLR90, § 25.5] — provide a profusion of algorithms. One of the simplest and most famous is due to Bellman and Ford, and runs in $\mathcal{O}(n \times s + n^2)$, where n is the number of nodes and s is the number of arcs in the graph. We do not insist on such techniques as we are about to provide an algorithm that will provide the emptiness information as a side-effect of solving a more complex problem.

3.3.2 Closure Operator

Whenever $\gamma^{Zone}(\mathbf{m})$, or equivalently $\gamma^{Pot}(\mathbf{m})$, is not empty, the potential graph $\mathcal{G}(\mathbf{m})$ has a *shortest-path closure* — or, more concisely, *closure* — that we will denote by $\mathcal{G}(\mathbf{m})^*$. This

shortest-path closure graph may be defined by its adjacency matrix $\mathbf{m}^*$: $\mathcal{G}(\mathbf{m})^* = \mathcal{G}(\mathbf{m}^*)$, where $\mathbf{m}^*$ is defined as follows:

Definition 3.3.1. Shortest-path closure.

$$\begin{cases} \mathbf{m}^*_{ii} \stackrel{\text{def}}{=} 0 \\ \mathbf{m}^*_{ij} \stackrel{\text{def}}{=} \min\limits_{\substack{1\leq m \\ \langle i=i_1,i_2,\ldots,i_m=j\rangle}} \sum\limits_{k=1}^{m-1} \mathbf{m}_{i_k i_{k+1}} & \text{if } i \neq j \end{cases}$$

where the min *and* + *operators are extended to* $\bar{\mathbb{I}}$ *as usual:*

$$\min(x, +\infty) \stackrel{\text{def}}{=} \min(+\infty, x) \stackrel{\text{def}}{=} x \text{ and } x + (+\infty) \stackrel{\text{def}}{=} (+\infty) + x \stackrel{\text{def}}{=} +\infty \ .$$

●

Existence of $\mathbf{m}^*$. We have supposed that $\gamma^{Zone}(\mathbf{m}) \neq \emptyset$ and so, by Thm. 3.3.1, $\mathbf{m}$ has no cycle with strictly negative total weight. This means that, in the definition of $\mathbf{m}^*$, we need to consider only *simple* paths as one can only increase the total weight of a path by inserting cycles. As there are only a finite number of simple paths from i to j, each $\mathbf{m}_{ij}$ is well-defined.

Propagating Constraints. Computing the closure of a DBM can be seen as propagating adjacent constraints. First note that $V_i - V_j \leq c$ and $V_j - V_k \leq d$ implies $V_i - V_k \leq c+d$. We now consider any path $\langle i = i_1, i_2, \ldots, i_m = j\rangle$ from V_i to V_j. By summing the constraints $V_{i_{k+1}} - V_{i_k} \leq \mathbf{m}_{i_k\, i_{k+1}}$, we deduce that any point $(v_1, \ldots, v_n) \in \gamma^{Pot}(\mathbf{m})$ will satisfy $v_j - v_i \leq \sum_k \mathbf{m}_{i_k\, i_{k+1}}$. We will call any constraint obtained by summing several adjacent constraints over a path in $\mathbf{m}$ an *implicit* constraint by opposition to constraints that appear explicitly in $\mathbf{m}$. What the closure $*$ does is to refine $\mathbf{m}$ by making all implicit constraints explicit. Additionally, all diagonal elements are replaced with 0, which is safe as they correspond to constraints of the form $V_i - V_i \leq 0$ and these are always true.

An immediate consequence of this reasoning is that the closure operator $*$ preserves the concretisation:

Theorem 3.3.2. Soundness of the closure $*$.

1. $\gamma^{Pot}(\mathbf{m}^*) = \gamma^{Pot}(\mathbf{m})$.
2. $\gamma^{Zone}(\mathbf{m}^*) = \gamma^{Zone}(\mathbf{m})$.

●

Saturation. A very useful property of the constraint conjunction associated to $\mathbf{m}^*$ is that it is *saturated*, that is, each constraint actually "touches" the set defined by the conjunction. We will see that many interesting properties are a consequence of this one.

Theorem 3.3.3. Saturation of closed DBMs.

1. $\forall i, j,$ *if* $\mathbf{m}^*_{ij} < +\infty,$ *then* $\exists (v_1, \ldots, v_n) \in \gamma^{Pot}(\mathbf{m})$ *such that* $v_j - v_i = \mathbf{m}^*_{ij}$.
2. $\forall i, j,$ *if* $\mathbf{m}^*_{ij} = +\infty,$ *then* $\forall M < +\infty$, $\exists (v_1, \ldots, v_n) \in \gamma^{Pot}(\mathbf{m})$ *such that* $v_j - v_i \geq M$.

●

Proof.

1. Set a pair (i_0, j_0) such that $\mathbf{m}^*_{i_0 j_0} < +\infty$.

   If $i_0 = j_0$, any $(v_1, \ldots, v_n) \in \gamma^{Pot}(\mathbf{m})$ will be such that $v_{j_0} - v_{i_0} = 0 = m^*_{i_0 j_0}$.

   Let us now consider the more complex case when $i_0 \neq j_0$.

   We denote by $\mathbf{m}'$ the matrix equal to $\mathbf{m}^*$ except for $\mathbf{m}'_{j_0 i_0} = -\mathbf{m}^*_{i_0 j_0}$. We first prove that $\gamma^{Pot}(\mathbf{m}') = \gamma^{Pot}(\mathbf{m}) \cap \{ (v_1, \ldots, v_n) \mid v_{j_0} - v_{i_0} = \mathbf{m}^*_{i_0 j_0} \}$.

   - By Thm. 3.3.2, $\gamma^{Pot}(\mathbf{m}^*) = \gamma^{Pot}(\mathbf{m}) \neq \emptyset$, so, by Thm. 3.3.1, there is no simple cycle in $\mathbf{m}^*$ with strictly negative weight. In particular, $\mathbf{m}^*_{i_0 j_0} + \mathbf{m}^*_{j_0 i_0} \geq 0$, so, $\mathbf{m}'_{j_0 i_0} = -\mathbf{m}^*_{i_0 j_0} \leq \mathbf{m}^*_{j_0 i_0}$.
     This means that $\forall i, j,\ \mathbf{m}'_{ij} \leq \mathbf{m}^*_{ij}$, so, $\gamma^{Pot}(\mathbf{m}') \subseteq \gamma^{Pot}(\mathbf{m}^*) = \gamma^{Pot}(\mathbf{m})$.
   - If $(v_1, \ldots, v_n) \in \gamma^{Pot}(\mathbf{m}')$, then $-\mathbf{m}'_{j_0 i_0} \leq v_{j_0} - v_{i_0} \leq \mathbf{m}'_{i_0 j_0}$. This means $\mathbf{m}^*_{i_0 j_0} \leq v_{j_0} - v_{i_0} \leq \mathbf{m}^*_{i_0 j_0}$.
     We just proved that $\gamma^{Pot}(\mathbf{m}') \subseteq \gamma^{Pot}(\mathbf{m}) \cap \{ (v_1, \ldots, v_n) \mid v_{j_0} - v_{i_0} = \mathbf{m}^*_{i_0 j_0} \}$.
   - If $(v_1, \ldots, v_n)$ is in $\gamma^{Pot}(\mathbf{m}) \cap \{ (v_1, \ldots, v_n) \mid v_{j_0} - v_{i_0} = \mathbf{m}^*_{i_0 j_0} \}$ then, by Thm. 3.3.2, $\forall i, j,\ v_j - v_i \leq \mathbf{m}^*_{ij}$ and $v_{i_0} - v_{j_0} \leq -\mathbf{m}^*_{i_0 j_0} = \mathbf{m}'_{j_0 i_0}$. So, $\forall i, j,\ v_j - v_i \leq \mathbf{m}'_{ij}$.

   Now suppose that $\gamma^{Pot}(\mathbf{m}')$ is empty. Then there exists a simple cycle in $\mathcal{G}(\mathbf{m}')$ with strictly negative weight. Either one of the two cases bellow occur, both leading to a contradiction.

   - If the new arc from j_0 to i_0 is not in this cycle, then we conclude that this cycle also exists in $\mathcal{G}(\mathbf{m})$ and that $\gamma^{Pot}(\mathbf{m})$ is empty, which is false.
   - If this cycle contains the arc from j_0 to i_0, since the cycle is simple, the arc cannot appear more than once and we can assume the cycle has the following form: $\langle i_0, i_1, \ldots, i_{N-1} = j_0, i_N = i_0 \rangle$. With this notation, we have:

$$\sum_{i=0}^{N-2} \mathbf{m}'_{i_k i_{k+1}} + \mathbf{m}'_{j_0 i_0} < 0,$$

so

$$\sum_{i=0}^{N-2} \mathbf{m}^*_{i_k i_{k+1}} < \mathbf{m}^*_{i_0 j_0}$$

and $\langle i_0, \ldots, i_{N-1} = j_0 \rangle$ is a path in $\mathcal{G}(\mathbf{m}^*)$ from i_0 to j_0 with weight strictly smaller than $\mathbf{m}^*_{i_0 j_0}$. As each $\mathbf{m}^*_{i_k i_{k+1}}$ is such that $i_k \neq i_{k+1}$, it can be replaced with the total weight along a path from i_k to i_{k+1} in $\mathcal{G}(\mathbf{m})$. By gluing these together, we construct a path from i_0 to j_0 in $\mathcal{G}(\mathbf{m})$ with total weight strictly smaller than $\mathbf{m}^*_{i_0 j_0}$, which contradicts the fact that $\mathbf{m}^*$ is closed.

We just proved that $\gamma^{Pot}(\mathbf{m}) \cap \{ (v_1, \ldots, v_n) \mid v_{j_0} - v_{i_0} = \mathbf{m}^*_{i_0 j_0} \} \neq \emptyset$ which proves the saturation property.

2. Set a pair (i_0, j_0) such that $\mathbf{m}^*_{i_0 j_0} = +\infty$ and $M \in \mathbb{I}$. We denote by $\mathbf{m}'$ the DBM equal to $\mathbf{m}^*$ except that $\mathbf{m}'_{j_0 i_0} = \min(\mathbf{m}^*_{j_0 i_0}, -M)$. We can prove the same way as for the first point that $\gamma^{Pot}(\mathbf{m}') = \gamma^{Pot}(\mathbf{m}) \cap \{ (v_1, \ldots, v_n) \mid v_{j_0} - v_{i_0} \geq M \}$ and $\gamma^{Pot}(\mathbf{m}') \neq \emptyset$.

○

Best Representation. We are now ready to prove that $*$ allows computing, among all the DBM representations for $\gamma^{Pot}(\mathbf{m})$ and $\gamma^{Zone}(\mathbf{m})$, the best, that is, the smallest for our abstract ordering $\sqsubseteq^{\mathsf{DBM}}$, provided that $\mathbf{m}$ has no simple cycle with a strictly negative weight:

Theorem 3.3.4. Best abstraction of potential sets and zones.

1. $\mathbf{m}^* = (\alpha^{Pot} \circ \gamma^{Pot})(\mathbf{m}) = \inf_{\sqsubseteq^{\mathsf{DBM}}} \{ \mathbf{n} \in \mathsf{DBM} \mid \gamma^{Pot}(\mathbf{m}) = \gamma^{Pot}(\mathbf{n}) \}$.
2. $\mathbf{m}^* = (\alpha^{Zone} \circ \gamma^{Zone})(\mathbf{m}) = \inf_{\sqsubseteq^{\mathsf{DBM}}} \{ \mathbf{n} \in \mathsf{DBM} \mid \gamma^{Zone}(\mathbf{m}) = \gamma^{Zone}(\mathbf{n}) \}$.

●

Proof.

1. $(\alpha^{Pot} \circ \gamma^{Pot})(\mathbf{m}) = \inf_{\sqsubseteq^{\mathsf{DBM}}} \{ \mathbf{n} \in \mathsf{DBM} \mid \gamma^{Pot}(\mathbf{m}) = \gamma^{Pot}(\mathbf{n}) \}$ is a direct consequence of choosing for α^{Pot} in Def. 3.2.3 the canonical abstraction associated to γ^{Pot}. We now prove that $\mathbf{m}^* = \inf_{\sqsubseteq^{\mathsf{DBM}}} \{ \mathbf{n} \in \mathsf{DBM} \mid \gamma^{Pot}(\mathbf{m}) = \gamma^{Pot}(\mathbf{n}) \}$. In order to do this, we first recall that, by Thm. 3.3.2, $\gamma^{Pot}(\mathbf{m}^*) = \gamma^{Pot}(\mathbf{m})$. Finally, we prove that $\gamma^{Pot}(\mathbf{n}) = \gamma^{Pot}(\mathbf{m}^*) \Longrightarrow \mathbf{m}^* \sqsubseteq^{\mathsf{DBM}} \mathbf{n}$. Suppose that $\gamma^{Pot}(\mathbf{n}) = \gamma^{Pot}(\mathbf{m}^*)$ but $\mathbf{m}^* \not\sqsubseteq^{\mathsf{DBM}} \mathbf{n}$. It means that for some i and j, $\mathbf{m}^*_{ij} > \mathbf{n}_{ij}$. We now use the saturation property — Thm. 3.3.3 — to construct $(v_1, \ldots, v_n) \in \gamma^{Pot}(\mathbf{m})$ such that $v_j - v_i \geq M$ for some M such that $\mathbf{n}^*_{ij} < M \leq \mathbf{m}^*_{ij}$. This point cannot satisfy the constraint $v_j - v_i \leq \mathbf{n}_{ij}$, and so, is not in $\gamma^{Pot}(\mathbf{n})$, which contradicts our hypothesis $\gamma^{Pot}(\mathbf{n}) = \gamma^{Pot}(\mathbf{m})$.

2. The proof is similar to that of the first point.

○

An immediate consequence is that $*$ is a normal form, that is $(\mathbf{m}^*)^* = \mathbf{m}^*$. Thus, we will say that $\mathbf{m}$ is *closed* if and only if $\mathbf{m}^* = \mathbf{m}$. Another consequence is that $\mathbf{m}^* \sqsubseteq^{\mathsf{DBM}} \mathbf{m}$. As an illustration, Fig. 3.3 presents two DBMs (a) and (b) together with their common closure (c).

3.3.3 Closure Algorithms

As for the problem of determining the shortest path from a node, mentioned in Sect. 3.3.1, the problem of determining the shortest-path closure graph has been widely studied and there exists several algorithms.

Floyd–Warshall Algorithm. The Floyd–Warshall algorithm [CLR90, § 26.2] computes $\mathbf{m}^*$ in cubic time by performing local transformations: for each node V_k, it checks for all pairs (V_i, V_j) whether it would be shorter to pass through V_k instead of taking the direct arc from V_i to V_j:

Definition 3.3.2. Floyd–Warshall algorithm.

$$\begin{cases} \mathbf{m}^0 \stackrel{\text{def}}{=} \mathbf{m} \\ \mathbf{m}^k_{ij} \stackrel{\text{def}}{=} \min(\mathbf{m}^{k-1}_{ij}, \mathbf{m}^{k-1}_{ik} + \mathbf{m}^{k-1}_{kj}) \quad \forall 1 \leq i,j,k \leq n \\ \mathbf{m}^*_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}^n_{ij} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \end{cases}$$

●

Def. 3.3.2 is fitted for DBMs representing potential sets where lines and columns are numbered from 1 to n. In order to adapt it to DBMs representing zones, whose lines and columns are numbered from 0 to n, one simply starts from $\mathbf{m}^{-1} \stackrel{\text{def}}{=} \mathbf{m}$ instead of $\mathbf{m}^0 \stackrel{\text{def}}{=} \mathbf{m}$ and applies the iterative steps for all $0 \leq i,j,k \leq n$.

A nice property of this algorithm is that it solves both problems of determining whether a graph has a simple cycle with strictly negative weight and computing its shortest-path closure when this is not the case:

Theorem 3.3.5. Floyd–Warshall algorithm properties.

1. $\mathbf{m}$ *has a cycle with strictly negative weight if and only if* $\exists i, \mathbf{m}^n_{ii} < 0$.
2. *If* $\forall i, \mathbf{m}^n_{ii} \geq 0$, *then* $\mathbf{m}^*$ *is the shortest-path closure of Def. 3.3.1.*

●

Proof.

The proof can be found, for instance, in [CLR90, § 26.2]. Nevertheless, we chose to include it here to pave the way towards the more complex proofs involved in the Floyd–Warshall algorithm extensions presented in Chaps. 4 and 5.

The algorithm invariant to prove is the following:

$$\forall i,j,k,\ \min_{\langle i=i_1,i_2,\ldots,i_m=j\rangle} \sum_{l=1}^{m-1} \mathbf{m}_{i_l i_{l+1}} \leq \mathbf{m}^k_{ij} \leq \min_{\substack{\langle i=i_1,i_2,\ldots,i_m=j\rangle \\ \text{simple paths such that} \\ i_l \leq k \text{ for } 1<l<m}} \sum_{l=1}^{m-1} \mathbf{m}_{i_l i_{l+1}} \ .$$

The first inequality is obvious, and we prove the second one by induction on k:

- This is obviously true for $k = 0$ because, on the one hand there is only one simple path from i to j with internal nodes lower than 0, which is $\langle i, j\rangle$, and, on the other hand, $\mathbf{m}^0 = \mathbf{m}$.
- Suppose the property is true for all $k' \leq k$. When $k' = k+1$, consider a simple path $\langle i = i_1, i_2, \ldots, i_m = j\rangle$ from i to j passing only through internal nodes such that $i_l \leq k+1$. One of the following occurs:
  - either $i_l \leq k$ for all $1 < l < m$, so, $\mathbf{m}^{k+1}_{ij} \leq \mathbf{m}^k_{ij} \leq \sum_{l=1}^{m-1} \mathbf{m}_{i_l i_{l+1}}$
  - or their exists a unique o between 1 and m such that $i_o = k+1$, and we can decompose the path into two sub-paths $\langle i = i_1, \ldots, i_o = k+1\rangle$ and $\langle k+1 = i_o, \ldots, i_m = j\rangle$ with internal nodes only smaller than k: $\mathbf{m}^{k+1}_{ij} \leq \mathbf{m}^k_{i\,(k+1)} + \mathbf{m}^k_{(k+1)\,j} \leq \sum_{l=1}^{o-1} \mathbf{m}_{i_l i_{l+1}} + \sum_{l=o}^{m-1} \mathbf{m}_{i_l i_{l+1}} = \sum_{l=1}^{m-1} \mathbf{m}_{i_l i_{l+1}}$.

The first inequality when $k = n$ states that if $\exists i$, $\mathbf{m}^n_{ii} < 0$, then there is a cycle passing through i that has a strictly negative weight. Conversely, if there is a cycle with strictly negative weight, then, by Thm. 3.3.1.2, there is also a *simple* cycle with strictly negative weight and the second inequality guarantees that $\exists i$, $\mathbf{m}^n_{ii} < 0$.

If there is no such cycle, the minimum weight on the set of paths from i to j is the minimum weight on the set of *simple* paths from i to j, so, by applying our induction hypothesis for $k = n$, we get:

$$\min_{\langle i=i_1,i_2,\ldots,i_m=j\rangle} \sum_{l=1}^{m-1} \mathbf{m}_{i_l i_{l+1}} = \mathbf{m}^n_{ij} = \min_{\substack{\langle i=i_1,i_2,\ldots,i_m=j\rangle \\ \text{simple path}}} \sum_{l=1}^{m-1} \mathbf{m}_{i_l i_{l+1}}$$

which concludes the proof.

○

In order to check whether $\mathbf{m}$ has a cycle with strictly negative weight, it is sufficient to check for trivial cycles $\langle i, i\rangle$ in $\mathbf{m}^n$, that is, diagonal elements $\mathbf{m}^n_{ii}$ of $\mathbf{m}^n$ such that

$\mathbf{m}^n_{ii} < 0$. In practice, it can be worth checking the diagonal elements of all the intermediate matrices $\mathbf{m}^k$: sometimes, an infeasible cycle is detected quite early, saving subsequent useless computations.

Other nice properties of the Floyd–Warshall algorithm are its ease of implementation and its interpretation as *local constraints propagation* that will allow us to generalise it to other types of constraints in the following two chapters.

In-Place Implementation. A naive implementation of Def. 3.3.2 would make use of n temporary matrices, $\mathbf{m}^1$ to $\mathbf{m}^n$, which is a waste of memory. A first optimisation is to remark that only two matrices are needed at a given time to implement Def. 3.3.2 as $\mathbf{m}^{k+1}$ is defined using solely $\mathbf{m}^k$: we need only keeping the currently computed DBM as well as the one computed in the last pass. In fact, we can do even better: we can update the matrix in-place without any extra storage requirement, as proposed by the following algorithm:

Definition 3.3.3. In-place Floyd–Warshall algorithm.

```
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
      m_ij ← min(m_ij, m_ik + m_kj)

for i = 1 to n
  if m_ii < 0 return ⊥^DBM else m_ii ← 0

return m
```

●

The intermediate matrices computed by this algorithm may be quite different from that of Def. 3.3.2, but the result can be proved to be equal — see [CLR90, § 26.2], for instance. Although Def. 3.3.3 is much nicer from an implementation point of view, we will keep reasoning using the original version of Def. 3.3.2 because it is much simpler to describe mathematically.

Johnson Algorithm. Another famous algorithm is the Johnson algorithm which computes the shortest-path closure in $\mathcal{O}(n \times s + n^2 \log n)$ time, where n is the number of nodes and s is the number of arcs in the graph. It is based on the Bellman–Ford algorithm and is much more complicated to implement, but it is more efficient than the Floyd–Warshall algorithm when there are only few arcs. Experimental results show that DBMs are often very full which means that the complexity benefit is thin in practice. Thus, we will stick to the Floyd–Warshall algorithm and refer the reader to [CLR90, § 26.3] for a presentation of the alternate Johnson algorithm.

Extending the Closure Operator $*$. A natural way to extend the definition of the closure $*$ as a total operator in DBM is to state that $(\bot^{\mathsf{DBM}})^* \stackrel{\text{def}}{=} \bot^{\mathsf{DBM}}$ and $\mathbf{m}^* \stackrel{\text{def}}{=} \bot^{\mathsf{DBM}}$ whenever $\gamma(\mathbf{m}) = \emptyset$. This way, the closure operator $*$ implements $\alpha^{Pot} \circ \gamma^{Pot}$ or, equivalently, $\alpha^{Zone} \circ \gamma^{Zone}$ in all cases.

3.3.4 Incremental Closure

We propose here an *incremental* version of the Floyd–Warshall algorithm that is able to compute the closure of an "almost closed" matrix. It can be used after a local modification on a closed matrix that resulted in the matrix being no longer closed: it will be faster than a full closure application.

We first propose a *local* characterisation of closed matrices:

Theorem 3.3.6. Local characterisation of closed matrices.

$$\mathbf{m} \textit{ is closed} \iff \forall i,j,k,\ \mathbf{m}_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj} \textit{ and } \forall i,\ \mathbf{m}_{ii} = 0 \ .$$

●

Proof.

The direction $\forall i,j,\ \mathbf{m}^*_{ij} = \mathbf{m}_{ij} \Longrightarrow \forall i,j,k,\ \mathbf{m}_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj}$ and $\forall i,\ \mathbf{m}_{ii} = 0$ is easy. For all $i \neq j$, we use the path $\langle i,k,j\rangle$ in the definition of the closure of Def. 3.3.1 to get $\mathbf{m}_{ij} = \mathbf{m}^*_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj}$. For all i we have, by definition, $\mathbf{m}_{ii} = 0 = \mathbf{m}^*_{ii}$.

For the other direction, suppose that $\forall i,j,k,\ \mathbf{m}_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj}$ and $\forall i,\ \mathbf{m}_{ii} = 0$.
Given i and j, we prove that $\mathbf{m}_{ij} = \mathbf{m}^*_{ij}$. If $i = j$, we know that $\mathbf{m}^*_{ii} = 0$, so $\mathbf{m}_{ii} = \mathbf{m}^*_{ii}$. If $i \neq j$, we already know that $\mathbf{m}^*_{ij} \leq \mathbf{m}_{ij}$ and we can use an induction on l to prove that:

$$\forall i,j,\ \forall l \geq 1,\ \forall \langle i = i_1, \ldots, i_l = j\rangle,\ \mathbf{m}_{ij} \leq \sum_{k=1}^{l-1} \mathbf{m}_{i_k i_{k+1}}$$

so that $\forall i \neq j,\ \mathbf{m}_{ij} \leq \mathbf{m}^*_{ij}$.
Here is a proof by induction of this statement:

- If $l = 2$, then the property is obvious.
- If $l > 2$ and the path is $\langle i = i_1, \ldots, i_{l-1}, i_l = j\rangle$, then we use the induction hypothesis to get:
$$\mathbf{m}_{i\, i_{l-1}} \leq \sum_{k=1}^{l-2} \mathbf{m}_{i_k i_{k+1}}$$
and the fact that $\mathbf{m}_{ij} \leq \mathbf{m}_{i\, i_{l-1}} + \mathbf{m}_{i_{l-1}\, j}$.

○

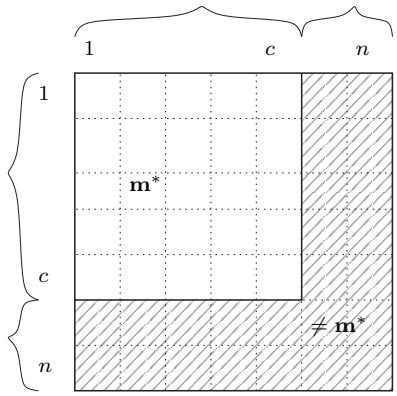


Figure 3.4: A matrix $\mathbf{n}$ equal to $\mathbf{m}^*$ except for the last $n-c$ lines and columns.

Suppose now that $\mathbf{m}$ is a closed DBM representing a potential set, that is, with lines and columns numbered from 1 to n. Suppose also that $\mathbf{n}$ equals $\mathbf{m} = \mathbf{m}^*$ except for the last $n-c$ lines and columns: $\forall 1 \leq i, j \leq c$, $\mathbf{n}_{ij} = \mathbf{m}_{ij}$, which is sketched in Fig. 3.4. From a constraint point of view, this means that we may have altered only constraints that contain at least a variable in $V_{c+1}, \ldots, V_n$. The following algorithm, inspired from the Floyd–Warshall algorithm, computes $\mathbf{n}^*$ in time proportional to $n^3 - c^3$:

Definition 3.3.4. Incremental Floyd–Warshall algorithm.

$$\begin{cases} \mathbf{n}^0 \stackrel{\text{def}}{=} \mathbf{n} \\ \mathbf{n}^k_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{n}^{k-1}_{ij} & \text{if } i \leq c,\ j \leq c, \text{ and } k \leq c \\ \min(\mathbf{n}^{k-1}_{ij},\ \mathbf{n}^{k-1}_{ik} + \mathbf{n}^{k-1}_{kj}) & \text{otherwise} \end{cases} \\ \mathbf{n}^*_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{n}^n_{ij} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \end{cases}$$

●

In the first c iterations, the algorithm only updates the last $n-c$ lines and columns of $\mathbf{n}$. The last $n-c$ iterations update the whole matrix, as the regular Floyd–Warshall algorithm does.

Theorem 3.3.7. Incremental Closure.

$\mathbf{n}^*$ as computed by the incremental Floyd–Warshall algorithm of Def. 3.3.4 is equal to $\mathbf{n}^*$ as computed by the vanilla Floyd–Warshall algorithm of Def. 3.3.2.

●

Proof.

By local closure property — Thm. 3.3.6 — we have $\mathbf{m}_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj}$, and so, $\forall i, j, k \leq c$, $\mathbf{n}_{ij} \leq \mathbf{n}_{ik} + \mathbf{n}_{kj}$. Thus, the c first steps of the regular Floyd–Warshall algorithm do not modify $\mathbf{n}_{ij}^k$ whenever $i, j \leq c$. The incremental closure simply uses this information to avoid useless computation.

○

By virtually reordering the variables, it is straightforward to design an incremental closure algorithm that works when the $n-c$ modified lines and columns are not necessarily at the end of the matrix. We will denote by $Inc^*_{i_1,\ldots,i_{n-c}}(\mathbf{m})$ the application of the preceding algorithm to the lines and columns numbered $i_1, \ldots, i_{n-c}$. A very common case is when only one line and the corresponding column have been changed; in this case the algorithm runs in quadratic time $\mathcal{O}(n^2)$. It is important to remark that $Inc^*_{i_1,i_2}$ is not equivalent to $Inc^*_{i_1} \circ Inc^*_{i_2}$: our incremental closure algorithm must treat all the modified lines and columns at once. Note also that an in-place version of the incremental closure algorithm in the spirit of Def. 3.3.3, easier to implement but harder to reason about, can be easily designed.

3.4 Set-Theoretic Operators

We now propose abstract operators on DBM that correspond to set-theoretic operators on *Pot* and *Zone*, including $\cup^\sharp$ and $\cap^\sharp$ that are necessary to define an abstract domain.

3.4.1 Equality Testing

We are able to compare two DBMs $\mathbf{m}_1$ and $\mathbf{m}_2$ using the $\sqsubseteq^{\mathsf{DBM}}$ order, but this does not always allow comparing two potential sets or zones as our γ operators are not injective. The following theorem uses the "normal form" property of the closure to solve this problem:

Theorem 3.4.1. Equality testing.

1. $\mathbf{m}_1^* = \mathbf{m}_2^* \iff \gamma^{Pot}(\mathbf{m}_1) = \gamma^{Pot}(\mathbf{m}_2)$.
2. $\mathbf{m}_1^* = \mathbf{m}_2^* \iff \gamma^{Zone}(\mathbf{m}_1) = \gamma^{Zone}(\mathbf{m}_2)$.

●

Proof.

1. The $\Longrightarrow$ part is a consequence of Thm. 3.3.2. To prove the $\Longleftarrow$ part, we first apply α^{Pot} to get $(\alpha^{Pot} \circ \gamma^{Pot})(\mathbf{m}_1) = (\alpha^{Pot} \circ \gamma^{Zone})(\mathbf{m}_2)$, and then use the fact that $\forall \mathbf{m} \in \mathsf{DBM}$, $\mathbf{m}^* = (\alpha^{Pot} \circ \gamma^{Pot})(\mathbf{m})$ proved by Thm. 3.3.4.

2. This is a consequence of the first point and the fact that $\gamma^{Zone}(\mathbf{m}_1) = \gamma^{Zone}(\mathbf{m}_2) \iff \gamma^{Pot}(\mathbf{m}_1) = \gamma^{Pot}(\mathbf{m}_2)$.

○

Testing for equality is done point-wisely, and so, has a quadratic cost, not counting the cubic cost of the closure operator.

3.4.2 Inclusion Testing

The situation is similar when testing inclusion: $\mathbf{m} \sqsubseteq^{\mathsf{DBM}} \mathbf{n} \Longrightarrow \gamma^{Pot}(\mathbf{m}) \subseteq \gamma^{Pot}(\mathbf{n})$ but the converse does not hold in general. As for equality testing, we will need to compute matrix closures first. However, it is not necessary to close the right argument:

Theorem 3.4.2. Inclusion testing.

1. $\mathbf{m}_1^* \sqsubseteq^{\mathsf{DBM}} \mathbf{m}_2 \iff \gamma^{Pot}(\mathbf{m}_1) \subseteq \gamma^{Pot}(\mathbf{m}_2)$.
2. $\mathbf{m}_1^* \sqsubseteq^{\mathsf{DBM}} \mathbf{m}_2 \iff \gamma^{Zone}(\mathbf{m}_1) \subseteq \gamma^{Zone}(\mathbf{m}_2)$.

●

Proof.

1. The $\Longrightarrow$ part is a consequence of Thm. 3.3.2 and the monotonicity of γ^{Pot}. To prove the $\Longleftarrow$ part, we first apply the monotonic α^{Pot} to get $(\alpha^{Pot} \circ \gamma^{Pot})(\mathbf{m}_1) \sqsubseteq^{\mathsf{DBM}} (\alpha^{Pot} \circ \gamma^{Pot})(\mathbf{m}_2)$, and then use the fact that $\forall \mathbf{m} \in \mathsf{DBM}$, $\mathbf{m}^* = (\alpha^{Pot} \circ \gamma^{Pot})(\mathbf{m})$, proved by Thm. 3.3.4, to get $\mathbf{m}_1^* \sqsubseteq^{\mathsf{DBM}} \mathbf{m}_2^*$. We conclude by remarking that $\mathbf{m}_2^* \sqsubseteq^{\mathsf{DBM}} \mathbf{m}_2$ always holds.
2. This is a consequence of the first point and the fact that $\gamma^{Zone}(\mathbf{m}_1) \subseteq \gamma^{Zone}(\mathbf{m}_2) \iff \gamma^{Pot}(\mathbf{m}_1) \subseteq \gamma^{Pot}(\mathbf{m}_2)$.

○

3.4.3 Union Abstraction

The union $\cup$ of two zones (resp. potential sets) may not be a zone. Indeed, zones are convex sets, which is not a property preserved by union. By monotonicity of γ^{Zone}, $\mathbf{m} \sqcup^{\mathsf{DBM}} \mathbf{n}$ gives a sound abstraction of $\gamma^{Zone}(\mathbf{m}) \cup \gamma^{Zone}(\mathbf{n})$. A less obvious fact is that the precision of this operator greatly depends on which DBM argument is used among all DBMs that represent the same zone. In particular, the *best* abstraction for the union is only reached when the arguments are closed DBMs, as illustrated in Fig. 3.5. Thus, we define our union abstractions as follows:

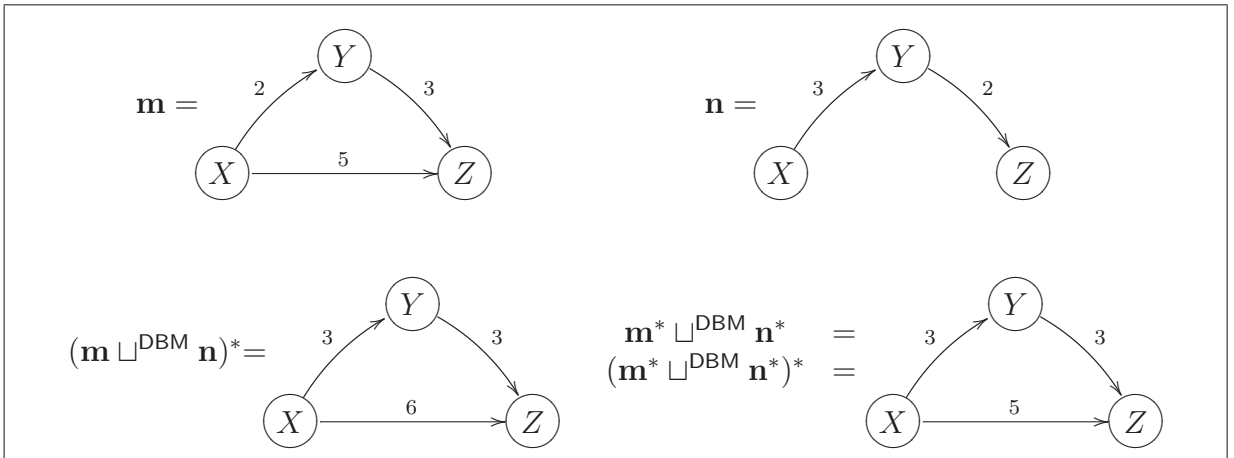


Figure 3.5: Least upper bounds of zones based on $\sqcup^{\mathsf{DBM}}$. DBMs should be closed for best precision. This also ensures that the result is closed.

Definition 3.4.1. Union abstractions.

$$\mathbf{m} \cup^{Pot} \mathbf{n} \stackrel{\text{def}}{=} (\mathbf{m}^*) \sqcup^{\mathsf{DBM}} (\mathbf{n}^*) \ .$$
$$\mathbf{m} \cup^{Zone} \mathbf{n} \stackrel{\text{def}}{=} (\mathbf{m}^*) \sqcup^{\mathsf{DBM}} (\mathbf{n}^*) \ .$$

●

Theorem 3.4.3. Properties of the union abstractions.

1. $\gamma^{Pot}(\mathbf{m} \cup^{Pot} \mathbf{n}) = \inf_{\subseteq}\{ S \in Pot \mid S \supseteq \gamma^{Pot}(\mathbf{m}) \cup \gamma^{Pot}(\mathbf{n}) \}$. *(best abstraction)*
2. $\gamma^{Zone}(\mathbf{m} \cup^{Zone} \mathbf{n}) = \inf_{\subseteq}\{ S \in Zone \mid S \supseteq \gamma^{Zone}(\mathbf{m}) \cup \gamma^{Zone}(\mathbf{n}) \}$. *(best abstraction)*
3. $\mathbf{m} \cup^{Pot} \mathbf{n}$ *and* $\mathbf{m} \cup^{Zone} \mathbf{n}$ *are closed.*

●

Proof.

1. We first prove $\mathbf{m} \cup^{Pot} \mathbf{n} = \inf_{\sqsubseteq^{\mathsf{DBM}}}\{ \mathbf{o} \mid \gamma^{Pot}(\mathbf{o}) \supseteq \gamma^{Pot}(\mathbf{m}) \cup \gamma^{Pot}(\mathbf{n}) \}$, which is a stronger result.

   Whenever $\gamma^{Pot}(\mathbf{m}) = \emptyset$ (resp. $\gamma^{Pot}(\mathbf{n}) = \emptyset$), $\mathbf{m}^* = \bot^{\mathsf{DBM}}$ and the property is obvious.

   We now suppose that $\gamma^{Pot}(\mathbf{m})$, $\gamma^{Pot}(\mathbf{n}) \neq \emptyset$. By Thm. 3.3.2 and the monotonicity of γ^{Pot}, we get that $\mathbf{m} \cup^{Pot} \mathbf{n}$ effectively over-approximates $\gamma^{Pot}(\mathbf{m}) \cup \gamma^{Pot}(\mathbf{n})$ because $\gamma^{Pot}(\mathbf{m}) \cup \gamma^{Pot}(\mathbf{n}) = \gamma^{Pot}(\mathbf{m}^*) \cup \gamma^{Pot}(\mathbf{n}^*) \subseteq \gamma^{Pot}(\mathbf{m}^* \sqcup^{\mathsf{DBM}} \mathbf{n}^*) = \gamma^{Pot}(\mathbf{m} \cup^{Pot} \mathbf{n})$.

   We now suppose that $\gamma^{Pot}(\mathbf{o}) \supseteq \gamma^{Pot}(\mathbf{m}) \cup \gamma^{Pot}(\mathbf{n})$ and prove that $\mathbf{m} \cup^{Pot} \mathbf{n} \sqsubseteq^{\mathsf{DBM}} \mathbf{o}$. Set two indexes i_0 and j_0 such that $\mathbf{m}^*_{i_0 j_0}$, $\mathbf{n}^*_{i_0 j_0} < +\infty$. Using the saturation property

of the closure (Thm. 3.3.3) we can find two points $v^{\mathbf{m}} \in \gamma^{Pot}(\mathbf{m})$ and $v^{\mathbf{n}} \in \gamma^{Pot}(\mathbf{n})$ such that $v^{\mathbf{m}}_{j_0} - v^{\mathbf{m}}_{i_0} = \mathbf{m}^*_{i_0 j_0}$ and $v^{\mathbf{n}}_{j_0} - v^{\mathbf{n}}_{i_0} = \mathbf{n}^*_{i_0 j_0}$. As these two elements are also in $\gamma^{Pot}(\mathbf{o})$, we have $v^{\mathbf{m}}_{j_0} - v^{\mathbf{m}}_{i_0} \leq \mathbf{o}_{i_0 j_0}$ and $v^{\mathbf{n}}_{j_0} - v^{\mathbf{n}}_{i_0} \leq \mathbf{o}_{i_0 j_0}$, which means that $\mathbf{o}_{i_0 j_0} \geq \max(\mathbf{m}^*_{i_0 j_0}, \mathbf{n}^*_{i_0 j_0}) = (\mathbf{m} \cup^{Pot} \mathbf{n})_{i_0 j_0}$. Whenever $\mathbf{m}^*_{i_0 j_0} = +\infty$ or $\mathbf{n}^*_{i_0 j_0} = +\infty$, the same reasoning allows proving that $\mathbf{o}_{i_0 j_0} \geq M$ for every M, that is, $\mathbf{o}_{i_0 j_0} = +\infty = (\mathbf{m} \cup^{Pot} \mathbf{n})_{i_0 j_0}$.

Because γ^{Pot} is a complete $\sqcap-$morphism, $\mathbf{m} \cup^{Pot} \mathbf{n} = \inf_{\sqsubseteq^{\mathsf{DBM}}} \{\ \mathbf{o} \mid \gamma^{Pot}(\mathbf{o}) \supseteq \gamma^{Pot}(\mathbf{m}) \cup \gamma^{Pot}(\mathbf{n})\ \}$ implies $\gamma^{Pot}(\mathbf{m} \cup^{Pot} \mathbf{n}) = \inf_{\subseteq} \{\ S \in Pot \mid S \supseteq \gamma^{Pot}(\mathbf{m}) \cup \gamma^{Pot}(\mathbf{n})\ \}$.

2. This is a consequence of the first point and the isomorphism between *Pot* and *Zone*.
3. This is a consequence of $\mathbf{m} \cup^{Pot} \mathbf{n} = \inf_{\sqsubseteq^{\mathsf{DBM}}} \{\ \mathbf{o} \mid \gamma^{Pot}(\mathbf{o}) \supseteq \gamma^{Pot}(\mathbf{m}) \cup \gamma^{Pot}(\mathbf{n})\ \}$, proved in the first point, and Thm. 3.3.4.

○

Related Work. No abstract union operator for DBMs is used in the model-checking community as it is not exact. The idea of taking point-wise maximums of upper bounds of constraints has already been proposed by Balasundram and Kennedy in [BK89]. Even though the authors are well-aware of the necessity to consider matrix arguments with the tightest possible constraints to achieve optimal precision, they fail to present an algorithmic way to construct such arguments, that is, our closure. The present abstract union operator seems to have been introduced in its complete form for the first time by Shaman, Kolodner, and Sagiv in [SKS00]. However, the authors perform an explicit closure on the result: they did not remark that this is useless, as explained by Thm. 3.4.3.3.

Disjunctive Completion. Unions appear very frequently in a static analysis: each time we encounter a control-flow join after a conditional or loop iteration. As our abstract union is not exact, this can lead to a severe precision loss in the analysis result. One solution to this problem is the *disjunctive completion* construction, proposed by Cousot and Cousot in [CC92b], which can enrich any abstract domain into a *distributive* abstract domain, that is, one that can represent disjunctions exactly. The idea is to consider, as new abstract elements, sets of abstract values; such a set represents the *union* of the concretisations of the elements. However, this is a theoretical construction and we still need to design efficient data-structures and algorithms. This must be done on a domain-per-domain basis and we now focus on the related work on the disjunctive completion for zones.

In the world of model-checking of timed-automata [Yov98], it is customary to represent an abstract environment as a finite set of DBMs to represent exactly both conjunctions and disjunctions of constraints of the form $X - Y \leq c$. This has several drawbacks:

- The number of DBMs in each abstract set can grow very quickly. In particular, intersecting two sets of m DBMs results in a set containing m^2 DBMs.

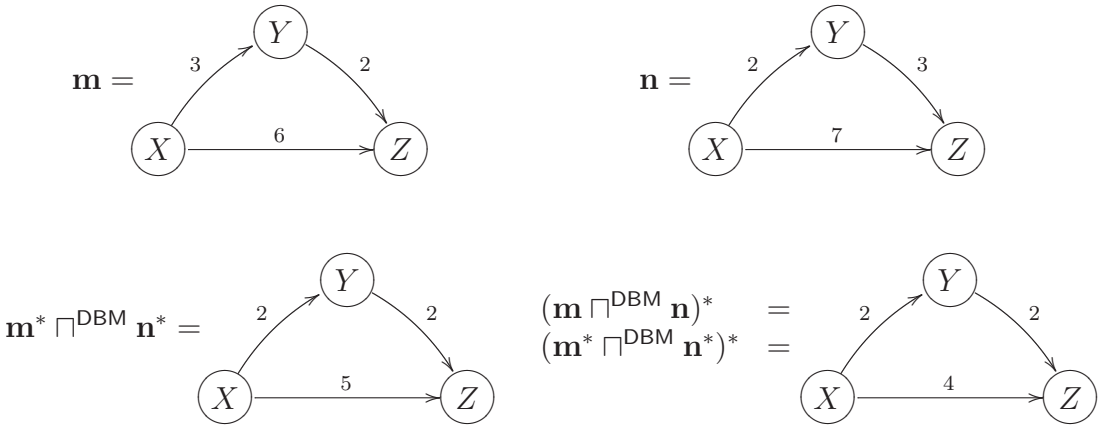


Figure 3.6: Intersection of zones based on $\sqcap^{\mathsf{DBM}}$. The arguments do not need to be closed, but the result is seldom closed.

- Many DBM sets can represent the same domain. Equality and inclusion testing become very costly.

To address these problems, several alternate data structures have been proposed. Two examples are *Clock Difference Diagrams*, introduced by Larsen, Weise, Wang, and Pearson [LWYP99] and *Difference Decision Diagrams*, introduced by Møller, Lichtenberg, Andersen, and Hulgaard in [MLAH99]. These tree-like data-structures resemble classical Binary Decision Diagrams, introduced by Bryant in [Bry86] to compactly represent boolean functions: each tree node is labelled by a potential constraint and its outgoing arcs represent different satisfiability choices for this constraint. Thanks to the *sharing* of isomorphic subtrees, much memory is saved. Despite the lack of canonical form for both data structures, inclusion, equality, and emptiness testing algorithms are proposed.

Adapting these data-structures to the needs of abstract interpretation is surely possible. There is already some work by Bagnara, Hill, and Zaffanella on the design of widening operators for disjunctive completions of abstract domains [BHZ03]. However, much work still needs to be done. A fundamental problem is that this disjunctive completion is costly due to its high precision. In order to be of practical use, the disjunctive completion of zones should be provided with operators able to abstract away information with a parametric cost versus precision trade-off. In the Astrée static analyser, whenever a more precise treatment of unions was required, we relied on *partitioning techniques* instead of disjunctive completion methods, as explained in Sect. 8.3.4.

3.4.4 Intersection Abstraction

The intersection of two zones (resp. potential sets) is always a zone. The $\sqcap^{\mathsf{DBM}}$ operator always computes a DBM representing the exact intersection of two zones, even when the DBM arguments are not closed, as shown in Fig. 3.6, so, we define $\cap^{Pot}$ and $\cap^{Zone}$ as $\sqcap^{\mathsf{DBM}}$:

Definition 3.4.2. Intersection abstractions.

$$\mathbf{m} \cap^{Pot} \mathbf{n} \stackrel{\text{def}}{=} \mathbf{m} \sqcap^{\mathsf{DBM}} \mathbf{n} \ .$$
$$\mathbf{m} \cap^{Zone} \mathbf{n} \stackrel{\text{def}}{=} \mathbf{m} \sqcap^{\mathsf{DBM}} \mathbf{n} \ .$$

●

Theorem 3.4.4. Properties of the abstract intersections.

1. $\gamma^{Pot}(\mathbf{m} \cap^{Pot} \mathbf{n}) = \gamma^{Pot}(\mathbf{m}) \cap \gamma^{Pot}(\mathbf{n})$. *(exact abstraction)*
2. $\gamma^{Zone}(\mathbf{m} \cap^{Zone} \mathbf{n}) = \gamma^{Zone}(\mathbf{m}) \cap \gamma^{Zone}(\mathbf{n})$. *(exact abstraction)*

●

Proof. This is a consequence of the fact that γ^{Zone} and γ^{Pot} are complete $\sqcap-$morphisms. ○

The intersection $\sqcap^{\mathsf{DBM}}$ has been used for a long time in the model-checking community [LLPY97, MB83]. It is important to remark that the result of an intersection is seldom closed, even when the arguments are, as demonstrated in Fig. 3.6.

3.5 Conversion Operators

Our presentation of a static analyser for the numerical properties of Simple programs in Sect. 2.4.2 is parametrised by a single numerical abstract domain. In practice, however, it can be useful to use several abstract domains and switch dynamically between them to adjust the cost versus precision trade-off. We propose here operators that allow converting between the zone, interval, and polyhedron domains.

3.5.1 Conversion Between Zones and Intervals

From Intervals to Zones. Given an interval environment $X^\sharp : \mathcal{V} \to \mathcal{B}^{Int}$, constructing a DBM $\mathbf{m}$ such that $\gamma^{Zone}(\mathbf{m}) = \gamma^{Int}(X^\sharp)$ is straightforward:

$$Zone(X^\sharp) \stackrel{\text{def}}{=} \begin{cases} \bot^{\mathsf{DBM}} & \text{if } \exists V_i,\ X^\sharp(V_i) = \bot^{Int}_{\mathcal{B}} \\ \mathbf{m} \text{ where } \begin{cases} \mathbf{m}_{0i} \stackrel{\text{def}}{=} \max\left(\gamma^{Int}_{\mathcal{B}}(X^\sharp(V_i))\right) \\ \mathbf{m}_{i0} \stackrel{\text{def}}{=} -\min\left(\gamma^{Int}_{\mathcal{B}}(X^\sharp(V_i))\right) & \text{otherwise} \\ \mathbf{m}_{ij} \stackrel{\text{def}}{=} +\infty \quad \text{if } i,j \neq 0 \end{cases} \end{cases}$$

Note that this is an exact abstraction.

From Zones to Intervals. We first define a *projection operator* $\pi_i(\mathbf{m})$ that is able to determine the maximum and minimum taken by the variable V_i in $\gamma^{Zone}(\mathbf{m})$. In order to do this, we need to compute the closure of $\mathbf{m}$:

Definition 3.5.1. Projection operator π_i.

$$\pi_i(\mathbf{m}) \stackrel{\text{def}}{=} \begin{cases} \bot_{\mathcal{B}}^{Int} & \text{if } \mathbf{m}^* = \bot^{\mathsf{DBM}} \\ [-\mathbf{m}^*_{i0},\ \mathbf{m}^*_{0i}] & \text{if } \mathbf{m}^* \neq \bot^{\mathsf{DBM}} \end{cases}$$

●

Theorem 3.5.1. Projection operator properties.

$$\gamma^{Int}(\pi_i(\mathbf{m})) = \{\ v \in \mathbb{I} \mid \exists (v_1, \ldots, v_n) \in \gamma^{Zone}(\mathbf{m}),\ v_i = v\ \}\ .$$

●

Proof. This is an easy consequence of the saturation property of Thm. 3.3.3. ○

Note that $[-\mathbf{m}_{i0},\ \mathbf{m}_{0i}]$ is a sound over-approximation of $\gamma^{Int}(\pi_i(\mathbf{m}))$, even when $\mathbf{m}$ is not closed, but it may not be exact in this case.

Given a zone represented by a DBM $\mathbf{m}$, we can compute its *best* abstraction as an interval domain element as follows:

$$Int(\mathbf{m}) \stackrel{\text{def}}{=} \lambda V_i.\pi_i(\mathbf{m})\ .$$

As the projection requires computing $\mathbf{m}$'s closure, this operator has a cubic worst-case time cost.

Galois Connection. Note that we have a Galois connection between DBMs representing zones and intervals: $\mathsf{DBM} \underset{Int}{\overset{Zone}{\leftrightarrows}} \mathcal{D}^{Int}$. In fact, $Int = \alpha^{Int} \circ \gamma^{Zone}$ and $Zone = \alpha^{Zone} \circ \gamma^{Int}$. Even though α^{Int} and α^{Zone} are only partial functions when $\mathbb{I} = \mathbb{Q}$, this Galois connection is total.

3.5.2 Conversion Between Zones and Polyhedra

From Zones to Polyhedra. Recall that a polyhedron can be internally represented as a conjunction of linear inequality constraints. As zone constraints are only special cases of linear inequality constraints, converting a DBM $\mathbf{m}$ into a polyhedron representing *exactly* $\gamma^{Zone}(\mathbf{m})$ is straightforward. We will denote by $Poly(\mathbf{m})$ the resulting polyhedron.

From Polyhedra to Zones. Converting a non-empty polyhedron P in $\mathbb{R}^n$ or $\mathbb{Q}^n$ into a zone is more subtle. Surprisingly, the *frame* representation of P is more handy here. Consider a finite set V of vertices and a finite set R of rays describing P. We denote respectively by v_i and r_i the $i-$th coordinate of a vertex $v \in V$ and a ray $r \in R$, starting from $i = 1$. We generate zone constraints using the following procedure:

- for every $i \geq 1$: if there is a ray $r \in R$ such that $r_i > 0$, we set $\mathbf{m}_{0i} = +\infty$, otherwise, we set $\mathbf{m}_{0i} = \max \{ v_i \mid v \in V \}$;
- for every $i \geq 1$: if there is a ray $r \in R$ such that $r_i < 0$, we set $\mathbf{m}_{i0} = +\infty$, otherwise, we set $\mathbf{m}_{i0} = -\min \{ v_i \mid v \in V \}$;
- for every $i, j \geq 1$, $i \neq j$: if there is a ray $r \in R$ such that $r_i > r_j$, we set $\mathbf{m}_{ij} = +\infty$, otherwise, we set $\mathbf{m}_{ij} = \max \{ v_j - v_i \mid v \in V \}$;
- we set $\mathbf{m}_{ii} = 0$ for all $i \geq 0$.

This procedure returns a DBM representing the smallest zone including the polyhedron in $\mathcal{O}(n^2 \times (|R| + |V|))$ time. The frame representation for P does not need to be minimised, however, it improves the conversion time if it is as there are fewer vertices and rays to consider. We will denote by $\mathit{Zone}(P)$ the resulting DBM. It is always closed.

The case $\mathbb{I} = \mathbb{Z}$ is a little more subtle. Recall that the polyhedron domain in $\mathbb{Z}$ uses the same representation — and algorithms — as the rational polyhedron domain, but with an altered semantics: only the points with integer coordinates inside rational polyhedra are considered; $\gamma_{\mathbb{Q}}^{Poly}(P)$ is replaced with $\gamma_{\mathbb{Q}}^{Poly}(P) \cap \mathbb{Z}^n$. Thus, the above algorithm would generate a DBM with non-integer elements. We argue that it is safe to round each $\mathbf{m}_{ij}$ to $\lfloor \mathbf{m}_{ij} \rfloor$; we may miss points in $\gamma_{\mathbb{Q}}^{Poly}(P)$ but not points in $\gamma_{\mathbb{Q}}^{Poly}(P) \cap \mathbb{Z}^n$. The rounded matrix might not be closed. For instance, if $n = 1$ and $\gamma_{\mathbb{Q}}^{Poly}(P) = \{(0.5)\}$, then the DBM will be

$$\mathbf{m} = \begin{array}{c|cc} & V_0 & V_1 \\ \hline V_0 & 0 & \lfloor -0.5 \rfloor \\ V_1 & \lfloor 0.5 \rfloor & 0 \end{array} = \begin{array}{c|cc} & V_0 & V_1 \\ \hline V_0 & 0 & -1 \\ V_1 & 0 & 0 \end{array}$$

and $\mathbf{m}^* = \bot^{\mathsf{DBM}} \neq \mathbf{m}$: the conversion has discovered that $\gamma_{\mathbb{Q}}^{Poly}(P) \cap \mathbb{Z}^n = \emptyset$. Finally, unlike what happened in the $\mathbb{R}^n$ and $\mathbb{Q}^n$ cases, our procedure does not always return the smallest zone that encompasses $\gamma_{\mathbb{Q}}^{Poly}(P) \cap \mathbb{Z}^n$. Indeed, the $\lfloor \cdot \rfloor$ operator enforces some "integerness" constraints on $\mathit{Zone}(P)$ that were not present in the original polyhedron representation, but it does not enforce *all* of them and we may still get points that are in $\gamma_{\mathbb{Q}}^{Poly}(P)$ but not in $\gamma_{\mathbb{Q}}^{Poly}(P) \cap \mathbb{Z}^n$.

Galois Connection. As for the conversion from and to the interval domain, there is a total Galois connection between polyhedra and DBMs representing zones: $\mathcal{D}^{Poly} \underset{Zone}{\overset{Poly}{\leftrightarrows}} \mathsf{DBM}$. As a matter of fact, $Poly = \alpha^{Poly} \circ \gamma^{Zone}$ and $Zone = \alpha^{Zone} \circ \gamma^{Poly}$, even though α^{Poly} is partial. We can deduce from this that $(\alpha^{Zone}, \gamma^{Zone})$ is a partial Galois connection with respect to transfer functions involving linear expressions.

3.6 Transfer Functions

We now present our transfer functions for the zone abstract domain. Whenever possible, we will propose several versions with different cost versus precision trade-offs. Also, some transfer functions have a less costly version whenever the argument is closed, or whenever we do not require the result to be closed. As all our transfer functions are strict in DBM, we simplify our presentation by considering only the case where the argument is a matrix, and not $\bot^{\mathsf{DBM}}$.

Related Work. In [SKS00], Shaham, Kolodner, and Sagiv only propose abstract assignment and test transfer functions for simple expressions that lead to exact abstractions. We recall these, but also propose more generic ones. We also explicit the relationship between the closure operator and the precision of our abstract transfer functions, which seems to be quite a novelty. Finally, our definitions are somewhat related to the "linearisation" technique that will be introduced later, in Chap. 6.

3.6.1 Forget Operators

Recall that the forget operator $\{\!|\, V_f \leftarrow ?\,|\!\}$ of Sect. 2.4.3 corresponds to a non-deterministic assignment into a variable and is quite useful as a fall-back assignment transfer function. A first idea to implement an abstract forget operator is to simply remove all constraints involving V_f:

Definition 3.6.1. Forget operator on zones $\{\!|\, V_f \leftarrow ?\,|\!\}^{Zone}$.

$$(\{\!|\, V_f \leftarrow ?\,|\!\}^{Zone}(\mathbf{m}))_{ij} \overset{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} & \text{if } i \neq f \text{ and } j \neq f \\ 0 & \text{if } i = j = f \\ +\infty & \text{otherwise} \end{cases}$$

●

This always gives us a valid over-approximation of the forget operator and, whenever the argument is a closed matrix, it is indeed exact:

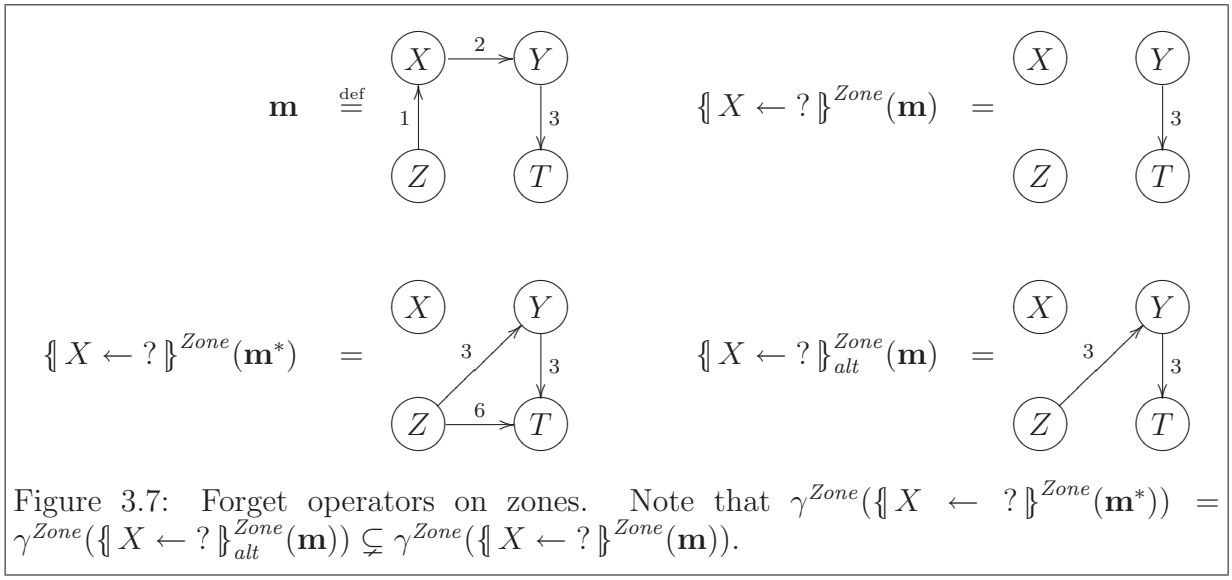


Figure 3.7: Forget operators on zones. Note that $\gamma^{Zone}(\{\!\!\{ X \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m}^*)) = \gamma^{Zone}(\{\!\!\{ X \leftarrow ? \}\!\!\}^{Zone}_{alt}(\mathbf{m})) \subsetneq \gamma^{Zone}(\{\!\!\{ X \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m}))$.

Theorem 3.6.1. Soundness and exactness of $\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}$.

1. $\gamma^{Zone}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m})) \supseteq \{\ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Zone}(\mathbf{m})\ \}$.
2. $\gamma^{Zone}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m}^*)) = \{\ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Zone}(\mathbf{m})\ \}$.
3. $\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m})$ *is closed whenever* $\mathbf{m}$ *is.*

●

Proof.

1. We will prove that $\gamma^{Pot}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m})) \supseteq \{\ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Pot}(\mathbf{m})\ \}$ which implies the desired property because $\gamma^{Zone}(\mathbf{m}) \subseteq \gamma^{Zone}(\mathbf{n}) \iff \gamma^{Pot}(\mathbf{m}) \subseteq \gamma^{Pot}(\mathbf{n})$.

   Let us take $t \in \mathbb{I}$ and $\vec{v} = (v_1, \ldots, v_n) \in \gamma^{Pot}(\mathbf{m})$. We want to prove that $\vec{v}' \stackrel{\text{def}}{=} (v_1, \ldots, v_{f-1}, t, v_{f+1}, \ldots, v_n) \in \gamma^{Pot}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m}))$; that is to say, if we denote by v'_k the $k-$th coordinates of $\vec{v}'$: $\forall i, j,\ v'_j - v'_i \leq (\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m}))_{ij}$.

   - If $i, j \neq f$, we have $v'_j - v'_i = v_j - v_i \leq \mathbf{m}_{ij} = (\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m}))_{ij}$.
   - If $i = f$ or $j = f$ but not both, then $v'_j - v'_i \leq +\infty = (\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m}))_{ij}$.
   - Finally, if $i = j = f$, $v'_j - v'_i = 0 = (\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m}))_{ij}$.

2. First, the property is obvious if $\mathbf{m}^* = \bot^{\mathsf{DBM}}$, that is, $\gamma^{Zone}(\mathbf{m}) = \emptyset$, so, we will only consider the case where $\mathbf{m}^* \neq \bot^{\mathsf{DBM}}$. By the first point, $\gamma^{Zone}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m}^*)) \supseteq$

$\{ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Zone}(\mathbf{m}^*) \} = \{ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Zone}(\mathbf{m}) \}$, so, we only need to prove the converse inclusion. As for the first point, we will prove instead that $\gamma^{Pot}(\{\!|\, V_f \leftarrow ?\, |\!\}^{Zone}(\mathbf{m}^*)) \subseteq \{ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Pot}(\mathbf{m}) \}$, which is indeed sufficient.

Let us take $\vec{v} = (v_1, \ldots, v_n) \in \gamma^{Pot}(\{\!|\, V_f \leftarrow ?\, |\!\}^{Zone}(\mathbf{m}^*))$. We want to prove that there exists a t such that $\vec{v}\,' = (v_1, \ldots, v_{f-1}, t, v_{f+1}, \ldots, v_n) \in \gamma^{Pot}(\mathbf{m})$.

Let us first prove that $\max_{j \neq f}(v_j - \mathbf{m}^*_{fj}) \leq \min_{i \neq f}(v_i + \mathbf{m}^*_{if})$. Suppose that this is not the case, then there exists two indexes $i \neq f$ and $j \neq f$ such that $v_j - \mathbf{m}^*_{fj} > v_i + \mathbf{m}^*_{if}$. By the local characterisation of the closure, Thm. 3.3.6, we have $\mathbf{m}^*_{ij} \leq \mathbf{m}^*_{if} + \mathbf{m}^*_{fj}$ and we get $v_j - v_i > \mathbf{m}^*_{if} + \mathbf{m}^*_{fj} \geq \mathbf{m}^*_{ij}$. This is absurd because $\vec{v} \in \gamma^{Pot}(\{\!|\, V_f \leftarrow ?\, |\!\}^{Zone}(\mathbf{m}^*))$ implies $v_j - v_i \leq (\{\!|\, V_f \leftarrow ?\, |\!\}^{Zone}(\mathbf{m}^*))_{ij} = \mathbf{m}^*_{ij}$ for every $i, j \neq f$.

So there exists at least a $t \in \mathbb{I}$ such that:

$$\max_{j \neq f}(v_j - \mathbf{m}^*_{fj}) \leq t \leq \min_{i \neq f}(v_i + \mathbf{m}^*_{if}) \ .$$

We now prove that any such t is a good choice, that is to say, $\vec{v}\,' \in \gamma^{Pot}(\mathbf{m})$. We will denote by v'_k the $k-$th coordinates of $\vec{v}\,'$.

- For all $i \neq f$ and $j \neq f$, $v'_j - v'_i = v_j - v_i \leq (\{\!|\, V_f \leftarrow ?\, |\!\}^{Zone}(\mathbf{m}^*))_{ij} = \mathbf{m}^*_{ij}$.
- If $i = f$ but $j \neq f$, then, we have $t \geq \max_{j \neq f}(v_j - \mathbf{m}^*_{fj})$, and so, $t \geq v_j - \mathbf{m}^*_{fj}$, which implies $v'_j - v'_i = v_j - t \leq \mathbf{m}^*_{ij}$.
- Similarly, when $j = f$ and $i \neq f$, $v'_j - v'_i = t - v_i \leq \mathbf{m}^*_{ij}$ because $t \leq \min_{i \neq f}(v_i + \mathbf{m}^*_{if})$.
- As $\mathbf{m}^*_{ff} = 0$, $\vec{v}\,'$ satisfies the constraint $t - t \leq \mathbf{m}^*_{ff}$.

3. Suppose that $\mathbf{m}$ is closed and let us denote $\{\!|\, V_f \leftarrow ?\, |\!\}^{Zone}(\mathbf{m})$ by $\mathbf{n}$. We use the local definition of closure: by Thm. 3.3.6 it is sufficient to prove that $\forall i, j, k,\ \mathbf{n}_{ij} \leq \mathbf{n}_{ik} + \mathbf{n}_{kj}$ and $\forall i,\ \mathbf{n}_{ii} = 0$.

   - We have easily $\forall i,\ \mathbf{n}_{ii} = 0$. When $i = f$, this is enforced by the definition of $\{\!|\, V_f \leftarrow ?\, |\!\}^{Zone}$, while, when $i \neq f$, we have $\mathbf{n}_{ii} = \mathbf{m}_{ii} = 0$ because $\mathbf{m}$ is itself closed.
   - Set three variables i, j, and k. If all of i, j, and k are different from f, then $\mathbf{n}_{ij} = \mathbf{m}_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj} = \mathbf{n}_{ik} + \mathbf{n}_{kj}$. If $i = j = k = f$, then $\mathbf{n}_{ij} = \mathbf{n}_{ik} + \mathbf{n}_{kj} = 0$. In all other cases, at least one of $\mathbf{n}_{ik}$ and $\mathbf{n}_{kj}$ is $+\infty$, and so, $\mathbf{n}_{ij} \leq \mathbf{n}_{ik} + \mathbf{n}_{kj} = +\infty$.

○

Whenever the argument is not closed, the result may not be exact. The intuitive reason is that, by forgetting constraints involving V_f, we also forget some implicit constraints on

unmodified variables as we break all former paths passing through V_f. If the argument is closed, however, all implicit constraints have been made explicit and this problem does not occur. This is exemplified in Fig. 3.7. Unfortunately, if we are to close the argument of the forget operator, its attractive linear cost becomes a less attractive cubic cost in the worst case.

It turns out that, unlike what happens for the union, not all the implicit constraints discovered by the closure are needed to get an exact operator. Only the information near V_f needs to be taken into account. Using this remark, we introduce the following quadratic time alternate forget operator:

Definition 3.6.2. Alternate forget operator on zones $\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}_{alt}$.

$$(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}_{alt}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \min(\mathbf{m}_{ij}, \mathbf{m}_{if} + \mathbf{m}_{fj}) & \textit{if } i \neq f \textit{ and } j \neq f \\ \mathbf{m}_{ff} & \textit{if } i = j = f \\ +\infty & \textit{otherwise} \end{cases}$$

●

This operator is exact, even when the argument is not a closed DBM. In that case, of course, the result will not be closed.

Theorem 3.6.2. Exactness of $\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}_{alt}$.

$$\gamma^{Zone}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}_{alt}(\mathbf{m})) = \{\ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Zone}(\mathbf{m})\ \}\ .$$

●

Proof.

We prove that $(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}_{alt}(\mathbf{m}))^* = \{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m}^*)$ which implies the desired property because of Thm. 3.6.1.2.

Firstly, we consider the case $\mathbf{m}^* \neq \bot^{\mathsf{DBM}}$. Let us consider a pair (i, j).

- If $i = j$, then $(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}_{alt}(\mathbf{m}))^*_{ij} = (\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m}^*))_{ij} = 0$ because both matrices are closed.
- If $i = f$ or $j = f$, then $\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m}^*)_{ij} = +\infty$. On the other hand, there is no path from i to j in $\mathcal{G}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}_{alt}(\mathbf{m}))$, so, $(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}_{alt}(\mathbf{m}))^*_{ij}$ is also $+\infty$.
- Suppose now that $i \neq f$, $j \neq f$, and $i \neq j$. We have $(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m}^*))_{ij} = \mathbf{m}^*_{ij}$.

  Let π be a path from i to j in $\mathcal{G}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}_{alt}(\mathbf{m}))$. Let us construct a path π' from i to j in $\mathcal{G}(\mathbf{m})$ as follows: we replace every arc (i_l, i_{l+1}) in π such that $i_l, i_{l+1} \neq f$ and $(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}_{alt}(\mathbf{m}))_{i_l i_{l+1}} = \mathbf{m}_{i_l f} + \mathbf{m}_{f i_{l+1}}$ with the sub-path $\langle i_l, f, i_{l+1} \rangle$. Then, the total weight of π' is exactly the same as π. This proves that $\mathbf{m}^*_{ij} \leq (\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}_{alt}(\mathbf{m}))^*_{ij}$.

Conversely, let π be a path in $\mathcal{G}(\mathbf{m})$ from i to j. We derive a path π' in $\mathcal{G}(\{\!|\, V_f \leftarrow ? \,|\!\}^{Zone}_{alt}(\mathbf{m}))$ as follows: while there is a sub-path of the form $\langle x, f, \ldots, f, y\rangle$ with $x, y \neq f$ in π, we replace it with $\langle x, y\rangle$. As $\mathbf{m}^* \neq \bot^{\mathsf{DBM}}$, $\mathbf{m}_{ff} \geq 0$, and so, π' has a weight that is smaller than or equal to that of π, which implies $(\{\!|\, V_f \leftarrow ? \,|\!\}^{Zone}_{alt}(\mathbf{m}))^*_{ij} \leq \mathbf{m}^*_{ij}$.

We now consider the case $\mathbf{m}^* = \bot^{\mathsf{DBM}}$. There exists a simple cycle π in $\mathbf{m}$ with strictly negative total weight. We now construct a cycle π' in $\{\!|\, V_f \leftarrow ? \,|\!\}^{Zone}_{alt}(\mathbf{m})$ with strictly negative weight; this will prove that $(\{\!|\, V_f \leftarrow ? \,|\!\}^{Zone}_{alt}(\mathbf{m}))^* = \bot^{\mathsf{DBM}} = \{\!|\, V_f \leftarrow ? \,|\!\}^{Zone}(\mathbf{m}^*)$. The cycle π' is derived from π as follows: while there is a sub-path of the form $\langle x, f, y\rangle$ in π, we replace it with $\langle x, y\rangle$. The resulting cycle π' is either $\langle f, f\rangle$, or a cycle not passing through f. In either case, its total weight in $\{\!|\, V_f \leftarrow ? \,|\!\}^{Zone}_{alt}(\mathbf{m})$ is smaller than or equal to that of π in $\mathbf{m}$.

○

The use of this alternate forget operator is also exemplified in Fig. 3.7.

3.6.2 Assignment Transfer Functions

Simple and Exact Cases. We first focus on the simpler case of assignments of the form $V_i \leftarrow [a, b]$ or $V_i \leftarrow V_j + [a, b]$. These kinds of assignment can be *exactly* modeled in the zone domain.

Definition 3.6.3. Abstraction of simple assignments.

Invertible assignments:

- $(\{\!|\, V_{j_0} \leftarrow V_{j_0} + [a, b] \,|\!\}^{Zone}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} - a & \text{if } i = j_0,\ j \neq j_0 \\ \mathbf{m}_{ij} + b & \text{if } i \neq j_0,\ j = j_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$

Non-invertible assignments ($j_0 \neq i_0$):

- $(\{\!|\, V_{j_0} \leftarrow [a, b] \,|\!\}^{Zone}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} -a & \text{if } i = j_0,\ j = 0 \\ b & \text{if } i = 0,\ j = j_0 \\ (\{\!|\, V_{j_0} \leftarrow ? \,|\!\}^{Zone}(\mathbf{m}^*))_{ij} & \text{otherwise} \end{cases}$
- $(\{\!|\, V_{j_0} \leftarrow V_{i_0} + [a, b] \,|\!\}^{Zone}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} -a & \text{if } i = j_0,\ j = i_0 \\ b & \text{if } i = i_0,\ j = j_0 \\ (\{\!|\, V_{j_0} \leftarrow ? \,|\!\}^{Zone}(\mathbf{m}^*))_{ij} & \text{otherwise} \end{cases}$

●

Note that non-invertible assignments make use of the forget operator $\{\!|\, V_{j_0} \leftarrow ? \,|\!\}^{Zone}$. Thus, in order to get an exact abstraction, we should either close the matrix argument

or use the quadratic $\{\!|\, V_{j_0} \leftarrow ?\,|\!\}^{Zone}_{alt}$ alternate version of the forget operator. Also, the result of a non-invertible assignment is generally not closed. However, as $\{\!|\, V_{j_0} \leftarrow ?\,|\!\}^{Zone}$ preserves the closure, if we use a closed argument, the closure of the result can be obtained by performing the incremental closure of Def. 3.3.4 on the line and column j_0. This is not possible if we choose to use $\{\!|\, V_{j_0} \leftarrow ?\,|\!\}^{Zone}_{alt}$ on a non-closed matrix argument. These remarks lead us to a quadratic algorithm that always returns the exact abstraction but also preserves the closure:

- if $\mathbf{m}$ is closed, return $Inc^*_{j_0}(\{\!|\, V_{j_0} \leftarrow expr\,|\!\}^{Zone}_{exact}(\mathbf{m}))$;
- if $\mathbf{m}$ is not closed, return $\{\!|\, V_{j_0} \leftarrow expr\,|\!\}^{Zone}_{alt}(\mathbf{m})$ computed using the alternate forget operator instead of the regular one.

The invertible assignment transfer function is much simpler: the constant time version presented in Def. 3.6.3 is always exact. Also, the result is closed whenever the argument is:

Theorem 3.6.3. $\{\!|\, V_{j_0} \leftarrow V_{j_0} + [a,b]\,|\!\}^{Zone}_{exact}$ **preserves the closure.**

If $\mathbf{m}$ *is closed, then so is* $\{\!|\, V_{j_0} \leftarrow V_{j_0} + [a,b]\,|\!\}^{Zone}_{exact}(\mathbf{m})$.

●

Proof.

Suppose that $\mathbf{m}$ is closed and denote by $\mathbf{n}$ the result $\{\!|\, V_{j_0} \leftarrow V_{j_0} + [a,b]\,|\!\}^{Zone}_{exact}(\mathbf{m})$. We will use the local characterisation of the closure, Thm. 3.3.6. $\forall i$, $\mathbf{n}_{ii} = 0$ because $\forall i$, $\mathbf{n}_{ii} = \mathbf{m}_{ii}$ and $\mathbf{m}$ is closed. Let us consider i, j, k, we now prove that $\mathbf{n}_{ij} \leq \mathbf{n}_{ik} + \mathbf{n}_{kj}$:

- If $i, j, k \neq j_0$, then $\mathbf{n}_{ij} = \mathbf{m}_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj} = \mathbf{n}_{ik} + \mathbf{n}_{kj}$.
- Suppose that $i = j_0$ but $j, k \neq j_0$, then $\mathbf{n}_{ij} = \mathbf{m}_{ij} - a \leq \mathbf{m}_{ik} + \mathbf{m}_{kj} - a = \mathbf{n}_{ik} + \mathbf{n}_{kj}$.
- Likewise, if $j = j_0$ but $i, k \neq j_0$, then $\mathbf{n}_{ij} = \mathbf{m}_{ij} + b \leq \mathbf{m}_{ik} + \mathbf{m}_{kj} + b = \mathbf{n}_{ik} + \mathbf{n}_{kj}$.
- Suppose that $k = j_0$ but $i, j \neq j_0$, then $\mathbf{n}_{ij} = \mathbf{m}_{ij} \leq \mathbf{m}_{ij} - a + b \leq \mathbf{m}_{ik} + \mathbf{m}_{kj} - a + b = \mathbf{n}_{ik} + \mathbf{n}_{kj}$ because $a \leq b$. The situation is similar if $i = j = j_0$ and $k \neq j_0$.
- Finally, the cases where $k = j_0$ and either $i = j_0$ or $j = j_0$ or both are obvious because $\mathbf{n}_{j_0 j_0} = \mathbf{m}_{j_0 j_0} = 0$.

○

In order to deal with assignments that cannot be exactly modeled in the zone domain, we propose several definitions, in increasing order of precision and cost. Also, more precise versions only work for limited forms of assigned expressions.

Interval-Based Assignment. A coarse method is to perform the assignment $V_i \leftarrow expr$ as in the interval domain: we first extract an interval abstract environment; then, we evaluate *expr* using the interval abstract arithmetic operators, as described in Sect. 2.4.6; finally, we force the resulting interval back into the zone domain. Using the conversion operator defined in Sect. 3.5.1, this can be formalised as:

$$\{\!\!\{ V_i \leftarrow expr \}\!\!\}^{Zone}_{nonrel}(\mathbf{m}) \stackrel{\text{def}}{=} \{\!\!\{ V_i \leftarrow (\llbracket expr \rrbracket^{Int}(Int(\mathbf{m}))) \}\!\!\}^{Zone}_{exact}(\mathbf{m}) \ .$$

In order to extract precise interval information, $\mathbf{m}$ should be in closed form. Depending on whether the argument is closed and whether we wish the result to be closed, the total cost varies from linear to cubic, plus the cost of the interval computation which is linear in the size of the expression *expr*.

The low precision of this transfer function comes from two facts. Firstly, we do not infer any relational information of the form $V_i - V_j$. Secondly, we do not use the existing relational information when computing the bounds of *expr*.

Deriving Relational Constraints. Our idea here is to solve the first cause of precision loss in the interval-based assignment, while not handling the second one. We still use the interval domain to compute bounds of expressions using only the information available in $Int(\mathbf{m})$. However, we compute the bounds of $expr - V_j$ for all $i \neq j$ to infer constraints of the form $V_i - V_j$. For instance, in the assignment $X \leftarrow Y + Z$, we would infer relations such as $\min Z \leq X - Y \leq \max Z$ and $\min Y \leq X - Z \leq \max Y$. In order to get that much precision, it is important to simplify formally each $expr - V_j$ before evaluating it in the interval domain: the precision improvement with respect to the interval-based assignment lies in the fact that the bounds for $expr - V_j$ may be tighter than $\llbracket expr \rrbracket^{Int}(Int(\mathbf{m})) - \pi_j(\mathbf{m})$ that one would get by merely closing the result of the interval-based assignment. In our running example, $(Y + Z) - Y$ has been replaced with Z which gives a big precision improvement when $\min Y \neq \max Y$: we get, for instance, $X - Y \leq \max Z$ instead of $X - Y \leq \max Y - \min Y + \max Z$.

We now propose a full formal definition of this idea, including the simplification step, but only for the simpler yet useful case of assignments of interval linear forms: $V_{j_0} \leftarrow [a_0, b_0] + \sum_k ([a_k, b_k] \times V_k)$. This is not a strong limitation as we will see, in Sect. 6.2.3, how to abstract an *arbitrary expression* into an interval linear form.

Definition 3.6.4. Abstraction of interval linear form assignments.

Let $e \stackrel{\text{def}}{=} [a_0, b_0] + \sum_k ([a_k, b_k] \times V_k)$. *We define:*

$$(\{\!\!\{ V_{j_0} \leftarrow e \}\!\!\}^{Zone}_{rel}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \max(\llbracket e \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } i = 0 \text{ and } j = j_0 \\ -\min(\llbracket e \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } i = j_0 \text{ and } j = 0 \\ \max(\llbracket e \boxminus V_i \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } i \neq 0, j_0 \text{ and } j = j_0 \\ -\min(\llbracket e \boxminus V_j \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } i = j_0 \text{ and } j \neq 0, j_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$$

where the interval linear form $e \boxminus V_i$ is defined as:[1]

$$e \boxminus V_i \stackrel{\text{def}}{=} [a_0, b_0] + \left(\sum_{k \neq i} ([a_k, b_k] \times V_k)\right) + ([a_i - 1, b_i - 1] \times V_i) \ .$$

●

This assignment definition is more precise than the interval-based one at the cost of more expression evaluations in the interval domain. It is not the best abstraction as we still do not use the relational information in the zone when computing bounds for each $expr - V_j$. For the sake of efficiency, a practical implementation would avoid evaluating, in the interval domain, $2n$ interval linear forms that are almost similar. For each i, it is possible to compute $[\![expr \boxminus V_i]\!]^{Int}(Int(\mathbf{m}))$ using only $[\![expr]\!]^{Int}(Int(\mathbf{m}))$ and $\pi_i(\mathbf{m})$, however, many different cases have to be considered because of the possible $+\infty$ coefficients, so, we do not present this optimisation here. It suffices to say that a careful implementation of this assignment operator leads to a cost similar to the interval-based assignment presented in the preceding paragraph: cubic if the argument is not closed, quadratic if the argument is closed and we wish the result to be closed, using the incremental closure $Inc^*_{j_0}$, and linear if the argument is closed but the result needs not be closed.

Polyhedron-Based Assignment. One can consider using temporarily the polyhedron abstract domain to perform the assignment transfer function, and convert the result back into a zone. This gives, using the conversion operators defined in Sect. 3.5.2:

$$\{\!| V_i \leftarrow expr |\!\}^{Zone}_{poly}(\mathbf{m}) \stackrel{\text{def}}{=} (Zone \circ \{\!| V_i \leftarrow expr |\!\}^{Poly} \circ Poly)(\mathbf{m}) \ .$$

This will work only when *expr* is a linear or a quasi-linear form — that is, of the form $[a_0, b_0] + \sum_k (a_k \times V_k)$ — as no polyhedron assignment is defined for other expression forms. Whenever $\mathbb{I} \neq \mathbb{Z}$, the conversion to a polyhedron and the polyhedron assignment are exact while the conversion to a zone is a best abstraction, so, we get the *best* abstract assignment in the zone domain. We will see, in Sect. 6.2.5, a generic way to abstract an arbitrary expression into a quasi-linear form, which greatly enlarges the scope of this definition.

The high precision reached by $\{\!| V_i \leftarrow expr |\!\}^{Zone}_{poly}$ calls for a great cost: because of the way our conversion operators work, the polyhedron assignment transfer function is fed with a constraint representation and we require its output in frame representation, which means that at least one representation conversion will occur, incurring an exponential cost at worse — consider, for instance, translating the box $\forall i \leq n, V_i \in [0, 1]$ by the assignment $V_1 \leftarrow V_1 + 1$, which requires computing a frame representation containing 2^n vertices. It is not known to the author whether there exists an algorithm to achieve best linear assignments for a smaller cost, without using the polyhedron domain.

[1] The $\boxminus$ operator used here is a special case of the $\boxplus$, $\boxminus$, $\boxtimes$, and $\boxslash$ operators we will introduce in Sect. 6.2.2 to manipulate interval linear forms.

Argument

$$\begin{cases} 0 \leq Y \leq 10 \\ 0 \leq Z \leq 10 \\ 0 \leq Y - Z \leq 10 \end{cases}$$

$$\Downarrow\ X \leftarrow Y - Z$$

Interval-based	**Def. 3.6.4**	**Polyhedron-based** (best)
$\begin{cases} \mathbf{-10} \leq X \leq 10 \\ \mathbf{-20} \leq X - Y \leq \mathbf{10} \\ \mathbf{-20} \leq X - Z \leq 10 \end{cases}$	$\begin{cases} \mathbf{-10} \leq X \leq 10 \\ -10 \leq X - Y \leq 0 \\ -10 \leq X - Z \leq 10 \end{cases}$	$\begin{cases} 0 \leq X \leq 10 \\ -10 \leq X - Y \leq 0 \\ -10 \leq X - Z \leq 10 \end{cases}$

Figure 3.8: Constraints on the variable X derived after the assignment $X \leftarrow Y - Z$ using three different assignment transfer functions. Non-optimal bounds are shown in boldface.

This idea behind $\{\!\!\{ V_i \leftarrow expr \}\!\!\}_{poly}^{Zone}$ can be carried further by considering any abstract domain that has an exact abstraction for assignments of a certain kind, a practical algorithm to compute the smallest zone encompassing an element in the domain, and the ability to represent zones exactly.

In our implementation, we chose to use Def. 3.6.3 when applicable, Def. 3.6.4 if we assign an interval linear form, and the interval-based assignment as a last resort. As the cost of the polyhedron-based assignment nullifies the gain in time obtained by choosing the zone domain instead of the polyhedron domain, we do not use it in actual static analyses; however, it is useful to perform regression tests and experimental studies of precision losses incurred when using non-optimal abstractions. Fig. 3.8 presents a comparison of our three definitions on a complex assignment example. We have closed the three results and presented only the constraints on the variable X to allow an easier comparison.

3.6.3 Test Transfer Functions

Preprocessing. In an attempt to simplify our definitions, we first show how the analysis of any test can be reduced to the case ($expr \leq 0$?). Consider a generic test: ($expr_1 \bowtie expr_2$?). A first step is to group the expressions on the left side as follows: ($expr_1 - expr_2 \bowtie 0$?). Whenever $\bowtie$ is not $\leq$ we perform one of the following:

- If $\bowtie$ is $=$, our test will be abstracted as:

$$\{\!\!\{\ expr_1 - expr_2 \leq 0\ ?\ \}\!\!\}^{Zone}(\mathbf{m}) \ \cap^{Zone}\ \{\!\!\{\ expr_2 - expr_1 \leq 0\ ?\ \}\!\!\}^{Zone}(\mathbf{m})\ .$$

- If $\bowtie$ is $<$ and $\mathbb{I} = \mathbb{Z}$, then we can use the test:

$$\{\!\!\{\ expr_1 - expr_2 + 1 \leq 0\ ?\ \}\!\!\}^{Zone}(\mathbf{m})\ .$$

- If $\bowtie$ is $<$ and $\mathbb{I} \neq \mathbb{Z}$, as we have no way represent exactly strict inequalities,[2] we relax it as a regular inequality:

$$\{\!\!\{\ expr_1 - expr_2 \leq 0\ ?\ \}\!\!\}^{Zone}(\mathbf{m})\ .$$

- If $\bowtie$ is $\neq$ and $\mathbb{I} = \mathbb{Z}$, we combine two inequalities:

$$\{\!\!\{\ expr_1 - expr_2 + 1 \leq 0\ ?\ \}\!\!\}^{Zone}(\mathbf{m}) \ \cup^{Zone}\ \{\!\!\{\ expr_2 - expr_1 + 1 \leq 0\ ?\ \}\!\!\}^{Zone}(\mathbf{m})\ .$$

- If $\bowtie$ is $\neq$ and $\mathbb{I} \neq \mathbb{Z}$, there is no better abstraction than the identity.

In order to abstract tests $\{\!\!\{\ expr \leq 0\ ?\ \}\!\!\}$, we use similar ideas as for assignments.

Simple and Exact Cases. If the test has the shape of a zone constraint, it can be modeled exactly by simply *adding* the constraint to the DBM:

Definition 3.6.5. Abstraction of simple tests.

1. $(\{\!\!\{\ V_{j_0} + [a,b] \leq 0\ ?\ \}\!\!\}^{Zone}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \min(\mathbf{m}_{ij},\ -a) & \text{if } i = 0,\ j = j_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$
2. $(\{\!\!\{\ V_{j_0} - V_{i_0} + [a,b] \leq 0\ ?\ \}\!\!\}^{Zone}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \min(\mathbf{m}_{ij},\ -a) & \text{if } i = i_0,\ j = j_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$

●

Note that the argument matrix does not need to be closed. However, if this is the case, the result, which is generally not closed, can be closed using the quadratic-cost incremental closure $Inc^*_{j_0}$.

[2]Note that, in Sect. 5.4.3, we will extend the zone abstract domain to include strict inequalities.

Interval-Based Abstraction. When *expr* has an arbitrary form, it is always possible to fall back to the interval domain abstract test:

$$\{\!\!\{\ expr \leq 0\ ?\ \}\!\!\}^{Zone}_{nonrel}(\mathbf{m}) \stackrel{\text{def}}{=} (Zone \circ \{\!\!\{\ expr \leq 0\ ?\ \}\!\!\}^{Int} \circ Int)(\mathbf{m}) \cap^{Zone} \mathbf{m}\ .$$

Because tests only *filter out* environments, it is safe to keep all the constraints of the argument DBM in the result, hence the intersection with $\mathbf{m}$ in our formula. This is quite fortunate because $(Zone \circ \{\!\!\{\ expr \leq 0\ ?\ \}\!\!\}^{Int} \circ Int)(\mathbf{m})$ does not contain any relational constraint by itself. As a conclusion, we do not infer new relational constraints but at least we keep all those that were valid before the test.

Because of the conversion to intervals, it is necessary for the argument $\mathbf{m}$ to be in closed form to achieve maximal accuracy. Unlike what happened for assignments, one pass of incremental closure is not sufficient to get a closed result as more than one line and column may be modified. If the argument is not closed or we wish the result to be closed, this gives a total cubic cost — in the number of variables $|\mathcal{V}|$ — plus the cost of the interval transfer function, without much room for optimisation.

Deriving New Relational Constraints. In order to derive some new relational constraints, we can remark that $expr \leq 0$ implies $V_j - V_i \leq V_j - V_i - expr$. Whenever V_i or V_j appears in *expr*, there is a possibility that $V_j - V_i - expr$ might be simplified and, once evaluated in the interval domain, may give a more precise upper bound for $V_j - V_i$ than the interval-based test. In the case of interval linear forms, using the $\boxminus$ operator introduced in the preceding section, we can define:

Definition 3.6.6. Interval linear form testing.

Let $expr \stackrel{\text{def}}{=} [a_0, b_0] + \sum_k ([a_k, b_k] \times V_k)$*, we define:*

$$(\{\!\!\{\ expr \leq 0\ ?\ \}\!\!\}^{Zone}_{rel}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \min(\mathbf{m}_{ij}, \mathbf{n}_{ij})$$

where $\mathbf{n}$ *is defined by:*

$$\mathbf{n}_{ij} \stackrel{\text{def}}{=} \begin{cases} \max([\![\ V_j \boxminus expr\ ]\!]^{Int}(Int(\mathbf{m}))) & \text{if } i = 0 \text{ and } j \neq 0 \\ \max([\![\ \boxminus V_i \boxminus expr\ ]\!]^{Int}(Int(\mathbf{m}))) & \text{if } i \neq 0 \text{ and } j = 0 \\ \max([\![\ V_j \boxminus V_i \boxminus expr\ ]\!]^{Int}(Int(\mathbf{m}))) & \text{if } i \neq 0 \text{ and } j \neq 0 \\ 0 & \text{if } i = j = 0 \end{cases}$$

●

A clever implementation would evaluate $[\![\ expr\ ]\!]^{Int}(Int(\mathbf{m}))$ only once and derive all other constraints by applying correcting terms — care must be taken when dealing with $+\infty$ bounds. Thus, deriving the constraints has a quadratic cost, plus the cost of an interval test transfer function. As for the interval-based test, the argument must be closed for maximum accuracy in $Int(\mathbf{m})$, but the result is not guaranteed to be closed.

Argument

$$\begin{cases} 5 \leq X \leq 25 \\ 0 \leq Y \leq 10 \\ 0 \leq Z \leq 10 \\ 0 \leq X - Y \leq 20 \\ -5 \leq X - Z \leq 25 \\ -10 \leq Y - Z \leq 10 \end{cases}$$

$$\Downarrow \; X \leq Y - Z \; ?$$

$\begin{cases} 5 \leq X \leq 10 \\ 5 \leq Y \leq 10 \\ 0 \leq Z \leq \mathbf{5} \\ 0 \leq X - Y \leq \mathbf{5} \\ \mathbf{0} \leq X - Z \leq 10 \\ \mathbf{0} \leq Y - Z \leq 10 \end{cases}$	$\begin{cases} 5 \leq X \leq 10 \\ 5 \leq Y \leq 10 \\ 0 \leq Z \leq \mathbf{5} \\ X - Y = 0 \\ \mathbf{0} \leq X - Z \leq 10 \\ \mathbf{0} \leq Y - Z \leq 10 \end{cases}$	$\begin{cases} 5 \leq X \leq 10 \\ 5 \leq Y \leq 10 \\ Z = 0 \\ X - Y = 0 \\ 5 \leq X - Z \leq 10 \\ 5 \leq Y - Z \leq 10 \end{cases}$
Interval-based	**Def. 3.6.6**	**Polyhedron-based** (best)

Figure 3.9: Constraints derived after the test ($X \leq Y - Z$?) using our three test transfer functions. Non-optimal bounds are shown in boldface.

Polyhedron-Based Test. As for the assignment, whenever *expr* has a quasi-linear form, the *best* abstraction can be computed at great cost by switching momentarily to the polyhedron abstract domain:

$$\{\!\!\{ \, expr \leq 0 \; ? \, \}\!\!\}^{Zone}_{poly}(\mathbf{m}) \stackrel{\text{def}}{=} (Zone \circ \{\!\!\{ \, expr \leq 0 \; ? \, \}\!\!\}^{Poly} \circ Poly)(\mathbf{m}) \; .$$

This has an exponential worst-case cost so, as for the polyhedron-based assignment, we will refrain from using it in practice; it is presented it here merely for the sake of completion. It is useful to compare, on theoretical examples, the preceding two methods with the best possible abstraction. Such an example is provided by Fig. 3.9. Note that Def. 3.6.6 is not guaranteed to be always at least as precise as the interval-based solution — even though this is the case on the example of Fig. 3.9 — so, it can be worth to actually compute both and return their intersection.

3.6.4 Backwards Assignment Transfer Functions

Simple and Exact Cases. The backward assignments that can be modeled exactly are similar to the exact assignments of Def. 3.6.3:

Definition 3.6.7. Abstraction of simple backward assignments.

Invertible assignments:

- $(\!\{ V_{j_0} \to V_{j_0} + [a,b] \}\!)^{Zone}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} + b & \text{if } i = j_0, j \neq j_0 \\ \mathbf{m}_{ij} - a & \text{if } i \neq j_0, j = j_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$

Non-invertible assignments ($j_0 \neq i_0$)*:*

- *if* $\mathbf{m}^*_{0j_0} \geq a$ *and* $\mathbf{m}^*_{j_0 0} \geq -b$*, then*

  $(\!\{ V_{j_0} \to [a,b] \}\!)^{Zone}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \min(\mathbf{m}^*_{ij}, \mathbf{m}^*_{j_0 j} + b) & \text{if } i = 0,\ j \neq 0, j_0 \\ \min(\mathbf{m}^*_{ij}, \mathbf{m}^*_{i j_0} - a) & \text{if } j = 0,\ i \neq 0, j_0 \\ +\infty & \text{if } i = j_0 \text{ or } j = j_0 \\ \mathbf{m}^*_{ij} & \text{otherwise} \end{cases}$

  otherwise, $(\!\{ V_{j_0} \to [a,b] \}\!)^{Zone}_{exact}(\mathbf{m}) \stackrel{\text{def}}{=} \bot^{\mathsf{DBM}}$

- *if* $\mathbf{m}^*_{i_0 j_0} \geq a$ *and* $\mathbf{m}^*_{j_0 i_0} \geq -b$*, then*

  $(\!\{ V_{j_0} \to V_{i_0} + [a,b] \}\!)^{Zone}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \min(\mathbf{m}^*_{ij}, \mathbf{m}^*_{j_0 j} + b) & \text{if } i = i_0,\ j \neq i_0, j_0 \\ \min(\mathbf{m}^*_{ij}, \mathbf{m}^*_{i j_0} - a) & \text{if } j = i_0,\ i \neq i_0, j_0 \\ +\infty & \text{if } i = j_0 \text{ or } j = j_0 \\ \mathbf{m}^*_{ij} & \text{otherwise} \end{cases}$

  otherwise, $(\!\{ V_{j_0} \to V_{i_0} + [a,b] \}\!)^{Zone}_{exact}(\mathbf{m}) \stackrel{\text{def}}{=} \bot^{\mathsf{DBM}}$

●

The definition in the invertible case $V_{j_0} \to V_{j_0} + [a,b]$ is exactly equivalent to $V_{j_0} \leftarrow V_{j_0} + [-b,-a]$, so, it shares equivalent properties: it does not require a closed argument, but it preserves the closure.

The non-invertible case $V_{j_0} \to V_{i_0} + [a,b]$ corresponds to *substituting* V_{j_0} with the assigned expressions in all the constraints in $\mathbf{m}$. This generates new constraints that refine all the constraints related to V_{i_0} but remove all information about V_{j_0}. Also, we may discover a trivially unsatisfiable constraint and return $\bot^{\mathsf{DBM}}$ directly. This requires a closed argument to achieve maximal precision. Provided the argument is closed, all elements unrelated to V_{i_0} or V_{j_0} are left intact, so, the result can be closed in quadratic time using the incremental closure $Inc^*_{i_0,j_0}$. The non-invertible case $V_{j_0} \to [a,b]$ is similar except that we replace the index i_0 with 0.

Interval-Based Backward Assignment. As for tests, we can use the interval backward assignment to discover interval information, but we need a way to recover some relational information as well. The idea is to keep in $\mathbf{m}$ all the constraints that are not invalidated by the backward assignment. Thus, we combine the interval transfer function together with the generic fall-back backward assignment which simply forgets the value of the assigned variable:

$$\begin{aligned}&\{\!\!\{ V_i \to \mathit{expr} \}\!\!\}^{\mathit{Zone}}_{\mathit{nonrel}}(\mathbf{m}) \stackrel{\text{def}}{=}\\ &\quad (\mathit{Zone} \circ \{\!\!\{ V_i \to \mathit{expr} \}\!\!\}^{\mathit{Int}} \circ \mathit{Int})(\mathbf{m}) \cap^{\mathit{Zone}} \{\!\!\{ V_i \leftarrow ? \}\!\!\}^{\mathit{Zone}}(\mathbf{m}^*) \ .\end{aligned}$$

Polyhedron-Based Backward Assignment. As for the other two transfer functions, whenever *expr* is quasi-linear, the *best* abstraction can be computed with exponential worst-case cost using the polyhedron abstract domain:

$$\{\!\!\{ V_i \to \mathit{expr} \}\!\!\}^{\mathit{Zone}}_{\mathit{poly}}(\mathbf{m}) \stackrel{\text{def}}{=} (\mathit{Zone} \circ \{\!\!\{ V_i \to \mathit{expr} \}\!\!\}^{\mathit{Poly}} \circ \mathit{Poly})(\mathbf{m}) \ .$$

Deriving New Relational Constraints. In the assignment and test transfer functions for interval linear forms, we were able to refine the interval-based transfer function by inferring some new relational constraints, for a linear or quadratic extra cost. This idea can be adapted to backward assignment, but with a cubic cost instead. Given the backward assignment $V_i \to \mathit{expr}$ on $\mathbf{m}$, we can derive for each variable $V_j \neq V_i$ two interval linear constraints by substitution: $\mathit{expr} - V_j \leq \mathbf{m}_{ji}$ and $V_j - \mathit{expr} \leq \mathbf{m}_{ij}$. We can then apply Def. 3.6.6 to derive an upper and lower bound for each $V_k - V_l$ by evaluating, in the interval domain, simplified versions of $V_k - V_l - \mathit{expr} + V_j + \mathbf{m}_{ji}$ and $V_k - V_l + \mathit{expr} - V_j + \mathbf{m}_{ij}$. As there is no obvious way to choose specific j, k, l among all possible triplets, we need to generate a cubic number of constraints.

3.7 Extrapolation Operators

As the zone domain has both strictly increasing and strictly decreasing infinite chains, widening and narrowing operators are required for the chaotic abstract iterations to converge within finite time.

3.7.1 Widenings

Simple Widening. The main idea of the standard widenings on intervals $\triangledown^{\mathit{Int}}$ and polyhedra $\triangledown^{\mathit{Poly}}$ is to remove the unstable constraints, and keep only the stable ones. A similar widening on zones can be defined point-wisely as follows:

Definition 3.7.1. Standard widening ∇_{std}^{Zone} on zones.

$$(\mathbf{m} \nabla_{std}^{Zone} \mathbf{n})_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} & \textit{if } \mathbf{m}_{ij} \geq \mathbf{n}_{ij} \\ +\infty & \textit{otherwise} \end{cases}$$

●

Proof. This is a special case of Def. 3.7.2, presented shortly. ○

Example 3.7.1. Using the standard zone widening.

Consider the following Simple program that iterates from 0 to N:

```
    X ← 0;
    N ← [0, +∞];
    while ❶ X < N {
②     X ← X + 1
③   }
④
```

We now present the zones computed by the chaotic iterations, with widening at point ❶. For each iteration i, the abstract value corresponding to only one label l is changed; we denote it by X_l^i and ignore unmodified components. For the sake of concision, we omit unimportant program labels, and do not show the iterations involving these omitted labels as they can be reconstructed easily:

iteration i	label l	zone X_l^i
0	❶	$X = 0 \wedge 0 \leq N \wedge X - N \leq 0$
1	②	$X = 0 \wedge 1 \leq N \wedge X - N \leq -1$
2	③	$X = 1 \wedge 1 \leq N \wedge X - N \leq 0$
3	❶ ∇	$0 \leq X \wedge 0 \leq N \wedge X - N \leq 0$
4	②	$0 \leq X \wedge 1 \leq N \wedge X - N \leq -1$
5	③	$1 \leq X \wedge 1 \leq N \wedge X - N \leq 0$
6	❶ ∇	$0 \leq X \wedge 0 \leq N \wedge X - N \leq 0$
7	④	$0 \leq X \wedge 0 \leq N \wedge X - N = 0$

Of particular interest is the sequence $X_{❶}^0 \sqsubset^{\mathsf{DBM}} X_{❶}^3 = X_{❶}^6$ that is able to infer the loop invariant $X \leq N$ after two iterations with widening. Combined with the loop exit condition $X \geq N$, this allows proving that, at the end of the program, $X = N$.

●

Enhanced Widenings. More generally, any widening on initial segments, that is, intervals of the form $[-\infty, a]$, $a \in \bar{\mathbb{I}}$, gives rise to a widening on zones by point-wise extension. For instance, [CC92c, § 8] proposes to improve the standard interval widening in order to infer sign information: if an interval not containing 0 is not stable, we first try to see if 0 is a stable bound instead of deciding it should be set to $\pm\infty$. A further generalisation, presented in [CC92c] and widely used in [BCC+02], is to design a widening parametric in a *finite* set $\mathbb{T} \subseteq \mathbb{I}$ of thresholds the stability of which should be tested before bailing out. This adapts nicely to a family of zone widenings parameterised by $\mathbb{T}$:

Definition 3.7.2. Widening with thresholds $\triangledown_{th}^{Zone}$ on zones.

$$(\mathbf{m} \triangledown_{th}^{Zone} \mathbf{n})_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} & \textit{if } \mathbf{m}_{ij} \geq \mathbf{n}_{ij} \\ \min \{\ x \in \mathbb{T} \cup \{+\infty\} \mid x \geq \mathbf{n}_{ij}\ \} & \textit{otherwise} \end{cases}$$

●

Proof.

We prove that the two requirements of Def. 2.2.3 are satisfied:

- $\mathbf{m}, \mathbf{n} \sqsubseteq^{\mathsf{DBM}} \mathbf{m} \triangledown_{th}^{Zone} \mathbf{n}$ is quite obvious.

- Consider a sequence $(\mathbf{m}^k)_{k\in\mathbb{N}}$ defined by $\mathbf{m}^{k+1} \stackrel{\text{def}}{=} \mathbf{m}^k \triangledown_{th}^{Zone} \mathbf{n}^k$. We can prove by induction on k that, for each matrix position (i, j), $\mathbf{m}_{ij}^k$ can only take values in the set $\mathbb{T} \cup \{+\infty, \mathbf{m}_{ij}^0\}$, which is finite. As a consequence, $\mathbf{m}^k$ can only take a value within a finite set of matrices and, as it is an increasing sequence, it must be stable after some finite k.

○

The sign-aware widening is a special instance of the widening with thresholds where $\mathbb{T} = \{0\}$, while $\mathbb{T} = \emptyset$ corresponds to the standard widening. The *height* of our widening, that is, the maximal length of an increasing sequence, is $n^2 \times |\mathbb{T} + 1|$: the worst case corresponds to stabilising to $\top^{\mathsf{DBM}}$ after having tested each threshold for each of the n^2 elements in turn.

Example 3.7.2. Using the widening with thresholds.

Consider the following Simple program that non-deterministically increments X and Y

for an unbounded amount of time while ensuring that $X \leq Y + 10$:

```
    X ← 5;
    Y ← 0;
    while ❶ rand {
      if rand { Y ← Y + 1 };
②     if rand {
        X ← X + 1;
        if Y + 10 < X { X ← Y + 10 }
③     }
④   }
```

The keyword `rand` in this program corresponds to a non-deterministic choice; it can be effectively implemented as the boolean expression $[0, 1] = 0$ and the `rand` keyword is only provided to increase the program readability.
The chaotic iterations with widening at ❶, using $\triangledown_{th}^{Zone}$ with $\mathbb{T} = \{10\}$, are:

iteration i	label l	zone X_l^i
0	❶	$X = 5 \wedge Y = 0 \wedge X - Y = 5$
1	②	$X = 5 \wedge 0 \leq Y \leq 1 \wedge 4 \leq X - Y \leq 5$
2	③	$X = 6 \wedge 0 \leq Y \leq 1 \wedge 5 \leq X - Y \leq 6$
3	④	$5 \leq X \leq 6 \wedge 0 \leq Y \leq 1 \wedge 4 \leq X - Y \leq 6$
4	❶ $\triangledown$	$5 \leq X \leq 10 \wedge 0 \leq Y \leq 10 \wedge -10 \leq X - Y \leq 10$
5	②	$5 \leq X \leq 10 \wedge 0 \leq Y \leq 11 \wedge -11 \leq X - Y \leq 10$
6	③	$6 \leq X \leq 11 \wedge 0 \leq Y \leq 11 \wedge -10 \leq X - Y \leq 10$
7	④	$5 \leq X \leq 11 \wedge 0 \leq Y \leq 11 \wedge -11 \leq X - Y \leq 10$
8	❶ $\triangledown$	$5 \leq X \wedge 0 \leq Y \wedge X - Y \leq 10$
9	②	$5 \leq X \wedge 0 \leq Y \wedge X - Y \leq 10$
10	③	$6 \leq X \wedge 0 \leq Y \wedge X - Y \leq 10$
11	④	$5 \leq X \wedge 0 \leq Y \wedge X - Y \leq 10$
12	❶ $\triangledown$	$5 \leq X \wedge 0 \leq Y \wedge X - Y \leq 10$

At iteration 4, the standard widening would lose all information about $X - Y$ while the widening with thresholds gives an opportunity for $X - Y \leq 10$ to stabilise, which is successful as $X_{❶}^{12} = X_{❶}^{8}$. Hence, it is able to prove that $X - Y \leq 10$ is indeed a valid loop invariant.

Note that it is not necessary to have exactly 10 in $\mathbb{T}$ if one simply wants to prove that $X - Y$ is bounded but not interested in the precise bound. Any constraint $X - Y \leq n$ is stable provided that $n \geq 10$, so, if $\mathbb{T}$ contains an element greater than 10, then the

following loop invariant will be found: $X - Y \leq \min(\mathbb{T} \cap [10, +\infty[)$.

●

Related Work. In [SKS00], Shaham, Kolodner, and Sagiv proposed the following widening for DBMs representing potential sets:

$$(\mathbf{m} \triangledown^{Zone}_{\text{[SKS00]}} \mathbf{n})_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} & \text{if } \mathbf{m}_{ij} = \mathbf{n}_{ij} \\ +\infty & \text{otherwise} \end{cases}$$

which resembles our standard widening $\triangledown^{Zone}_{std}$ but is less precise unless $\mathbf{m} \sqsubseteq^{\mathsf{DBM}} \mathbf{n}$. There is no reason for $\mathbf{m} \sqsubseteq^{\mathsf{DBM}} \mathbf{n}$ to hold: $\mathbf{n}$ is obtained from $\mathbf{m}$ by applying abstract transfer functions that need not be extensive nor monotonic. Moreover, [SKS00] insists on manipulating only closed matrices. This means that the widening sequence $\mathbf{m}_{i+1} \stackrel{\text{def}}{=} \mathbf{m}_i \triangledown \mathbf{n}_i$ is effectively replaced with $\mathbf{m}_{i+1} \stackrel{\text{def}}{=} (\mathbf{m}_i \triangledown \mathbf{n}_i)^*$ which is incorrect as we will see shortly.

3.7.2 Interactions between the Closure and our Widenings

Most operators and transfer functions we presented can be used on closed or non-closed DBMs, generally at the cost of the some precision loss when the arguments are not closed. The widenings presented in the preceding section differ in that the sequence $\mathbf{m}_{i+1} \stackrel{\text{def}}{=} (\mathbf{m}_i^*) \triangledown^{Zone} \mathbf{n}_i$ may not converge in finite time even though $\mathbf{m}_{i+1} \stackrel{\text{def}}{=} \mathbf{m}_i \triangledown^{Zone} \mathbf{n}_i$ always does.

An intuitive explanation for this problem is that the proof of termination for the sequence $\mathbf{m}_i$ relies on replacing more and more matrix coefficients with $+\infty$, while the closure tends to reduce the number of $+\infty$ coefficients. We now give a practical example where the termination of the iterations with widening is jeopardised by incorrect closure applications.

Example 3.7.3. Incorrect widening usage on zones.

Consider the matrices $\mathbf{m}_0$ and $\mathbf{n}_i$ defined in Fig. 3.10. Then, the infinite sequence $\mathbf{m}_{i+1} \stackrel{\text{def}}{=} (\mathbf{m}_i^*) \triangledown^{Zone}_{std} \mathbf{n}_i$ is strictly increasing. This sequence corresponds to the analysis of the following program:

```
    X ← 0;
    Y ← [−1, 1];
    while ❶ rand {
      if X = Y {
        if rand { Y ← X + [−1, 1] }
        else    { X ← Y + [−1, 1] }
      }
②   }
```

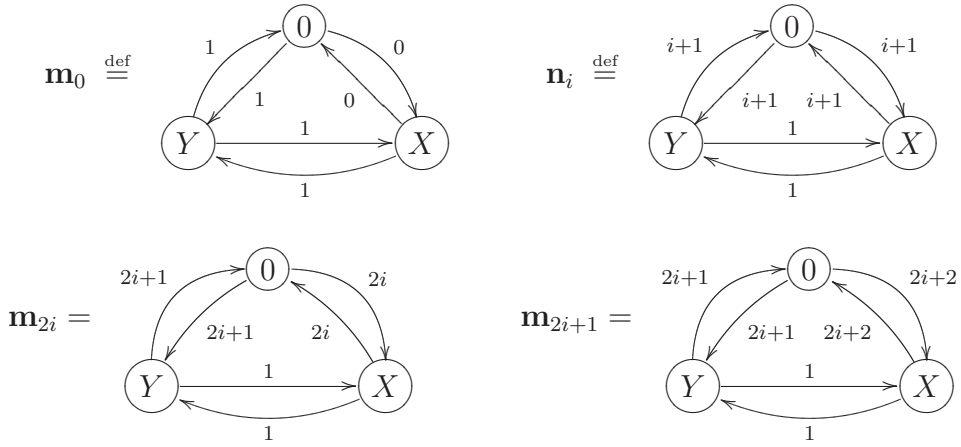


Figure 3.10: Example of infinite increasing chain defined by $\mathbf{m}_{i+1} \stackrel{\text{def}}{=} (\mathbf{m}_i^*) \triangledown_{std}^{Zone} \mathbf{n}_i$.

where $\mathbf{m}_i$ corresponds to the chaotic iterations with widening at ❶ while $\mathbf{n}_i$ is the abstract element derived from $\mathbf{m}_i$ at ②. If, however, we use the iteration sequence $\mathbf{m}_{i+1} \stackrel{\text{def}}{=} \mathbf{m}_i \triangledown_{std}^{Zone} \mathbf{n}_i$, then, after two iterations, we find the stable loop invariant $-1 \leq X - Y \leq 1$.

●

Intuitively, a DBM representing a zone can be seen as the reduced product, as defined in Sect. 2.2.6, of a quadratic number of abstract domains, each one of them focusing on the bounds of a single variable X or expression $X - Y$. With this in mind, our closure $*$ can be seen as a global n^2-way reduction between all these abstract elements while our widenings $\triangledown^{Zone}$ are component-wise extensions of classical interval widenings. The unfortunate interaction between the closure and our widenings is then only an illustration of the danger of performing a reduction after a component-wise widening on the reduced product of abstract domains.

Remark, however, that closing the right argument is not a problem as the sequence $\mathbf{m}_{i+1} \stackrel{\text{def}}{=} \mathbf{m}_i \triangledown^{Zone} (\mathbf{n}_i^*)$ always converges in finite time. This amounts to using a widening $\triangledown'$, defined by $\mathbf{m} \triangledown' \mathbf{n} \stackrel{\text{def}}{=} \mathbf{m} \triangledown (\mathbf{n}^*)$ which does not satisfy the requirement $\mathbf{n} \sqsubseteq^{\mathsf{DBM}} \mathbf{m} \triangledown' \mathbf{n}$. Fortunately, although Thm. 2.2.9 does not apply directly, it is still safe to use chaotic iterations in this case, provided that the termination condition $\mathbf{m}_{i+1} \sqsubseteq^{\mathsf{DBM}} \mathbf{m}_i$ is replaced with $\gamma^{Zone}(\mathbf{m}_{i+1}) \subseteq \gamma^{Zone}(\mathbf{m}_i)$, that is, by Thm. 3.4.2, $\mathbf{m}_{i+1}^* \sqsubseteq^{\mathsf{DBM}} \mathbf{m}_i$. It should be noted, however, that widenings are generally non-monotonic, and so, the sequence $\mathbf{m}_{i+1} \stackrel{\text{def}}{=} \mathbf{m}_i \triangledown^{Zone} (\mathbf{n}_i^*)$ may not converge towards a more precise limit than $\mathbf{m}_{i+1} \stackrel{\text{def}}{=} \mathbf{m}_i \triangledown^{Zone} \mathbf{n}_i$.

Future Work. It is possible in all our transfer functions and operators to safely replace any argument $\mathbf{m}$ with another DBM $\mathbf{m}'$ such that $\gamma^{Zone}(\mathbf{m}') = \gamma^{Zone}(\mathbf{m})$, except when it comes to our widenings. Their convergence indeed relies on information encoded in the specific choice of a set of zone constraints and are no longer available when considering only the zone it represents by γ^{Zone}. This is unlike the standard widening on polyhedra, introduced in [CH78] and then refined in [Hal79] and [BHRZ03], which is insensible to the chosen polyhedron representation. An interesting future work would be to try and design such a *semantical* widening on zones.

3.7.3 Narrowings

As for the widening, any narrowing on initial segments gives rise to a narrowing on the zone domain by point-wise extension. We present here a "standard" narrowing which is based on the standard narrowing $\triangle^{Int}$ on intervals; it refines only constraints involving $+\infty$:

Definition 3.7.3. Standard narrowing $\triangle^{Zone}_{std}$ on zones.

$$(\mathbf{m} \triangle^{Zone}_{std} \mathbf{n})_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{n}_{ij} & \textit{if } \mathbf{m}_{ij} = +\infty \\ \mathbf{m}_{ij} & \textit{otherwise} \end{cases}$$

●

Proof.

We prove that the two requirements of Def. 2.2.4 are satisfied:

- We obviously have $\mathbf{m} \sqcap^{\mathsf{DBM}} \mathbf{n} \sqsubseteq^{\mathsf{DBM}} \mathbf{m} \triangle^{Zone}_{std} \mathbf{n} \sqsubseteq^{\mathsf{DBM}} \mathbf{m}$.

- Consider a sequence defined by $\mathbf{m}^{k+1} \stackrel{\text{def}}{=} \mathbf{m}^k \triangle^{Zone}_{std} \mathbf{n}^k$. Consider the set S^k of matrix positions (i, j) such that $\mathbf{m}^k_{ij} = +\infty$. A consequence of Def. 3.7.3 is that S^k is decreasing for $\subseteq$. So, there exists some k such that $S^k = S^{k+1}$. For such a k, whenever $\mathbf{m}^k_{ij} = +\infty$, we also have $\mathbf{m}^{k+1}_{ij} = +\infty$. If $\mathbf{m}^k_{ij} \neq +\infty$, then by Def. 3.7.3 we have $\mathbf{m}^{k+1}_{ij} = \mathbf{m}^k_{ij}$. We thus have proved that $\mathbf{m}^{k+1} = \mathbf{m}^k$ for this k.

○

Example 3.7.4. Using the standard narrowing.

Consider the following Simple program that can increment X at most eleven times:

```
    X ← 0;
    I ← 0;
    while ❶ I ≤ 10 {
②     if rand { X ← X + 1 };
      I ← I + 1
③   }
④
```

which gives the following iterations using the standard widening and narrowing at ❶:

iteration i	label l	zone X_l^i
0	❶	$X = 0 \wedge I = 0 \wedge I - X = 0$
1	②	$X = 0 \wedge I = 0 \wedge I - X = 0$
2	③	$0 \leq X \leq 1 \wedge I = 1 \wedge 0 \leq I - X \leq 1$
3	❶ ∇	$0 \leq X \wedge 0 \leq I \wedge 0 \leq I - X$
4	②	$0 \leq X \leq 10 \wedge 0 \leq I \leq 10 \wedge 0 \leq I - X \leq 10$
5	③	$0 \leq X \leq 11 \wedge 1 \leq I \leq 11 \wedge 0 \leq I - X \leq 11$
6	❶ ∇	$0 \leq X \wedge 0 \leq I \wedge 0 \leq I - X$
7	❶ Δ	$0 \leq X \leq 11 \wedge 0 \leq I \leq 11 \wedge 0 \leq I - X \leq 11$
8	②	$0 \leq X \leq 10 \wedge 0 \leq I \leq 10 \wedge 0 \leq I - X \leq 10$
9	③	$0 \leq X \leq 11 \wedge 1 \leq I \leq 11 \wedge 0 \leq I - X \leq 11$
10	❶ Δ	$0 \leq X \leq 11 \wedge 0 \leq I \leq 11 \wedge 0 \leq I - X \leq 11$
11	④	$0 \leq X \leq 11 \wedge I = 11 \wedge 0 \leq I - X \leq 11$

Of particular interest are the increasing iterations with widening $X_❶^0 \sqsubset^{\mathsf{DBM}} X_❶^3 = X_❶^6$, followed by the decreasing iterations with narrowing $X_❶^6 \sqsupset^{\mathsf{DBM}} X_❶^7 = X_❶^{10}$. The iterations with widening are able to prove that $X \leq I$ is a loop invariant. One narrowing application is enough to recover the invariants $I \leq 11$ and $X \leq 11$ which cannot be inferred with the standard widening solely. This invariants allows proving that, at the end of the loop, $I = 11$ and $X \leq 11$.

In the interval domain, we would still find the loop invariant $I \leq 11$ and the invariant $I = 11$ at ④. However, as the upper bound on X is recovered after narrowing the upper bound of I and using the relational invariant $X \leq I$, the interval domain would not find any upper bound on X despite the use of a narrowing.

Note that the invariant $I \leq 11$ can be inferred using the widening with thresholds of Def. 3.7.2 provided that $11 \in \mathbb{T}$. The narrowing method is more satisfying as it does not necessitate *a priori* knowledge of the invariant to be found. Unlike Ex. 3.7.2, the value 11 to put in $\mathbb{T}$ does not appear syntactically in the program source. Conversely, the precise

invariant $X - Y \leq 10$ on Ex. 3.7.2 cannot be found using the standard widening and narrowing because the test $Y + 10 < X$ that ensures that X does not grow too much is not executed at each loop iteration, a condition *sine qua non* for the narrowing to refine a loop invariant — a discussion on the subject can be found in [BCC+02, § 6.4].

●

Unlike what happened for our widenings, the sequence $\mathbf{m}_{k+1} \stackrel{\text{def}}{=} (\mathbf{m}_k^*) \triangle_{std}^{Zone} \mathbf{n}_k$ will always converge. However, $\triangle_{std}^{Zone}$ is hardly monotonic in its left argument, so, unlike many operators on zones, the result may be less precise if the left argument is closed.

3.8 Cost Considerations

3.8.1 Ways to close

Our most costly algorithm on DBMs is the closure with its cubic cost. As many operators require their arguments to be closed for best precision, the closure appears to be the bottleneck of any static analysis based on the zone abstract domain. Fortunately, the algorithm is sufficiently simple and regular to be optimised automatically by compilers. It is also well-fitted for low-level parallelisation using the SIMD instruction sets of modern processors and can be worth the effort of writing hand-optimised assembly code.

An orthogonal way to optimise the analysis cost without compromising the precision is to avoid using the cubic closure algorithm and use instead, whenever possible, incremental closures or algorithms that preserve the closure. Ideally, we would like to know, when computing an abstract function, whether its result will be fed to an abstract function requiring a closed argument or not. In the later case, we can choose the version of the algorithm that does not preserve the closure as it is generally cheaper. Consider, for instance, a sequence of m tests of the form $V_i - V_j \leq c$ on a closed argument and whose result should be in closed form. Using an incremental closure at each step would cost $\mathcal{O}(m \times n^2)$ while using only one full closure at the end of the sequence would cost $\mathcal{O}(m + n^3)$, which is lighter if m is sufficiently large.

If the complete sequence of abstract functions to be called is known in advance — such as in *abstract compilation* as opposed to abstract interpretation — a global optimisation pass can select the optimal version for each function to minimise the overall cost. In the context of the abstract interpretation of Simple programs, we have experimented with two simple strategies: always choose the closure-preserving version of our transfer functions, and always choose the non-preserving one. The former gives somewhat better results. Experience shows that calls to abstract functions that do not need a closed argument are rather isolated — for instance, long sequences of tests without any control-flow join do not happen frequently and abstract intersections are rare.

3.8.2 Hollow Representation

Whenever memory is a concern, one can think of using a *sparse* matrix representation where the $+\infty$ elements are left implicit instead of full matrices that have a fixed quadratic memory cost. Equivalently, we can use a graph-oriented representation using pairs of pointers and (finite) weights for arcs. However, such representations are only effective when there are lots of $+\infty$ elements as they impose an overhead on each represented matrix element — or graph arc. In their [LLPY97] article, Larsen, Larsson, Pettersson, and Yi propose an algorithm to remove redundant constraints and maximise the number of $+\infty$ elements. We now briefly recall their algorithm.

Computing the Hollow Representation. Let $\mathbf{m}$ be a DBM. We first need to compute its closure $\mathbf{m}^*$. A constraint $\mathbf{m}^*_{ij}$ is called *redundant* if and only if there exists some k such that $\mathbf{m}^*_{ij} = \mathbf{m}^*_{ik} + \mathbf{m}^*_{kj}$. It seems that any such $\mathbf{m}^*_{ij}$ can be safely replaced with $+\infty$ because the precise constraint still exists implicitly in the matrix and can be recovered by a closure application. Consider, however, the case where $\mathbf{m}^*_{ij} = \mathbf{m}^*_{ik} + \mathbf{m}^*_{kj}$ and the cycle $\langle i, k, j\rangle$ has a total weight of 0. One can prove, using the local characterisation of the closure, Thm. 3.3.6, that we have $\mathbf{m}^*_{ij} + \mathbf{m}^*_{ji} = \mathbf{m}^*_{ik} + \mathbf{m}^*_{ki} = \mathbf{m}^*_{jk} + \mathbf{m}^*_{kj} = 0$, and so:

$$\begin{array}{l|l} \mathbf{m}^*_{ij} = \mathbf{m}^*_{ik} + \mathbf{m}^*_{kj} & \mathbf{m}^*_{ji} = \mathbf{m}^*_{jk} + \mathbf{m}^*_{ki} \\ \mathbf{m}^*_{ki} = \mathbf{m}^*_{kj} + \mathbf{m}^*_{ji} & \mathbf{m}^*_{ik} = \mathbf{m}^*_{ij} + \mathbf{m}^*_{jk} \\ \mathbf{m}^*_{jk} = \mathbf{m}^*_{ji} + \mathbf{m}^*_{ik} & \mathbf{m}^*_{kj} = \mathbf{m}^*_{ki} + \mathbf{m}^*_{ij} \end{array}$$

so, our naive approach would remove all the constraints between i, j, and k at once, and lose information. The solution proposed by [LLPY97] is to:

- First, determine equivalence classes for variables that appear on a cycle with a zero total weight. By Thm. 3.3.6, two variables V_i and V_j are in the same equivalence class if and only if $\mathbf{m}^*_{ij} = -\mathbf{m}^*_{ji}$. We denote by $p(i)$ the smallest index of all the variables in the same equivalence class as V_i.
- Then, construct a matrix from $\mathbf{m}^*$ as follows:
  - keep the element at line $p(i)$, column $p(j)$ if there does not exist $k \neq i, j$ such that $\mathbf{m}^*_{p(i)\,p(j)} = \mathbf{m}^*_{p(i)\,p(k)} + \mathbf{m}^*_{p(k)\,p(j)}$;
  - for each equivalence class $(i_1, \ldots, i_m)$ where $i_1 < i_2 < \cdots < i_m$, keep all the elements at position (i_k, i_{k+1}) and the element at position (i_m, i_1);
  - set all other elements to $+\infty$.

  We call the resulting matrix the *hollow* representation of $\mathbf{m}$ and denote it by $\mathit{Hollow}(\mathbf{m})$.

The hollow representation is not unique as it depends upon an ordering among variables. However, it is a representation for γ^{Zone} that has as many $+\infty$ as possible:

Theorem 3.8.1. Properties of the hollow representation.

1. $(Hollow(\mathbf{m}))^* = \mathbf{m}^*$.
   As a consequence $\gamma^{Zone}(Hollow(\mathbf{m})) = \gamma^{Zone}(\mathbf{m})$ *and* $\gamma^{Pot}(Hollow(\mathbf{m})) = \gamma^{Pot}(\mathbf{m})$.
2. *If* $\mathbf{o}^* = \mathbf{m}^*$, *then* $\mathbf{o}$ *has a smaller or equal number of* $+\infty$ *elements than* $Hollow(\mathbf{m})$.
3. $Hollow(\mathbf{m})$ *can be computed in cubic time.*

●

Proof. Done in [LLPY97]. ○

Space Versus Time Trade-Off. We have seen in Sect. 3.8.1 that, except for the widening iterates, it is best to keep the matrices in closed form for precision and time efficiency purposes. This requirement is in conflict with the use of a hollow representation. Also, using a sparse matrix data-structure, which is mandatory to actually benefit from the space improvement of hollow representations, may incur an extra time cost on all algorithms. Thus, a trade-off between space and time must be chosen.

A lazy approach, such as the ones used in memory swapping can be devised: all computations are done using a full and closed matrix representation but, when memory runs low, some DBMs among those not used recently are transformed into sparse and hollow DBMs. Whenever needed, these "swapped-out" DBMs are restored into their full and closed DBM representation. Special care must be taken for DBMs used in widenings: in order to prevent the sequence from diverging, an iterate should never be replaced with a different DBM representing the same zone. In particular, it cannot be closed or put in hollow form — which requires closing. A simple solution is to mark such DBMs as "unswappable".

3.9 Conclusion

We have presented, in this chapter, a fully-featured relational numerical abstract domain for invariants of the form $X - Y \leq c$ and $\pm X \leq c$, called the *zone abstract domain*. It is based on a matrix representation with a memory cost *quadratic* in the number of variables, and algorithms based on shortest-path closure, so that it features a *cubic* worst-case time cost per abstract operation. Thus, it is, in terms of precision and cost, strictly between the interval and the polyhedron domains. Even though the zone abstract domain can be used "as is", it can also be seen as a step towards the design of the strictly more precise *octagon abstract domain* that is presented in the next chapter.

Chapter 4

The Octagon Abstract Domain

Le domaine abstrait des octogones est une extension du domaine des zones qui permet de représenter et de manipuler des invariants de la forme $\pm X \pm Y \leq c$ sans augmentation du coût asymptotique en temps et en mémoire. Un gros travail est nécessaire pour adapter l'algorithme de plus-court chemin utilisé dans le domaine des zones. Les autres algorithmes s'adaptent plus aisément.

The octagon abstract domain is an extension of the zone domain that can represent and manipulate invariants of the form $\pm X \pm Y \leq c$ without increasing the asymptotic memory and time cost. Much work is needed to adapt the shortest-path closure algorithm used in the zone domain, but then, all other algorithms adapt more easily.

4.1 Introduction

In this chapter, we extend the zone abstract domain to infer constraints of the form $\pm X \pm Y \leq c$ while keeping a quadratic memory cost and a cubic worst-case time cost per abstract operation. Constraints of the form $X+Y \in [a, b]$ are an interesting addition to the potential constraints $X - Y \leq c$. They allow representing, for instance, properties such as *mutual exclusion* $\neg(X \wedge Y)$, encoded as $X \geq 0 \wedge Y \geq 0 \wedge X+Y \leq 1$, but also, numerical properties on absolute values such as $|X| \leq Y + 1$, encoded as $X - Y \leq 1 \wedge -X - Y \leq 1$.

Previous Work. Much work has been done to extend the shortest-path closure satisfiability algorithm for potential constraints proposed initially by Bellmann [Bel58]. A natural question is whether graph-based algorithms can apply to richer constraint forms. For instance, the works of Shostak [Sho81] and Nelson [Nel78] present algorithms to test

the satisfiability of conjunctions of constraints of the form $\alpha X + \beta Y \leq c$ in $\mathbb{R}$ and $\mathbb{Q}$. In [JMSY94] and [HS97], Jaffar, Maher, Stuckey, Yap, and Harvey focus on constraints with unit coefficients: $\pm X \pm Y \leq c$; they also treat the more complex case of integers. These works focus on satisfiability and do not study the more complex problem of *manipulating* constraint conjunctions.

In [BK89], Balasundaram and Kennedy propose to use constraint of the form $\pm X \pm Y \leq c$ to represent data access patterns in arrays in order to perform automatic loop parallelisation. The authors present an intersection and a union algorithms, which are unfortunately a little flawed — see Sect. 4.4.1 — and a transfer function to derive the bounds in nested loops of a simple form, which is too specific for our purpose.

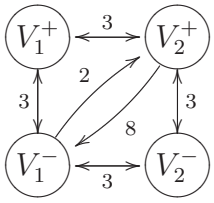
Our Contribution. As [JMSY94, HS97, BK89], we focus on unit constraints involving two variables, $\pm X \pm Y \leq c$, in $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$, but propose a full abstract domain. Our construction mimics that of the zone domain presented in the previous chapter. In particular, we first introduce a representation, then provide a normal form and state a saturation property that is used to guarantee the precision of the subsequently introduced operators and transfer functions.

4.2 Modified Representation

We call *octagonal constraint* any constraint of the form $\pm V_i \pm V_j \leq c$. Octagonal constraints include potential constraints $V_i - V_j \leq c$. Also, interval constraints $V_i \leq a$, $V_j \geq b$, can be encoded respectively as $V_i + V_i \leq 2a$ and $-V_j - V_j \leq -2b$, so, octagonal constraints include all the zone constraints.

The set of points that satisfy a conjunction of octagonal constraints will be called *an octagon*. We denote by *Oct* the set of all octagons for a given set of variables $\mathcal{V}$. The name "octagon" comes from the fact that, in two dimensions, the set of points satisfying a conjunction of octagonal constraints is a planar convex polyhedron with *at most eight sides*. More generally, in dimension n, an octagon has at most $2n^2$ faces. Note that the exact same sets are refereed in [BK89] as *simple sections* by Balasundaram and Kennedy.

An important idea of the previous chapter, introduced in Sect. 3.2.1, was to use the Difference Bound Matrix (DBM) representation for conjunction of potential constraints and encode zone constraints as potential constraints using an additional variable V_0 with a special meaning. We use a similar idea here and also encode octagonal constraints as potential constraints to benefit from the nice properties of DBMs and legacy algorithms.

(a) $\begin{cases} V_1 + V_2 \leq 3 \\ V_2 - V_1 \leq 3 \\ V_1 - V_2 \leq 3 \\ -V_1 - V_2 \leq -3 \\ 2V_2 \leq 2 \\ -2V_2 \leq 8 \end{cases}$ (b) 

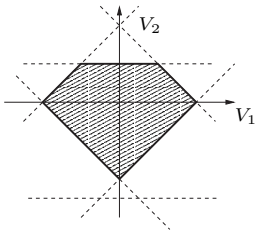
(c) 

Figure 4.1: A conjunction of octagonal constraints (a), its encoding as a potential graph on $\mathcal{V}'$ (b), and the octagon it defines by γ^{Oct} (c).

4.2.1 Octagonal Constraints Encoding

Given a set of variables $\mathcal{V} = \{V_1, \ldots, V_n\}$, we introduce the set $\mathcal{V}' \stackrel{\text{def}}{=} \{V'_1, \ldots, V'_{2n}\}$ containing twice as many variables. Each variable in $V_i \in \mathcal{V}$ has both a *positive form* V'_{2i-1}, also denoted by V_i^+, and a *negative form* V'_{2i}, also denoted by V_i^-, in $\mathcal{V}'$. We will encode octagonal constraints in $\mathcal{V}$ as potential constraints in $\mathcal{V}'$. Intuitively, in a potential constraint, V_i^+ will represent V_i while V_i^- will represent $-V_i$. More formally:

Definition 4.2.1. Encoding octagonal constraints as potential constraints.

The constraint				*is represented as*						
$V_i - V_j$	$\leq$	c	$(i \neq j)$	$V'_{2i-1} - V'_{2j-1}$	$\leq$	c	*and*	$V'_{2j} - V'_{2i}$	$\leq$	c
$V_i + V_j$	$\leq$	c	$(i \neq j)$	$V'_{2i-1} - V'_{2j}$	$\leq$	c	*and*	$V'_{2j-1} - V'_{2i}$	$\leq$	c
$-V_i - V_j$	$\leq$	c	$(i \neq j)$	$V'_{2i} - V'_{2j-1}$	$\leq$	c	*and*	$V'_{2j} - V'_{2i-1}$	$\leq$	c
V_i	$\leq$	c		$V'_{2i-1} - V'_{2i}$	$\leq$	$2c$				
V_i	$\geq$	c		$V'_{2i} - V'_{2i-1}$	$\leq$	$-2c$				

●

Thus, a conjunction of octagonal constraints in $\mathcal{V}$ can be represented as a DBM of dimension $2n$, that is, a $2n \times 2n$ matrix with elements in $\overline{\mathbb{I}} = \mathbb{I} \cup \{+\infty\}$ or, equivalently, a potential graph with nodes in $\mathcal{V}'$ and weights in $\mathbb{I}$. As for DBMs representing potential sets, we number the lines and columns from 1 to $2n$, and the line or column i corresponds to the variable V'_i, that is $V_{\lceil i/2 \rceil}$. Our encoding is exemplified in Fig. 4.1.

Concretisation. Given a DBM $\mathbf{m}$ of dimension $2n$, we can define formally the octagon in $\mathcal{P}(\mathbb{I}^n)$ described by $\mathbf{m}$ as follows:

Definition 4.2.2. Octagon concretisation γ^{Oct} of a DBM.

$$\gamma^{Oct}(\mathbf{m}) \stackrel{\text{def}}{=} \{ (v_1, \ldots, v_n) \in \mathbb{I}^n \mid (v_1, -v_1, \ldots, v_n, -v_n) \in \gamma^{Pot}(\mathbf{m}) \} \ .$$
$$Oct \stackrel{\text{def}}{=} \{ \gamma^{Oct}(\mathbf{m}) \mid \mathbf{m} \in \mathsf{DBM} \} \ .$$

●

As for γ^{Zone} defined in Def. 3.2.2, γ^{Oct} combines the semantics of potential constraints, expressed using γ^{Pot}, with constraints inherent to our encoding, that is, $\forall i \geq 1$, $V'_{2i-1} = -V'_{2i}$. If we denote by Π the plane $\{ \vec{v}\,' \in \mathbb{I}^{2n} \mid v'_{2i-1} = -v'_{2i} \}$, there is a bijection between $\gamma^{Pot}(\mathbf{m}) \cap \Pi$ and $\gamma^{Oct}(\mathbf{m})$.

4.2.2 Coherence

Note that in Def. 4.2.1 some octagonal constraints have two different encodings as potential constraints in $\mathcal{V}'$, and so, are defined by two elements in the DBM representation. We will say that a DBM is *coherent* if each constraint in such a related pair is equivalent to the other one. More formally:

Definition 4.2.3. Coherent DBMs and the $\bar{\cdot}$ operator.

$$\mathbf{m} \in \mathsf{DBM} \textit{ is coherent} \iff^{\text{def}} \forall i, j, \ \mathbf{m}_{ij} = \mathbf{m}_{\bar{j}\bar{i}}$$
$$\textit{where the } \bar{\cdot} \textit{ operator on indexes is defined as:} \quad \bar{i} \stackrel{\text{def}}{=} \begin{cases} i+1 & \textit{if } i \textit{ is odd} \\ i-1 & \textit{if } i \textit{ is even} \end{cases}$$

●

Intuitively, the $\bar{\cdot}$ operator corresponds to switching between the positive and the negative form of a variable. Obviously, $\bar{\bar{i}} = i$. Also, the $\bar{\cdot}$ operator can be easily implemented using the `xor` bit-wise exclusive or operator as $\bar{i} - 1 = (i - 1)$ `xor` 1.

4.2.3 Lattice Structure

From now on in this chapter, we will only work with coherent DBMs. We denote by $c\mathsf{DBM}$ the set of coherent DBMs enriched with a bottom element $\bot^{\mathsf{DBM}}$. The exact lattice construction of Thm. 3.2.2 can be restricted to $c\mathsf{DBM}$ because $\sqcup^{\mathsf{DBM}}$ and $\sqcap^{\mathsf{DBM}}$ preserve the coherence, and $\top^{\mathsf{DBM}}$ is itself coherent. We thus obtain a lattice $(c\mathsf{DBM}, \sqsubseteq^{\mathsf{DBM}}, \sqcup^{\mathsf{DBM}}, \sqcap^{\mathsf{DBM}}, \bot^{\mathsf{DBM}}, \top^{\mathsf{DBM}})$. Moreover, when $\mathbb{I} = \mathbb{Z}$ or $\mathbb{I} = \mathbb{R}$, this lattice is complete. Finally, γ^{Oct}, extended to $c\mathsf{DBM}$ by $\gamma^{Oct}(\bot^{\mathsf{DBM}}) = \emptyset$, is a complete $\sqcap-$morphism — and, thus, it is monotonic.

Partial Galois Connection. By Thm. 2.2.3, we can define a canonical partial α^{Oct} and get a partial Galois connection $\mathcal{P}(\mathcal{V} \to \mathbb{I}) \underset{\alpha^{Oct}}{\overset{\gamma^{Oct}}{\leftrightarrows}} c\mathsf{DBM}$:

- $\alpha^{Oct}(R) \stackrel{\text{def}}{=} \bot^{\mathsf{DBM}}$ if $R = \emptyset$,
- $\left(\alpha^{Oct}(R)\right)_{ij} \stackrel{\text{def}}{=} \begin{cases} \min \{ \rho(V_l) - \rho(V_k) \mid \rho \in R \} & \text{when } i = 2k-1,\ j = 2l-1 \\ & \text{or } i = 2l,\ j = 2k \\ \min \{ \rho(V_l) + \rho(V_k) \mid \rho \in R \} & \text{when } i = 2k,\ j = 2l-1 \\ \min \{ -\rho(V_l) - \rho(V_k) \mid \rho \in R \} & \text{when } i = 2k-1,\ j = 2l \end{cases}$
  if $R \neq \emptyset$.

As for the zone abstract domain, the Galois connection is partial for all transfer functions involving only interval linear forms, at least. Moreover, if $\mathbb{I} \in \{\mathbb{Z}, \mathbb{R}\}$, then α^{Oct} is total and we obtain a regular Galois connection.

4.3 Modified Closure Algorithms

As γ^{Oct} is not injective, we are interested in finding a canonical representation of octagons as coherent DBMs. This is always possible because $(\alpha^{Oct}, \gamma^{Oct})$ is $Id-$partial: $\forall \mathbf{m} \in c\mathsf{DBM}$, $(\alpha^{Oct} \circ \gamma^{Oct})(\mathbf{m})$ always exist. We will see, however, that the actual computation of this canonical form is a little more complex than it was for potential sets and zones.

4.3.1 Emptiness Testing

We first focus on the problem of determining the satisfiability of a conjunction of octagonal constraints when $\mathbb{I}$ is $\mathbb{Q}$ or $\mathbb{R}$. Thanks to the following theorem, it can be reduced to the satisfiability of potential constraints, a problem already solved in the previous chapter:

Theorem 4.3.1. Satisfiability of a conjunction of octagonal constraints.
When $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$, $\gamma^{Oct}(\mathbf{m}) = \emptyset \iff \gamma^{Pot}(\mathbf{m}) = \emptyset$.

●

Proof.
If $\gamma^{Pot}(\mathbf{m}) = \emptyset$ then, obviously, $\gamma^{Oct}(\mathbf{m}) = \emptyset$ by Def. 4.2.2.
We now suppose that $\gamma^{Pot}(\mathbf{m}) \neq \emptyset$ and prove that $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$ as well. Take $\vec{v}\,' = (v'_1, \ldots, v'_{2n}) \in \gamma^{Pot}(\mathbf{m})$. We have $\forall i, j,\ v'_j - v'_i \leq \mathbf{m}_{ij}$. By coherence of $\mathbf{m}$, this implies $\forall i, j,\ v'_{\bar{i}} - v'_{\bar{j}} \leq \mathbf{m}_{\bar{j}\bar{i}} = \mathbf{m}_{ij}$, which means that $\vec{w}\,' \stackrel{\text{def}}{=} (-v'_2, -v'_1, \ldots, -v'_{2n}, -v'_{2n-1}) \in \gamma^{Pot}(\mathbf{m})$ as well. As $\gamma^{Pot}(\mathbf{m})$ is defined by an intersection of half-spaces, it is convex, so, the point $\vec{z}\,' \stackrel{\text{def}}{=} (\vec{v}\,' + \vec{w}\,')/2$ is also in $\gamma^{Pot}(\mathbf{m})$. Moreover, the coordinates z'_i of $\vec{z}\,'$ verify:

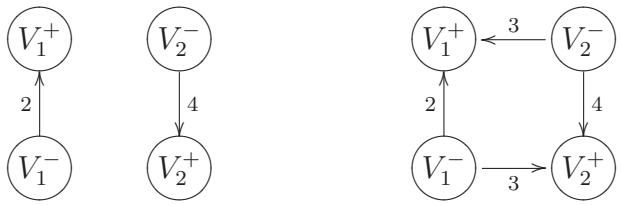


Figure 4.2: Two different closed potential graphs that represent the same octagon: $V_1 \leq 1 \wedge V_2 \leq 2$.

$\forall i$, $z'_{2i} = (v'_{2i} - v'_{2i-1})/2 = -z'_{2i-1}$. By Def. 4.2.2, this means that $(z_1, z_3, \ldots, z_{2n-1}) \in \gamma^{Oct}(\mathbf{m})$.

When $\mathbb{I} = \mathbb{Z}$, this proof does not hold because $\vec{z}'$ may not be in $\gamma^{Pot}(\mathbf{m})$ whenever some $v'_{2i} - v'_{2i-1}$ is not even.

○

Using Thm. 4.3.1 in conjunction with Thm. 3.3.1, $\gamma^{Oct}(\mathbf{m}) = \emptyset$ is equivalent to the existence of a cycle — or equivalently a simple cycle — with strictly negative total weight, and can be tested using the Bellman–Ford algorithm, for instance. Note that this theorem is not true when $\mathbb{I} = \mathbb{Z}$; the integer case will be discussed in more details in Sect. 4.3.5.

4.3.2 Strong Closure

Let us now consider a DBM $\mathbf{m}$ such that $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$. We saw in the previous chapter that $\mathbf{m}$'s shortest-path closure $\mathbf{m}^*$ exists and is the canonical representation for $\gamma^{Pot}(\mathbf{m})$ and $\gamma^{Zone}(\mathbf{m})$. It is easy to see that, if $\mathbf{m}$ is coherent, so is $\mathbf{m}^*$. However, $\gamma^{Pot}(\mathbf{m}) = \gamma^{Pot}(\mathbf{n}) \Longrightarrow \gamma^{Oct}(\mathbf{m}) = \gamma^{Oct}(\mathbf{n})$, but the converse is not true, so, $\mathbf{m}^*$ may not be $\mathbf{m}$'s canonical representation for $\gamma^{Oct}(\mathbf{m})$. Indeed, Fig. 4.2 presents two *closed* DBMs representing the same octagon, but different potential sets.

Intuition. Recall that, on the one hand, the local characterisation of the closure expresses that $\mathbf{m}$ is closed if and only if $\forall i, j$, $\mathbf{m}_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj}$ and, on the other hand, the Floyd–Warshall closure algorithm performs steps of the form $\mathbf{m}^k_{ij} \stackrel{\text{def}}{=} \min(\mathbf{m}^{k-1}_{ij}, \mathbf{m}^{k-1}_{ik} + \mathbf{m}^{k-1}_{kj})$. Hence, we can see the Floyd–Warshall algorithm as performing *local* constraint propagations of the form:

$$\left\{ \begin{array}{l} V'_i - V'_k \leq c \\ V'_k - V'_j \leq d \end{array} \right. \Longrightarrow V'_i - V'_j \leq c + d$$

until no further propagation can be done. Our idea is to add a second form of local constraint propagation:

$$\begin{cases} V'_{\bar{i}} - V'_i \leq c \\ V'_j - V'_{\bar{j}} \leq d \end{cases} \Longrightarrow V'_j - V'_i \leq (c+d)/2$$

that is, replacing $\mathbf{m}_{ij}$ with $\min(\mathbf{m}_{ij}, (\mathbf{m}_{i\bar{i}} + \mathbf{m}_{\bar{j}j})/2)$. This second transformation is valid because we are interested only in points such that $V'_i = -V'_{\bar{i}}$. On $\mathcal{V}$, it corresponds to adding two unary constraints $\pm 2V_i \leq c$ and $\pm 2V_j \leq d$ to derive a binary constraint $\pm V_j \pm V_i \leq (c+d)/2$. However, the second transformation works on pairs of edges that do not form a path in the potential graph, and so, cannot be reduced to the first transformation.

Formalisation. A DBM in $\mathbb{R}$ or $\mathbb{Q}$ that is stable by our two local transformations will be said to be *strongly closed*, which is formalised as:

Definition 4.3.1. Strong closure.

A coherent DBM $\mathbf{m}$ *in* $\mathbb{R}$ *or* $\mathbb{Q}$ *is said to be strongly closed if and only if:*

$$\begin{cases} \forall i,j,k, & \mathbf{m}_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj} \\ \forall i,j, & \mathbf{m}_{ij} \leq (\mathbf{m}_{i\bar{i}} + \mathbf{m}_{\bar{j}j})/2 \\ \forall i, & \mathbf{m}_{ii} = 0 \end{cases}$$

●

Note that strong closure implies regular closure.

As for the emptiness test of Thm. 4.3.1, we restrict ourselves to the case where $\mathbb{I} \neq \mathbb{Z}$. Indeed, our definition of strong closure uses of a division by 2 which is ill-defined on integers. The precise treatment of the case $\mathbb{I} = \mathbb{Z}$ is postponed to Sect. 4.3.5.

Saturation. As closed DBMs, strongly closed DBMs enjoy a saturation property, that is, every octagonal constraint in a strongly closed DBM actually "touches" the octagon:

Theorem 4.3.2. Saturation of strongly closed DBMs.

If $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$ *and* $\mathbf{m}$ *is strongly closed, then:*

1. $\forall i,j,$ *if* $\mathbf{m}_{ij} < +\infty,$ *then* $\exists (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m})$ *such that* $v'_j - v'_i = \mathbf{m}_{ij}$.
2. $\forall i,j,$ *if* $\mathbf{m}_{ij} = +\infty,$ *then* $\forall M < +\infty, \exists (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m})$ *such that* $v'_j - v'_i \geq M$.

where the v'_k *are derived from the* v_k *by* $v'_{2k-1} \stackrel{\text{def}}{=} v_k$ *and* $v'_{2k} \stackrel{\text{def}}{=} -v_k$.

●

Proof.

1. Set a pair (i_0, j_0) such that $\mathbf{m}_{i_0 j_0} < +\infty$.

   The case $i_0 = j_0$ is easy: as $\mathbf{m}$ is strongly closed, we have $\mathbf{m}_{i_0 j_0} = 0$ and any point $(v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}) \neq \emptyset$ is such that $v'_{i_0} - v'_{i_0} \leq 0$.

   We now consider the much more complex case $i_0 \neq j_0$. We denote by $\mathbf{m}'$ the matrix equal to $\mathbf{m}$ except that $\mathbf{m}'_{j_0 i_0} \stackrel{\text{def}}{=} \mathbf{m}'_{\overline{i_0}\,\overline{j_0}} \stackrel{\text{def}}{=} -\mathbf{m}_{i_0 j_0}$. It is a coherent DBM.

   Let us define S as $S \stackrel{\text{def}}{=} \{\ (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}) \mid v'_{j_0} - v'_{i_0} = \mathbf{m}_{i_0 j_0}\ \}$ where the v'_k are derived from the v_k by stating that $v'_{2k-1} \stackrel{\text{def}}{=} v_k$ and $v'_{2k} \stackrel{\text{def}}{=} -v_k$. We first prove that $\gamma^{Oct}(\mathbf{m}') = S$.

   - As $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$, there is no cycle with strictly negative weight in $\mathbf{m}$, so, $\mathbf{m}_{j_0 i_0} \geq -\mathbf{m}_{i_0 j_0} = \mathbf{m}'_{j_0 i_0}$ and, similarly $\mathbf{m}_{\overline{i_0}\,\overline{j_0}} \geq -\mathbf{m}_{\overline{j_0}\,\overline{i_0}} = \mathbf{m}'_{\overline{i_0}\,\overline{j_0}}$. This means that $\mathbf{m}' \sqsubseteq^{\mathsf{DBM}} \mathbf{m}$, and so, $\gamma^{Oct}(\mathbf{m}') \subseteq \gamma^{Oct}(\mathbf{m})$.
   - Consider $(v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}')$. Then, $-\mathbf{m}'_{j_0 i_0} \leq v'_{j_0} - v'_{i_0} \leq \mathbf{m}'_{i_0 j_0}$, which, by definition of $\mathbf{m}'$, implies $v'_{j_0} - v'_{i_0} = \mathbf{m}_{i_0 j_0}$. Together with the preceding point, this implies $\gamma^{Oct}(\mathbf{m}') \subseteq S$
   - Conversely, if $(v_1, \ldots, v_n) \in S$, then it is easy to see that $\forall i, j,\ v'_j - v'_i \leq \mathbf{m}'_{ij}$.

   To prove the desired property, it is now sufficient to check that $\gamma^{Oct}(\mathbf{m}')$ is not empty, that is, that $\mathbf{m}'$ has no simple cycle with a strictly negative total weight. Suppose that there exists such a simple cycle $\langle i = k_1, \ldots, k_l = i \rangle$. We distinguish several cases that all lead to an absurdity:

   - If neither of the two modified arcs — from j_0 to i_0 and from $\overline{i_0}$ to $\overline{j_0}$ — are in this strictly negative cycle, this cycle also exists in $\mathcal{G}(\mathbf{m})$ and $\gamma^{Oct}(\mathbf{m}) = \emptyset$, which is not true.
   - Suppose now that the strictly negative cycle contains only one of these two modified arcs say, the arc from j_0 to i_0. It contains this arc only once, as the cycle is simple. By adequately shifting the indexes of the cycle, we can assume the existence of a strictly negative cycle of the form: $\langle k_1 = j_0,\ k_2 = i_0,\ k_3,\ \ldots,\ k_l = j_0 \rangle$, where $\langle k_2 = i_0,\ k_3,\ \ldots,\ k_l = j_0 \rangle$ is a valid path in $\mathcal{G}(\mathbf{m})$. This path is such that
     $$\sum_{x=2}^{l-1} \mathbf{m}_{k_x k_{x+1}} < -\mathbf{m}'_{j_0 i_0} = \mathbf{m}_{i_0 j_0}$$
     that is, there is a path in $\mathbf{m}$ from i_0 to j_0 with a weight strictly smaller than $\mathbf{m}_{i_0 j_0}$, which is absurd because, $\mathbf{m}$ being strongly closed, it is also closed.

- Finally, suppose that the two modified arcs are in this cycle. Each one of them can appear only once, so, we can — without loss of generality — rewrite the cycle as: $\langle k_1 = \overline{j_0}, \ldots, k_a = j_0, k_{a+1} = i_0, \ldots, k_b = \overline{i_0}, k_{b+1} = \overline{j_0} \rangle$, where the sub-paths $\langle k_1 = \overline{j_0}, \ldots, k_a = j_0 \rangle$ and $\langle k_{a+1} = i_0, \ldots, k_b = \overline{i_0} \rangle$ are in $\mathcal{G}(\mathbf{m})$. We then have:

$$\left(\sum_{x=1}^{a-1} \mathbf{m}_{k_x k_{x+1}}\right) + \mathbf{m}'_{j_0 i_0} + \left(\sum_{x=a+1}^{b-1} \mathbf{m}_{k_x k_{x+1}}\right) + \mathbf{m}'_{\overline{i_0}\,\overline{j_0}} < 0 \ .$$

Because $\mathbf{m}$ is strongly closed, it is also closed, and we have:

$$\mathbf{m}_{\overline{j_0} j_0} \leq \sum_{x=1}^{a-1} \mathbf{m}_{k_x k_{x+1}} \quad \text{and} \quad \mathbf{m}_{i_0 \overline{i_0}} \leq \sum_{x=a+1}^{b-1} \mathbf{m}_{k_x k_{x+1}}$$

which combines with the preceding inequality to give:

$$\mathbf{m}_{\overline{j_0} j_0} + \mathbf{m}'_{j_0 i_0} + \mathbf{m}_{i_0 \overline{i_0}} + \mathbf{m}'_{\overline{i_0}\,\overline{j_0}} < 0$$

that is:

$$\mathbf{m}_{i_0 j_0} > (\mathbf{m}_{\overline{j_0} j_0} + \mathbf{m}_{i_0 \overline{i_0}})/2$$

which contradicts the fact that $\mathbf{m}$ is strongly closed.

2. Set a pair (i_0, j_0) such that $\mathbf{m}_{ij} = +\infty$ and $M \in \mathbb{I}$. We denote by $\mathbf{m}'$ the DBM equal to $\mathbf{m}$ except that $\mathbf{m}'_{j_0 i_0} \stackrel{\text{def}}{=} \mathbf{m}'_{\overline{i_0}\,\overline{j_0}} \stackrel{\text{def}}{=} \min(\mathbf{m}_{j_0 i_0}, -M)$. We can prove the same way as in the first point that $\gamma^{Oct}(\mathbf{m}') = \{\ (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}) \mid v'_{j_0} - v'_{i_0} \geq M\ \}$ and $\gamma^{Oct}(\mathbf{m}') \neq \emptyset$.

○

This strong closure property will be used pervasively in our subsequent proofs: it provides a strong link between octagons and their representations.

Best Representation. A first consequence of the saturation property is that, if $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$, then there is a *unique* strongly closed DBM that has the same concretisation as $\gamma^{Oct}(\mathbf{m})$. We denote by $\mathbf{m}^\bullet$ this DBM. By extending $\bullet$ to a full function from $c\mathsf{DBM}$ to $c\mathsf{DBM}$ by stating that $\mathbf{m}^\bullet = \bot^{\mathsf{DBM}}$ whenever $\gamma^{Oct}(\mathbf{m}) = \emptyset$, we obtain the normal form we seek:

Theorem 4.3.3. Best abstraction of octagons.

If $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$,
then $\mathbf{m}^\bullet = (\alpha^{Oct} \circ \gamma^{Oct})(\mathbf{m}) = \inf_{\sqsubseteq^{\mathsf{DBM}}} \{\ \mathbf{n} \in \mathsf{DBM} \mid \gamma^{Oct}(\mathbf{m}) = \gamma^{Oct}(\mathbf{n})\ \}$.

●

Proof. This is an easy consequence of Thm. 4.3.2. ○

4.3.3 Floyd–Warshall Algorithm for Strong Closure

We now present a modified version of the Floyd–Warshall algorithm that uses our two local transformations to compute $\mathbf{m}^\bullet$ in cubic time:

Definition 4.3.2. Floyd–Warshall algorithm for strong closure.

$$(\mathbf{m}^\bullet)_{ij} \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } i = j \\ \mathbf{m}^n_{ij} & \text{if } i \neq j \end{cases}$$

$$\text{where} \quad \mathbf{m}^k \stackrel{\text{def}}{=} \begin{cases} \mathbf{m} & \text{if } k = 0 \\ S(C^{2k-1}(\mathbf{m}^{k-1})) & \text{if } 1 \leq k \leq n \end{cases}$$

$$\text{and} \quad (S(\mathbf{n}))_{ij} \stackrel{\text{def}}{=} \min(\mathbf{n}_{ij}, (\mathbf{n}_{i\bar{i}} + \mathbf{n}_{\bar{j}j})/2)$$

$$\text{and} \quad \left(C^k(\mathbf{n})\right)_{ij} \stackrel{\text{def}}{=} \min (\begin{array}{l} \mathbf{n}_{ij},\ \mathbf{n}_{ik} + \mathbf{n}_{kj},\ \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}j}, \\ \mathbf{n}_{ik} + \mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{k}j},\ \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}k} + \mathbf{n}_{kj} \end{array}) .$$

●

As the classical Floyd–Warshall algorithm, this algorithm performs n steps. Each step computes a new matrix in quadratic time. However, each step now uses two passes: a S pass and a C^k pass. We recognise in S our second local transformation. C^k looks like an inflated version of the classical Floyd–Warshall local transformation: we try and see if a shorter way to go from i to j is to pass through k, but we also try passing through $\overline{k}$, and through k and then $\overline{k}$, and finally through $\overline{k}$ and then k. This increase in complexity as well as the interleaving of S and C^k passes is very important to ensure that the local characterisation of the strong closure is verified for more and more elements, that is, to ensure that what is enforced by a pass is not destroyed later. Unfortunately, there does not seem to exist a simple and intuitive reason for the exact formulas of Def. 4.3.2: this should be considered as a technicality required for the proof of the following Thm. 4.3.4.

Not only does this algorithm compute the strong closure $\mathbf{m}^\bullet$ of any DBM $\mathbf{m}$ that represents a non-empty octagon, but it can also be used to determine whether a DBM represents an empty octagon:

Theorem 4.3.4. Properties of the Floyd–Warshall algorithm for strong closure.

1. $\gamma^{Oct}(\mathbf{m}) = \emptyset \iff \exists i,\ \mathbf{m}^n_{ii} < 0$, *where* $\mathbf{m}^n$ *is defined as in Def. 4.3.2.*
2. *If* $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$ *then* $\mathbf{m}^\bullet$ *computed by Def. 4.3.2 is the strong closure as defined by Def. 4.3.1 and Thm. 4.3.3.*

●

Proof.

As the full proof is quite complex, it is postponed to the appendix, in Sect. A.1, and we give here only a proof sketch.

The first, and easiest, property steams from the fact that each coefficient in the strong closure of the matrix is smaller than the corresponding one in the classical closure and that a strictly negative diagonal coefficient in the classical closure is equivalent to the octagon emptiness. To prove the second property, we need to prove that $\forall i,\ \mathbf{m}^\bullet_{ii} = 0$, $\forall i, j,\ \mathbf{m}^\bullet_{ij} \leq (\mathbf{m}^\bullet_{i\bar{i}} + \mathbf{m}^\bullet_{\bar{j}j})/2$, and $\forall i, j, k,\ \mathbf{m}^\bullet_{ij} \leq \mathbf{m}^\bullet_{ik} + \mathbf{m}^\bullet_{kj}$. The first two properties are easy. The third is quite complex: we prove by induction on o that $\forall 1 \leq k \leq o,\ \forall i, j,\ \mathbf{m}^o_{ij} \leq \mathbf{m}^o_{i\,(2k-1)} + \mathbf{m}^o_{(2k-1)\,j}$ and $\mathbf{m}^o_{ij} \leq \mathbf{m}^o_{i\,(2k)} + \mathbf{m}^o_{(2k)\,j}$. This requires proving that, not only each application of C^k makes the property true for one more line and column k, but also that S applications do not destroy the induction hypothesis.

○

In-Place Version. Finally, it should be noted that it is possible to modify the argument matrix in-place, as we did for the vanilla Floyd–Warshall algorithm in Def. 3.3.3:

Definition 4.3.3. In-place Floyd–Warshall algorithm for strong closure.

```
for k = 1 to n {
  for i = 1 to 2n
    for j = 1 to 2n
      m_ij ← min ( m_ij, m_i(2k-1) + m_(2k-1)j, m_i(2k) + m_(2k)j,
                   m_i(2k-1) + m_(2k-1)(2k) + m_(2k)j,
                   m_i(2k) + m_(2k)(2k-1) + m_(2k-1)j )
  for i = 1 to 2n
    for j = 1 to 2n
      m_ij ← min(m_ij, (m_ii' + m_j'j)/2)
}
for i = 1 to 2n
  if m_ii < 0 return ⊥^DBM else m_ii ← 0

return m
```

●

The computed intermediate matrices are slightly different, but the overall result is the same. Def. 4.3.3 is easier to implement than Def. 4.3.2 but harder to express as a mathematical formula.

4.3.4 Incremental Strong Closure Algorithm

As for the classical Floyd–Warshall algorithm, our modified Floyd–Warshall algorithm features an incremental version that is quite useful to quickly compute the strong closure of an almost strongly closed matrix. Suppose that $\mathbf{m}$ is strongly closed and that $\mathbf{n}_{ij} = \mathbf{m}_{ij}$ when $i, j \leq 2c$. From a constraint point of view, this means that we may only have altered constraints that contain at least one variable in $V_{c+1}, \ldots, V_n$. We use the following algorithm:

Definition 4.3.4. Incremental Floyd–Warshall algorithm for strong closure.

$$(\mathbf{n}^\bullet)_{ij} \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } i = j \\ \mathbf{n}^n_{ij} & \text{if } i \neq j \end{cases}$$

$$\text{where} \quad \mathbf{n}^k \stackrel{\text{def}}{=} \begin{cases} \mathbf{n} & \text{if } k = 0 \\ S'^{2k-1}(C'^{2k-1}(\mathbf{n}^{k-1})) & \text{if } 1 \leq k \leq n \end{cases}$$

$$\text{and} \quad \left(S'^k(\mathbf{n})\right)_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{n}_{ij} & \text{if } i, j, k \leq 2c \\ \min(\mathbf{n}_{ij}, (\mathbf{n}_{i\bar{i}} + \mathbf{n}_{\bar{j}j})/2) & \text{otherwise} \end{cases}$$

$$\text{and} \quad \left(C'^k(\mathbf{n})\right)_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{n}_{ij} & \text{if } i, j, k \leq 2c \\ \min\left(\begin{array}{l} \mathbf{n}_{ij},\ \mathbf{n}_{ik} + \mathbf{n}_{kj},\ \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}j}, \\ \quad \mathbf{n}_{ik} + \mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{k}j},\ \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}k} + \mathbf{n}_{kj}\ \end{array}\right) & \text{otherwise} \end{cases}$$

●

Theorem 4.3.5. Incremental strong closure properties.

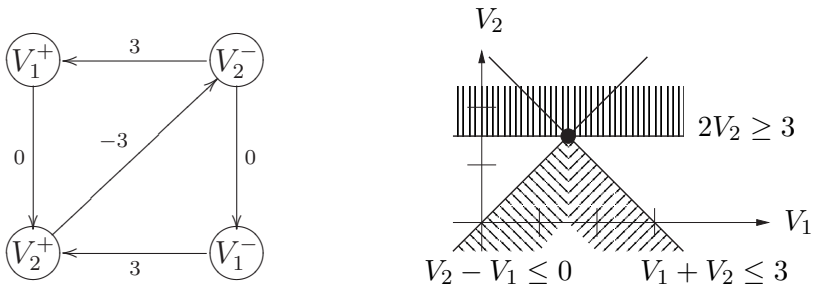
$\mathbf{n}^\bullet$ as computed by the incremental strong closure algorithm of Def. 4.3.4 is equal to $\mathbf{n}^\bullet$ as computed by the vanilla strong closure algorithm of Def. 4.3.2.

●

Proof.

Because $\mathbf{n}_{ij}$ equals $\mathbf{m}_{ij}$ when $i, j \leq 2c$ and $\mathbf{m}$ is strongly closed, Def. 4.3.1 gives $\forall i, j, k \leq 2c$:

- $\mathbf{n}_{ij} \leq \mathbf{n}_{ik} + \mathbf{n}_{kj}$,
- $\mathbf{n}_{ij} \leq \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}j}$ because $\bar{k} \leq 2c$,
- $\mathbf{n}_{kj} \leq \mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{k}j}$, so, $\mathbf{n}_{ij} \leq \mathbf{n}_{ik} + \mathbf{n}_{kj} \leq \mathbf{n}_{ik} + \mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{k}j}$,



(a) (b)

Figure 4.3: A potential graph $\mathcal{G}(\mathbf{m})$ in $\mathbb{Z}$ with no strictly negative cycle (a) and the corresponding octagon (b) $\gamma^{Oct}(\mathbf{m}) = \{(\frac{3}{2}, \frac{3}{2})\}$ which is empty in $\mathbb{Z}^2$.

- likewise $\mathbf{n}_{ij} \leq \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}k} + \mathbf{n}_{kj}$,
- finally, $\bar{\imath}, \bar{\jmath} \leq 2c$, so, $\mathbf{n}_{ij} \leq (\mathbf{n}_{i\bar{\imath}} + \mathbf{n}_{\bar{\jmath}j})/2$.

The incremental closure algorithm simply uses this knowledge to avoid some useless computation in the first c steps of the strong closure algorithm.

○

As the sub-matrix from indexes $(1,1)$ to $(2c, 2c)$ is left unmodified for the first c iterations, we have a time cost proportional to $n^3 - c^3$. By virtually exchanging columns and lines, this algorithm extends to the case where the $n-c$ pairs of modified lines and columns are anywhere in the matrix, not necessarily at the end. We will denote by $Inc^{\bullet}_{i_1,\ldots,i_{n-c}}(\mathbf{n})$ the result of the algorithm when the modified lines and columns correspond to variables V_{i_1} to $V_{i_{n-c}}$. Finally, note that an in-place version of this incremental algorithm, in the spirit of Def. 4.3.3, may be easily designed.

4.3.5 Integer Case

Whenever $\mathbb{I} = \mathbb{Z}$, the emptiness test of Thm. 4.3.1 no longer works, that is, $\gamma^{Pot}(\mathbf{m}) = \emptyset \Longrightarrow \gamma^{Oct}(\mathbf{m}) = \emptyset$ but the converse is not true. Indeed, a conjunction of integer octagonal constraints may have only non-integer solutions, as exemplified in Fig. 4.3, which was not possible for potential or zone constraints. As a consequence, Thms. 4.3.2, 4.3.3, 4.3.4, and 4.3.5 are no longer true. In [JMSY94], Jaffar et al. propose to consider constraint conjunctions that are not only closed by *transitivity* — that is, the addition of two constraints — but also by *tightening*, a new operation that allows deriving the constraint $x \leq \lfloor c/2 \rfloor$ from the constraint $2x \leq c$. They prove that constraint systems closed by transitivity and tightening are satisfiable if and only if no trivially unsatisfiable constraint $0 < 0$ appears in the

system. Later, in [HS97], Harvey and Stuckey propose a practical algorithm to maintain the tightened transitive closure of a constraint system when new constraints are added.

Even though [JMSY94, HS97] are only interested in checking satisfiability and not constructing abstract domains, their ideas can be of use. As for the preceding section, what we need is a way to compute $\alpha^{Oct} \circ \gamma^{Oct}$, that is, a normal form with the saturation property.

Strong Closure with Tightening. We first restate the notion of tightened transitive closure from [JMSY94, HS97] using our encoding of constraint conjunctions as DBMs:

Definition 4.3.5. Tight closure.

A coherent DBM $\mathbf{m}$ *in* $\mathbb{Z}$ *is said to be* tightly closed *if and only if:*

$$\begin{cases} \forall i,j,k, & \mathbf{m}_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj} \\ \forall i,j, & \mathbf{m}_{ij} \leq (\mathbf{m}_{i\bar{\imath}} + \mathbf{m}_{\bar{\jmath}j})/2 \\ \forall i, & \mathbf{m}_{i\bar{\imath}} \text{ is even} \\ \forall i, & \mathbf{m}_{ii} = 0 \end{cases}$$

●

This simply amounts to stating that $\mathbf{m}$ is strongly closed with the extra requirement that elements $\mathbf{m}_{i\bar{\imath}}$, encoding unary constraints of the form $\pm 2V_k \leq c$, are even.

Harvey and Stuckey's Algorithm. Unlike our modified Floyd–Warshall algorithm that is able to compute at once the strong closure of a DBM, or the incremental version that can recover the strong closure after *all* constraints concerning one or more variable(s) have been changed, the algorithm proposed by Harvey and Stuckey in [HS97] can recover the tight closure only if *one constraint* has changed. We now present this algorithm adapted to our DBM notation. Suppose that the coherent DBM $\mathbf{m}$ is equal to a tightly closed DBM, except for the element at position $(i_0 j_0)$ — and, by coherence, the element at position $(\overline{\jmath_0}\,\overline{\imath_0})$. Moreover, suppose that, if $i_0 = \overline{\jmath_0}$, then the changed element $\mathbf{m}_{i_0 j_0}$ is even. The incremental tight closure on $\mathbf{m}$ with respect to the position (i_0, j_0) is denoted by $Inc^T_{i_0 j_0}(\mathbf{m})$ and defined as follows:

Definition 4.3.6. Incremental tight closure algorithm from [HS97].

$$\left(Inc^T_{i_0 j_0}(\mathbf{m})\right)_{ij} \stackrel{\text{def}}{=} \min\left(\mathbf{m}'_{ij}, (\mathbf{m}'_{i\bar{i}} + \mathbf{m}'_{\bar{j}j})/2\right)$$

where

$$\mathbf{m}'_{ij} \stackrel{\text{def}}{=} \begin{cases} \min\left(\mathbf{m}_{ij}, \mathbf{m}_{ii_0} + \mathbf{m}_{i_0 j_0} + \mathbf{m}_{j_0 j}, \mathbf{m}_{i\overline{j_0}} + \mathbf{m}_{\overline{j_0}\,\overline{i_0}} + \mathbf{m}_{\overline{i_0} j}\right) & \text{if } i \neq \bar{j} \\ \min\left(\mathbf{m}_{ij}, \begin{array}{l} 2(\mathbf{m}_{i_0 j_0} + \mathbf{m}_{i\overline{j_0}} + (\mathbf{m}_{\overline{i_0} i_0}/2)), \\ 2(\mathbf{m}_{i_0 j_0} + \mathbf{m}_{ii_0} + (\mathbf{m}_{j_0\overline{j_0}}/2)), \\ 2\lfloor(\mathbf{m}_{ii_0} + \mathbf{m}_{i_0 j_0} + \mathbf{m}_{j_0\bar{i}})/2\rfloor \end{array}\right) & \text{if } i = \bar{j} \end{cases}$$

●

In $Inc^T_{i_0 j_0}$, we first propagate the new constraint $\mathbf{m}_{i_0 j_0}$ to obtain new unary and binary constraints in $\mathbf{m}'$. Note that the terms $2(\mathbf{m}_{i_0 j_0} + \mathbf{m}_{i\overline{j_0}} + (\mathbf{m}_{\overline{i_0} i_0}/2))$ and $2(\mathbf{m}_{i_0 j_0} + \mathbf{m}_{ii_0} + (\mathbf{m}_{j_0\overline{j_0}}/2))$ correspond respectively to the sum along the paths $\langle i, \overline{j_0}, \overline{i_0}, i_0, j_0, \bar{i}\rangle$ and $\langle i, i_0, j_0, \overline{j_0}, \overline{i_0}, \bar{i}\rangle$. We use *tightening* for the third derived unary constraint: $\mathbf{m}'_{i\bar{i}} \leq 2\lfloor(\mathbf{m}_{ii_0} + \mathbf{m}_{i_0 j_0} + \mathbf{m}_{j_0\bar{i}})/2\rfloor$. In $\mathbf{m}'$, all the derived matrix coefficients corresponding to unary constraints are even. Finally, the new unary constraints are combined to derive new binary constraints: $(Inc^T_{i_0 j_0}(\mathbf{m}))_{ij} \leq (\mathbf{m}'_{i\bar{i}} + \mathbf{m}'_{\bar{j}j})/2$. Whenever the result of a division by 2 is not fed to the floor operator $\lfloor\cdot\rfloor$, this means that the division cannot produce half-integers.

We now prove that this algorithm indeed allows testing for emptiness, and finding a saturated DBM when $\mathbb{I} = \mathbb{Z}$:

Theorem 4.3.6. Incremental tight closure properties.

Let $\mathbf{m}$ *be a coherent tightly closed DBM, and* $\mathbf{n}$ *a coherent DBM equal to* $\mathbf{m}$ *except at positions* (i_0, j_0) *and* $(\overline{j_0}, \overline{i_0})$*, then:*

1. $\gamma^{Oct}(\mathbf{n}) = \emptyset \iff \exists i, (Inc^T_{i_0 j_0}(\mathbf{n}))_{ii} < 0$.
2. *If* $\gamma^{Oct}(\mathbf{n}) \neq \emptyset$*, then* $Inc^T_{i_0 j_0}(\mathbf{n})$*, with diagonal elements set to 0, is tightly closed.*

●

Proof.

1. This is a restatement of Thm. 2 from [JMSY94], also recalled as Thm. 1 in [HS97].
2. This is a restatement of Thm. 2 from [HS97].

○

Theorem 4.3.7. Saturation property.

If $\mathbf{m}$ *is tightly closed, then:*

1. $\forall i, j$, *if* $\mathbf{m}_{ij} < +\infty$, *then* $\exists (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m})$ *such that* $v'_j - v'_i = \mathbf{m}_{ij}$.

2. $\forall i, j,$ *if* $\mathbf{m}_{ij} = +\infty,$ *then* $\forall M < +\infty, \exists (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m})$ *such that* $v'_j - v'_i \geq M$.

where the v'_k *are derived from the* v_k *by* $v'_{2k-1} \stackrel{\text{def}}{=} v_k$ *and* $v'_{2k} \stackrel{\text{def}}{=} -v_k$.

●

Proof.

1. Set a pair (i_0, j_0). Let us consider a DBM $\mathbf{n}$ equal to $\mathbf{m}$ except that $\mathbf{n}_{j_0 i_0} \stackrel{\text{def}}{=} \mathbf{n}_{\overline{i_0}\,\overline{j_0}} \stackrel{\text{def}}{=} -\mathbf{m}_{i_0 j_0}$. We can prove as in Thm. 4.3.2 that $\gamma^{Oct}(\mathbf{n}) = \{\ (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}) \mid v'_{j_0} - v'_{i_0} = \mathbf{m}_{i_0 j_0}\ \}$, and so, proving the desired property reduces to proving that $\gamma^{Oct}(\mathbf{n}) \neq \emptyset$.

   Suppose that $\gamma^{Oct}(\mathbf{n}) = \emptyset$. As $\mathbf{m}$ is tightly closed, $\mathbf{n}$ can be tightly closed by one application of the incremental tight closure algorithm, at position (j_0, i_0). We will denote by $\mathbf{n}''$ the matrix $Inc^T_{j_0 i_0}(\mathbf{n})$, and by $\mathbf{n}'$ the intermediate matrix computed in Def. 4.3.6. By Thm. 4.3.6.1, we have $\exists i,\ \mathbf{n}''_{ii} < 0$. Several cases can occur, each one of them leading to an absurdity:

   - Suppose that $\mathbf{n}''_{ii} = \mathbf{n}'_{ii}$. This means that $\min(\mathbf{n}_{ii},\ \mathbf{n}_{ij_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0 i},\ \mathbf{n}_{i\overline{i_0}} + \mathbf{n}_{\overline{i_0}\,\overline{j_0}} + \mathbf{n}_{\overline{j_0} i}) < 0$. Thus, one of the three following cases occurs: either $0 > \mathbf{n}_{ii} = \mathbf{m}_{ii}$, $0 > \mathbf{n}_{ij_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0 i} = \mathbf{m}_{ij_0} - \mathbf{m}_{i_0 j_0} + \mathbf{m}_{i_0 i}$, or $0 > \mathbf{n}_{i\overline{i_0}} + \mathbf{n}_{\overline{i_0}\,\overline{j_0}} + \mathbf{n}_{\overline{j_0} i} = \mathbf{m}_{i\overline{i_0}} - \mathbf{m}_{\overline{j_0}\,\overline{i_0}} + \mathbf{m}_{\overline{j_0} i}$. Each inequality contradicts the fact that $\mathbf{m}$ is closed.
   - If $\mathbf{n}''_{ii} = (\mathbf{n}'_{i\bar{i}} + \mathbf{n}'_{\bar{i}i})/2$, there are many cases depending on the value of $\mathbf{n}'_{i\bar{i}}$ and $\mathbf{n}'_{\bar{i}i}$.

     Suppose that no tightening is used to derive $\mathbf{n}'_{i\bar{i}}$ nor $\mathbf{n}'_{\bar{i}i}$, that is, $\mathbf{n}'_{i\bar{i}} \in \{\mathbf{n}_{i\bar{i}},\ 2(\mathbf{n}_{j_0 i_0} + \mathbf{n}_{i\overline{i_0}} + (\mathbf{n}_{\overline{j_0} j_0}/2)),\ 2(\mathbf{n}_{j_0 i_0} + \mathbf{n}_{ij_0} + (\mathbf{n}_{i_0\overline{i_0}}/2)),\ \mathbf{n}_{ij_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0\bar{i}}\}$ and $\mathbf{n}'_{\bar{i}i} \in \{\mathbf{n}_{\bar{i}i},\ 2(\mathbf{n}_{j_0 i_0} + \mathbf{n}_{\bar{i}\,\overline{i_0}} + (\mathbf{n}_{\overline{j_0} j_0}/2)),\ 2(\mathbf{n}_{j_0 i_0} + \mathbf{n}_{\bar{i}j_0} + (\mathbf{n}_{i_0\overline{i_0}}/2)),\ \mathbf{n}_{\bar{i}j_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0 i}\}$. This can be rewritten as: $\mathbf{n}'_{i\bar{i}} \in \{\mathbf{n}_{i\bar{i}},\ \mathbf{n}_{i\overline{i_0}} + \mathbf{n}_{\overline{i_0}\,\overline{j_0}} + \mathbf{n}_{\overline{j_0} j_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0\bar{i}},\ \mathbf{n}_{ij_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0\overline{i_0}} + \mathbf{n}_{\overline{i_0}\,\overline{j_0}} + \mathbf{n}_{\overline{j_0}\bar{i}},\ \mathbf{n}_{ij_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0\bar{i}}\}$ and $\mathbf{n}'_{\bar{i}i} \in \{\mathbf{n}_{\bar{i}i},\ \mathbf{n}_{\bar{i}\,\overline{i_0}} + \mathbf{n}_{\overline{i_0}\,\overline{j_0}} + \mathbf{n}_{\overline{j_0} j_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0 i},\ \mathbf{n}_{\bar{i}j_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0\overline{i_0}} + \mathbf{n}_{\overline{i_0}\,\overline{j_0}} + \mathbf{n}_{\overline{j_0} i},\ \mathbf{n}_{\bar{i}j_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0 i}\}$. Then, $\mathbf{n}'_{i\bar{i}} + \mathbf{n}'_{\bar{i}i}$ can be expressed as the sum along a cycle in $\mathbf{m}$ that passes exactly zero times, once, or twice through $\mathbf{n}_{j_0 i_0} = -\mathbf{m}_{i_0 j_0}$. If it does not pass through $\mathbf{n}_{j_0 i_0}$, then we have a cycle in $\mathbf{m}$ with a strictly negative weight, which is absurd because $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$. If it passes once, then $\mathbf{m}_{i_0 j_0}$ is strictly greater than the sum along a path from i_0 to j_0 in $\mathbf{m}$, which is also absurd because $\mathbf{m}$ is closed. If it passes twice, then $2\mathbf{m}_{i_0 j_0}$ is strictly greater than the sum along *two* paths from i_0 to j_0 in $\mathbf{m}$, which is only possible if $\mathbf{m}_{i_0 j_0}$ is strictly greater than the sum along at least one of them, which is also absurd.

     Suppose now that $\mathbf{n}_{ij_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0\bar{i}} = 2k+1$ is odd and, by tightening, $\mathbf{n}'_{i\bar{i}} = 2k$, but no tightening is involved in the computation of $\mathbf{n}'_{\bar{i}i}$. Suppose, in our first

sub-case, that $\mathbf{n}'_{\bar{\imath}i} = \mathbf{n}_{\bar{\imath}i}$. Then, we have $(\mathbf{n}_{ij_0} + \mathbf{n}_{j_0i_0} + \mathbf{n}_{i_0\bar{\imath}} - 1) + \mathbf{n}_{\bar{\imath}i} < 0$, that is, $\mathbf{m}_{i_0j_0} + 1 > \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath}i} + \mathbf{m}_{ij_0}$. As $\mathbf{m}_{i_0j_0} \leq \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath}i} + \mathbf{m}_{ij_0}$ by closure of $\mathbf{m}$, we must have $\mathbf{m}_{i_0j_0} = \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath}i} + \mathbf{m}_{ij_0}$. By hypothesis, $2k+1 = \mathbf{m}_{ij_0} - \mathbf{m}_{i_0j_0} + \mathbf{m}_{i_0\bar{\imath}}$, and so, $\mathbf{m}_{\bar{\imath}i} = -2k-1$. This is absurd: by tightness of $\mathbf{m}$, $\mathbf{m}_{\bar{\imath}i}$ cannot be odd. Our second sub-case is $\mathbf{n}'_{\bar{\imath}i} \neq \mathbf{n}_{\bar{\imath}i}$. Then, $\mathbf{n}'_{i\bar{\imath}} + \mathbf{n}'_{\bar{\imath}i}$ can be expressed as the sum *minus one* along a cycle in $\mathbf{n}$ that passes exactly twice through $\mathbf{n}_{j_0i_0} = -\mathbf{m}_{i_0j_0}$. Thus, $2\mathbf{m}_{i_0j_0} + 1$ is strictly greater than the sum along two paths from i_0 to j_0. Moreover, this sum along the two paths is *odd*, and so, the weight of the these two paths cannot be the same and $2\mathbf{m}_{i_0j_0}$ is strictly greater than twice the weight of the path with smallest weight. We thus have proved that $\mathbf{m}_{i_0j_0}$ is strictly greater than the weight of a path from i_0 to j_0 in $\mathbf{m}$, which is absurd. The situation where tightening is used for $\mathbf{n}'_{\bar{\imath}i}$ but not for $\mathbf{n}'_{i\bar{\imath}}$ is similar.

For our last case, we suppose that tightening is used in both $\mathbf{n}'_{i\bar{\imath}}$ and $\mathbf{n}'_{\bar{\imath}i}$, that is, $\mathbf{n}'_{i\bar{\imath}} = 2k$ and $\mathbf{n}'_{\bar{\imath}i} = 2l$ where $\mathbf{n}_{ij_0} + \mathbf{n}_{j_0i_0} + \mathbf{n}_{i_0\bar{\imath}} = 2k+1$ and $\mathbf{n}_{\bar{\imath}j_0} + \mathbf{n}_{j_0i_0} + \mathbf{n}_{i_0i} = 2l+1$. This means, in particular, that $\mathbf{m}_{i_0j_0} = \mathbf{m}_{ij_0} + \mathbf{m}_{i_0\bar{\imath}} - (2k+1) = \mathbf{m}_{\bar{\imath}j_0} + \mathbf{m}_{i_0i} - (2l+1)$, that is, $\mathbf{m}_{ij_0} + \mathbf{m}_{i_0\bar{\imath}}$ and $\mathbf{m}_{\bar{\imath}j_0} + \mathbf{m}_{i_0i}$ are either both odd, or both even. Our hypothesis $\mathbf{n}'_{i\bar{\imath}} + \mathbf{n}'_{\bar{\imath}i} < 0$ can be rewritten as: $2\mathbf{m}_{i_0j_0} + 2 > \mathbf{m}_{ij_0} + \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath}j_0} + \mathbf{m}_{i_0i}$. As, by closure, $2\mathbf{m}_{i_0j_0} \leq \mathbf{m}_{ij_0} + \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath}j_0} + \mathbf{m}_{i_0i}$ and $\mathbf{m}_{ij_0} + \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath}j_0} + \mathbf{m}_{i_0i}$ is even, we have $2\mathbf{m}_{i_0j_0} = \mathbf{m}_{ij_0} + \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath}j_0} + \mathbf{m}_{i_0i}$. If we had $\mathbf{m}_{i_0i} + \mathbf{m}_{ij_0} \neq \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath}j_0}$, we would have $\mathbf{m}_{i_0j_0} > \min(\mathbf{m}_{i_0i} + \mathbf{m}_{ij_0}, \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath}j_0})$, which is absurd because $\mathbf{m}$ is closed. We can now suppose that $\mathbf{m}_{i_0i} + \mathbf{m}_{ij_0} = \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath}j_0}$. This implies that $\mathbf{m}_{i_0j_0} = \mathbf{m}_{i_0i} + \mathbf{m}_{ij_0} = \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath}j_0}$. On the one hand, $(2k+1) = \mathbf{m}_{ij_0} - \mathbf{m}_{i_0j_0} + \mathbf{m}_{i_0\bar{\imath}} = \mathbf{m}_{i_0\bar{\imath}} - \mathbf{m}_{i_0i}$. On the other hand, $(2l+1) = \mathbf{m}_{\bar{\imath}j_0} - \mathbf{m}_{i_0j_0} + \mathbf{m}_{i_0i} = \mathbf{m}_{i_0i} - \mathbf{m}_{i_0\bar{\imath}}$. So, $k = -l$ and $\mathbf{n}'_{i\bar{\imath}} + \mathbf{n}'_{\bar{\imath}i} = 2(k+l) = 0$, which is in contradiction with $\mathbf{n}'_{i\bar{\imath}} + \mathbf{n}'_{\bar{\imath}i} < 0$.

2. The second point can be proved almost as the first one, except that $\mathbf{n}$ is constructed by changing $\mathbf{m}$'s elements (j_0, i_0) and $(\overline{i_0}, \overline{j_0})$ into $\mathbf{n}_{j_0i_0} \stackrel{\text{def}}{=} \mathbf{n}_{\overline{i_0}\,\overline{j_0}} \stackrel{\text{def}}{=} \min(\mathbf{m}_{j_0i_0}, -M)$. We can then prove that $\gamma^{Oct}(\mathbf{n}) = \{\ (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}) \mid v'_{j_0} - v'_{i_0} \geq M\ \}$ and $\gamma^{Oct}(\mathbf{n}) \neq \emptyset$.

○

Cost Considerations. The incremental tight closure algorithm has a $\mathcal{O}(n^2)$ cost. In order to get the tight closure of an arbitrary DBM $\mathbf{m}$, we must start from $\top^{\mathsf{DBM}}$ and add all the constraints $\mathbf{m}_{ij}$ one by one and perform an incremental tight closure Inc^T_{ij} after each addition. This leads to a $\mathcal{O}(n^4)$ total cost while our strong closure algorithm had a $\mathcal{O}(n^3)$ cost. It is not known to the author whether a better cost than $\mathcal{O}(n^4)$ can be achieved. Nothing is less certain as many equation systems are strictly more difficult to solve on

integers than on rationals or reals.

If time cost is a concern, one may consider using the strong closure algorithm where the S pass has been changed into:

$$(S(\mathbf{n}))_{ij} \stackrel{\text{def}}{=} \min(\mathbf{n}_{ij}, \lfloor(\mathbf{n}_{i\bar{\imath}} + \mathbf{n}_{\bar{\jmath}j})/2\rfloor)$$

or, better, into:

$$(S(\mathbf{n}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \min(\mathbf{n}_{ij}, \lfloor\mathbf{n}_{i\bar{\imath}}/2\rfloor + \lfloor\mathbf{n}_{\bar{\jmath}j}/2\rfloor) & \text{if } i \neq \bar{\jmath} \\ 2\lfloor\mathbf{n}_{ij}/2\rfloor & \text{if } i = \bar{\jmath} \end{cases}$$

which is more precise and ensures that unary constraints are tight. The two resulting strong closure algorithms indeed return DBMs $\mathbf{m}^\bullet$ in $\mathbb{Z}$, such that $\gamma^{Oct}(\mathbf{m}^\bullet) = \gamma^{Oct}(\mathbf{m})$, which are much smaller than $\mathbf{m}$ with respect to $\sqsubseteq^{\mathsf{DBM}}$. However, they do not return the *smallest* one as none of the modified S functions preserve the transitive closure property enforced by the C steps. As a consequence, the saturation property is not verified. This will affect most of the operators and transfer functions that will be presented in the following: our inclusion and equality tests will become incomplete — they can fail to detect that $\gamma^{Oct}(\mathbf{m}) \subseteq \gamma^{Oct}(\mathbf{n})$ or $\gamma^{Oct}(\mathbf{m}) = \gamma^{Oct}(\mathbf{m})$ — and our abstractions for the union and the forget operators — among others — will not be the best ones. They will *remain sound* in every situation, but they will not be as precise as they might be: this loosely amounts to abstracting integers as rationals by forgetting their "integerness property", something which is commonly done in the polyhedron abstract domain.[1] Whether to choose the strong closure or the tight closure when $\mathbb{I} = \mathbb{Z}$ is just another cost versus precision trade-off.

In practice, we have chosen to gain time and use the strong closure; we have yet to find a real-life example where the tight closure is needed to prove a meaningful invariant. In the following, we will focus on properties of strongly closed matrices on $\mathbb{Q}$ and $\mathbb{R}$, and leave implicit the fact that most of these properties are also true for tightened matrices on $\mathbb{Z}$.

4.4 Operators and Transfer Functions

We now present our operators and transfer functions for the octagon domain. Except for the transfer functions that use the polyhedron domain internally, the most time-consuming operation they use is a call to the strong closure algorithm to get the normal form of their arguments, and so, they are cubic in the worst case.

[1] More precisely, we use *some* but not all properties of integers when rounding our matrix coefficients while the polyhedron domain uses none. As a consequence, some constraints derived by the octagon domain might be more precise when working on integers than if we used the polyhedron domain.

4.4.1 Adapted Set-Theoretic Operators

The situation is very similar to what happened in the zone abstract domain except that we must use the strong closure instead of the regular closure to get complete tests and a best union abstraction. As before, the intersection does not need strongly closed arguments but does not preserve the strong closure, while this is exactly the converse for the union abstraction:

Definition 4.4.1. Set-theoretic operators on octagons.

$$\mathbf{m} \cap^{Oct} \mathbf{n} \stackrel{\text{def}}{=} \mathbf{m} \sqcap^{\mathsf{DBM}} \mathbf{n} \ .$$

$$\mathbf{m} \cup^{Oct} \mathbf{n} \stackrel{\text{def}}{=} (\mathbf{m}^\bullet) \sqcup^{\mathsf{DBM}} (\mathbf{n}^\bullet) \ .$$

●

Theorem 4.4.1. Properties of set-theoretic operators on octagons.

1. $\gamma^{Oct}(\mathbf{m} \cap^{Oct} \mathbf{n}) = \gamma^{Oct}(\mathbf{m}) \cap \gamma^{Oct}(\mathbf{n})$. *(exact abstraction)*
2. $\gamma^{Oct}(\mathbf{m} \cup^{Oct} \mathbf{n}) = \inf_{\subseteq}\{ S \in Oct \mid S \supseteq \gamma^{Oct}(\mathbf{m}) \cup \gamma^{Oct}(\mathbf{n}) \}$. *(best abstraction)*
3. $\mathbf{m} \cup^{Oct} \mathbf{n}$ *is strongly closed.*
4. $\mathbf{m}^\bullet = \mathbf{n}^\bullet \iff \gamma^{Oct}(\mathbf{m}) = \gamma^{Oct}(\mathbf{n})$.
5. $\mathbf{m}^\bullet \sqsubseteq^{\mathsf{DBM}} \mathbf{n} \iff \gamma^{Oct}(\mathbf{m}) \subseteq \gamma^{Oct}(\mathbf{n})$.

●

Proof. Thanks to the saturation property of Thm. 4.3.2, the proofs are very similar to that of Thms. 3.4.1, 3.4.2, and 3.4.1 for the zone abstract domain. ○

Comparison with Previous Work. The idea of computing the point-wise minimum and maximum of upper bounds to compute, respectively, the union and intersection of octagons is already present in the work of Balasundaram and Kennedy [BK89]. Although the authors remark that the bounds should be the tightest possible for the union to be precise while the result of an intersection may have loose bounds, they do not propose any way to actually "tighten" them. Our main contribution in this respect is the introduction of the strong closure algorithm to enforce tight bounds and achieve practical best precision results.

4.4.2 Adapted Forget Operator

A straightforward adaptation of our first forget operator on zones — Def. 3.6.1 — to octagons is to put $+\infty$ elements in the *two* lines and columns corresponding to the constraints containing the variable to forget:

Definition 4.4.2. Forget operator on octagons $\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Oct}$.

$$(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Oct}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} & \text{if } i \neq 2f-1, 2f \text{ and } j \neq 2f-1, 2f \\ 0 & \text{if } i = j = 2f-1 \text{ or } i = j = 2f \\ +\infty & \text{otherwise} \end{cases}$$

●

Note that this operator can be derived from the zone forget operator as follows:

$$\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Oct} = \{\!\!\{ V'_{2f-1} \leftarrow ? \}\!\!\}^{Zone} \circ \{\!\!\{ V'_{2f} \leftarrow ? \}\!\!\}^{Zone} \ .$$

This operator is always sound. However, it is exact only when the argument is strongly closed. Otherwise, it can lose definitively some implicit constraints. Moreover, this operator preserves the strong closure:

Theorem 4.4.2. Soundness and exactness of $\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Oct}$.

1. $\gamma^{Oct}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Oct}(\mathbf{m})) \supseteq \{ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Oct}(\mathbf{m}) \}$.
2. $\gamma^{Oct}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Oct}(\mathbf{m}^\bullet)) = \{ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Oct}(\mathbf{m}) \}$.
3. $\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Oct}(\mathbf{m})$ *is strongly closed whenever* $\mathbf{m}$ *is.*

●

Proof.

1. We have actually proved in Thm. 3.6.1.1 that $\gamma^{Pot}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Zone}(\mathbf{m})) \supseteq \{ \vec{v}' \in \mathbb{I}^{2n} \mid \exists t \in \mathbb{I},\ \vec{v}'[V'_f \mapsto t] \in \gamma^{Pot}(\mathbf{m}) \}$. Using this with the fact that $\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Oct} = \{\!\!\{ V'_{2f-1} \leftarrow ? \}\!\!\}^{Zone} \circ \{\!\!\{ V'_{2f} \leftarrow ? \}\!\!\}^{Zone}$, we get $\gamma^{Pot}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Oct}(\mathbf{m})) \supseteq \{ \vec{v}' \in \mathbb{I}^{2n} \mid \exists t, t' \in \mathbb{I},\ \vec{v}'[V'_{2f-1} \mapsto t,\ V'_{2f} \mapsto t'] \in \gamma^{Pot}(\mathbf{m}) \}$ (1).

   Let us take $\vec{v} = (v_1, \ldots, v_n) \in \mathbb{I}^n$ such that for some $t \in \mathbb{I}$, $\vec{v}[V_f \mapsto t] \in \gamma^{Oct}(\mathbf{m})$. Let us denote $\vec{v}' \stackrel{\text{def}}{=} (v_1, -v_1, \ldots, v_n, -v_n)$. By definition of γ^{Oct}, we have that $\vec{v}'[V'_{2f-1} \mapsto t, V'_{2f} \mapsto -t] \in \gamma^{Pot}(\mathbf{m})$. By (1), this implies that $\vec{v}' \in \gamma^{Pot}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Oct}(\mathbf{m}))$, so, $\vec{v} \in \gamma^{Oct}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Oct}(\mathbf{m}))$.

2. Firstly, the property is obvious if $\mathbf{m}^\bullet = \bot^{\mathsf{DBM}}$, that is, $\gamma^{Oct}(\mathbf{m}) = \emptyset$, so, we will consider only the case where $\mathbf{m}^\bullet \neq \bot^{\mathsf{DBM}}$. By the first point and the fact that $\gamma^{Oct}(\mathbf{m}^\bullet) = \gamma^{Oct}(\mathbf{m})$, we get the first part of the equality: $\gamma^{Oct}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Oct}(\mathbf{m}^\bullet)) \supseteq \{ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Oct}(\mathbf{m}^\bullet) \} = \{ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Oct}(\mathbf{m}) \}$.

   For the converse inequality, let us take $\vec{v} = (v_1, \ldots, v_n) \in \gamma^{Oct}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Oct}(\mathbf{m}^\bullet))$. We want to prove that there exists a t such that $\vec{v}[V_f \mapsto t] \in \gamma^{Oct}(\mathbf{m})$. As in Thm. 3.6.1, we first prove that

$$\begin{array}{l}\max\{\ v'_j - \mathbf{m}^\bullet_{(2f-1)\,j},\ -\mathbf{m}^\bullet_{(2f-1)\,(2f)}/2 \mid j \neq 2f-1,\ 2f\ \} \quad \leq \\ \min\{\ \mathbf{m}^\bullet_{i\,(2f-1)} + v'_i,\ \mathbf{m}^\bullet_{(2f)\,(2f-1)}/2 \mid i \neq 2f-1,\ 2f\ \}\end{array}$$

where, as usual, $v'_{2i-1} \stackrel{\text{def}}{=} v_i$ and $v'_{2i} \stackrel{\text{def}}{=} -v_i$.

Suppose that this is not true; this can only be for one of the following reasons, each one of them leading to an absurdity:

- Either $\exists i, j \neq 2f-1,\ 2f$ such that $v'_j - \mathbf{m}^\bullet_{(2f-1)\,j} > \mathbf{m}^\bullet_{i\,(2f-1)} + v'_i$, which means that $v'_j - v'_i > \mathbf{m}^\bullet_{i\,(2f-1)} + \mathbf{m}^\bullet_{(2f-1)\,j} \geq \mathbf{m}^\bullet_{ij}$ by closure of $\mathbf{m}^\bullet$. This contradicts the fact that $v'_j - v'_i \leq (\{\!\{ V_f \leftarrow ? \}\!\}^{Oct}(\mathbf{m}^\bullet))_{ij} = \mathbf{m}^\bullet_{ij}$ when $i, j \neq 2f-1,\ 2f$.
- Either $\exists j \neq 2f-1,\ 2f$ such that $v'_j - \mathbf{m}^\bullet_{(2f-1)\,j} > \mathbf{m}^\bullet_{(2f)\,(2f-1)}/2$, which means that $2v'_j > \mathbf{m}^\bullet_{(2f)\,(2f-1)} + 2\mathbf{m}^\bullet_{(2f-1)\,j} = \mathbf{m}^\bullet_{\bar{\jmath}\,(2f)} + \mathbf{m}^\bullet_{(2f)\,(2f-1)} + \mathbf{m}^\bullet_{(2f-1)\,j} \geq \mathbf{m}^\bullet_{\bar{\jmath}j}$ by closure of $\mathbf{m}^\bullet$. This is also impossible.
- The situation would be similar if we had $\exists i \neq 2f-1,\ 2f$ such that $\mathbf{m}^\bullet_{i\,(2f-1)} + v'_i > -\mathbf{m}^\bullet_{(2f-1)\,(2f)}/2$.
- The last possibility is to have $-\mathbf{m}^\bullet_{(2f-1)\,(2f)}/2 > \mathbf{m}^\bullet_{(2f)\,(2f-1)}/2$, which would imply $\mathbf{m}^\bullet_{(2f)\,(2f-1)} + \mathbf{m}^\bullet_{(2f-1)\,(2f)} < 0$, and so, $\gamma^{Oct}(\mathbf{m}) = \emptyset$, which contradicts our hypothesis.

So, there exists at least one $t \in \mathbb{I}$ such that:

$$\begin{array}{ccccc}\max_{j\neq 2f-1,2f}\left(v'_j - \mathbf{m}^\bullet_{(2f-1)\,j}\right) & \leq & t & \leq & \min_{i\neq 2f-1,2f}\left(v'_i + \mathbf{m}^\bullet_{i\,(2f-1)}\right) \\ -\mathbf{m}^\bullet_{(2f-1)\,(2f)}/2 & \leq & t & \leq & \mathbf{m}^\bullet_{(2f)\,(2f-1)}/2\end{array}$$

We now prove that any such a t is a good choice, *i.e.*, $\vec{v}[V_f \mapsto t] \in \gamma^{Oct}(\mathbf{m})$. We will denote $(v_1, -v_1, \ldots, v_{f-1}, -v_{f-1}, t, -t, v_{f+1}, -v_{f+1}, \ldots, v_n, -v_n)$ by $\vec{w}\,'$, and by w'_k its $k-$th coordinate. We only need to prove that $\forall i, j,\ w'_j - w'_i \leq \mathbf{m}^\bullet_{ij}$:

- If $i \neq 2f-1,\ 2f$ and $j \neq 2f-1,\ 2f$, then $w'_j - w'_i = v'_j - v'_i \leq (\{\!\{ V_f \leftarrow ? \}\!\}^{Oct}(\mathbf{m}^\bullet))_{ij} = \mathbf{m}^\bullet_{ij}$.
- If $i = j = 2f-1$ or $i = j = 2f$, then $w'_j - w'_i = 0 = \mathbf{m}^\bullet_{ij}$.
- If $i = 2f-1$ and $j \neq 2f-1,\ 2f$, $w'_j - w'_i = v'_j - t \leq \mathbf{m}^\bullet_{ij}$ because $t \geq \max_{j\neq 2f-1,2f}\left(v'_j - \mathbf{m}^\bullet_{(2f-1)\,j}\right)$.
- If $j = 2f-1$ and $i \neq 2f-1,\ 2f$, $w'_j - w'_i = t - v'_i \leq \mathbf{m}^\bullet_{ij}$ because $t \leq \min_{i\neq 2f-1,2f}\left(v'_i + \mathbf{m}^\bullet_{i\,(2f-1)}\right)$.
- If $i = 2f$ and $j \neq 2f-1,\ 2f$, $w'_j - w'_i = v'_j + t \leq \mathbf{m}^\bullet_{\bar{\jmath}\,(2f-1)} = \mathbf{m}^\bullet_{ij}$ using the case where $j = 2f-1$ and the coherence.

- If $j = 2f$ and $i \neq 2f-1, 2f$, $w'_j - w'_i = -t - v'_i \leq \mathbf{m}^{\bullet}_{(2f-1)\,\bar{\imath}} = \mathbf{m}^{\bullet}_{ij}$ using the case where $i = 2f-1$ and the coherence.
- When $j = \bar{\imath} = 2f-1$, $w'_j - w'_i = 2t \leq \mathbf{m}^{\bullet}_{(2f)\,(2f-1)}$ by definition of t. The case $i = \bar{\jmath} = 2f-1$ is similar.

3. Suppose that $\mathbf{m}$ is strongly closed and let us denote by $\mathbf{m}'$ the matrix $\{\!|\, V_f \leftarrow ? \,|\!\}^{Oct}(\mathbf{m})$. Then, $\mathbf{m}$ is also closed and, by Thm. 3.6.1.3, we deduce that $\mathbf{m}'$ is closed. To prove its strong closure, we only have to prove that $\forall i, j,\ \mathbf{m}'_{ij} \leq (\mathbf{m}'_{i\bar{\imath}} + \mathbf{m}'_{\bar{\jmath}j})/2$:
   - If $i \neq 2f-1, 2f$ and $j \neq 2f-1, 2f$, then $\mathbf{m}'_{ij} = \mathbf{m}_{ij}$, $\mathbf{m}'_{i\bar{\imath}} = \mathbf{m}_{i\bar{\imath}}$ and $\mathbf{m}'_{\bar{\jmath}j} = \mathbf{m}_{\bar{\jmath}j}$, so, the property is a consequence of $\mathbf{m}$ being strongly closed.
   - If $i = 2f-1$ or $i = 2f$ or $j = 2f-1$ or $j = 2f$, then at least one of $\mathbf{m}'_{i\bar{\imath}}$ and $\mathbf{m}'_{\bar{\jmath}j}$ is $+\infty$, so $(\mathbf{m}'_{i\bar{\imath}} + \mathbf{m}'_{\bar{\jmath}j})/2 = +\infty \geq \mathbf{m}'_{ij}$.

○

We do not present here any alternate forget operator, such as the one that was proposed on zones in Def. 3.6.2, that is exact even when the argument matrix is not strongly closed. We will simply remark that the straightforward definition:

$$\{\!|\, V'_{2f-1} \leftarrow ? \,|\!\}^{Zone}_{alt} \circ \{\!|\, V'_{2f} \leftarrow ? \,|\!\}^{Zone}_{alt}$$

gives an upper approximation of the forget operator that is not always exact. Designing an exact alternate forget operator on octagons would thus require some more work.

4.4.3 Adapted Conversion Operators

Conversions from intervals to octagons and from octagons to polyhedra are both straightforward and exact. Thus, we focus here only on the conversions from octagons to intervals and from polyhedra to octagons, which differ only slightly from the corresponding operators introduced in Sect. 3.5.2 on the zone abstract domain.

From Octagons to Intervals. A straightforward application of the saturation property of the strong normal form is the ability to easily project an octagon $\mathbf{m}$ along a variable V_i to get an interval, denoted by $\pi_i(\mathbf{m})$ and defined as follows:

Definition 4.4.3. Projection operator π_i.

$$\pi_i(\mathbf{m}) \stackrel{\text{def}}{=} \begin{cases} \bot^{Int}_{\mathcal{B}} & \text{if } \mathbf{m}^{\bullet} = \bot^{\mathsf{DBM}} \\ [-\mathbf{m}^{\bullet}_{(2i-1)\,(2i)}/2,\ \mathbf{m}^{\bullet}_{(2i)\,(2i-1)}/2] & \text{if } \mathbf{m}^{\bullet} \neq \bot^{\mathsf{DBM}} \end{cases}$$

is such that $\gamma^{Int}(\pi_i(\mathbf{m})) = \{\ v \in \mathbb{I} \mid \exists (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}),\ v_i = v\ \}$.

●

As in Sect. 3.5.2, an interval abstract domain element is obtained by projecting each variable independently. If we do not use the strong normal form of $\mathbf{m}$ in each π_i, we get sound intervals that are not as tight as possible; otherwise, we get the best abstraction $\mathit{Int}(\mathbf{m})$ of an octagon $\mathbf{m}$ in the interval abstract domain.

From Polyhedra to Octagons. Given a polyhedron in its frame representation (V, R), that is, a set of vertices V and rays R in $\mathbb{I}^n$, we can find the *smallest* octagon enclosing this polyhedron by generating a set of octagonal constraints as follows:

- for every i: if there is a ray $r \in R$ such that $r_i > 0$, we set $\mathbf{m}_{(2i)\,(2i-1)} = +\infty$, otherwise, we set $\mathbf{m}_{(2i)\,(2i-1)} = 2 \max \{ v_i \mid v \in V \}$;
- for every i: if there is a ray $r \in R$ such that $r_i < 0$, we set $\mathbf{m}_{(2i-1)\,(2i)} = +\infty$, otherwise, we set $\mathbf{m}_{(2i-1)\,(2i)} = -2 \min \{ v_i \mid v \in V \}$;
- for every $i \neq j$:
  if there is a ray $r \in R$ such that $r_i > r_j$, we set $\mathbf{m}_{(2i-1)\,(2j-1)} = \mathbf{m}_{(2j)\,(2i)} = +\infty$,
  otherwise, we set $\mathbf{m}_{(2i-1)\,(2j-1)} = \mathbf{m}_{(2j)\,(2i)} = \max \{ v_j - v_i \mid v \in V \}$;
- for every $i \neq j$:
  if there is a ray $r \in R$ such that $r_i > -r_j$, we set $\mathbf{m}_{(2i-1)\,(2j)} = +\infty$,
  otherwise, we set $\mathbf{m}_{(2i-1)\,(2j)} = \max \{ -v_j - v_i \mid v \in V \}$;
- for every $i \neq j$:
  if there is a ray $r \in R$ such that $-r_i > r_j$, we set $\mathbf{m}_{(2i)\,(2j-1)} = +\infty$,
  otherwise, we set $\mathbf{m}_{(2i)\,(2j-1)} = \max \{ v_j + v_i \mid v \in V \}$;
- we set $\mathbf{m}_{ii} = 0$ for all i.

The result is, by construction, a coherent DBM. It is also strongly closed as all inferred octagonal constraints saturate the initial polyhedron, and so, the resulting octagon.

In the following, we will denote by *Oct* both operators that convert from an interval or polyhedron abstract element into a coherent DBM representing an octagon — which one is actually considered will depend on the context. Likewise, we will denote by *Int* and *Poly* the operators that convert octagons into intervals and polyhedra. As for the zone abstract domain, we have the following full Galois connections:

$$\mathcal{D}^{\mathit{Poly}} \underset{\alpha^{\mathit{Oct}} \circ \gamma^{\mathit{Poly}} = \mathit{Oct}}{\overset{\alpha^{\mathit{Poly}} \circ \gamma^{\mathit{Oct}} = \mathit{Poly}}{\leftrightarrows}} c\mathsf{DBM} \underset{\alpha^{\mathit{Int}} \circ \gamma^{\mathit{Oct}} = \mathit{Int}}{\overset{\alpha^{\mathit{Oct}} \circ \gamma^{\mathit{Int}} = \mathit{Oct}}{\leftrightarrows}} \mathcal{D}^{\mathit{Int}} \ .$$

This proves, in particular, that $(\alpha^{\mathit{Oct}}, \gamma^{\mathit{Oct}})$ is a partial Galois connection with respect to all transfer functions involving linear expressions, at least, when $\mathbb{I} = \mathbb{Q}$. When $\mathbb{I} = \mathbb{Z}$ or $\mathbb{I} = \mathbb{R}$, we already saw that the Galois connection is total.

4.4.4 Adapted Transfer Functions

We adapt here all the ideas presented in Sect. 3.6 to design several abstract transfer functions for the octagon domain, with different cost versus precision trade-offs. In particular, we suppose that generic tests have been pre-processed into tests of the simpler form ($expr \leq 0$?), as in Sect. 3.6.3, but using our octagon union and intersection operators instead of the zone ones.

Exact Abstractions. The class of assignments and tests that can be exactly modeled in the octagon domain is slightly larger than for the zone domain. We propose the following straightforward definitions that resemble Defs. 3.6.3, 3.6.5, and 3.6.7:

Definition 4.4.4. Exact octagonal transfer functions.

We suppose that $i_0 \neq j_0$. We define the following exact *assignment transfer functions:*

- $(\{\!|\, V_{j_0} \leftarrow [a,b] \,|\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} -2a & \text{if } i = 2j_0 - 1,\ j = 2j_0 \\ 2b & \text{if } i = 2j_0,\ j = 2j_0 - 1 \\ (\{\!|\, V_{j_0} \leftarrow ? \,|\!\}^{Oct}(\mathbf{m}^\bullet))_{ij} & \text{otherwise} \end{cases}$

- $(\{\!|\, V_{j_0} \leftarrow V_{j_0} + [a,b] \,|\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} - a & \text{if } i = 2j_0 - 1,\ j \neq 2j_0 - 1, 2j_0 \\ & \text{or } j = 2j_0,\ i \neq 2j_0 - 1, 2j_0 \\ \mathbf{m}_{ij} + b & \text{if } i \neq 2j_0 - 1, 2j_0,\ j = 2j_0 - 1 \\ & \text{or } j \neq 2j_0 - 1, 2j_0,\ i = 2j_0 \\ \mathbf{m}_{ij} - 2a & \text{if } i = 2j_0 - 1,\ j = 2j_0 \\ \mathbf{m}_{ij} + 2b & \text{if } i = 2j_0,\ j = 2j_0 - 1 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$

- $(\{\!|\, V_{j_0} \leftarrow V_{i_0} + [a,b] \,|\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} -a & \text{if } i = 2j_0 - 1,\ j = 2i_0 - 1 \\ & \text{or } i = 2i_0,\ j = 2j_0 \\ b & \text{if } i = 2i_0 - 1,\ j = 2j_0 - 1 \\ & \text{or } i = 2j_0,\ j = 2i_0 \\ (\{\!|\, V_{j_0} \leftarrow ? \,|\!\}^{Oct}(\mathbf{m}^\bullet))_{ij} & \text{otherwise} \end{cases}$

- $(\{\!|\, V_{j_0} \leftarrow -V_{j_0} \,|\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{\bar{i}j} & \text{if } i \in \{2j_0 - 1, 2j_0\} \text{ and } j \notin \{2j_0 - 1, 2j_0\} \\ \mathbf{m}_{i\bar{j}} & \text{if } i \notin \{2j_0 - 1, 2j_0\} \text{ and } j \in \{2j_0 - 1, 2j_0\} \\ \mathbf{m}_{\bar{i}\bar{j}} & \text{if } i \in \{2j_0 - 1, 2j_0\} \text{ and } j \in \{2j_0 - 1, 2j_0\} \\ \mathbf{m}_{ij} & \text{if } i \notin \{2j_0 - 1, 2j_0\} \text{ and } j \notin \{2j_0 - 1, 2j_0\} \end{cases}$

- $\{\!|\, V_{j_0} \leftarrow -V_{i_0} \,|\!\}^{Oct}_{exact}(\mathbf{m}) \stackrel{\text{def}}{=} (\{\!|\, V_{j_0} \leftarrow -V_{j_0} \,|\!\}^{Oct}_{exact} \circ \{\!|\, V_{j_0} \leftarrow V_{i_0} \,|\!\}^{Oct}_{exact})(\mathbf{m})$

- $\{\!\!\{ V_{j_0} \leftarrow -V_{j_0} + [a,b] \}\!\!\}^{Oct}_{exact}(\mathbf{m}) \stackrel{\text{def}}{=} (\{\!\!\{ V_{j_0} \leftarrow V_{j_0} + [a,b] \}\!\!\}^{Oct}_{exact} \circ \{\!\!\{ V_{j_0} \leftarrow -V_{j_0} \}\!\!\}^{Oct}_{exact})(\mathbf{m})$

- $\{\!\!\{ V_{j_0} \leftarrow -V_{i_0} + [a,b] \}\!\!\}^{Oct}_{exact}(\mathbf{m}) \stackrel{\text{def}}{=} (\{\!\!\{ V_{j_0} \leftarrow V_{j_0} + [a,b] \}\!\!\}^{Oct}_{exact} \circ \{\!\!\{ V_{j_0} \leftarrow -V_{i_0} \}\!\!\}^{Oct}_{exact})(\mathbf{m})$

The invertible cases $V_{j_0} \leftarrow V_{j_0} + [a,b]$ *and* $V_{j_0} \leftarrow -V_{j_0} + [a,b]$ *do not require strongly closed matrix arguments but preserve the strong closure. Other, non-invertible, cases require a strong closure computation due to the embedded forget operator; the result can be strongly closed by merely performing the incremental strong closure* $Inc^{\bullet}_{j_0}$.

We define the following exact *tests transfer functions:*

- $(\{\!\!\{ V_{j_0} + [a,b] \leq 0 \ ? \}\!\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \min(\mathbf{m}_{ij}, -2a) & \text{if } i = 2j_0,\ j = 2j_0 - 1 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$

- $(\{\!\!\{ -V_{j_0} + [a,b] \leq 0 \ ? \}\!\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \min(\mathbf{m}_{ij}, -2a) & \text{if } i = 2j_0 - 1,\ j = 2j_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$

- $(\{\!\!\{ V_{j_0} - V_{i_0} + [a,b] \leq 0 \ ? \}\!\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \min(\mathbf{m}_{ij}, -a) & \text{if } i = 2i_0 - 1,\ j = 2j_0 - 1 \\ & \text{or } i = 2j_0,\ j = 2i_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$

- $(\{\!\!\{ V_{j_0} + V_{i_0} + [a,b] \leq 0 \ ? \}\!\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \min(\mathbf{m}_{ij}, -a) & \text{if } i = 2i_0,\ j = 2j_0 - 1 \\ & \text{or } i = 2j_0,\ j = 2i_0 - 1 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$

- $(\{\!\!\{ -V_{j_0} - V_{i_0} + [a,b] \leq 0 \ ? \}\!\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \min(\mathbf{m}_{ij}, -a) & \text{if } i = 2i_0 - 1,\ j = 2j_0 \\ & \text{or } i = 2j_0 - 1,\ j = 2i_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$

Test transfer functions do not require a strongly closed argument. If, however, the argument is strongly closed, the result can be made strongly closed by applying the incremental strong closure $Inc^{\bullet}_{j_0}$.

We define the following exact *backward assignment transfer functions:*

- $\{\!\!\{ V_{j_0} \rightarrow V_{j_0} + [a,b] \}\!\!\}^{Oct}_{exact}(\mathbf{m}) \stackrel{\text{def}}{=} \{\!\!\{ V_{j_0} \leftarrow V_{j_0} + [-b,-a] \}\!\!\}^{Oct}_{exact}(\mathbf{m})$

- $\{\!\!\{ V_{j_0} \rightarrow -V_{j_0} + [a,b] \}\!\!\}^{Oct}_{exact}(\mathbf{m}) \stackrel{\text{def}}{=} \{\!\!\{ V_{j_0} \leftarrow -V_{j_0} + [a,b] \}\!\!\}^{Oct}_{exact}(\mathbf{m})$

- *if* $\mathbf{m}^\bullet_{(2j_0)\,(2j_0-1)} \geq 2a$ *and* $\mathbf{m}^\bullet_{(2j_0-1)\,(2j_0)} \geq -2b$, *then*

$$(\{\!\!\{ V_{j_0} \to [a,b] \}\!\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$$

$$\begin{cases} \min\left(\mathbf{m}^\bullet_{ij},\, 2(\mathbf{m}^\bullet_{i\,(2j_0-1)} - a),\, 2(\mathbf{m}^\bullet_{i\,(2j_0)} + b)\right) & \text{if } i = \bar{\jmath},\ i \notin \{2j_0-1, 2j_0\} \\ +\infty & \text{if } i \in \{2j_0-1, 2j_0\} \\ & \text{or } j \in \{2j_0-1, 2j_0\} \\ \mathbf{m}^\bullet_{ij} & \text{otherwise} \end{cases}$$

otherwise, $\{\!\!\{ V_{j_0} \to [a,b] \}\!\!\}^{Oct}_{exact}(\mathbf{m}) = \bot^{\mathsf{DBM}}$

- *if* $\mathbf{m}^\bullet_{(2i_0-1)\,(2j_0-1)} \geq a$ *and* $\mathbf{m}^\bullet_{(2j_0-1)\,(2i_0-1)} \geq -b$, *then*

$$(\{\!\!\{ V_{j_0} \to V_{i_0} + [a,b] \}\!\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$$

$$\begin{cases} \min\left(\mathbf{m}^\bullet_{ij},\, \mathbf{m}^\bullet_{(2j_0-1)\,j} + b\right) & \text{if } i = 2i_0-1,\ j \notin \{2i_0-1, 2j_0-1, 2i_0, 2j_0\} \\ \min\left(\mathbf{m}^\bullet_{ij},\, \mathbf{m}^\bullet_{i\,(2j_0)} + b\right) & \text{if } j = 2i_0,\ i \notin \{2i_0-1, 2j_0-1, 2i_0, 2j_0\} \\ \min\left(\mathbf{m}^\bullet_{ij},\, \mathbf{m}^\bullet_{(2j_0)\,j} - a\right) & \text{if } i = 2i_0,\ j \notin \{2i_0-1, 2j_0-1, 2i_0, 2j_0\} \\ \min\left(\mathbf{m}^\bullet_{ij},\, \mathbf{m}^\bullet_{i\,(2j_0-1)} - a\right) & \text{if } j = 2i_0-1,\ i \notin \{2i_0-1, 2j_0-1, 2i_0, 2j_0\} \\ \min\left(\mathbf{m}^\bullet_{ij},\, \mathbf{m}^\bullet_{(2j_0)\,(2j_0-1)} - 2a\right) & \text{if } i = 2i_0,\ j = 2i_0-1 \\ \min\left(\mathbf{m}^\bullet_{ij},\, \mathbf{m}^\bullet_{(2j_0-1)\,(2j_0)} + 2b\right) & \text{if } i = 2i_0-1,\ j = 2i_0 \\ +\infty & \text{if } i \in \{2j_0-1, 2j_0\} \text{ or } j \in \{2j_0-1, 2j_0\} \\ \mathbf{m}^\bullet_{ij} & \text{otherwise} \end{cases}$$

otherwise, $\{\!\!\{ V_{j_0} \to V_{i_0} + [a,b] \}\!\!\}^{Oct}_{exact}(\mathbf{m}) = \bot^{\mathsf{DBM}}$

- $\{\!\!\{ V_{j_0} \to -V_{i_0} + [a,b] \}\!\!\}^{Oct}_{exact}(\mathbf{m}) \stackrel{\text{def}}{=} (\{\!\!\{ V_{j_0} \to V_{i_0} \}\!\!\}^{Oct}_{exact} \circ \{\!\!\{ V_{j_0} \to -V_{j_0} + [a,b] \}\!\!\}^{Oct}_{exact})(\mathbf{m})$

The invertible backward assignments, $V_{j_0} \to V_{j_0} + [a,b]$ *and* $V_{j_0} \to -V_{j_0} + [a,b]$ *do not require strongly closed arguments but preserve the strong closure. Other, non-invertible, cases require a strongly closed argument but, as they only modify constraints involving the variables* V_{i_0} *and* V_{j_0}*, the* $2-$*variable incremental strong closure* $Inc^\bullet_{i_0,j_0}$ *is sufficient to recover the strong closure of the result.*

●

Abstractions Based on Intervals and Polyhedra. As for the zone domain, we can always revert to existing interval or polyhedron transfer functions by using the conversion operators presented in the preceding section:

Definition 4.4.5. Interval-based octagonal transfer functions.

1. $\{\!\!\{ V_i \leftarrow expr \}\!\!\}^{Oct}_{nonrel}(\mathbf{m}) \stackrel{\text{def}}{=} \{\!\!\{ V_i \leftarrow ([\![expr]\!]^{Int}(Int(\mathbf{m}))) \}\!\!\}^{Oct}_{exact}(\mathbf{m})$.
2. $\{\!\!\{ expr \leq 0 \ ? \}\!\!\}^{Oct}_{nonrel}(\mathbf{m}) \stackrel{\text{def}}{=} (Oct \circ \{\!\!\{ expr \leq 0 \ ? \}\!\!\}^{Int} \circ Int)(\mathbf{m}) \cap^{Oct} \mathbf{m}$.
3. $\{\!\!\{ V_i \rightarrow expr \}\!\!\}^{Oct}_{nonrel}(\mathbf{m}) \stackrel{\text{def}}{=} (Oct \circ \{\!\!\{ V_i \rightarrow expr \}\!\!\}^{Int} \circ Int)(\mathbf{m}) \cap^{Oct} \{\!\!\{ V_i \leftarrow ? \}\!\!\}^{Oct}(\mathbf{m}^\bullet)$.

●

Definition 4.4.6. Polyhedron-based octagonal transfer functions.

1. $\{\!\!\{ V_i \leftarrow expr \}\!\!\}^{Oct}_{poly}(\mathbf{m}) \stackrel{\text{def}}{=} (Oct \circ \{\!\!\{ V_i \leftarrow expr \}\!\!\}^{Poly} \circ Poly)(\mathbf{m})$.
2. $\{\!\!\{ expr \leq 0 \ ? \}\!\!\}^{Oct}_{poly}(\mathbf{m}) \stackrel{\text{def}}{=} (Oct \circ \{\!\!\{ expr \leq 0 \ ? \}\!\!\}^{Poly} \circ Poly)(\mathbf{m})$.
3. $\{\!\!\{ V_i \rightarrow expr \}\!\!\}^{Oct}_{poly}(\mathbf{m}) \stackrel{\text{def}}{=} (Oct \circ \{\!\!\{ V_i \rightarrow expr \}\!\!\}^{Poly} \circ Poly)(\mathbf{m})$.

●

These operators have the same advantages and disadvantages as the zone domain ones: the interval-based abstractions are fast but coarse while the polyhedron-based ones are precise — they are, in fact, best abstractions — but very costly. Also, the polyhedron-based abstractions are defined only when the involved expressions are linear or quasi-linear while the interval domain can abstract expressions of arbitrary form. This last problem can be partially solved using the methods that will be presented in Sect. 6.2.5 to abstract an arbitrary expression into a quasi-linear form, at the cost of losing the best abstraction property.

Deriving New Constraints. When assigning or testing an interval linear form that cannot be exactly abstracted, we can synthesise new relational constraints to increase the precision of the interval-based abstractions using a technique similar to the one we used in the zone domain. For a linear or quadratic cost, we can derive constraints of the form $\pm V_i \pm V_j \leq c$ for all V_i and V_j by computing an upper bound of an appropriate interval linear expression in the interval domain, as we did in Defs. 3.6.4 and 3.6.6:

Definition 4.4.7. More precise octagon transfer functions.

Let $expr \stackrel{\text{def}}{=} [a_0, b_0] + \sum_k ([a_k, b_k] \times V_k)$. *We define:*

- $(\!\{ V_{j_0} \leftarrow expr \}\!)^{Oct}_{rel}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$

$$\begin{cases} 2\max(\llbracket expr \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } i = 2j_0 \text{ and } j = 2j_0 - 1 \\ 2\max(\llbracket \boxminus expr \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } i = 2j_0 - 1 \text{ and } j = 2j_0 \\ \max(\llbracket expr \boxminus V_{i_0} \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } i = 2i_0 - 1,\ j = 2j_0 - 1,\ \text{and } i_0 \neq j_0 \\ & \text{or } i = 2j_0,\ j = 2i_0,\ \text{and } i_0 \neq j_0 \\ \max(\llbracket expr \boxplus V_{i_0} \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } i = 2i_0,\ j = 2j_0 - 1,\ \text{and } i_0 \neq j_0 \\ & \text{or } i = 2j_0,\ j = 2i_0 - 1,\ \text{and } i_0 \neq j_0 \\ \max(\llbracket V_{i_0} \boxminus expr \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } i = 2j_0 - 1,\ j = 2i_0 - 1,\ \text{and } i_0 \neq j_0 \\ & \text{or } i = 2i_0,\ j = 2j_0,\ \text{and } i_0 \neq j_0 \\ \max(\llbracket \boxminus expr \boxminus V_{i_0} \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } i = 2i_0 - 1,\ j = 2j_0,\ \text{and } i_0 \neq j_0 \\ & \text{or } i = 2j_0 - 1,\ j = 2i_0,\ \text{and } i_0 \neq j_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$$

- $(\!\{ expr \leq 0\ ? \}\!)^{Oct}_{rel}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \min(\mathbf{m}_{ij},$

$$\left.\begin{cases} 2\max(\llbracket V_{j_0} \boxminus expr \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } \exists j_0,\ i = 2j_0,\ j = 2j_0 - 1 \\ 2\max(\llbracket \boxminus V_{j_0} \boxminus expr \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } \exists j_0,\ i = 2j_0 - 1,\ j = 2j_0 \\ \max(\llbracket V_{j_0} \boxminus V_{i_0} \boxminus expr \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } \exists i_0 \neq j_0,\ i = 2i_0 - 1,\ j = 2j_0 - 1 \\ & \text{or } \exists i_0 \neq j_0,\ i = 2j_0,\ j = 2i_0 \\ \max(\llbracket V_{j_0} \boxplus V_{i_0} \boxminus expr \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } \exists i_0 \neq j_0,\ i = 2i_0,\ j = 2j_0 - 1 \\ \max(\llbracket \boxminus V_{j_0} \boxminus V_{i_0} \boxminus expr \rrbracket^{Int}(Int(\mathbf{m}))) & \text{if } \exists i_0 \neq j_0,\ i = 2i_0 - 1,\ j = 2j_0 \end{cases}\right)$$

The addition $\boxplus$ and subtraction $\boxminus$ operators on interval linear forms are defined by respectively adding or subtracting the interval coefficients corresponding to the same variable. A formal definition will be presented in Sect. 6.2.2.

●

We restrict ourselves to interval linear forms so that some simplification can be easily performed using the $\boxplus$ and $\boxminus$ operators. This is not such a big restriction as we will see in Sect. 6.2.3 how an arbitrary expression can be abstracted into an interval linear form. Also, the same technique can be used for backward assignments of interval linear forms; however, this result in a cubic-cost algorithm which is less attractive than the linear-cost assignment and the quadratic-cost test.

4.4.5 Adapted Extrapolation Operators

Any widening or narrowing in the zone abstract domain that preserves the coherence is, respectively, a valid widening or narrowing in the octagon abstract domain. We can simply

define:

$$\begin{cases} \nabla^{Oct} & \stackrel{\text{def}}{=} & \nabla^{Zone} \\ \triangle^{Oct} & \stackrel{\text{def}}{=} & \triangle^{Zone} \end{cases}$$

and refer the reader to Sect. 3.7 for some possible definitions of ∇^{Zone} and $\triangle^{Zone}$ to provide, for instance, a standard widening ∇^{Oct}_{std} and narrowing $\triangle^{Oct}_{std}$.

There is, however, a little subtlety concerning the widening with thresholds ∇^{Zone}_{th} proposed in Def. 3.7.2. As unary constraints $V_i \leq a$ and $-V_j \leq b$ are now encoded as $V_i + V_i \leq 2a$ and $-V_j - V_j \leq 2b$, we should check for the stability of values in $2\mathbb{T}$ and not in $\mathbb{T}$ for matrix elements that represent such constraints if we wish to get a behavior similar to that of the zone domain. Hence, the following modified definition:

Definition 4.4.8. Widening with thresholds ∇^{Oct}_{th}.

$$(\mathbf{m} \nabla^{Oct}_{th} \mathbf{n})_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} & \textit{if } \mathbf{m}_{ij} \geq \mathbf{n}_{ij} \\ \min \{ x \mid x \in \mathbb{T} \cup \{+\infty\}, x \geq \mathbf{n}_{ij} \} & \textit{otherwise when } i \neq \bar{\jmath} \\ \min \{ 2x \mid x \in \mathbb{T} \cup \{+\infty\}, 2x \geq \mathbf{n}_{ij} \} & \textit{otherwise when } i = \bar{\jmath} \end{cases}$$

●

Finally, we stress the fact that one should restrain from closing strongly the left argument of the widening as the sequence $\mathbf{m}_{i+1} \stackrel{\text{def}}{=} (\mathbf{m}^{\bullet}_i) \nabla^{Oct} \mathbf{n}_i$ may not converge in finite time, even though the sequence $\mathbf{m}_{i+1} \stackrel{\text{def}}{=} \mathbf{m}_i \nabla^{Oct} \mathbf{n}_i$ always does. The example of non-terminating analysis presented in the zone domain, in Ex. 3.7.3, is also valid in the octagon domain. It is not a problem, however, to strongly close the right argument of our widenings, or either or both argument(s) of the standard narrowing. But, due to the non-monotonicity of the widening and narrowing operators, there is no guarantee that closing any argument will result in a gain of precision, unlike what happens for all the other operators and transfer functions.

4.5 Alternate Octagon Encodings

4.5.1 Efficient Representation

In the octagon abstract domain, we only manipulate coherent DBMs. From an implementation point-of-view, coherent DBMs are redundant and waste precious memory. In our implementation [Mina], we chose to actually store only the lower left part of the DBM, that is, elements at line i, column j such that $i \geq j$ or $i = \bar{\jmath}$, as pictured in Fig. 4.4. Other elements can be recovered by coherence, using the fact that $\mathbf{m}_{ij} = \mathbf{m}_{\bar{\jmath}\bar{\imath}}$. The elements are stored into a flat array in left-to-right, top-down, row-major ordering. Note that this layout uses $2n(n+1)$ elements — instead of $4n^2$ for a straightforward redundant DBM storage — while, in theory, $2n^2$ elements suffice. By coherence, $\mathbf{m}_{ii} = \mathbf{m}_{\bar{\imath}\bar{\imath}}$, so, every other element on

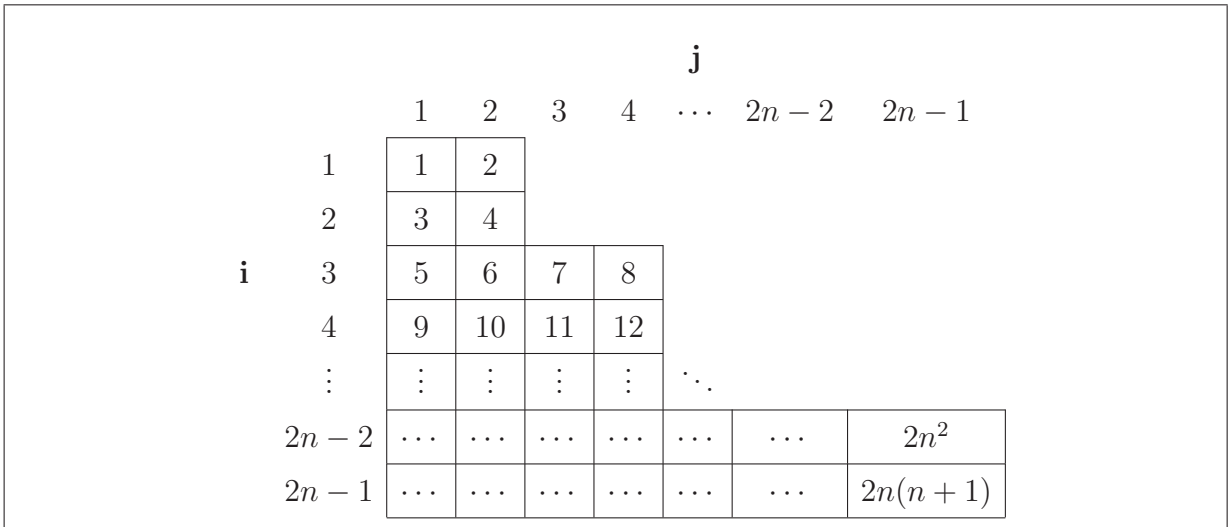


Figure 4.4: Efficient memory representation of a coherent DBM. For each of the $2n(n+1)$ matrix elements actually present in memory, we give its index in a flat array with left-to-right, top-down, row-major ordering. All absent elements can be recovered by coherence.

the diagonal is redundant. They are kept in our representation to avoid "holes". This also permits to easily access elements: $\mathbf{m}_{ij}$, for $i \geq j$ or $i = \bar{j}$, is stored at position $j + \lfloor i^2/2 \rfloor$ in the flat array.

4.5.2 Adapted Hollow Form

In Sect. 3.8.2, we explained how the "hollow" representation of DBMs proposed by Larsen, Larsson, Pettersson, and Yi in [LLPY97] can be useful when memory consumption is an issue: an algorithm is provided to set to $+\infty$ the maximal number of elements of a DBM without changing its zone concretisation γ^{Zone}. This method can also be applied directly to DBMs representing octagons; however it is not optimal. Indeed, a DBM representing an octagon has a little more redundancy than a DBM representing a zone. Firstly, coherence ensures that half the elements can be safely ignored. Secondly, we can choose as hollow DBM a matrix that has the same strong closure than that of the original matrix, which is less restrictive than imposing that they have the same closure, as many distinct closed matrices may have the same strong closure. In practice, whenever $\mathbf{m}^{\bullet}_{ij} = (\mathbf{m}^{\bullet}_{i\bar{i}} + \mathbf{m}^{\bullet}_{\bar{j}j})/2$ and $i \neq \bar{j}$, the value of $\mathbf{m}^{\bullet}_{ij}$ can be forgotten as it will be restored by a strong closure application.

We propose the following algorithm that extends the one presented in Sect. 3.8.2:

1. Firstly, apply the algorithm *Hollow* presented in Sect. 3.8.2 on a strongly closed DBM

$\mathbf{m}$. We call $\mathbf{m}'$ the resulting matrix.

2. Then, set to $+\infty$ elements at positions (i, j) such that $\mathbf{m}'_{ij} = (\mathbf{m}'_{i\bar{i}} + \mathbf{m}'_{\bar{j}j})/2$ when $j \neq \bar{i}$. We call $\mathbf{m}''$ the obtained matrix.

3. Finally, set to $+\infty$ elements in $\mathbf{m}''$ at position (i, j) such that $\lfloor i/2 \rfloor < \lfloor j/2 \rfloor$ or $i = j$. We denote by $\mathit{Hollow}^\bullet(\mathbf{m})$ the result of this algorithm.

Theorem 4.5.1. Properties of the hollow representation.

1. $\mathbf{m}^\bullet$ *can be retrieved from* $\mathit{Hollow}^\bullet(\mathbf{m})$.

2. $\mathit{Hollow}^\bullet(\mathbf{m})$ *can be computed in cubic time.*

●

Proof.

1. We prove that each step of $\mathit{Hollow}^\bullet$ can be reverted:

   - $\mathbf{m}''$ can be reconstructed from $\mathit{Hollow}^\bullet(\mathbf{m})$ by coherence:

$$\mathbf{m}''_{ij} \stackrel{\text{def}}{=} \begin{cases} (\mathit{Hollow}^\bullet(\mathbf{m}))_{ij} & \text{if } \lfloor i/2 \rfloor \geq \lfloor j/2 \rfloor \text{ and } i \neq j \\ (\mathit{Hollow}^\bullet(\mathbf{m}))_{\bar{j}\bar{i}} & \text{if } \lfloor i/2 \rfloor < \lfloor j/2 \rfloor \\ 0 & \text{if } i = j \end{cases}$$

   - Because our algorithm ensures that $\forall i$, $\mathbf{m}'_{i\bar{i}} = \mathbf{m}''_{i\bar{i}}$, $\mathbf{m}'$ can be reconstructed from $\mathbf{m}''$ by setting:

$$\mathbf{m}'_{ij} \stackrel{\text{def}}{=} \min(\mathbf{m}''_{ij}, (\mathbf{m}''_{i\bar{i}} + \mathbf{m}''_{\bar{j}j})/2) \ .$$

   - Thm. 3.8.1 states that $\gamma^{Pot}(\mathit{Hollow}(\mathbf{m}^*)) = \gamma^{Pot}(\mathbf{m})$. If we apply this theorem to $\mathbf{m}^\bullet$, we get $\gamma^{Pot}(\mathit{Hollow}((\mathbf{m}^\bullet)^*)) = \gamma^{Pot}(\mathbf{m}^\bullet)$, and so, $\gamma^{Oct}(\mathit{Hollow}((\mathbf{m}^\bullet)^*)) = \gamma^{Oct}(\mathbf{m}^\bullet)$. As strongly closed matrices are also closed, this implies $\gamma^{Oct}(\mathit{Hollow}(\mathbf{m}^\bullet)) = \gamma^{Oct}(\mathbf{m})$, and so, $(\mathit{Hollow}(\mathbf{m}^\bullet))^\bullet = \mathbf{m}^\bullet$. Thus, by applying the strong closure to the matrix $\mathbf{m}'$, we get $\mathbf{m}^\bullet$ back.

2. The computation of $\mathit{Hollow}^\bullet(\mathbf{m})$ is not very different from that of $\mathit{Hollow}(\mathbf{m})$, which can be done in cubic time according to Thm. 3.8.1.3. We only change the closure requirement into a strong closure requirement and perform two quadratic time additional steps, hence, the cost remains cubic.

○

It is not obvious whether $\mathit{Hollow}^\bullet$ returns a DBM with as many $+\infty$ coefficients as possible. However, in practice, it performs quite well and is an improvement over using Hollow on DBMs representing octagons.

4.6 Analysis Examples

All the examples presented in the previous chapter that were in the scope of the zone abstract domain can also be precisely analysed in the octagon abstract domain. We present here new examples that require the inference of relational invariants of the form $c \leq X+Y \leq d$, and so, cannot be precisely analysed in the zone abstract domain.

4.6.1 Decreasing Loop

In the following example, the loop counter I is decremented at each iteration while the index X is incremented:

Example 4.6.1. Decreasing loop.

```
    I ← 16;
    X ← 1;
    while ❶ 0 < I {
②     X ← X + 1;
      I ← I − 1
③   }
④
```

●

The iterates are as follows:

iteration i	label l	octagon X_l^i
0	❶	$I = 16 \wedge X = 1 \wedge I + X = 17 \wedge I - X = 15$
1	②	$I = 16 \wedge X = 1 \wedge I + X = 17 \wedge I - X = 15$
2	③	$I = 15 \wedge X = 2 \wedge I + X = 17 \wedge I - X = 13$
3	❶ $\triangledown$	$I \leq 16 \wedge 1 \leq X \wedge I + X = 17 \wedge I - X \leq 15$
4	②	$1 \leq I \leq 16 \wedge 1 \leq X \leq 16 \wedge I + X = 17$ $\wedge -15 \leq I - X \leq 15$
5	③	$0 \leq I \leq 15 \wedge 2 \leq X \leq 17 \wedge I + X = 17$ $\wedge -17 \leq I - X \leq 13$
6	❶ $\triangledown$	$I \leq 16 \wedge 1 \leq X \wedge I + X = 17 \wedge I - X \leq 15$
7	❶ $\triangle$	$0 \leq I \leq 16 \wedge 1 \leq X \leq 17 \wedge I + X = 17$ $\wedge -17 \leq I - X \leq 15$
8	④	$I = 0 \wedge X = 17 \wedge I + X = 17 \wedge I - X = -17$

We are able to discover the relational loop invariant $I + X = 17$ at ❶. Moreover,

iterations with widening and narrowing are able to prove that, at the end of the loop, $I = 0$. Thus, we can conclude that $X = 17$ holds at ④.

Thanks to its ability to exactly represent the negation of variables, the octagon abstract domain is able to abstract precisely loops that mix increasing and decreasing counters, which is not possible using the plain zone abstract domain.

4.6.2 Absolute Value

Consider the following code that computes the absolute value Y of X before testing whether it is smaller than 69:

Example 4.6.2. Absolute value analysis.

$X \leftarrow [-100, 100]$;
① $Y \leftarrow X$;
② `if` $Y \leq 0$ `{` ③ $Y \leftarrow -Y$ ④ `}` `else` `{` ⑤ `};`
⑥ `if` $Y \leq 69$ `{` ⑦ $\cdots$ `}`

●

The interval domain is able to prove easily that, at ⑦, $Y \leq 69$, however, it is not able to deduce any bound on X. The octagon domain, however, can discover that $X \in [-69, 69]$ at ⑦ because it is able to keep the relationship between X and Y. The iterates are as follows:

iteration i	label l	octagon X_l^i
0	①	$-100 \leq X \leq 100$
1	②	$-100 \leq X \leq 100 \ \wedge \ -100 \leq Y \leq 100$ $\wedge \ X - Y = 0 \ \wedge \ -200 \leq X + Y \leq 200$
2	③	$-100 \leq X \leq 0 \ \wedge \ -100 \leq Y \leq 0$ $\wedge \ X - Y = 0 \ \wedge \ -200 \leq X + Y \leq 0$
3	④	$-100 \leq X \leq 0 \ \wedge \ 0 \leq Y \leq 100$ $\wedge \ -200 \leq X - Y \leq 0 \ \wedge \ X + Y = 0$
4	⑤	$0 \leq X \leq 100 \ \wedge \ 0 \leq Y \leq 100$ $\wedge \ X - Y = 0 \ \wedge \ 0 \leq X + Y \leq 200$
5	⑥	$-100 \leq X \leq 100 \ \wedge \ 0 \leq Y \leq 100$ $\wedge \ -200 \leq X - Y \leq 0 \ \wedge \ 0 \leq X + Y \leq 200$
6	⑦	$-69 \leq X \leq 69 \ \wedge \ 0 \leq Y \leq 69$ $\wedge \ -138 \leq X - Y \leq 0 \ \wedge \ 0 \leq X + Y \leq 138$

Intuitively, one may think that the most precise bounds for X can only be discovered by an abstract domain able to represent the constraint $Y = |X|$. In fact, this intuition is

false and the octagon domain, which cannot represent such a non-linear and non-convex constraint, finds the most precise bounds for X. The important point is that, at ⑥, we are able to infer the information $-Y \leq X \leq Y$ that will be combined by closure with the information $Y \leq 69$ at ⑦. This analysis works equally well if we modify the 100 and 69 constants. As this analysis requires the manipulation of bounds on $X + Y$, it cannot be performed accurately using the zone abstract domain.

4.6.3 Rate Limiter

Consider the following code implementing a rate limiter:

Example 4.6.3. Rate limiter analysis.

```
    Y ← 0;
    while ❶ rand {
      X ← [−128, 128];
      D ← [0, 16];
      S ← Y;
②     R ← X − S;
      Y ← X;
      if R ≤ −D { ③ Y ← S − D ④ } else
      if D ≤ R  { ⑤ Y ← S + D ⑥ }
⑦   }
```

●

At each loop iteration, a new value for the entry X is fetched within $[-128, 128]$ and a new maximum rate D is chosen in $[0, 16]$. The program then computes an output Y that tries to follow X but is compelled to change slowly: the difference between Y and its value in the preceding iteration is bounded, in absolute value, by the current value of D. The S state variable is used to remember the value of Y at the last iteration while R is a temporary variable used to avoid computing the difference $X - S$ twice.

The output Y is bounded by the range of X, that is, $Y \in [-128, 128]$. To prove this, suppose that $Y \in [-128, 128]$ at ②. One of the three following cases may occur at ⑦ in the same loop iteration:

- If $-D < R < D$, then $Y = X$.
- If $R \leq -D$, then $Y = S - D$. As $R = X - S$, we have $X - S \leq -D$, so, $S - D \geq X$. Finally, $X \leq Y \leq S$, so, $Y \in [-128, 128]$.
- If $R \geq D$, then $Y = S + D$. As $R = X - S$, we have $X - S \geq D$, so, $S + D \leq X$. Finally, $S \leq Y \leq X$, so, $Y \in [-128, 128]$.

Interval Analysis. The interval analysis is not able to keep any relation between R, X, and S. As a consequence, the tests $R \leq -D$ and $R \geq D$ do not refine the bounds for $S - D$ nor $S + D$. The analysis behaves exactly as if these tests were ignored, that is, as if Y were non-deterministically incremented or decremented by D at *each* loop iteration. An abstract semantics using the interval transfer functions but exact fixpoint computation without widening would find that, at ❶, Y is unbounded, so, *a fortiori*, no computable loop abstraction using widenings can find finite bounds for Y.

Octagonal Analysis. In order to find the most precise bounds for Y, that is $Y \in [-128, 128]$, one needs to represent exactly the constraint $R = X - S$, which is not possible in the octagon domain. Nevertheless, the non-exact abstract assignment of Def. 4.4.7 is smart enough to derive the constraint $R + S \in [-128, 128]$. Suppose that, at a given abstract loop iteration, $Y \in [-M, M]$ at ❶. Then, at ②, we also have $S \in [-M, M]$ and the following computation occurs:

- At ③, the test implies $R + D \leq 0$, which implies $-R \geq 0$ and $S = (S + R) - R \geq S + R \geq -128$, so, $S \in [-128, M]$. At ④, $Y - S \in [-16, 0]$, which gives $Y = (Y - S) + S \in [-144, M]$.
- At ⑤, the test implies $R - D \geq 0$, which implies $-R \leq 0$ and $S = (S + R) - R \leq S + R \leq 128$, so, $S \in [M, 128]$. At ⑥, $Y - S \in [0, 16]$, which gives $Y = (Y - S) + S \in [-M, 144]$.
- At ⑦, by union, we get $Y \in [-\max(M, 144), \max(M, 144)]$.

Thus, the iteration is stable if and only if $M \geq 144$. As a consequence, a static analysis using the widening with thresholds will find as bound for Y the smallest threshold greater than 144. Even though this result is not optimal, we are still able to derive *finite bounds* for Y provided our widening has sufficiently many threshold steps.

4.7 Related Works and Extensions

It seems that at the time we published some early results on the zone and octagon domains [Min00, Min01a, Min01b], the search for numerical abstract domains with a precision versus cost trade-off between that of the interval and the polyhedron domains became of particular interest to the static analysis research community. We present here two recent extensions to the octagon abstract domain of particular interest.

Two Variables Per Linear Inequality Abstract Domain. In [SKH02], Simon, King, and Howe present an abstract domain for invariants containing constraints of the form

$\alpha X + \beta Y \leq c$, that is, octagonal constraints refined to allow arbitrary coefficients instead of unit ones. An abstract element is conceived as a map that stores a planar polyhedron for each unordered pair of distinct variables. This domain is called Two Variables Per Linear Inequality (TVPLI, for short).

Several efficient satisfiability algorithms for such constraint conjunctions have been proposed in the late 70's: one is from Nelson [Nel78] and the other one, inspired from Bellman's satisfiability algorithm for potential constraints [Bel58], is from Shostak [Sho81]. Following our methodology, the authors choose in [SKH02] to derive a normal form algorithm from Nelson's satisfiability algorithm, which is then used to construct best abstractions for the union and projection operators, as well as an inclusion test. Also, the authors propose to use any polyhedron widening point-wisely on each planar polyhedron.

As for polyhedra, the algorithms are complete only for rationals and reals. Integers can be abstracted as rationals with a little precision degradation. Indeed, even the satisfiability problem for conjunctions of TVPLI constraints on integers is known to be NP-complete.

The TVPLI domain is much more precise than our octagon domain, but it also has a greater cost. The most costly operation is, as expected, the closure that is used pervasively and performs in $\mathcal{O}(k^2n^3 \log n(\log k + \log n))$, where n is the number of variables and k is the maximum number of inequalities over all variable pairs. As for the polyhedron domain, there is no upper bound to the size of an abstract element, and so, no theoretical time cost bound can be given. More experiments are needed to determine the scope of this new abstract domain.

Octahedra Abstract Domain. In [CC04], Clarisó and Cortadella introduce another, orthogonal, extension to the octagon abstract domain allowing *any* number of variables in each inequality while keeping unit coefficients: $\sum_i \epsilon_i V_i \leq c$, $\epsilon_i \in \{-1, 0, 1\}$. They name it the *octahedra* abstract domain, meaning "octagons with more dimensions". The authors are only interested in inferring delays, so, they designed the octahedra abstract domain to abstract sets of *positive* variables only, but it can be adapted to abstract variables that can be positive or negative.

Unlike what happens for zone, octagonal, and TVPLI constraints, no complete satisfiability algorithm seems to have been discovered for such constraints — except, of course, the generic ones used in linear programming. The authors rely on a propagation algorithm that saturates a constraint set by adding the sum of two constraints if it has only unit coefficients. This does not give a full normal form, and so, the derived inclusion and equality tests are sound but incomplete — they can return either a definitive "yes" or "I don't know" — and there is no best abstraction result. However, the authors introduce a very clever representation for octahedra that compactly encodes a conjunction of constraints using a tree-like structure with maximal sharing that resembles the Binary Decision Diagrams introduced by Bryant in [Bry86] to represent boolean functions.

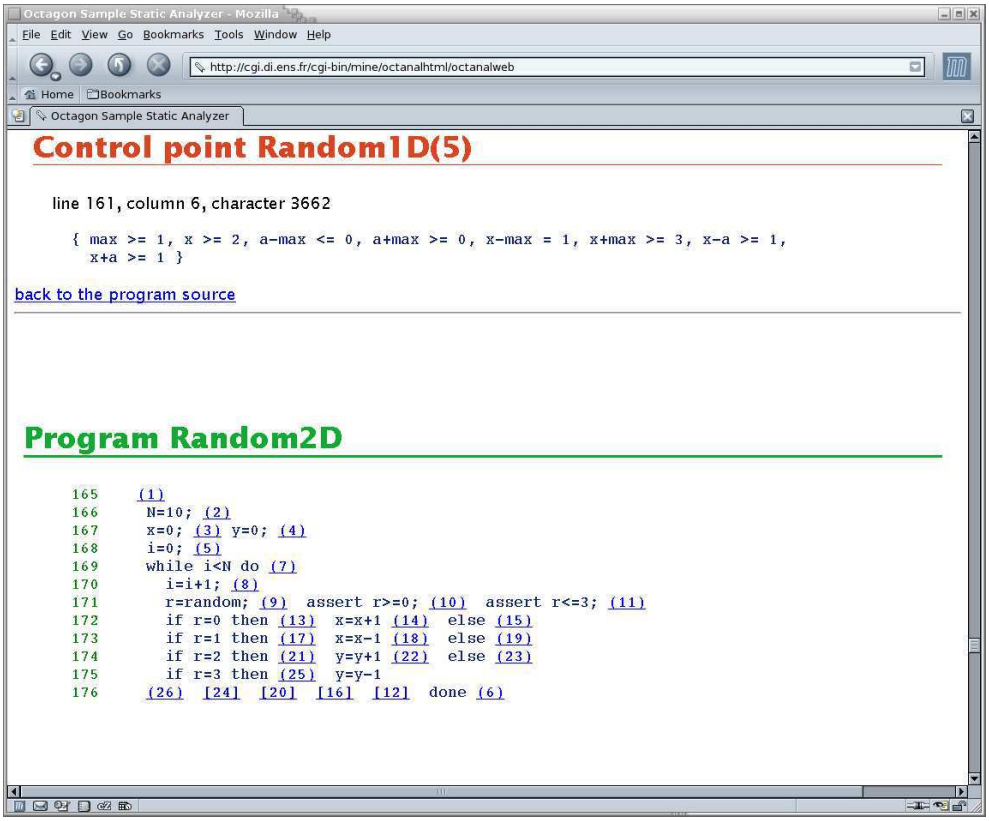


Figure 4.5: On-line octagon sample analyser.

The octahedra abstract domain has been successfully applied to the problem of automatically generating timing constraints for asynchronous circuits where it exhibited a precision comparable to that of the polyhedron abstract domain, more time consumption, but much less memory consumption — which is a positive result as, in this application, memory was the limiting factor.

4.8 Conclusion

We have presented, in this chapter, an extension to the zone abstract domain that is able to discover invariants of the form $\pm X \pm Y \leq c$ for a similar asymptotic cost: a $\mathcal{O}(n^2)$ memory cost and a $\mathcal{O}(n^3)$ time cost per abstract operation in the worst case. When $\mathbb{I} = \mathbb{Q}$ and $\mathbb{I} = \mathbb{R}$, we have provided as much best abstract transfer functions and operators as possible. When $\mathbb{I} = \mathbb{Z}$ the choice is given to either lose a little precision, or retain best operators and transfer functions and have a $\mathcal{O}(n^4)$ worst-case time cost.

The octagon abstract domain presented here has been implemented as a robust and fast library in the C programming language — an OCaml binding is also available. This library is publicly available on the WEB [Mina]. An academic analyser using the octagon domain is included within the library. It can be queried on-line [Minb] using the WEB-interface pictured in Fig. 4.5. The library has also been used in the Astrée industrial-strength static analyser. Chap. 8 will present its integration within Astrée and give experimental results on the analysis of real-life programs that show the practical usefulness of the octagon domain.

Future Work. Our work on the octagon domain may be extended in several directions. In particular, we left in our presentation a few dark corners that should be elucidated. One issue is the closure algorithm for *integer* octagonal constraints. Unlike the integer zone constraints and the rational and real octagonal constraints, which enjoy Floyd–Warshall-related cubic algorithms, the only closure algorithm for integer octagonal constraints we are aware of has a $\mathcal{O}(n^4)$ cost — see Sect. 4.3.5. Determining the exact complexity of the closure problem for integer octagonal constraints is interesting from a theoretic point-of-view, even though, from a practical point-of-view, we can safely use an incomplete closure if we do not mind a small precision loss. The next points are common to the octagon domain and the zone domain of the previous chapter. Firstly, we only presented widening and narrowing operators that are point-wise extensions of those on the interval domain and we may ask ourselves whether there exists other ways to construct them. In particular, it would be quite interesting if we could design a widening operator that is insensible to the chosen DBM representation of a zone or an octagon, so that it is possible to close the iterates — see Sect. 3.7.2. Secondly, we designed inexact transfer functions on interval linear forms that are able to derive new relational constraints but only use non-relational information — see Def. 4.4.7. New transfer functions that are, in terms of precision and cost, between these transfer functions and the costly polyhedron-based ones would be welcome. One may investigate whether best linear assignments and tests can be computed using a less costly technique than switching temporarily into the polyhedron domain. Finally, we may wish to extend the expressiveness of the octagon domain and infer invariants of a more complex form while keeping a cost that is much smaller than that of the polyhedron domain. We cited two newly developed abstract domains that are in-between, in terms of precision and cost, the octagon and the polyhedron domains: the octahedra domain by Clarisó and Cortadella and the TVPLI domain by Simon, King, and Howe. Such domains seem very promising but are works in progress: they still need a few operators and transfer functions before the whole Def. 2.4.1 is implemented and they can be "plugged" into static analysers such as Astrée. Only experimentation — such as our work with Astrée, related in Chap. 8 — will tell us which abstract domain is best for a given application.

Chapter 5

A Family of Zone-Like Abstract Domains

Nous nous intéressons maintenant à une autre extension du domaine des zones permettant de représenter et de manipuler des invariants de la forme $X - Y \in B$ où B vit dans une base non relationnelle $\mathcal{B}$. Des hypothèses suffisantes sur $\mathcal{B}$ sont présentées pour construire une notion de clôture par plus-courts chemins, dont nos fonctions de transfert et opérateurs abstraits découlent. Nous donnons quelques exemples de bases valides $\mathcal{B}$ qui permettent la construction de nouveaux domaines numériques abstraits faiblement relationnels tels que, par exemple, un domaine pour les contraintes de zone strictes $X - Y < c$ et un autre pour les contraintes de congruence $X \equiv Y + c\ [d]$.

We turn to another, orthogonal, extension of zones, that allows representing and manipulating invariants of the form $X - Y \in B$ where B lives in a suitable non-relational basis $\mathcal{B}$. We present sufficient hypotheses on $\mathcal{B}$ that permit the existence of a shortest-path closure notion, from which our abstract operators and transfer functions are derived. We also give suitable examples of acceptable bases $\mathcal{B}$ that permit the construction of new weakly relational numerical abstract domains, such as a domain for strict zone constraints $X - Y < c$ and another for congruence constraints $X \equiv Y + c\ [d]$.

5.1 Introduction

We are interested, in this chapter, in generalising the zone abstract domain, that was presented in Chap. 3 and manipulated conjunctions of constraints of the form $X - Y \leq c$, to different forms of constraints while keeping its lightweight cubic worst-case time cost

per abstract operation. Recall that this cost comes from our ability to compute a normal form that enjoys a saturation property, using a mere shortest-path closure computation. A natural question is whether it is possible to apply such algorithms to other types of constraints. We already discussed in Chap. 4 some extensions that consists in generalising the left-hand side of constraints: octagonal constraints ($\pm X \pm Y \leq c$) but also Two Variables Per Linear Inequality ($\alpha X + \beta Y \leq c$, [SKH02]) and octahedra constraints ($\sum_i \epsilon_i V_i \leq c$, $\epsilon_i \in \{-1, 0, 1\}$, [CC04]). In this chapter, we propose to try and extend the *right-hand* side of constraints instead: given an abstraction $\mathcal{B}$ of $\mathcal{P}(\mathbb{I})$, where $\mathbb{I}$ is a numerical set that can be $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$, we propose a generic way to construct an abstract domain for conjunctions of constraints of the form $X - Y \in \gamma_{\mathcal{B}}(B)$, $B \in \mathcal{B}$. Adapting the closure is a lot of work and we will need restrictive hypotheses on the algebraic structure of the basis $\mathcal{B}$. The subsequent construction of the abstract domain is then straightforward. We provide several bases examples $\mathcal{B}$ that motivate our construction. We retrieve the zone abstract domain in Sect. 5.4.2, but also extend it to allow both non-strict $X - Y \leq c$ and strict $X - Y < c$ constraints in Sect. 5.4.3. We will also present new domains for congruence relations $X \equiv Y + a$ [b]; either integer congruences, in Sect. 5.4.4, or rational congruences, in Sect. 5.4.5.

Related Work. We have already seen several examples of closure-based satisfiability algorithms for conjunctions of constraints of the form $X - Y \leq c$ [Bel58], $\pm X \pm Y \leq c$ [HS97], and $\alpha X + \beta Y \leq c$ [Nel78, Sho81] that can be converted into normalisation algorithms. In [TCR94], Toman, Chomicki, and Rogers propose a normalisation procedure for conjunctions of constraints of the form $X \equiv Y + c$ [d] and present derived equality, inclusion, and emptiness testing, as well as intersection and projection operators. Unfortunately, their algorithm has a $\mathcal{O}(n^4)$ time cost. We will see that, by propagating constraints using the same pattern as in the Floyd–Warshall shortest-path closure algorithm, the normalisation can be performed with a better, cubic, worst-case time cost.

Methods purely based on constraint propagation following a simple transitive closure scheme become quickly incomplete when we consider more complex constraint forms. For instance, the satisfiability problem for constraints of the form $X - Y \in (a\mathbb{Z} + b) \cap [x, y]$ is undecidable [TCR94]. In his Ph. D. thesis, Bagnara [Bag97, Chap. 5.7] proposes to use the family of constraints $X - Y \in B$ where B lives in a subset $\mathcal{B}$ of $\mathcal{P}(\mathbb{R})$, but without any restrictive hypotheses on $\mathcal{B}$, and so, without our completeness and best abstraction results. Also, it is quite customary in the field of *Constraint Logic Programming* (CLP) to manipulate complex constraints using such incomplete propagation methods. Incompleteness is not an issue for CLP as only a *finite* universe is tested for satisfiability. Constraint propagation is merely performed for efficiency purposes but each input point will be ultimately checked against each initial and inferred constraint, resulting in a complete and terminating algorithm.

When it comes to abstract interpretation, even though we manipulate potentially infinite sets, the completeness of the constraint propagation in our domain $X - Y \in B$, $B \in \mathcal{B}$ is not required to get a *sound* abstraction. It matters, however, for *precision* purposes. A complete closure enjoying saturation ensures full equality, inclusion, and emptiness tests — instead of incomplete tests that can return either a definitive "yes" or "I don't know" — and *best* or even *exact* abstractions for union, projection, assignments of the form $X \leftarrow Y + C$, and tests of the form $(X - Y \in C\ ?)$ whenever C can be exactly represented in $\mathcal{B}$. We will see that obtaining sound constructions is quite easy while giving these precision guarantees is much harder.

5.2 Constraint Matrices

5.2.1 Representing Constraints

Recall that a basis $\mathcal{B}$, defined as in Sect. 2.4.4, is an abstraction of $\mathcal{P}(\mathbb{I})$, where $\mathbb{I} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$, suitable to build non-relational abstract domains in a generic way. Given a basis $\mathcal{B}$, we wish now to lift it to a *weakly relational* abstract domain able to represent and manipulate conjunctions of constraints of the form $V_i - V_j \in C$ and $V_i \in C$ where the C's are subsets of $\mathbb{I}$ representable in $\mathcal{B}$.

Constraint Matrix. Let $\mathcal{V} \stackrel{\text{def}}{=} \{V_1, \cdots, V_n\}$ be the finite set of program variables. A *constraint matrix* $\mathbf{m}$ is a $(n+1) \times (n+1)$ matrix with elements in $\mathcal{B}$. We denote by *Weak* the set of such constraint matrices. As for zones, we use the first row and column of the matrix to encode unary constraints; intuitively, this amounts to adding a phantom variable $V_0 \notin \mathcal{V}$ that is considered to be always equal to zero, and encode unary constraints $V_i \in C$ as $V_i - V_0 \in C$. Whenever there is no information about $V_j - V_i$, we can safely set $\mathbf{m}_{ij}$ to $\top^\sharp_{\mathcal{B}}$ as $\gamma_{\mathcal{B}}(\top^\sharp_{\mathcal{B}}) = \mathbb{I}$. Unlike what happened for the zone domain, we do not need to define a special $+\infty$ element that represents "no constraint". A constraint matrix $\mathbf{m}$ represents the following set of points in $\mathbb{I}^n$:

Definition 5.2.1. Constraint matrix concretisation γ^{Weak} of a DBM.

$$\gamma^{Weak}(\mathbf{m}) \stackrel{\text{def}}{=} \{\ (v_1, \ldots, v_n) \in \mathbb{I}^n \mid \begin{array}{l} \forall i, j \geq 1,\ v_j - v_i \in \gamma_{\mathcal{B}}(\mathbf{m}_{ij})\ \textit{and} \\ \forall i \geq 1,\ v_i \in \gamma_{\mathcal{B}}(\mathbf{m}_{0i}),\ -v_i \in \gamma_{\mathcal{B}}(\mathbf{m}_{i0})\ \} \end{array} .$$

●

Alternatively, we can picture $\mathbf{m}$ as a *constraint graph*, that is, a directed complete graph where nodes are variables in $\mathcal{V} \cup \{V_0\}$ and each arc is labelled with an element in $\mathcal{B}$. Thus, we will follow the graph terminology of paths and cycles as in the preceding chapters.

Partial Order. We can define the following partial order $\sqsubseteq^{Weak}$ on $Weak$ as the pointwise extension to matrices of the order $\sqsubseteq^\sharp_{\mathcal{B}}$ on $\mathcal{B}$:

Definition 5.2.2. $\sqsubseteq^{Weak}$ order.

$$\begin{array}{lcl} \mathbf{m} \sqsubseteq^{Weak} \mathbf{n} & \stackrel{\text{def}}{\iff} & \forall i,j,\ \mathbf{m}_{ij} \sqsubseteq^\sharp_{\mathcal{B}} \mathbf{n}_{ij} \\ (\bot^{Weak})_{ij} & \stackrel{\text{def}}{=} & \bot^\sharp_{\mathcal{B}} \\ (\top^{Weak})_{ij} & \stackrel{\text{def}}{=} & \top^\sharp_{\mathcal{B}} \end{array}$$

●

Obviously, γ^{Weak} is monotonic for $\sqsubseteq^{Weak}$: $\mathbf{m} \sqsubseteq^{Weak} \mathbf{n} \Longrightarrow \gamma^{Weak}(\mathbf{m}) \subseteq \gamma^{Weak}(\mathbf{n})$. We also have $\gamma^{Weak}(\bot^{Weak}) = \emptyset$ while $\gamma^{Weak}(\top^{Weak}) = \mathbb{I}^n$.

Coherence. Given two variables V_i and V_j, two elements in $\mathbf{m}$ give direct information on $V_j - V_i$: $\mathbf{m}_{ij}$ and $\mathbf{m}_{ji}$. Similarly, $\mathbf{m}_{i0}$ and $\mathbf{m}_{0i}$ both give some information about the acceptable values of V_i. In the following we impose a *coherence* condition stating that each element in such a pair gives the same amount of information as the other one:

Definition 5.2.3. Coherence constraint matrix.

$$\mathbf{m} \textit{ is coherent } \iff \forall i,j,\ \gamma_{\mathcal{B}}(\mathbf{m}_{ij}) = \{\ -x \mid x \in \gamma_{\mathcal{B}}(\mathbf{m}_{ji})\ \}\ .$$

●

From now on, all our constraint matrices will be coherent. This implicitly requires that $\mathcal{B}$ has an exact counterpart for the unary minus operator so that $\{\ -x \mid x \in \gamma_{\mathcal{B}}(\mathbf{m}_{ji})\ \}$ is exactly representable as an abstract basis element.

Constraint Propagation. Constraints are not independent. For instance:

$$V_j - V_k \in C \text{ and } V_k - V_i \in D \Longrightarrow V_j - V_i \in \{\ c + d \mid c \in C,\ d \in D\ \}\ .$$

If $+^\sharp$ is a sound abstraction in $\mathcal{B}$ for the $+$ operator, then we have:

$$V_j - V_k \in \gamma_{\mathcal{B}}(\mathbf{m}_{kj}) \text{ and } V_k - V_i \in \gamma_{\mathcal{B}}(\mathbf{m}_{ik}) \Longrightarrow V_j - V_i \in \gamma_{\mathcal{B}}(\mathbf{m}_{ik} +^\sharp \mathbf{m}_{kj})\ .$$

If we suppose moreover that $\cap^\sharp_{\mathcal{B}}$ is a sound abstraction for $\cap$, then we can replace $\mathbf{m}_{ij}$ with $\mathbf{m}_{ij} \cap^\sharp_{\mathcal{B}} (\mathbf{m}_{ik} +^\sharp \mathbf{m}_{kj})$ for all i,j,k without changing $\gamma^{Weak}(\mathbf{m})$. More generally, given a path $\langle i = i_1, \ldots, i_m = j \rangle$ from i to j we can derive the implicit constraint $V_j - V_i \in \gamma_{\mathcal{B}}(\mathbf{m}_{i_1 i_2} +^\sharp \cdots +^\sharp \mathbf{m}_{i_{m-1} i_m})$ to refine $\mathbf{m}_{ij}$. Thus, we can perform constraint propagation in the abstract to replace a matrix with a smaller matrix representing the same set. Of course, if we wish this abstract constraint propagation to be as precise as in $\mathcal{P}(\mathbb{I})$, the basis operators $\cap^\sharp_{\mathcal{B}}$ and $+^\sharp$ need to be *exact* abstractions. We saw that, in the zone domain, it is possible to gather *all* implicit constraints in cubic time to get an emptiness test, and a

normal form enjoying constraint saturation. We will present sufficient conditions on $\mathcal{B}$ so that such a normal form exists and is computable by a slightly modified Floyd–Warshall algorithm.

5.2.2 Previous Work on Closed Half-Rings

We recall here the notion of *closed half-ring* presented, for instance, in [AHU74] and [CLR90, § 26], that provides a well-known framework for generalising the notion of shortest-path closure and Floyd–Warshall algorithm:

Definition 5.2.4. Closed half-ring.

A closed half-ring is a set C together with:

1. *an operator $\oplus$ that is associative: $a \oplus (b \oplus c) = (a \oplus b) \oplus c$, commutative: $a \oplus b = b \oplus a$, and idempotent: $a \oplus a = a$, and*
2. *an operator $\odot$ that is associative, such that*
3. *$\odot$ distributes $\oplus$: $a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$ and $(a \oplus b) \odot c = (a \odot c) \oplus (b \odot c)$, and*
4. *$\oplus$ has a neutral element $\overline{0}$: $\overline{0} \oplus a = a \oplus \overline{0} = a$, such that $\overline{0} \odot a = a \odot \overline{0} = \overline{0}$, and*
5. *$\odot$ has a neutral element $\overline{1}$: $\overline{1} \odot a = a \odot \overline{1} = a$, and*
6. *given a countable set of elements $(a_i)_{i \in \mathbb{N}}$, the infinite sum $\bigoplus_i a_i$ exists, and $\odot$ distributes infinite sums.*

●

Properties. A core property of closed half-rings is that a notion of shortest-path closure exists. Given a matrix $\mathbf{m}$ with elements in C, we can define its closure $\mathbf{m}^\star$ by:

$$\mathbf{m}^\star_{ij} \stackrel{\text{def}}{=} \bigoplus_{m \geq 1,\, \langle i = i_1, \ldots, i_m = j \rangle} \left(\bigodot_{k=1}^{m-1} \mathbf{m}_{i_k i_{k+1}} \right) .$$

According to [CLR90, § 26.4], $\mathbf{m}^\star$ can be computed by the following algorithm derived from the Floyd–Warshall algorithm:

$$\begin{cases} \mathbf{m}^0_{ij} & \stackrel{\text{def}}{=} & \begin{cases} \mathbf{m}_{ij} & \text{if } i \neq j \\ (\overline{1} \oplus \mathbf{m}_{ij}) & \text{if } i = j \end{cases} \\ \mathbf{m}^{k+1}_{ij} & \stackrel{\text{def}}{=} & \mathbf{m}^k_{ij} \oplus (\mathbf{m}^k_{ik} \odot (\mathbf{m}^k_{kk})^* \odot \mathbf{m}^k_{kj}) \quad \forall\, 0 \leq i, j, k \leq n \\ \mathbf{m}^\star & \stackrel{\text{def}}{=} & \mathbf{m}^{n+1} \end{cases}$$

where c^* for $c \in C$ is defined as follows:

$$c^* = \overline{1} \oplus c \oplus (c \odot c) \oplus (c \odot c \odot c) \oplus \cdots$$

Examples. The numerical set $\overline{\mathbb{I}} \stackrel{\text{def}}{=} \mathbb{I} \cup \{+\infty\}$ we have used in the zone and the octagon domains is almost a closed half-ring. In order for the sum $\oplus$ of arbitrary sets to be defined, we need to take $\mathbb{I} \in \{\mathbb{Z}, \mathbb{R}\}$ and add a $-\infty$ element as well as a $+\infty$ element:

$$\begin{array}{llll} C & \stackrel{\text{def}}{=} & \mathbb{I} \cup \{+\infty, -\infty\} & \\ \oplus & \stackrel{\text{def}}{=} & \min & \overline{0} \stackrel{\text{def}}{=} +\infty \\ \odot & \stackrel{\text{def}}{=} & + & \overline{1} \stackrel{\text{def}}{=} 0 \end{array}$$

$\mathbf{m}^\star$ corresponds to a slightly modified shortest-path closure where $-\infty$ coefficients appear whenever there exists a cycle with a strictly negative total weight.

Another classical example is the boolean half-ring:

$$\begin{array}{llll} C & \stackrel{\text{def}}{=} & \{\mathbf{T}, \mathbf{F}\} & \\ \oplus & \stackrel{\text{def}}{=} & \vee & \overline{0} \stackrel{\text{def}}{=} \mathbf{F} \\ \odot & \stackrel{\text{def}}{=} & \wedge & \overline{1} \stackrel{\text{def}}{=} \mathbf{T} \end{array}$$

A matrix $\mathbf{m}$ in the boolean half-ring effectively encodes a relation between elements of $\mathcal{V}$. Its closure $\mathbf{m}^\star$ corresponds to the transitive closure of the relation.

Bases as Closed Half-Rings. Given a basis $\mathcal{B}$, we would like to compute the shortest-path closure using the operators $\odot \stackrel{\text{def}}{=} \mathbf{+}^\sharp$ and $\oplus \stackrel{\text{def}}{=} \cap^\sharp_{\mathcal{B}}$ so that it corresponds to making explicit all implicit constraints. The neutral element for $\cap^\sharp_{\mathcal{B}}$ must be such that $\gamma_{\mathcal{B}}(\overline{0}) = \mathbb{I}$. Unfortunately, by Def. 5.2.4.4, $\forall a$, $\overline{0} \odot a = \overline{0}$ which implies $\overline{0}\ \mathbf{+}^\sharp \perp^\sharp_{\mathcal{B}} = \overline{0}$, and so, our basis must abstract $\{\ a + b \mid a \in \mathbb{I},\ b \in \emptyset\ \} = \emptyset$ by $\mathbb{I}$ which is not natural. As encountering $\perp^\sharp_{\mathcal{B}}$ while combining constraints corresponds to discovering an infeasible path in the matrix, $\perp^\sharp_{\mathcal{B}}$ should be absorbing — *i.e.*, $\overline{0}\ \mathbf{+}^\sharp \perp^\sharp_{\mathcal{B}} = \perp^\sharp_{\mathcal{B}}$ — so that the contradiction appears explicitly in the closed matrix. As a consequence, the kind of closure computed by closed half-rings does not correspond to the constraint propagation we seek. Moreover, the theory of closed half-rings can only help us in the propagation of adjacent constraints but does not guarantee that such propagations are sufficient to get the tightest possible bounds — recall that the octagon abstract domain required the combination of non-adjacent constraints, as well as tightening in the integer case. As a consequence, we will need to adapt the algebraic structure a little bit for our purposes.

5.2.3 Acceptable Bases

We introduce the concept of *acceptable basis*, that is, sets $\mathcal{B}$ such that:

Definition 5.2.5. Acceptable basis.

An acceptable basis $\mathcal{B}$ is a basis in the sense of Def. 2.4.2 such that:

1. *Arbitrary sets $B \subseteq \mathcal{B}$ have an exact abstract intersection $\cap^\sharp_{\mathcal{B}} B$:*

$$\gamma_{\mathcal{B}}(\cap^\sharp_{\mathcal{B}} B) = \bigcap \{ \gamma_{\mathcal{B}}(b) \mid b \in B \} \ .$$

   Note that $\cap^\sharp_{\mathcal{B}}$ may be different from the optional greatest lower bound, denoted by $\sqcap^\sharp_{\mathcal{B}}$.

2. *$\mathbf{+}^\sharp$ is exact: $\gamma_{\mathcal{B}}(X^\sharp \mathbf{+}^\sharp Y^\sharp) = \{ x + y \mid x \in \gamma_{\mathcal{B}}(X^\sharp), y \in \gamma_{\mathcal{B}}(Y^\sharp) \}$.*

3. *$\mathbf{-}^\sharp$ is exact: $\gamma_{\mathcal{B}}(\mathbf{-}^\sharp X^\sharp) = \{ -x \mid x \in \gamma_{\mathcal{B}}(X^\sharp) \}$.*

4. *Singletons are abstracted exactly: $\gamma_{\mathcal{B}}([a,a]^\sharp) = \{a\}$.*

5. *Given a finite set $B \subseteq \mathcal{B}$,*

$$\gamma_{\mathcal{B}}\left(\bigcap^\sharp_{\mathcal{B}} B\right) = \emptyset \implies \exists b, b' \in B,\ \gamma_{\mathcal{B}}(b \cap^\sharp_{\mathcal{B}} b') = \emptyset \ .$$

6. *Given an arbitrary set $B \subseteq \mathcal{B}$,*

$$\gamma_{\mathcal{B}}\left(\bigcap^\sharp_{\mathcal{B}} B\right) \neq \emptyset \implies \forall b \in \mathcal{B},\ \gamma_{\mathcal{B}}\left(\bigcap^\sharp_{\mathcal{B}} \{ b \mathbf{+}^\sharp b' \mid b' \in B \}\right) = \gamma_{\mathcal{B}}\left(b \mathbf{+}^\sharp \left(\bigcap^\sharp_{\mathcal{B}} B\right)\right) \ .$$

●

As this definition is quite complex, it deserves some comments:

- First of all, remark that $\gamma_{\mathcal{B}}$ may not be injective. We introduce the following equivalence relation $\simeq$:

$$X^\sharp \simeq Y^\sharp \iff^{\text{def}} \gamma_{\mathcal{B}}(X^\sharp) = \gamma_{\mathcal{B}}(Y^\sharp) \ .$$

- The exactness of $\mathbf{-}^\sharp$ in Def. 5.2.5.3 allows redefining the coherence as follows:

$$\mathbf{m} \text{ is coherent} \iff \forall i, j,\ \mathbf{m}_{ij} \simeq \mathbf{-}^\sharp\, \mathbf{m}_{ji} \ .$$

- Defs. 5.2.5.1–2 state the existence of exact abstractions $\cap^\sharp_{\mathcal{B}}$ and $\mathbf{+}^\sharp$ for $\cap$ and $\mathbf{+}$. This ensures that constraint propagation in the abstract is as precise as in the concrete:

$$\gamma_{\mathcal{B}}(\mathbf{m}_{ij} \ \cap^\sharp_{\mathcal{B}} \ (\mathbf{m}_{ik} \mathbf{+}^\sharp \mathbf{m}_{kj})) = \gamma_{\mathcal{B}}(\mathbf{m}_{ij}) \ \cap \ \{ x + y \mid x \in \gamma_{\mathcal{B}}(\mathbf{m}_{ik}), y \in \gamma_{\mathcal{B}}(\mathbf{m}_{kj}) \} \ .$$

- Def. 5.2.5.1 requires that the abstract intersection of arbitrary many elements exists so that we can actually define the shortest path from i to j as an intersection over the infinite set of paths from i to j:

$$\bigcap^{\sharp}_{\mathcal{B}\ m\geq 1,\,\langle i=i_1,\ldots,i_m=j\rangle} (\mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}) \ .$$

  As this requirement is quite strong, we also propose the following, relaxed yet sufficient, alternate definition of an acceptable basis:

  Definition 5.2.6. Acceptable basis revisited.

  In Def. 5.2.5, point 1 can be weakened into: for each finite *set $C \subseteq \mathcal{B}$ and for each finite or infinite subset D of $\{\ c_1 \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} c_m \mid m \geq 1,\ c_1, \ldots, c_m \in C\ \}$, $\cap^{\sharp}_{\mathcal{B}} D$ exists and is such that: $\gamma_{\mathcal{B}}(\cap^{\sharp}_{\mathcal{B}} D) = \bigcap \{\ \gamma_{\mathcal{B}}(d) \mid d \in D\ \}$.*

  ●

  By taking $C = \{\ \mathbf{m}_{ij} \mid \forall i, j\ \}$, Def. 5.2.6 implies the existence of the intersection of sums of constraints along paths, for arbitrary sets of paths in $\mathbf{m}$.

- Def. 5.2.5.4 ensures that $[0,0]^{\sharp}$ exists and corresponds to $\{0\}$. It is a neutral element for $\mathbin{+^{\sharp}}$, up to $\gamma_{\mathcal{B}}$. It will also be necessary for our saturation proof in Thm. 5.2.1 to consider the exact abstraction of arbitrary singletons.

- Def. 5.2.5.6 replaces the distributivity of $\odot$ over infinite sums $\oplus$ with a weaker version which is compatible with $\forall X^{\sharp} \in \mathcal{B}$, $\bot^{\sharp}_{\mathcal{B}} \mathbin{+^{\sharp}} X^{\sharp} \simeq \bot^{\sharp}_{\mathcal{B}}$. Indeed, one may find, for instance, $B \subseteq \mathcal{B}$ such that $\bigcap^{\sharp}_{\mathcal{B}} B \simeq \bot^{\sharp}_{\mathcal{B}}$ but $\bigcap^{\sharp}_{\mathcal{B}} \{\ b \mathbin{+^{\sharp}} \top^{\sharp}_{\mathcal{B}} \mid b \in B\ \} \not\simeq \bot^{\sharp}_{\mathcal{B}}$. In this case $\left(\bigcap^{\sharp}_{\mathcal{B}} B\right) \mathbin{+^{\sharp}} \top^{\sharp}_{\mathcal{B}} \simeq \bot^{\sharp}_{\mathcal{B}} \not\simeq \bigcap^{\sharp}_{\mathcal{B}} \{\ b \mathbin{+^{\sharp}} \top^{\sharp}_{\mathcal{B}} \mid b \in B\ \}$, so, $\cap^{\sharp}_{\mathcal{B}}$ cannot distribute $\mathbin{+^{\sharp}}$ up to $\gamma_{\mathcal{B}}$.

- Finally, remark that hypotheses Defs. 5.2.5.5–6 are very strong. Arbitrary subsets of $\mathcal{P}(\mathbb{I})$ are not likely to verify them. Sect. 5.4 will present a few acceptable bases as well as bases that form classical non-relational domains but are not acceptable.

Some hypotheses that do not seem to be relevant here will in fact be crucial in the proof of the theorems related to the closure. Unfortunately, they do not always have a very intuitive meaning: they should be considered as mandatory technicalities. As these hypotheses were stated specifically in order for our proofs to work out, they are sufficient but certainly not necessary conditions; it is probably possible to trade some hypotheses for new ones, at the expense of non-trivial changes in our proofs.

5.2.4 Adapted Closure Algorithm

Our closure algorithm is exactly the Floyd–Warshall algorithm used in Def. 3.3.2 for zones, starting from column and line number 0, except that we have redefined the meaning of min, +, and 0 as $\cap^\sharp_{\mathcal{B}}$, $+^\sharp$, and $[0,0]^\sharp$. It is much simpler than the algorithm presented in Sect. 5.2.2 for regular closed half-rings. We will denote its result on the matrix $\mathbf{m}$ by $\mathbf{m}^\star$:

Definition 5.2.7. Floyd–Warshall algorithm for constraint matrices $\star$.

$$\begin{cases} \mathbf{m}^0 & \stackrel{\text{def}}{=} & \mathbf{m} \\ \mathbf{m}^{k+1}_{ij} & \stackrel{\text{def}}{=} & \mathbf{m}^k_{ij} \cap^\sharp_{\mathcal{B}} (\mathbf{m}^k_{ik} +^\sharp \mathbf{m}^k_{kj}) \qquad \forall\, 0 \leq i,j,k \leq n \\ \mathbf{m}^\star_{ij} & \stackrel{\text{def}}{=} & \begin{cases} \mathbf{m}^{n+1}_{ij} & \text{if } i \neq j \\ [0,0]^\sharp & \text{if } i = j \end{cases} \end{cases}$$

●

We retrieve the classical emptiness test, using the diagonal elements of $\mathbf{m}^{n+1}$, as well as the closure, normal form, and saturation properties of $\mathbf{m}^\star$. They are but a little complicated by the possible non-injectivity of $\gamma_{\mathcal{B}}$:

Theorem 5.2.1. Properties of the Floyd–Warshall algorithm for constraint matrices.

1. $\gamma^{Weak}(\mathbf{m}^\star) = \gamma^{Weak}(\mathbf{m})$.
2. $\gamma^{Weak}(\mathbf{m}) = \emptyset \iff \exists i,\ 0 \notin \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ii})$.
3. *If* $\gamma^{Weak}(\mathbf{m}^\star) \neq \emptyset$, *then:*
   - $\mathbf{m}^\star$ *is coherent,*
   - $\mathbf{m}^\star$ *is the* transitive closure *of* $\mathbf{m}$, *up to* $\gamma_{\mathcal{B}}$*:*

$$\forall i,j,\ \mathbf{m}^\star_{ij} \simeq \bigcap^\sharp_{\substack{\mathcal{B} \\ \langle i=i_1,\ldots,i_m=j\rangle}} \mathbf{m}_{i_1 i_2} +^\sharp \cdots +^\sharp \mathbf{m}_{i_{m-1} i_m}$$

   - *all constraints in* $\mathbf{m}^\star$ saturate $\gamma^{Weak}(\mathbf{m})$*:*

$$\begin{array}{l} \forall i,j,\ \forall c \in \gamma_{\mathcal{B}}(\mathbf{m}^\star_{ij}), \\ \exists \vec{v} \in \mathbb{I}^{n+1} \text{ such that } v_0 = 0,\ (v_1,\ldots,v_n) \in \gamma^{Weak}(\mathbf{m}),\ \text{and } v_j - v_i = c \end{array}$$

   - $\mathbf{m}^\star$ *is a normal form, up to* $\gamma_{\mathcal{B}}$*:*

$$\forall i,j,\ \gamma_{\mathcal{B}}(\mathbf{m}^\star_{ij}) = \inf_{\subseteq} \{\ \gamma_{\mathcal{B}}(\mathbf{n}_{ij}) \mid \gamma^{Weak}(\mathbf{m}) = \gamma^{Weak}(\mathbf{n})\ \}$$

- *the closure has the following* local characterisation*:*

$$\mathbf{m} = \mathbf{m}^\star \iff \begin{cases} \forall i,j,k, & \gamma_{\mathcal{B}}(\mathbf{m}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}_{ik} \mathbin{+^\sharp} \mathbf{m}_{kj}) \\ \forall i, & \gamma_{\mathcal{B}}(\mathbf{m}_{ii}) = \{0\} \end{cases}$$

●

Proof.

The proof is quite complex, and so, it is postponed to the appendix, in Sect. A.2, and we only give here a proof sketch.

The proof is similar to that of the classical Floyd–Warshall algorithm for potential constraints — Thm. 3.3.5. We prove, by induction on k, that, for all i, j:

$$\gamma_{\mathcal{B}}(\mathbf{m}^k_{ij}) \supseteq \bigcap_{\langle i=i_1,\ldots,i_m=j\rangle} \gamma_{\mathcal{B}}(\mathbf{m}_{i_1 i_2} \mathbin{+^\sharp} \cdots \mathbin{+^\sharp} \mathbf{m}_{i_{m-1} i_m})$$

$$\gamma_{\mathcal{B}}(\mathbf{m}^k_{ij}) \subseteq \bigcap_{\substack{\langle i = i_1, \ldots, i_m = j\rangle \\ \text{simple path such that} \\ \forall 1 < l < m,\, i_l < k}} \gamma_{\mathcal{B}}(\mathbf{m}_{i_1 i_2} \mathbin{+^\sharp} \cdots \mathbin{+^\sharp} \mathbf{m}_{i_{m-1} i_m})$$

Thanks to the exactness of $+^\sharp$, each $\gamma_{\mathcal{B}}(\mathbf{m}_{i_1 i_2} \mathbin{+^\sharp} \cdots \mathbin{+^\sharp} \mathbf{m}_{i_{m-1} i_m})$ is well-defined independently from any evaluation order, although $\mathbf{m}_{i_1 i_2} \mathbin{+^\sharp} \cdots \mathbin{+^\sharp} \mathbf{m}_{i_{m-1} i_m}$ may depend upon the chosen evaluation order. Then, we prove that, when $k = n$, these upper and lower bounds are indeed equal. This requires the distributivity property Def. 5.2.5.6. The requirement that singletons are exactly representable is used when proving the saturation property by induction on the matrix size n. Finally, Def. 5.2.5.5 is used to prove the emptiness test.

○

Incremental Closure. Thanks to the local characterisation of closed constraint matrices, it is possible to derive, as for zones — see Def. 3.3.4 — and octagons — see Def. 4.3.4 — an incremental version of the modified Floyd–Warshall algorithm. If $\mathbf{m}$ is closed and $\mathbf{n}$ equals $\mathbf{m}$ except for the last $n - c$ lines and columns, we can compute the closure $\mathbf{n}^\star$ of $\mathbf{n}$ using the following algorithm.

Definition 5.2.8. Incremental Floyd–Warshall algorithm for constraint matrices.

$$\begin{cases} \mathbf{n}^0 & \stackrel{\text{def}}{=} & \mathbf{n} \\ \mathbf{n}^{k+1}_{ij} & \stackrel{\text{def}}{=} & \begin{cases} \mathbf{n}^k_{ij} & \textit{if } 0 \leq i,j,k \leq c \\ \mathbf{n}^k_{ij} \cap^\sharp_{\mathcal{B}} (\mathbf{n}^k_{ik} +^\sharp \mathbf{n}^k_{kj}) & \textit{otherwise} \end{cases} \\ \mathbf{n}^\bigstar_{ij} & \stackrel{\text{def}}{=} & \begin{cases} \mathbf{n}^{n+1}_{ij} & \textit{if } i \neq j \\ [0,0]^\sharp & \textit{if } i = j \end{cases} \end{cases}$$

●

By virtually reordering the variables, this algorithm can be extended to the case where only lines and columns at indexes i_1 to i_{n-c} have been modified. We denote by $Inc^\bigstar_{i_1,\ldots,i_{n-c}}(\mathbf{n})$ the resulting matrix. It is computed in time proportional to $(n+1)^3 - (c+1)^3$.

Extending the Normal Form. The transitive closure and local closure characterisation properties of Thm. 5.2.1.3 are only valid when $\gamma^{\mathit{Weak}}(\mathbf{m}) \neq \emptyset$. However, the saturation and normal form properties can be extended to all constraint matrices by slightly modifying the definition of the closure $\bigstar$ of matrices representing an empty set:

$$\mathbf{m}^\bigstar \stackrel{\text{def}}{=} \bot^{\mathit{Weak}} \quad \text{if } \exists i,\ 0 \notin \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ii}) \text{ in Def. 5.2.7 .}$$

In-Place Implementation. It is possible to design an in-place version of the closure and incremental closure algorithms, in the spirit of Def. 3.3.3. Such algorithms would be easier to implement, but harder to reason about.

5.2.5 Galois Connection

We required, in Def. 5.2.5, $\gamma_{\mathcal{B}}$ to be a complete $\cap^\sharp_{\mathcal{B}}$–morphism so that we can compute an exact abstraction of the shortest-path closure by the Floyd–Warshall algorithm of Def. 5.2.7. However, $\cap^\sharp_{\mathcal{B}}$ may be distinct from $\sqcap^\sharp_{\mathcal{B}}$ and compute an exact abstraction of the concrete intersection $\cap$ that is not the smallest one with respect the abstract order on $\mathcal{B}$. This can happen when $\gamma_{\mathcal{B}}$ is not injective.

If we suppose, moreover, that $\gamma_{\mathcal{B}}$ is a complete $\sqcap^\sharp_{\mathcal{B}}$–morphism, then, by Thm. 2.2.1, there exists a canonical abstraction $\alpha_{\mathcal{B}}$ from $\mathcal{P}(\mathbb{I})$ to $\mathcal{B}$

$$\alpha_{\mathcal{B}}(X) \stackrel{\text{def}}{=} \bigsqcap^\sharp_{\mathcal{B}} \{\ Y \mid X \sqsubseteq^\sharp_{\mathcal{B}} \gamma_{\mathcal{B}}(Y)\ \}$$

such that $(\alpha_{\mathcal{B}}, \gamma_{\mathcal{B}})$ is a Galois connection. $\alpha_{\mathcal{B}}$ can then be "lifted" to form an abstraction function from $\mathcal{P}(\mathbb{I}^n)$ to *Weak* as follows:

$$\left(\alpha^{\mathit{Weak}}(S)\right)_{ij} \stackrel{\text{def}}{=} \alpha_{\mathcal{B}}\left(\{\ v_j - v_i \mid v_0 = 0,\ (v_1,\ldots,v_n) \in S\ \}\right) .$$

$\alpha^{Weak} \circ \gamma^{Weak}$ is then a normalisation function on *Weak*. As seen in Thm. 5.2.1.3, $\bigstar$ is also a kind of normalisation, but only "up to $\gamma_{\mathcal{B}}$", that is:

$$\forall i, j,\ \Big((\alpha^{Weak} \circ \gamma^{Weak})(\mathbf{m})\Big)_{ij} \simeq \mathbf{m}^{\bigstar}_{ij} \ .$$

When $\gamma_{\mathcal{B}}$ is a not complete $\sqcap^{\sharp}_{\mathcal{B}}-$morphism — because Def. 5.2.6 is used instead of Def. 5.2.5 or $\sqcap^{\sharp}_{\mathcal{B}}$ is distinct from $\cap^{\sharp}_{\mathcal{B}}$ — we only have a *partial* Galois connection and a best abstract representation $\alpha^{Weak} \circ \gamma^{Weak}$ may not exist for all constraint matrices. Nevertheless, $\mathbf{m}^{\bigstar}$ is still well-defined and gives a best abstract representation "up to $\gamma_{\mathcal{B}}$".

5.3 Operators and Transfer Functions

Once a normal form operator $\bigstar$ enjoying the saturation property has been defined, there is not much work to do to adapt the zone abstract domain transfer functions and operators to construct a fully-featured weakly relational abstract domain *Weak*. Their time complexity is dominated by that of the modified closure algorithm that sometimes must be performed on their arguments, and so, their complexity is cubic in the worst case.

5.3.1 Set-Theoretic Operators

We must use our modified closure to get complete inclusion and equality tests. A best union abstraction $\cup^{Weak}$ can also be constructed using our closure operator provided that the basis $\mathcal{B}$ has a best union abstraction $\cup^{\sharp}_{\mathcal{B}}$; moreover, $\cup^{Weak}$ preserves the closure up to $\gamma_{\mathcal{B}}$.

Definition 5.3.1. Set-theoretic operators on constraint matrices.

$$(\mathbf{m} \cap^{Weak} \mathbf{n})_{ij} \stackrel{\text{def}}{=} \mathbf{m}_{ij} \cap^{\sharp}_{\mathcal{B}} \mathbf{n}_{ij} \ .$$

$$(\mathbf{m} \cup^{Weak} \mathbf{n})_{ij} \stackrel{\text{def}}{=} (\mathbf{m}^{\bigstar}_{ij}) \cup^{\sharp}_{\mathcal{B}} (\mathbf{n}^{\bigstar}_{ij}) \ .$$

●

Theorem 5.3.1. Properties of set-theoretic operators on constraint matrices.

1. $\gamma^{Weak}(\mathbf{m} \cap^{Weak} \mathbf{n}) = \gamma^{Weak}(\mathbf{m}) \cap \gamma^{Weak}(\mathbf{n})$. *(exact $\cap$ abstraction)*
2. $\gamma^{Weak}(\mathbf{m} \cup^{Weak} \mathbf{n}) \supseteq \gamma^{Weak}(\mathbf{m}) \cup \gamma^{Weak}(\mathbf{n})$. *($\cup$ abstraction)*
3. $\gamma^{Weak}(\mathbf{m} \cup^{Weak} \mathbf{n}) = \inf_{\subseteq}\{\ \gamma^{Weak}(\mathbf{o}) \mid \gamma^{Weak}(\mathbf{o}) \supseteq \gamma^{Weak}(\mathbf{m}) \cup \gamma^{Weak}(\mathbf{n})\ \}$
   if $\cup^{\sharp}_{\mathcal{B}}$ is the best abstraction for $\cup$. *(best $\cup$ abstraction)*
4. $\forall i, j,\ (\mathbf{m} \cup^{Weak} \mathbf{n})^{\bigstar}_{ij} \simeq (\mathbf{m} \cup^{Weak} \mathbf{n})_{ij}$
   if $\cup^{\sharp}_{\mathcal{B}}$ is the best abstraction for $\cup$. *($\cup^{Weak}$ preserves the closure)*

5. $\gamma^{Weak}(\mathbf{m}) = \gamma^{Weak}(\mathbf{n}) \iff \forall i,j,\ \mathbf{m}^{\bigstar}_{ij} \simeq \mathbf{n}^{\bigstar}_{ij}$. *(equality testing)*

6. $\gamma^{Weak}(\mathbf{m}) \subseteq \gamma^{Weak}(\mathbf{n}) \iff \forall i,j,\ \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{n}_{ij})$. *(inclusion testing)*

●

Proof.

1. This is an easy consequence of the exactness of $\cap^{\sharp}_{\mathcal{B}}$: $\gamma_{\mathcal{B}}(X^{\sharp} \cap^{\sharp}_{\mathcal{B}} Y^{\sharp}) = \gamma_{\mathcal{B}}(X^{\sharp}) \cap \gamma_{\mathcal{B}}(X^{\sharp})$.

2. This is an easy consequence of the fact that $\cup^{\sharp}_{\mathcal{B}}$ is a sound abstraction for $\cup$: $\gamma_{\mathcal{B}}(X^{\sharp} \cup^{\sharp}_{\mathcal{B}} Y^{\sharp}) \supseteq \gamma_{\mathcal{B}}(X^{\sharp}) \cup \gamma_{\mathcal{B}}(Y^{\sharp})$ and the fact that $\gamma^{Weak}(\mathbf{m}^{\bigstar}) = \gamma^{Weak}(\mathbf{m})$.

3. By the preceding point, $\gamma^{Weak}(\mathbf{m} \cup^{Weak} \mathbf{n}) \supseteq \inf_{\subseteq} \{\ \gamma^{Weak}(\mathbf{o}) \mid \gamma^{Weak}(\mathbf{o}) \supseteq \gamma^{Weak}(\mathbf{m}) \cup \gamma^{Weak}(\mathbf{n})\ \}$. For the converse inequality, suppose that $\mathbf{o}$ is such that $\gamma^{Weak}(\mathbf{o}) \supseteq \gamma^{Weak}(\mathbf{m}) \cup \gamma^{Weak}(\mathbf{n})$. We prove that, for all i and j, $\gamma_{\mathcal{B}}(\mathbf{o}_{ij}) \supseteq \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij} \cup^{\sharp}_{\mathcal{B}} \mathbf{n}^{\bigstar}_{ij})$ which implies the desired result. Because $\cup^{\sharp}_{\mathcal{B}}$ is the best abstraction for $\cup$, $\gamma_{\mathcal{B}}(X^{\sharp} \cup^{\sharp}_{\mathcal{B}} Y^{\sharp}) = \inf_{\subseteq} \{\ S \in \gamma^{\sharp}_{\mathcal{B}}(\mathcal{B}) \mid S \supseteq \gamma_{\mathcal{B}}(X^{\sharp}) \cup \gamma_{\mathcal{B}}(Y^{\sharp})\ \}$, and so, it is sufficient to prove that $\gamma_{\mathcal{B}}(\mathbf{o}_{ij}) \supseteq \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij}) \cup \gamma_{\mathcal{B}}(\mathbf{n}^{\bigstar}_{ij})$. Consider $c \in \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij}) \cup \gamma_{\mathcal{B}}(\mathbf{n}^{\bigstar}_{ij})$. Either $c \in \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij})$ or $c \in \gamma_{\mathcal{B}}(\mathbf{n}^{\bigstar}_{ij})$. We consider only the case $c \in \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij})$ as the other case is symmetric. By the saturation property of Thm. 5.2.1.3, we have $(v_0, , \ldots, v_n)$ such that $v_0 = 0$, $(v_1, \ldots, v_n) \in \gamma^{Weak}(\mathbf{m})$ and $v_j - v_i = c$. By definition of $\mathbf{o}$, we also have $(v_1, \ldots, v_n) \in \gamma^{Weak}(\mathbf{o})$, so $v_j - v_i \in \gamma_{\mathcal{B}}(\mathbf{o}_{ij})$, and so, $c \in \gamma_{\mathcal{B}}(\mathbf{o}_{ij})$ which concludes the proof.

4. Obviously, for all i and j, $\gamma_{\mathcal{B}}((\mathbf{m} \cup^{Weak} \mathbf{n})^{\bigstar}_{ij}) \subseteq \gamma_{\mathcal{B}}((\mathbf{m} \cup^{Weak} \mathbf{n})_{ij})$.

   To prove the converse inequality, recall that we have proved in the preceding point that $\forall i,j,\ \gamma_{\mathcal{B}}(\mathbf{o}_{ij}) \supseteq \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij} \cup^{\sharp}_{\mathcal{B}} \mathbf{n}^{\bigstar}_{ij})$ whenever $\gamma^{Weak}(\mathbf{o}) \supseteq \gamma^{Weak}(\mathbf{m}) \cup \gamma^{Weak}(\mathbf{n})$. The desired result is obtained by choosing $\mathbf{o} \stackrel{\text{def}}{=} (\mathbf{m} \cup^{Weak} \mathbf{n})^{\bigstar}$.

5. This is an easy consequence of the normal form property of Thm. 5.2.1.3.

6. Using the normal form property of Thm. 5.2.1.3, we have $\forall i,j,\ \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{n}^{\bigstar}_{ij}) \iff \gamma^{Weak}(\mathbf{m}) \subseteq \gamma^{Weak}(\mathbf{n})$. We conclude by remarking that, on the one hand $\forall i,j,\ \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{n}_{ij}) \Longrightarrow \gamma^{Weak}(\mathbf{m}) \subseteq \gamma^{Weak}(\mathbf{n})$ and on the other hand, as $\forall i,j,\ \gamma_{\mathcal{B}}(\mathbf{n}^{\bigstar}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{n}_{ij})$, $\forall i,j,\ \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{n}^{\bigstar}_{ij}) \Longrightarrow \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{n}_{ij})$.

○

5.3.2 Forget and Projection Operators

Forget Operators. The non-deterministic assignment $\{\!|\, V_f \leftarrow ?\,|\!\}^{Weak}$ simply amounts to forgetting the row and column associated to the variable V_f. If the matrix argument is closed, then we have an exact abstract transfer function and the resulting matrix is closed.

If the matrix argument is not closed, we may lose precision. We propose also an alternate forget operator $\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}_{alt}$ that does not require a closed argument, but has a quadratic cost instead of a linear one. These operators are inspired from the forget operators on the zone abstract domain presented in Defs. 3.6.1 and 3.6.2. The main difference is that we take care not to replace elements $\mathbf{m}_{ij}$ such that $\gamma_{\mathcal{B}}(\mathbf{m}_{ij}) = \emptyset$ with $\top^\sharp_{\mathcal{B}}$ as this could destroy the information that $\gamma^{Weak}(\mathbf{m}) = \emptyset$. The resulting operators are naturally strict:

Definition 5.3.2. Forget operators on constraint matrices.

1. $$(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} & \textit{if } i \neq f \textit{ and } j \neq f \\ \mathbf{m}_{ij} & \textit{if } i = j = f \\ \mathbf{m}_{ij} & \textit{otherwise if } \mathbf{m}_{ij} \simeq \bot^\sharp_{\mathcal{B}} \\ \top^\sharp_{\mathcal{B}} & \textit{otherwise} \end{cases}$$

2. $$(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}_{alt}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} \cap^\sharp_{\mathcal{B}} (\mathbf{m}_{if} +^\sharp \mathbf{m}_{fj}) & \textit{if } i \neq f,\ j \neq f \\ \mathbf{m}_{ij} & \textit{if } i = j = f \\ \mathbf{m}_{ij} & \textit{otherwise if } \mathbf{m}_{ij} \simeq \bot^\sharp_{\mathcal{B}} \\ \top^\sharp_{\mathcal{B}} & \textit{otherwise} \end{cases}$$

●

Theorem 5.3.2. Soundness and exactness of $\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}$ and $\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}_{alt}$.

1. $\gamma^{Weak}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m})) \supseteq \{\ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Weak}(\mathbf{m})\ \}$.
2. $\gamma^{Weak}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m}^\bigstar)) = \{\ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Weak}(\mathbf{m})\ \}$.
3. $\gamma^{Weak}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}_{alt}(\mathbf{m})) = \{\ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Weak}(\mathbf{m})\ \}$.
4. $\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m})$ *is closed whenever* $\mathbf{m}$ *is.*

●

Proof.

1. The property is obvious if there is some element in $\mathbf{m}$ representing $\emptyset$ as such elements are preserved in $\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m})$, which gives $\gamma^{Weak}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m})) = \gamma^{Weak}(\mathbf{m}) = \emptyset$. We now consider the case where $\forall i, j,\ \gamma_{\mathcal{B}}(\mathbf{m}_{ij}) \neq \emptyset$.

   Let us take $t \in \mathbb{I}$ and $\vec{v} = (v_1, \ldots, v_n) \in \gamma^{Weak}(\mathbf{m})$. We want to prove that $\vec{v}\,' = (v_1, \ldots, v_{f-1}, t, v_{f+1}, \ldots, v_n) \in \gamma^{Weak}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m}))$, that is to say, if we denote by v'_k the $k-$th coordinates of $\vec{v}\,'$ and state that $v_0 \stackrel{\text{def}}{=} v'_0 \stackrel{\text{def}}{=} 0,\ \forall i, j \geq 0,\ v'_j - v'_i \in \gamma_{\mathcal{B}}\left((\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m}))_{ij}\right)$.

   - If $i, j \neq f$, we have $v'_j - v'_i = v_j - v_i \in \gamma_{\mathcal{B}}(\mathbf{m}_{ij}) = \gamma_{\mathcal{B}}\left((\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m}))_{ij}\right)$.

- If $i = j = f$, we also have $v'_j - v'_i = v_j - v_i = 0$ and $\mathbf{m}_{ij} = (\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m}))_{ij}$.
- If $i = f$ or $j = f$ but not both, then $v'_j - v'_i \in \mathbb{I} = \gamma_{\mathcal{B}}\left((\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m}))_{ij}\right)$.

2. By the first point, $\gamma^{Weak}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m}^\star)) \supseteq \{\ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Weak}(\mathbf{m}^\star)\ \} = \{\ \vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Weak}(\mathbf{m})\ \}$, so, we only need to prove the converse inclusion. The case when $\mathbf{m}^\star = \bot^{Weak}$ is obvious, so we will consider only the case $\mathbf{m}^\star \neq \bot^{Weak}$, which implies $\forall i, j,\ \gamma_{\mathcal{B}}(\mathbf{m}^\star_{ij}) \neq \emptyset$.

   Let us take $\vec{v} = (v_1, \ldots, v_n) \in \gamma^{Weak}(\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m}^\star))$. We want to prove that there exists a t such that $\vec{v}\,' = (v_1, \ldots, v_{f-1}, t, v_{f+1}, \ldots, v_n) \in \gamma^{Weak}(\mathbf{m})$.

   Let us first prove that, provided that $v_0 \stackrel{\text{def}}{=} 0$, we have:

$$\bigcap^\sharp_{\substack{\mathcal{B} \\ i \neq f}} ([v_i, v_i]^\sharp \mathbin{+^\sharp} \mathbf{m}^\star_{if}) \not\simeq \bot^\sharp_{\mathcal{B}}\ .$$

   Suppose that this is not the case, then, by Def. 5.2.5.5, there exists two abstract elements $X^\sharp = [v_i, v_i]^\sharp \mathbin{+^\sharp} \mathbf{m}^\star_{if}$ and $Y^\sharp = [v_j, v_j]^\sharp \mathbin{+^\sharp} \mathbf{m}^\star_{jf}$, with $i, j \neq f$, in this big intersection such that $X^\sharp \cap^\sharp_{\mathcal{B}} Y^\sharp \not\simeq \bot^\sharp_{\mathcal{B}}$. By exactness of $\cap^\sharp_{\mathcal{B}}$, $[c, c]^\sharp$, $\mathbin{-^\sharp}$, and $\mathbin{+^\sharp}$, this gives:

$$\{\ v_i + a \mid a \in \gamma_{\mathcal{B}}(\mathbf{m}^\star_{if})\ \} \cap \{\ v_j + b \mid b \in \gamma_{\mathcal{B}}(\mathbf{m}^\star_{jf})\ \} = \emptyset$$

   that is:

$$v_j - v_i \notin \gamma_{\mathcal{B}}(\mathbf{m}^\star_{if} \mathbin{-^\sharp} \mathbf{m}^\star_{jf})$$

   which is absurd because:

$$\begin{array}{rcll} v_j - v_i & \in & \gamma_{\mathcal{B}}\left((\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m}^\star))_{ij}\right) & \\ & = & \gamma_{\mathcal{B}}(\mathbf{m}^\star_{ij}) & \text{(by definition of } \{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}) \\ & \subseteq & \gamma_{\mathcal{B}}(\mathbf{m}^\star_{if} \mathbin{+^\sharp} \mathbf{m}^\star_{fj}) & \text{(by local closure)} \\ & = & \gamma_{\mathcal{B}}(\mathbf{m}^\star_{if} \mathbin{-^\sharp} \mathbf{m}^\star_{jf}) & \text{(by coherence)} \end{array}$$

   So, there exists at least one t in $\gamma_{\mathcal{B}}\left(\bigcap^\sharp_{\substack{\mathcal{B} \\ i \neq f}} ([v_i, v_i]^\sharp \mathbin{+^\sharp} \mathbf{m}^\star_{if})\right)$.

   We now prove that any such t is a good choice, that is to say, $\vec{v}\,' \in \gamma^{Weak}(\mathbf{m})$. We will denote by v'_k the $k-$th coordinate of $\vec{v}\,'$ and state that $v'_0 \stackrel{\text{def}}{=} 0$.

   - For all $i \neq f$ and $j \neq f$, $v'_j - v'_i = v_j - v_i \in \gamma_{\mathcal{B}}\left((\{\!\!\{ V_f \leftarrow ? \}\!\!\}^{Weak}(\mathbf{m}^\star))_{ij}\right) = \gamma_{\mathcal{B}}(\mathbf{m}^\star_{ij})$.

- This equalities also holds when $i = j = f$, as $v'_j - v'_i = v_j - v_i = 0$ and $(\{\!\{ V_f \leftarrow ? \}\!\}^{Weak}(\mathbf{m}^\star))_{ff} = \mathbf{m}^\star_{ff}$.
- If $i = f$ but $j \neq f$, then, we have $t \in \gamma_{\mathcal{B}}([v_j, v_j]^\sharp +^\sharp \mathbf{m}^\star_{jf})$, and so, $v'_i - v'_j = t - v_j \in \gamma_{\mathcal{B}}(\mathbf{m}^\star_{ji})$.
- The case $j = f$ and $i \neq f$ is similar, by coherence.

3. We prove that $\forall i, j,\ (\{\!\{ V_f \leftarrow ? \}\!\}^{Weak}_{alt}(\mathbf{m}))^\star_{ij} \not\simeq \{\!\{ V_f \leftarrow ? \}\!\}^{Weak}(\mathbf{m}^\star)_{ij}$ which implies the desired property.

   The proof is similar to that of Thm. 3.6.2: any path between two elements distinct from f in $\{\!\{ V_f \leftarrow ? \}\!\}^{Weak}_{alt}(\mathbf{m})$ can be transformed into a path in $\mathbf{m}$ with an equal or smaller total weight, and the other way round.

4. Suppose that $\mathbf{m}$ is closed and let us denote $\{\!\{ V_f \leftarrow ? \}\!\}^{Weak}(\mathbf{m})$ by $\mathbf{n}$.

   Whenever $\mathbf{m} = \bot^{Weak}$, by strictness, we also have $\mathbf{n} = \bot^{Weak}$.

   We now consider the case $\mathbf{m} \neq \bot^{Weak}$, which implies $\forall i, j,\ \gamma_{\mathcal{B}}(\mathbf{m}_{ij}) \neq \emptyset$ because $\mathbf{m}$ is closed. We use the local characterisation of the closure: by Thm. 5.2.1.3 it is sufficient to prove that $\forall i, j, k,\ \gamma_{\mathcal{B}}(\mathbf{n}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{n}_{ik} +^\sharp \mathbf{n}_{kj})$ and $\forall i,\ \gamma_{\mathcal{B}}(\mathbf{n}_{ii}) = \{0\}$.

   Set three variables i, j, and k. If all of i, j, and k are distinct from f, then $\gamma_{\mathcal{B}}(\mathbf{n}_{ij}) = \gamma_{\mathcal{B}}(\mathbf{m}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}_{ik} +^\sharp \mathbf{m}_{kj}) = \gamma_{\mathcal{B}}(\mathbf{n}_{ik} +^\sharp \mathbf{n}_{kj})$. If $i = j = k = f$, then $\gamma_{\mathcal{B}}(\mathbf{n}_{ij}) = \gamma_{\mathcal{B}}(\mathbf{n}_{ik} +^\sharp \mathbf{n}_{kj}) = \{0\}$. In all other cases, at least one of $\gamma_{\mathcal{B}}(\mathbf{n}_{ik})$ and $\gamma_{\mathcal{B}}(\mathbf{n}_{kj})$ is $\mathbb{I}$ while none are $\emptyset$, and so, $\gamma_{\mathcal{B}}(\mathbf{n}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{n}_{ik} +^\sharp \mathbf{n}_{kj}) = \mathbb{I}$.

   As we have $\forall i,\ \mathbf{n}_{ii} = \mathbf{m}_{ii}$, $\gamma(\mathbf{n}_{ii}) = \{0\}$ is a consequence of the closure of $\mathbf{m}$.

○

Projection Operator. In order to extract the information about a single variable V_i, it is sufficient to look at the first line of the closure $\mathbf{m}^\star$ of $\mathbf{m}$:

Theorem 5.3.3. Projection operator.

$$\gamma_{\mathcal{B}}(\mathbf{m}^\star_{0i}) = \{\ v \in \mathbb{I} \mid \exists (v_1, \ldots, v_n) \in \gamma^{Weak}(\mathbf{m}),\ v_i = v\ \}\ .$$

●

Proof. This is an easy consequence of the saturation property of Thm. 5.2.1.3. ○

Thus, the top row of $\mathbf{m}^\star$, $(\mathbf{m}^\star_{01}, \ldots, \mathbf{m}^\star_{0n})$, can be seen as an abstract element in the non-relational abstract domain derived from $\mathcal{B}$. We call *NonRel* the conversion operator from *Weak* into the non-relational abstract domain constructed on $\mathcal{B}$; it is a best abstraction.

Conversely, a non-relational abstract element can be converted into a constraint matrix containing $\top^\sharp_{\mathcal{B}}$ elements everywhere except on the top row and, by coherence, the leftmost

column. We denote by $Weak(X^\sharp)$ the constraint matrix derived from the non-relational element $X^\sharp$. This conversion is exact.

5.3.3 Transfer Functions

We use the very same ideas that we used in the zone abstract domain to defined several abstract transfer functions on our weakly relational domains *Weak*.

Simple Cases. Recall that, in the zone abstract domain, assignments and backward assignments of the form $X \leftarrow Y + [a,b]$ and $X \leftarrow [a,b]$, as well as tests of the form $X - Y \leq [a,b]$ and $X \leq [a,b]$ could be modeled exactly. In a weakly relational abstract domain based on a basis $\mathcal{B}$, we can only abstract exactly assignments and backward assignments of the form $X \leftarrow Y + C$ and $X \leftarrow C$ and tests of the form $X - Y \in C$ and $X \in C$, whenever $C \subseteq \mathbb{I}$ is exactly representable in $\mathcal{B}$, *i.e.*, $\exists C^\sharp \in \mathcal{B}$, $\gamma_{\mathcal{B}}(C^\sharp) = C$. For instance, singletons are guaranteed to be exactly representable in $\mathcal{B}$, so, $X \leftarrow Y + [a,a]$ has an exact abstract counterpart, but $X \leftarrow Y + [a,b]$ when $b \neq a$ may not be exactly representable. If C is not exactly representable, we can still take any $C^\sharp \in \mathcal{B}$ that over-approximates C, that is $\gamma_{\mathcal{B}}(C^\sharp) \supseteq C$, to get a sound but not exact abstract transfer function. Note that we have taken the liberty to enrich the syntax of expressions with constant sets C that may not be intervals and to replace the comparison operators in tests by the $\in$ predicate so that expressions match more closely the expressiveness of the base $\mathcal{B}$. The following abstract transfer functions for such simple tests, assignments, and backward assignments are adapted from Defs. 3.6.3, 3.6.5, and 3.6.7 as follows:

Definition 5.3.3. Simple transfer functions for constraint matrices.

We suppose that $i_0 \neq j_0$ and $\gamma_{\mathcal{B}}(C^\sharp) \supseteq C$. We can defined the following transfer functions:

- $(\!|\, V_{j_0} \leftarrow V_{j_0} + C \,|\!)^{Weak}_{simple}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} -^\sharp C^\sharp & \text{if } i = j_0, j \neq j_0 \\ \mathbf{m}_{ij} +^\sharp C^\sharp & \text{if } i \neq j_0, j = j_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$
- $(\!|\, V_{j_0} \leftarrow C \,|\!)^{Weak}_{simple}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} -^\sharp C^\sharp & \text{if } i = j_0,\ j = 0 \\ C^\sharp & \text{if } i = 0,\ j = j_0 \\ (\!|\, V_{j_0} \leftarrow ? \,|\!)^{Weak}(\mathbf{m}^\bigstar)_{ij} & \text{otherwise} \end{cases}$
- $(\!|\, V_{j_0} \leftarrow V_{i_0} + C \,|\!)^{Weak}_{simple}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} -^\sharp C^\sharp & \text{if } i = j_0,\ j = i_0 \\ C^\sharp & \text{if } i = i_0,\ j = j_0 \\ (\!|\, V_{j_0} \leftarrow ? \,|\!)^{Weak}(\mathbf{m}^\bigstar)_{ij} & \text{otherwise} \end{cases}$

- $\{\!\!\{ V_{j_0} \in C \,?\, \}\!\!\}^{Weak}_{simple}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} \cap^\sharp_{\mathcal{B}} (-^\sharp C^\sharp) & \text{if } i = j_0, j = 0 \\ \mathbf{m}_{ij} \cap^\sharp_{\mathcal{B}} C^\sharp & \text{if } i = 0, j = j_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$

- $\{\!\!\{ V_{j_0} - V_{i_0} \in C \,?\, \}\!\!\}^{Weak}_{simple}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} \cap^\sharp_{\mathcal{B}} (-^\sharp C^\sharp) & \text{if } i = j_0, j = i_0 \\ \mathbf{m}_{ij} \cap^\sharp_{\mathcal{B}} C^\sharp & \text{if } i = i_0, j = j_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$

- $(\{\!\!\{ V_{j_0} \to V_{j_0} + C \}\!\!\}^{Weak}_{simple}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} +^\sharp C^\sharp & \text{if } i = j_0, j \neq j_0 \\ \mathbf{m}_{ij} -^\sharp C^\sharp & \text{if } i \neq j_0, j = j_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$

- *if* $\gamma_{\mathcal{B}}(\mathbf{m}^\bigstar_{0j_0} \cap^\sharp_{\mathcal{B}} C^\sharp) \neq \emptyset$, *then*

  $(\{\!\!\{ V_{j_0} \to C \}\!\!\}^{Weak}_{simple}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}^\bigstar_{ij} \cap^\sharp_{\mathcal{B}} (\mathbf{m}^\bigstar_{j_0 j} +^\sharp C^\sharp) & \text{if } i = 0,\ j \neq 0, j_0 \\ \mathbf{m}^\bigstar_{ij} \cap^\sharp_{\mathcal{B}} (\mathbf{m}^\bigstar_{i j_0} -^\sharp C^\sharp) & \text{if } j = 0,\ i \neq 0, j_0 \\ \top^\sharp_{\mathcal{B}} & \text{if } i = j_0 \text{ or } j = j_0 \\ \mathbf{m}^\bigstar_{ij} & \text{otherwise} \end{cases}$

  otherwise, $\{\!\!\{ V_{j_0} \to C \}\!\!\}^{Weak}_{simple}(\mathbf{m}) \stackrel{\text{def}}{=} \bot^{Weak}$

- *if* $\gamma_{\mathcal{B}}(\mathbf{m}^\bigstar_{i_0 j_0} \cap^\sharp_{\mathcal{B}} C^\sharp) \neq \emptyset$, *then*

  $(\{\!\!\{ V_{j_0} \to V_{i_0} + C \}\!\!\}^{Weak}_{simple}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}^\bigstar_{ij} \cap^\sharp_{\mathcal{B}} (\mathbf{m}^\bigstar_{j_0 j} +^\sharp C^\sharp) & \text{if } i = i_0,\ j \neq i_0, j_0 \\ \mathbf{m}^\bigstar_{ij} \cap^\sharp_{\mathcal{B}} (\mathbf{m}^\bigstar_{i j_0} -^\sharp C^\sharp) & \text{if } j = i_0,\ i \neq i_0, j_0 \\ \top^\sharp_{\mathcal{B}} & \text{if } i = j_0 \text{ or } j = j_0 \\ \mathbf{m}^\bigstar_{ij} & \text{otherwise} \end{cases}$

  otherwise, $\{\!\!\{ V_{j_0} \to V_{i_0} + C \}\!\!\}^{Weak}_{simple}(\mathbf{m}) \stackrel{\text{def}}{=} \bot^{Weak}$

Whenever $\gamma_{\mathcal{B}}(C^\sharp) = C$, *these transfer functions are* exact.

●

Reverting to Non-Relational Abstractions. Whenever the assignment, backward assignment, or test is of a more complex form, we can always fall back to transfer functions on the non-relational domain *NonRel* constructed over the same basis $\mathcal{B}$. Special care must

be taken in the test and backward assignment transfer functions to keep as much relational information as possible.

Definition 5.3.4. Non-relational transfer functions.

- $\{\!\!\{ V_i \leftarrow expr \}\!\!\}^{Weak}_{nonrel}(\mathbf{m}) \stackrel{\text{def}}{=} \{\!\!\{ V_i \leftarrow (\![expr]\!]^{NonRel}(NonRel(\mathbf{m})) \}\!\!\}^{Weak}_{simple}(\mathbf{m})$
- $\{\!\!\{ expr \in C\ ? \}\!\!\}^{Weak}_{nonrel}(\mathbf{m}) \stackrel{\text{def}}{=} (Weak \circ \{\!\!\{ expr \in C\ ? \}\!\!\}^{NonRel} \circ NonRel)(\mathbf{m}) \cap^{Weak} \mathbf{m}$
- $\{\!\!\{ V_i \rightarrow expr \}\!\!\}^{Weak}_{nonrel}(\mathbf{m}) \stackrel{\text{def}}{=}$
  $(Weak \circ \{\!\!\{ V_i \rightarrow expr \}\!\!\}^{NonRel} \circ NonRel)(\mathbf{m}) \cap^{Weak} \{\!\!\{ V_i \leftarrow ?\ \}\!\!\}^{Weak}(\mathbf{m}^\star)$

Where $(\![expr]\!]^{NonRel}$, $\{\!\!\{ expr \in C\ ? \}\!\!\}^{NonRel}$, *and* $\{\!\!\{ V_i \rightarrow expr \}\!\!\}^{NonRel}$ *live in the non-relational domain based on* $\mathcal{B}$ *— see Sect. 2.4.4 — and NonRel and Weak are the conversion operators described in Sect. 5.3.2.*

●

More Precise Abstractions for Interval Linear Forms. The non-relational transfer functions merely merge inferred non-relational information with the subset of known relational information that is not invalidated by the operation. When assigning or testing an interval linear form that cannot be exactly abstracted, we can synthesise *new* relational constraints to increase the precision of the non-relational transfer functions, for a linear or quadratic cost, using a technique similar to the one that was used for the zone domain in Defs. 3.6.4 and 3.6.6: we derive an abstract basis element for each expression of the form $V_i - V_j$ using the non-relational expression evaluation operator $(\![expr]\!]^{NonRel}$. After an assignment $V_j \leftarrow expr$, we know that $V_j - V_i = expr - V_i$, and so, we can derive the constraint $V_j - V_i \in \gamma_{\mathcal{B}}((\![expr - V_i]\!]^{NonRel}(NonRel(\mathbf{m})))$ after simplification of $expr - V_i$. After a test $expr \in C\ ?$, we know that $V_j - V_i = (V_j - V_i - expr) + expr \in (V_j - V_i - expr) + C$, and so, we can derive the constraint $V_j - V_i \in \gamma_{\mathcal{B}}((\![V_j - V_i - expr]\!]^{NonRel}(NonRel(\mathbf{m}))) + C$ after simplification of $V_j - V_i - expr$. This gives the following definitions:

Definition 5.3.5. Weakly relational transfer functions.

- $(\{\!\!\{ V_{j_0} \leftarrow expr \}\!\!\}^{Weak}_{rel}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$

$$\begin{cases} (\![expr]\!]^{NonRel}(NonRel(\mathbf{m})) & \text{if } i = 0 \text{ and } j = j_0 \\ (\![\boxminus expr]\!]^{NonRel}(NonRel(\mathbf{m})) & \text{if } i = j_0 \text{ and } j = 0 \\ (\![expr \boxminus V_i]\!]^{NonRel}(NonRel(\mathbf{m})) & \text{if } i \neq 0 \text{ and } j = j_0 \\ (\![V_j \boxminus expr]\!]^{NonRel}(NonRel(\mathbf{m})) & \text{if } i = j_0 \text{ and } j \neq 0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$$

- $(\{\!|\, expr \in C \,?\, |\!\}^{Weak}_{rel}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \mathbf{m}_{ij} \cap^\sharp_{\mathcal{B}}$
$$\begin{cases} C^\sharp +^\sharp [\![V_j \boxminus expr]\!]^{NonRel}(NonRel(\mathbf{m})) & \text{if } i = 0 \text{ and } j \neq 0 \\ C^\sharp +^\sharp [\![\boxminus V_i \boxminus expr]\!]^{NonRel}(NonRel(\mathbf{m})) & \text{if } i \neq 0 \text{ and } j = 0 \\ C^\sharp +^\sharp [\![V_j \boxminus V_i \boxminus expr]\!]^{NonRel}(NonRel(\mathbf{m})) & \text{if } i \neq 0 \text{ and } j \neq 0 \\ [0,0]^\sharp & \text{if } i = j = 0 \end{cases}$$

where $\gamma_{\mathcal{B}}(C^\sharp) \supseteq C$ and the addition $\boxplus$ and subtraction $\boxminus$ operators on interval linear forms are defined by respectively adding and subtracting the interval coefficients corresponding to the same variable. A formal definition will be presented in Sect. 6.2.2.

●

Whenever the expression is not of interval linear form, it is still possible to apply Def. 5.3.5 if we take care to first use the technique that will be presented in Sect. 6.2.3 to abstract expressions into interval linear forms.

Finally, note that backward assignments of interval linear forms, $V_i \rightarrow expr$, can be analysed by substituting *expr* for V_i in every constraint within $\mathbf{m}$, and then applying Def. 5.3.5 to each such constraint. This gives a cubic-time algorithm.

5.3.4 Extrapolation Operators

Weak has strictly increasing (resp. decreasing) infinite chains for $\sqsubseteq^{Weak}$ whenever the basis $\mathcal{B}$ has. In this case there exists a widening $\triangledown_{\mathcal{B}}$ (resp. narrowing $\triangle_{\mathcal{B}}$) on $\mathcal{B}$ that can be extended point-wisely on *Weak*:

$$\begin{array}{rcl} (\mathbf{m} \triangledown^{Weak} \mathbf{n})_{ij} & \stackrel{\text{def}}{=} & \mathbf{m}_{ij} \triangledown_{\mathcal{B}} \mathbf{n}_{ij} \\ (\mathbf{m} \triangle^{Weak} \mathbf{n})_{ij} & \stackrel{\text{def}}{=} & \mathbf{m}_{ij} \triangle_{\mathcal{B}} \mathbf{n}_{ij} \end{array}$$

Stabilisation, in finite time, of increasing iterations with widening — resp. decreasing iterations with narrowing — is only guaranteed automatically for sequences of the form $X^{i+1} \stackrel{\text{def}}{=} X^i \triangledown^{Weak} Y^i$ — resp. $X^{i+1} \stackrel{\text{def}}{=} X^i \triangle^{Weak} Y^i$. If the iterates are closed before being fed as left argument to the extrapolation operator, such as in $X^{i+1} \stackrel{\text{def}}{=} (X^i)^\bigstar \triangledown^{Weak} Y^i$, then the iterates may or may not converge in finite time. We have seen, for instance, in the zone domain, that the standard interval widening results in non-terminating sequences while the standard interval narrowing still terminates.

5.4 Instance Examples

In this section, we give several examples of acceptable bases to illustrate our generic relational abstract domain family. We will retrieve existing abstract domains, such as the zone domain, but also construct new ones, such as the congruence domain.

5.4.1 Translated Equalities

Let us take $\mathbb{I} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$. By Def. 5.2.5.4, an acceptable basis must contain all singletons in $\mathbb{I}$. The simplest such basis is the *constant basis* $\mathcal{B}^{Cst}$. It is derived from the constant propagation technique introduced by Kildall in [Kil73] and subsequently restated in the abstract interpretation framework by Cousot and Cousot in [CC77]. It is presented here using our own notations:

Definition 5.4.1. Constant basis $\mathcal{B}^{Cst}$.

1. $\mathcal{B}^{Cst} \stackrel{\text{def}}{=} \mathbb{I} \cup \{\bot_{\mathcal{B}}^{Cst}, \top_{\mathcal{B}}^{Cst}\}$.
2. *The order* $\sqsubseteq_{\mathcal{B}}^{Cst}$ *is a flat one:* $\forall c \in \mathbb{I},\ \bot_{\mathcal{B}}^{Cst} \sqsubseteq_{\mathcal{B}}^{Cst} c \sqsubseteq_{\mathcal{B}}^{Cst} \top_{\mathcal{B}}^{Cst}$.
3. *The* $(\alpha_{\mathcal{B}}^{Cst}, \gamma_{\mathcal{B}}^{Cst})$ *function pair defined as follows forms a Galois connection:*

$$\gamma_{\mathcal{B}}^{Cst}(X^\sharp) \stackrel{\text{def}}{=} \begin{cases} \{c\} & \text{if } X^\sharp = c \in \mathbb{I} \\ \emptyset & \text{if } X^\sharp = \bot_{\mathcal{B}}^{Cst} \\ \mathbb{I} & \text{if } X^\sharp = \top_{\mathcal{B}}^{Cst} \end{cases} \qquad \alpha_{\mathcal{B}}^{Cst}(R) \stackrel{\text{def}}{=} \begin{cases} c & \text{if } R = \{c\} \\ \bot_{\mathcal{B}}^{Cst} & \text{if } R = \emptyset \\ \top_{\mathcal{B}}^{Cst} & \text{if } |R| > 1 \end{cases}$$

4. *The best* $\cup_{\mathcal{B}}^{Cst}$ *and* $\cap_{\mathcal{B}}^{Cst}$ *operators are exactly* $\sqcup_{\mathcal{B}}^{Cst}$ *and* $\sqcap_{\mathcal{B}}^{Cst}$ *defined as follows:*

- $X^\sharp \cup_{\mathcal{B}}^{Cst} Y^\sharp \stackrel{\text{def}}{=} \begin{cases} \bot_{\mathcal{B}}^{Cst} & \text{if } X^\sharp = \bot_{\mathcal{B}}^{Cst} \text{ and } Y^\sharp = \bot_{\mathcal{B}}^{Cst} \\ X^\sharp & \text{if } X^\sharp = Y^\sharp \text{ or } Y^\sharp = \bot_{\mathcal{B}}^\sharp \\ Y^\sharp & \text{if } X^\sharp = Y^\sharp \text{ or } X^\sharp = \bot_{\mathcal{B}}^\sharp \\ \top_{\mathcal{B}}^{Cst} & \text{otherwise} \end{cases}$
- $X^\sharp \cap_{\mathcal{B}}^{Cst} Y^\sharp \stackrel{\text{def}}{=} \begin{cases} \top_{\mathcal{B}}^{Cst} & \text{if } X^\sharp = \top_{\mathcal{B}}^{Cst} \text{ and } Y^\sharp = \top_{\mathcal{B}}^{Cst} \\ X^\sharp & \text{if } X^\sharp = Y^\sharp \text{ or } Y^\sharp = \top_{\mathcal{B}}^\sharp \\ Y^\sharp & \text{if } X^\sharp = Y^\sharp \text{ or } X^\sharp = \top_{\mathcal{B}}^\sharp \\ \bot_{\mathcal{B}}^{Cst} & \text{otherwise} \end{cases}$

5. *The best forward arithmetic operators can be defined as follows:*

- $[a,b]^{Cst} \stackrel{\text{def}}{=} \begin{cases} a & \text{if } a = b \\ \top_{\mathcal{B}}^{Cst} & \text{if } a \neq b \end{cases}$
- $X^\sharp \mathbin{+}^{Cst} Y^\sharp \stackrel{\text{def}}{=} \begin{cases} \bot_{\mathcal{B}}^{Cst} & \text{if } X^\sharp = \bot_{\mathcal{B}}^{Cst} \text{ or } Y^\sharp = \bot_{\mathcal{B}}^{Cst} \\ \top_{\mathcal{B}}^{Cst} & \text{otherwise if } X^\sharp = \top_{\mathcal{B}}^{Cst} \text{ or } Y^\sharp = \top_{\mathcal{B}}^{Cst} \\ X^\sharp + Y^\sharp & \text{if } X^\sharp \in \mathbb{I} \text{ and } Y^\sharp \in \mathbb{I} \end{cases}$
- $X^\sharp \mathbin{-}^{Cst} Y^\sharp \stackrel{\text{def}}{=} \begin{cases} \bot_{\mathcal{B}}^{Cst} & \text{if } X^\sharp = \bot_{\mathcal{B}}^{Cst} \text{ or } Y^\sharp = \bot_{\mathcal{B}}^{Cst} \\ \top_{\mathcal{B}}^{Cst} & \text{otherwise if } X^\sharp = \top_{\mathcal{B}}^{Cst} \text{ or } Y^\sharp = \top_{\mathcal{B}}^{Cst} \\ X^\sharp - Y^\sharp & \text{if } X^\sharp \in \mathbb{I} \text{ and } Y^\sharp \in \mathbb{I} \end{cases}$
- $X^\sharp \mathbin{\times}^{Cst} Y^\sharp \stackrel{\text{def}}{=} \begin{cases} \bot_{\mathcal{B}}^{Cst} & \text{if } X^\sharp = \bot_{\mathcal{B}}^{Cst} \text{ or } Y^\sharp = \bot_{\mathcal{B}}^{Cst} \\ 0 & \text{otherwise if } X^\sharp = 0 \text{ or } Y^\sharp = 0 \\ \top_{\mathcal{B}}^{Cst} & \text{otherwise if } X^\sharp = \top_{\mathcal{B}}^{Cst} \text{ or } Y^\sharp = \top_{\mathcal{B}}^{Cst} \\ X^\sharp \times Y^\sharp & \text{if } X^\sharp \in \mathbb{I} \setminus \{0\} \text{ and } Y^\sharp \in \mathbb{I} \setminus \{0\} \end{cases}$
- $X^\sharp \mathbin{/}^{Cst} Y^\sharp \stackrel{\text{def}}{=} \begin{cases} \bot_{\mathcal{B}}^{Cst} & \text{if } X^\sharp = \bot_{\mathcal{B}}^{Cst} \text{ or } Y^\sharp \in \{0, \bot_{\mathcal{B}}^{Cst}\} \\ 0 & \text{otherwise if } X^\sharp = 0 \\ \top_{\mathcal{B}}^{Cst} & \text{otherwise if } X^\sharp = \top_{\mathcal{B}}^{Cst} \text{ or } Y^\sharp = \top_{\mathcal{B}}^{Cst} \\ adj(X^\sharp/Y^\sharp) & \text{if } X^\sharp \in \mathbb{I} \setminus \{0\} \text{ and } Y^\sharp \in \mathbb{I} \setminus \{0\} \end{cases}$

6. *As backward arithmetic operators, we choose the generic ones derived, as in Sect. 2.4.4, from the forward ones.*

7. *We now present the backward tests:*

- $\overleftarrow{=}^{Cst}(X^\sharp, Y^\sharp) \stackrel{\text{def}}{=} (X^\sharp \cap_{\mathcal{B}}^{Cst} Y^\sharp,\ X^\sharp \cap_{\mathcal{B}}^{Cst} Y^\sharp)$
- $\overleftarrow{\leq}^{Cst}(X^\sharp, Y^\sharp) \stackrel{\text{def}}{=} \begin{cases} (\bot_{\mathcal{B}}^{Cst}, \bot_{\mathcal{B}}^{Cst}) & \text{if } X^\sharp = \bot_{\mathcal{B}}^{Cst} \text{ or } Y^\sharp = \bot_{\mathcal{B}}^{Cst} \\ (\bot_{\mathcal{B}}^{Cst}, \bot_{\mathcal{B}}^{Cst}) & \text{if } X^\sharp \in \mathbb{I},\ Y^\sharp \in \mathbb{I}, \text{ and } X^\sharp > Y^\sharp \\ (X^\sharp, Y^\sharp) & \text{otherwise} \end{cases}$

- $\overleftarrow{<}^{Cst}(X^\sharp, Y^\sharp) \stackrel{\text{def}}{=} \begin{cases} (\bot_{\mathcal{B}}^{Cst}, \bot_{\mathcal{B}}^{Cst}) & \text{if } X^\sharp = \bot_{\mathcal{B}}^{Cst} \text{ or } Y^\sharp = \bot_{\mathcal{B}}^{Cst} \\ (\bot_{\mathcal{B}}^{Cst}, \bot_{\mathcal{B}}^{Cst}) & \text{if } X^\sharp \in \mathbb{I},\ Y^\sharp \in \mathbb{I}, \text{ and } X^\sharp \geq Y^\sharp \\ (X^\sharp, Y^\sharp) & \text{otherwise} \end{cases}$
- $\overleftarrow{\neq}^{Cst}(X^\sharp, Y^\sharp) \stackrel{\text{def}}{=} \begin{cases} (\bot_{\mathcal{B}}^{Cst}, \bot_{\mathcal{B}}^{Cst}) & \text{if } X^\sharp = \bot_{\mathcal{B}}^{Cst} \text{ or } Y^\sharp = \bot_{\mathcal{B}}^{Cst} \\ (\bot_{\mathcal{B}}^{Cst}, \bot_{\mathcal{B}}^{Cst}) & \text{if } X^\sharp \in \mathbb{I},\ Y^\sharp \in \mathbb{I}, \text{ and } X^\sharp = Y^\sharp \\ (X^\sharp, Y^\sharp) & \text{otherwise} \end{cases}$

8. $\mathcal{B}^{Cst}$ *is of finite height, so, it does not need a widening nor a narrowing.*

●

When plugged into the generic non-relational abstract domain construction of Sect. 2.4.4, $\mathcal{B}^{Cst}$ leads to a constant propagation domain *à la Kildall*. We now state that $\mathcal{B}^{Cst}$ is acceptable for our generic relational domain construction:

Theorem 5.4.1. Acceptability of the constant basis.

$\mathcal{B}^{Cst}$ *is an acceptable basis for Def. 5.2.5.*

●

Proof.

All properties of Def. 5.2.5 are obvious, except Defs. 5.2.5.5–6 that we now prove.

1. Suppose that $\gamma_{\mathcal{B}}^{Cst}\left(\bigcap_{\mathcal{B}}^{Cst} B\right) = \emptyset$. This means that $\bigcap_{\mathcal{B}}^{Cst} B = \bot_{\mathcal{B}}^{Cst}$. We prove that $\exists b, b' \in B,\ b \cap_{\mathcal{B}}^{Cst} b' = \bot_{\mathcal{B}}^{Cst}$. If $\bot_{\mathcal{B}}^{Cst} \in B$, we simply take $b = b' = \bot_{\mathcal{B}}^{Cst}$. Otherwise, there exists $b, b' \in B \cap \mathbb{I}$ such that $b \neq b'$.

2. Suppose that $\gamma_{\mathcal{B}}^{Cst}\left(\bigcap_{\mathcal{B}}^{Cst} B\right) \neq \emptyset$. Then either $\bigcap_{\mathcal{B}}^{Cst} B \in \top_{\mathcal{B}}^{Cst}$ or $\bigcap_{\mathcal{B}}^{Cst} B \in \mathbb{I}$. In the first case, $B = \{\top_{\mathcal{B}}^{Cst}\}$ and in the other case, either $B = \{\top_{\mathcal{B}}^{Cst}, c\}$ or $B = \{c\}$ with $c \in \mathbb{I}$. Whenever B is a singleton, the distributivity is trivial. Whenever $B = \{\top_{\mathcal{B}}^{Cst}, c\}$, if moreover $b' \neq \bot_{\mathcal{B}}^{Cst}$, then $(\top_{\mathcal{B}}^{Cst} \cap_{\mathcal{B}}^{Cst} c) +^{Cst} b' = c +^{Cst} b' = (c +^{Cst} b') \cap_{B}^{Cst} \top_{\mathcal{B}}^{Cst} = (c +^{Cst} b') \cap_{B}^{Cst} (\top_{\mathcal{B}}^{Cst} +^{Cst} b')$ and, if $b' = \bot_{\mathcal{B}}^{Cst}$, then $(\top_{\mathcal{B}}^{Cst} \cap_{\mathcal{B}}^{Cst} c) +^{Cst} b' = (c +^{Cst} b') \cap_{B}^{Cst} (\top_{B}^{Cst} +^{Cst} b') = \bot_{\mathcal{B}}^{Cst}$.

○

When applying our weakly relational generic construction, we obtain an abstract domain that can infer relations of the form $X = Y + c$ and $X = c$, $c \in \mathbb{I}$.

Discussion. This domain is not very expressive, yet it is sufficient to prove interesting properties. For instance, the quantitative shape analysis of Rugina [Rug04] only requires such numerical relational invariants — combined with non-relational invariants and a shape analysis — to prove the correctness of the re-balancing algorithm performed after each insertion in an AVL tree.

A drawback of this domain is its high cost regarded to the obtained precision. One the one hand, it has the same cubic worst-case time cost per abstract operation and quadratic memory cost as the zone domain which is strictly more expressive. On the other hand, there exists quasi-linear algorithms — such as the union–find algorithm — to manipulate conjunctions of constraints of the form $X = Y$. Intuitively, the manipulation of conjunctions of constraints of the form $X = Y + c$ should have a complexity strictly included between these bounds, and so, we postulate for the existence of better representations and algorithms for this abstract domain.

5.4.2 Retrieving the Zone Domain

In order to retrieve the zone abstract domain, we need to express constraints of the form $V_j - V_i \leq c$, $c \in \mathbb{I}$, as $V_j - V_i \in \gamma_{\mathcal{B}}(B)$ for some B in $\mathcal{B}$. A natural idea is to take, as basis $\mathcal{B}$, the set of *initial segments*: $]-\infty, c]$, $c \in \mathbb{I}$. Unfortunately, this does not form an acceptable basis as it is not stable under the unary minus operator. The smallest superset of the set of initial segments that is stable by opposite is the interval abstract domain $\mathcal{B}^{Int}$ described in details in Sect. 2.4.6. We now prove that $\mathcal{B}^{Int}$ is indeed acceptable for our generic relational domain construction:

Theorem 5.4.2. Acceptability of the interval basis.

1. *$\mathcal{B}^{Int}$ is an acceptable basis for Def. 5.2.5 when $\mathbb{I} = \mathbb{Z}$ or $\mathbb{I} = \mathbb{R}$.*
2. *$\mathcal{B}^{Int}$ is an acceptable basis for Def. 5.2.6 when $\mathbb{I} = \mathbb{Q}$.*

●

Proof.

All properties of Def. 5.2.5 are obvious, except Defs. 5.2.5.5–6, as well as Def. 5.2.6 when $\mathbb{I} = \mathbb{Q}$, that we now prove.

1. Suppose that $B \subseteq \mathcal{B}^{Int}$ is a finite set such that $\gamma_{\mathcal{B}}^{Int}\left(\bigcap_{\mathcal{B}}^{Int} B\right) = \emptyset$. This means that $\bigcap_{\mathcal{B}}^{Int} B = \bot_{\mathcal{B}}^{Int}$. We prove that $\exists b, b' \in B$, $b \cap_{\mathcal{B}}^{Int} b' = \bot_{\mathcal{B}}^{Int}$. If $\bot_{\mathcal{B}}^{Int} \in B$, we simply take $b = b' = \bot_{\mathcal{B}}^{Int}$. Otherwise, as B has at least one element, and all elements are non-empty intervals, we can consider:

$$\begin{array}{rcll} m & \stackrel{\text{def}}{=} & \max \{\ a \mid [a,b] \in B\ \} & \in \mathbb{I} \cup \{-\infty\} \\ M & \stackrel{\text{def}}{=} & \min\ \{\ b \mid [a,b] \in B\ \} & \in \mathbb{I} \cup \{+\infty\} \end{array}$$

$\bigcap_{\mathcal{B}}^{Int} B = \bot_{\mathcal{B}}^{Int}$ implies $m > M$. As B is finite, there is some $b, b' \in B$ such that $\min b = m$ and $\max b' = M$. As $\min b > \max b'$, we have $b \cap_{\mathcal{B}}^{Int} b' = \bot_{\mathcal{B}}^{Int}$.

2. Suppose that $\gamma_{\mathcal{B}}^{Int}\left(\bigcap_{\mathcal{B}}^{Int} B\right) \neq \emptyset$. Then $\bot_{\mathcal{B}}^{Int} \notin B$. Suppose that $b' = [a, b] \neq \bot_{\mathcal{B}}^{Int}$, then $(\bigcap_{\mathcal{B}}^{Int} B) \mathbin{+}^{Int} [a', b'] = [\max \{ a \mid [a, b] \in B \} + a', \min \{ b \mid [a, b] \in B \} + b'] = [\max \{ a + a' \mid [a, b] \in B \}, \min \{ b + b' \mid [a, b] \in B \}] = \bigcap_{\mathcal{B}}^{Int} \{ b \mathbin{+}^{Int} [a', b'] \mid b \in B \}$. If, however, $b' = \bot_{\mathcal{B}}^{Int}$, then obviously $(\bigcap_{\mathcal{B}}^{Int} B) \mathbin{+}^{Int} \bot_{\mathcal{B}}^{Int} = \bigcap_{\mathcal{B}}^{Int} \{ b \mathbin{+}^{Int} \bot_{\mathcal{B}}^{Int} \mid b \in B \} = \bot_{\mathcal{B}}^{Int}$.

3. Let us consider a finite set C of intervals with rational bounds. Denominators appearing in sums of arbitrary many elements in C are bounded by the least common multiple of all denominators appearing in C, which is a finite number. Given a finite or infinite subset D of $\{ c_1 \mathbin{+}^{Int} \cdots \mathbin{+}^{Int} c_m \mid m \geq 1, c_1, \ldots, c_m \in C \}$, $\cap_{\mathcal{B}}^{Int} D$ is thus well-defined.

○

When instancing our relational domain family with the $\mathcal{B}^{Int}$ basis, we get the same expressive power as the zone abstract domain of Chap. 3 and similar operators and transfer functions. However, the representation is a little more redundant here as each matrix element stores both an upper bound and a lower bound instead of a single upper bound.

5.4.3 Zones With Strict Constraints

We propose here to extend the zone domain to consider strict as well as non-strict inequalities. This will give us our first interesting and new application of the generic domain construction introduced in the chapter.

In order to do this, we first need to extend the interval basis $\mathcal{B}^{Int}$ to consider strict inequalities. We denote by $\mathcal{B}^{XInt}$ the basis containing all intervals in $\mathcal{P}(\mathbb{I})$ where each bound may be independently included or excluded — except infinite bounds which are always excluded. The operators and transfer functions on $\mathcal{B}^{XInt}$ are quite long and boring due to the numerous cases to consider, yet, they are straightforward to derive from those on $\mathcal{B}^{Int}$ described in Sect. 2.4.6, and so, we will not present them here. As the case $\mathbb{I} = \mathbb{Z}$ reduces to standard integer intervals, we consider only the cases $\mathbb{I} = \mathbb{R}$ and $\mathbb{I} = \mathbb{Q}$.

Theorem 5.4.3. Acceptability of the extended interval basis.

1. *$\mathcal{B}^{XInt}$ is an acceptable basis for Def. 5.2.5 when $\mathbb{I} = \mathbb{R}$.*
2. *$\mathcal{B}^{XInt}$ is an acceptable basis for Def. 5.2.6 when $\mathbb{I} = \mathbb{Q}$.*

●

Proof.

All properties of Def. 5.2.5 are obvious, except maybe Defs. 5.2.5.5–6 that we prove now.

1. Suppose that $B \subseteq \mathcal{B}^{XInt}$ is a finite set such that $\gamma_{\mathcal{B}}^{XInt}\left(\bigcap_{\mathcal{B}}^{XInt} B\right) = \emptyset$. This means that $\bigcap_{\mathcal{B}}^{XInt} B = \bot_{\mathcal{B}}^{XInt}$. We prove that $\exists b, b' \in B,\ b \cap_{\mathcal{B}}^{XInt} b' = \bot_{\mathcal{B}}^{XInt}$. If $\bot_{\mathcal{B}}^{XInt} \in B$, we simply take $b = b' = \bot_{\mathcal{B}}^{XInt}$. Otherwise, we consider, as in Thm. 5.4.2.1, the maximum m of all lower bounds and the minimum M of all upper bounds. There exists two intervals $b \stackrel{\text{def}}{=} \left[m, x\right]$ and $b' \stackrel{\text{def}}{=} \left[y, M\right]$ in B, where we use the $\left[$ symbol instead of $]$ or $[$ when we do not know whether the bound is strict or not. Whenever $m > M$, $[m, x] \cap_{\mathcal{B}}^{XInt} [y, M] = \bot_{\mathcal{B}}^{XInt}$, so, *a fortiori*, $b \cap_{\mathcal{B}}^{XInt} b' = \bot_{\mathcal{B}}^{XInt}$. Whenever $m = M$, there exists either a b such that $b =]m, x\left]$ or a b' such that $b' = \left[y, M\right[$ in B. In both cases, we get $b \cap_{\mathcal{B}}^{XInt} b' = \bot_{\mathcal{B}}^{XInt}$.

2. For the limited distributivity of $+^{XInt}$ over $\cap_{\mathcal{B}}^{XInt}$, we reason independently on the value of each bound of the result, and on the bound strictness. The distributivity of the values of bounds is a consequence of the distributivity in $\cap_{\mathcal{B}}^{Int}$ that we proved in Thm. 5.4.2.2. If a bound of b is strict, then the corresponding bounds of both $b +^{XInt} \left(\bigcap_{\mathcal{B}}^{XInt} B\right)$ and $\bigcap_{\mathcal{B}}^{XInt} \{\ b +^{XInt} b' \mid b' \in B\ \}$ are strict. If the lower bound of b is non-strict, then the lower bound for $b +^{XInt} \left(\bigcap_{\mathcal{B}}^{XInt} B\right)$ is strict if and only if there exists at least one interval $b' \in B$ such that $\min b' = \max \{\ \min b'' \mid b'' \in B\ \}$ and b' has a strict lower bound, which is equivalent to having a strict lower bound for $\bigcap_{\mathcal{B}}^{XInt} \{\ b +^{XInt} b' \mid b' \in B\ \}$. A similar argument applies to the upper bound.

○

When instancing our abstract domain family with $\mathcal{B}^{XInt}$, we obtain a relational abstract domain that can infer constraints of the form $\pm X \leq c$ and $X - Y \leq c$, but also $\pm X < c$ and $X - Y < c$.

Application. Because $\mathcal{B}^{XInt}$ is able to describe exactly the interval $]0, +\infty[$, the weakly relational domain constructed on $\mathcal{B}^{XInt}$ can exactly model strict comparisons such as $X < Y$.

5.4.4 Integer Congruences

In [TCR94], Toman, Chomicki, and Rogers propose to represent conjunctions of constraints of the form $X \equiv Y + a\ [b]$ using "periodicity graphs" resembling our constraint graphs. We propose here to retrieve such constraint systems using our parametric construction.

We first present the *integer congruence basis* denoted by $\mathcal{B}^{Cong}$ and inspired from [Gra89]. But before, we recall and extend the classical arithmetic notions of divisor, greatest common divisor, and lowest common multiple:

Theorem 5.4.4. Arithmetic lattice.

Let us denote by $\mathbb{N}^*$ *the set of strictly positive integers.*

Then $(\mathbb{N}^* \cup \{\infty\}, /, 1, \infty, \vee, \wedge)$ *where:*

$$\begin{array}{lcll} x/y & \stackrel{\text{def}}{\iff} & y = \infty \text{ or } \exists k \in \mathbb{N}^*,\, y = kx & \text{(x divides y)} \\ \bigwedge A & \stackrel{\text{def}}{=} & \max_{/} \{\, x \mid \forall a \in A,\, x/a \,\} & \text{(greatest common divisor)} \\ \bigvee A & \stackrel{\text{def}}{=} & \min_{/} \{\, x \mid \forall a \in A,\, a/x \,\} & \text{(least common multiple)} \end{array}$$

is a complete, fully distributive, lattice.

●

Proof.

1. / is obviously a partial order and it is easy to see that $\forall a,\, 1/a/\infty$. Moreover, there exists an isomorphism between $\mathbb{N}^*$ and the set of countable sequences in $\mathbb{N}$ with a finite number of non-zero elements: $x \in \mathbb{N}^*$ is assimilated to the only sequence $\alpha(x) = (\alpha_1, \ldots, \alpha_n, \ldots)$ such that $x = \prod_i p_i^{\alpha_i}$ where p_i denotes the $i-$th prime number. By this isomorphism, / is corresponds to the point-wise extension of the $\leq$ order. The least upper bound and greatest lower bound of two sequences are respectively the point-wise maximum and minimum, and so, we have a lattice in $\mathbb{N}^*$. We now prove that the lattice is indeed complete if we add the element ∞. Note that we have chosen to complete $\mathbb{N}^*$ with an element named ∞ and not 0 so that $a/b \Longrightarrow a \leq b$, which seems more natural.

   Given a finite or infinite subset A of $\mathbb{N}^*$, the point-wise minimum of sequences is always defined, and so, $\bigwedge A$ exists. If we now consider a subset A of $\mathbb{N}^* \cup \{\infty\}$, then $\bigwedge A = \bigwedge(A \setminus \{\infty\})$ and, in particular, $\bigwedge\{\infty\} = \bigwedge \emptyset = \infty$. Thus, $\wedge$ is defined for arbitrary subsets of $\mathbb{N}^* \cup \{\infty\}$.

   In order to prove that $\bigvee A$ always exists, we consider three cases. If $\infty \in A$, then $\bigvee A = \infty$. If A is infinite, it is also unbounded for /, and we also have $\bigvee A = \infty$. If $\infty \notin A$ and A is finite, then, by isomorphism, the point-wise maximum of sequences is well-defined and so is $\bigvee A$.

2. We now prove that $a \vee (\bigwedge B) = \bigwedge \{\, a \vee b \mid b \in B \,\}$. As $\forall x,\, \infty \vee x = \infty$, if $a = \infty$, we have $a \vee (\bigwedge B) = \bigwedge \{\, a \vee b \mid b \in B \,\} = \infty$. Suppose now that $a \neq \infty$. Suppose moreover that $|B| > 1$ as the property is obvious when $|B| \leq 1$. Then B contains at least one non-∞ element. If we denote by B' the set $B \setminus \{\infty\}$, then we have: $a \vee (\bigwedge B) = a \vee (\bigwedge B')$ and $\bigwedge \{\, a \vee b \mid b \in B \,\} = \bigwedge \{\, a \vee b \mid b \in B' \,\}$. Now that we only consider finite elements, we can use the isomorphism between $\mathbb{N}^*$ and the set of countable integer sequences with a finite number of non-zero elements. As the $\wedge$ and $\vee$ operators correspond, respectively, to the point-wise minimum and maximum on such sequences, the distributivity follows from the fact that $\max(x, \min Y) = \min \{\, \max(x, y) \mid y \in Y \,\}$.

3. We now prove that $a \wedge (\bigvee B) = \bigvee \{ a \wedge b \mid b \in B \}$.

   If $a = \infty$, then $a \wedge (\bigvee B) = \bigvee \{ a \wedge b \mid b \in B \} = \bigvee B$.

   We now suppose that $a \neq \infty$. If $\infty \in B$, then $a \wedge (\bigvee B) = a$. On the one hand, $\forall b$, $(a \wedge b)/a$, so, $(\bigvee \{ a \wedge b \mid b \in B \})/a$. On the other hand, $a \wedge \infty = a$, so, $a \in \{ a \wedge b \mid b \in B \}$ and $a/(\bigvee \{ a \wedge b \mid b \in B \})$. Thus, we have $\bigvee \{ a \wedge b \mid b \in B \} = a = a \wedge (\bigvee B)$.

   Suppose now that $a \neq \infty$ and $\infty \notin B$. We can still have $\bigvee B = \infty$, so, we cannot apply our isomorphism by prime factor decomposition to $\bigvee B$. Instead, we apply it to all *finite* subsets $B' \subseteq B$, as no ∞ can appear in $\bigvee B'$ nor in $\bigvee \{ a \wedge b \mid b \in B' \}$. Given that $\min(x, \max Y) = \max \{ \min(x, y) \mid y \in Y \}$, by our isomorphism, we get $a \wedge (\bigvee B') = \bigvee \{ a \wedge b \mid b \in B' \}$. This set increases with B', but, as $a \wedge (\bigvee B) \neq \infty$, it cannot increase indefinitely. Thus, there exists some finite subset $B' \subseteq B$ such that $a \wedge (\bigvee B') = a \wedge (\bigvee B)$ and $\bigvee \{ a \wedge b \mid b \in B' \} = \bigvee \{ a \wedge b \mid b \in B \}$, which proves the statement.

○

The *congruence* relation $\equiv$, for $x, x' \in \mathbb{Z}$ and $y \in \mathbb{N}^* \cup \{\infty\}$, is defined formally as:

$$x \equiv x' \ [y] \iff^{\text{def}} x = x' \text{ or } (x \neq x' \text{ and } y/|x - x'|)$$

so that $x \equiv x' \ [\infty]$ simply means $x = x'$.

Using these notions, $\mathcal{B}^{Cong}$ can be now defined as:

Definition 5.4.2. Integer congruence basis $\mathcal{B}^{Cong}$.

1. $\mathcal{B}^{Cong} \stackrel{\text{def}}{=} \{\bot_{\mathcal{B}}^{Cong}\} \cup \{ a\mathbb{Z} + b \mid a \in \mathbb{N}^* \cup \{\infty\}, b \in \mathbb{Z} \}$.

2. *The preorder $\sqsubseteq_{\mathcal{B}}^{Cong}$ is defined as:*

   $$(a\mathbb{Z} + b) \sqsubseteq_{\mathcal{B}}^{Cong} (a'\mathbb{Z} + b') \iff^{\text{def}} a'/a \text{ and } b \equiv b' \ [a'] \ .$$

   This preorder is made into a partial order by ordering equivalent elements, that is, elements $(a\mathbb{Z} + b)$ and $(a'\mathbb{Z} + b')$ such that $a = a'$ and $b \equiv b' \ [a']$, the following way: $(a\mathbb{Z} + b) \sqsubseteq_{\mathcal{B}}^{Cong} (a\mathbb{Z} + b') \iff^{\text{def}} |b| < |b'|$ or $(|b| = |b'|$ and $b > 0)$.

   The greatest element for this (pre)order is $\top_{\mathcal{B}}^{Cong} \stackrel{\text{def}}{=} (1\mathbb{Z} + 0)$.

3. *The function pair $(\alpha_{\mathcal{B}}^{Cong}, \gamma_{\mathcal{B}}^{Cong})$ defined as follows:*

   $$\gamma_{\mathcal{B}}^{Cong}(X^\sharp) \stackrel{\text{def}}{=} \begin{cases} \emptyset & \text{if } X^\sharp = \bot_{\mathcal{B}}^{Cong} \\ \{ b \} & \text{if } X^\sharp = (\infty\mathbb{Z} + b) \\ \{ ak + b \mid k \in \mathbb{Z} \} & \text{if } X^\sharp = (a\mathbb{Z} + b),\ a \in \mathbb{N}^* \end{cases}$$

   $$\alpha_{\mathcal{B}}^{Cong}(R) \stackrel{\text{def}}{=} \bigsqcup_{\mathcal{B}}^{Cong} \{ (\infty\mathbb{Z} + b) \mid b \in R \}$$

forms a Galois connection. Note that $\gamma_{\mathcal{B}}^{Cong}$ *is* not *injective.*

4. *The best* $\cup_{\mathcal{B}}^{Cong}$ *and* $\cap_{\mathcal{B}}^{Cong}$ *operators, up to* $\gamma_{\mathcal{B}}$*, can be defined as follows for non-*$\bot_{\mathcal{B}}^{Cong}$ *elements:*

$$(a\mathbb{Z}+b) \cup_{\mathcal{B}}^{Cong} (a'\mathbb{Z}+b') \stackrel{\text{def}}{=} (a \wedge a' \wedge |b-b'|)\mathbb{Z} + b$$

$$(a\mathbb{Z}+b) \cap_{\mathcal{B}}^{Cong} (a'\mathbb{Z}+b') \stackrel{\text{def}}{=} \begin{cases} (a \vee a')\mathbb{Z} + b'' & \textit{if } b \equiv b' \ [a \wedge a'] \\ \bot_{\mathcal{B}}^{Cong} & \textit{otherwise,} \end{cases}$$

where b'' *is defined as:*

$$\begin{cases} k & \textit{is such that} & b - b' = k(a \wedge a') \\ u, u' & \textit{are such that} & au + a'u' = a \wedge a' \\ b'' & \textit{is such that} & b'' = b - kau = b' + ka'u' \end{cases}$$

Note that the existence of u *and* u' *when* $a, a' \neq \infty$ *is guaranteed by Bézout's theorem. If* $a = \infty$ *or* $a' = \infty$*, we can take* $b'' = b' = b$*.*

The operators $\sqcup_{\mathcal{B}}^{Cong}$ *and* $\sqcap_{\mathcal{B}}^{Cong}$ *are equivalent to* $\cup_{\mathcal{B}}^{Cong}$ *and* $\cap_{\mathcal{B}}^{Cong}$*, except that we must take care to choose the right element among all those with the same concretisation* $\gamma_{\mathcal{B}}^{Cong}$*. As a consequence of Thm. 5.4.4,* $\mathcal{B}^{Cong}$ *is a complete lattice.*

5. *We recall here only the* $[a,b]^{Cong}$*,* $+^{Cong}$*, and* $-^{Cong}$ *operators, and reefer the reader to [Gra89] for the definition of other operators:*

$$[a,b]^{Cong} \stackrel{\text{def}}{=} \begin{cases} (\infty\mathbb{Z}+a) & \textit{if } a = b \\ (1\mathbb{Z}+0) & \textit{if } a \neq b \end{cases}$$

$$(a\mathbb{Z}+b) +^{Cong} (a'\mathbb{Z}+b') \stackrel{\text{def}}{=} (a \wedge a')\mathbb{Z} + (b+b')$$

$$-^{Cong}(a\mathbb{Z}+b) \stackrel{\text{def}}{=} a\mathbb{Z} + (-b)$$

6. $\mathcal{B}^{Cong}$ *has no strictly increasing infinite chain, and so, it does not require a widening. It has, however, strictly decreasing infinite chains. A simple narrowing, inspired from the one on intervals, is to refine only elements that are* $\top_{\mathcal{B}}^{Cong}$*:*

$$X^\sharp \vartriangle_{\mathcal{B}}^{Cong} Y^\sharp \stackrel{\text{def}}{=} \begin{cases} Y^\sharp & \textit{if } X^\sharp = (1\mathbb{Z}+b) \\ X^\sharp & \textit{otherwise} \end{cases}$$

●

When applying the congruence basis $\mathcal{B}^{Cong}$ to the non-relational abstract domain construction of Sect. 2.4.4, we obtain Granger's simple congruence domain [Gra89]. Moreover, $\mathcal{B}^{Cong}$ is acceptable for our generic relational domain construction:

Theorem 5.4.5. Acceptability of the simple congruence basis.

$\mathcal{B}^{Cong}$ is an acceptable basis for Def. 5.2.5.

●

Proof.

We now prove that the requirements of Def. 5.2.5 hold.

1. $\gamma^{Cong}(\infty\mathbb{Z}+b) = \{b\}$, so, singletons can be represented exactly. The exactness of $\cap_{\mathcal{B}}^{Cong}$, $+^{Cong}$, and $-^{Cong}$ follows from classical arithmetic properties.

2. In order to prove that $\bigcap_{\mathcal{B}}^{Cong} B = \bot_{\mathcal{B}}^{Cong} \Longrightarrow \exists x, x' \in B, x \cap_{\mathcal{B}}^{Cong} x' = \bot_{\mathcal{B}}^{Cong}$, it is sufficient to prove that, if $x \cap_{\mathcal{B}}^{Cong} x'$, $x \cap_{\mathcal{B}}^{Cong} x''$, and $x' \cap_{\mathcal{B}}^{Cong} x'' \neq \bot_{\mathcal{B}}^{Cong}$, then $x \cap_{\mathcal{B}}^{Cong} x' \cap_{\mathcal{B}}^{Cong} x'' \neq \bot_{\mathcal{B}}^{Cong}$. We consider several cases:

   - We cannot have any of x, x', x'' be $\bot_{\mathcal{B}}^{Cong}$.
   - Suppose that $x = (\infty\mathbb{Z}+b)$, then $x \cap_{\mathcal{B}}^{Cong} x' = x \cap_{\mathcal{B}}^{Cong} x'' = x$, so, $x \cap_{\mathcal{B}}^{Cong} x' \cap_{\mathcal{B}}^{Cong} x'' = x \neq \bot_{\mathcal{B}}^{Cong}$. The cases where $x' = (\infty\mathbb{Z}+b)$ or $x'' = (\infty\mathbb{Z}+b)$ are similar.
   - Suppose that $x = (a\mathbb{Z}+b)$, $x' = (a'\mathbb{Z}+b')$, and $x'' = (a''\mathbb{Z}+b'')$ where $a, a', a'' \neq \infty$. If we denote $x' \cap_{\mathcal{B}}^{Cong} x''$ by $a^*\mathbb{Z}+b^*$, then $b^* \equiv b' \ [a']$ and $b^* \equiv b'' \ [a'']$. Moreover, as $x \cap_{\mathcal{B}}^{Cong} x' \neq \bot_{\mathcal{B}}^{Cong}$, $b \equiv b' \ [a \wedge a']$ and, likewise, $b \equiv b'' \ [a \wedge a'']$, so, $b^* \equiv b' \ [a \wedge a']$ and $b^* \equiv b'' \ [a \wedge a'']$. We deduce that $b^* \equiv b \ [(a \wedge a') \vee (a \wedge a'')]$. By distributivity, $(a \wedge a') \vee (a \wedge a'') = a \wedge (a' \vee a'') = a \wedge a^*$, so, $b^* \equiv b \ [a \wedge a^*]$, which proves that $x \cap_{\mathcal{B}}^{Cong} (x' \cap_{\mathcal{B}}^{Cong} x'') \neq \bot_{\mathcal{B}}^{Cong}$.

3. We suppose that $\bigcap_{\mathcal{B}}^{Cong} B \neq \bot_{\mathcal{B}}^{Cong}$ and prove that $\bigcap_{\mathcal{B}}^{Cong} \{ x +^{Cong} x' \mid x \in B \} = \left(\bigcap_{\mathcal{B}}^{Cong} B\right) +^{Cong} x'$. The case $x' = \bot_{\mathcal{B}}^{Cong}$ is obvious, so, we now consider the case $x' = (a'\mathbb{Z}+b')$. On the one hand, $\left(\bigcap_{\mathcal{B}}^{Cong} B\right) +^{Cong} x' = ((\bigvee_{(a\mathbb{Z}+b)\in B} a) \wedge a')\mathbb{Z} + (x+b')$ where x is such that $\forall (a\mathbb{Z}+b) \in B$, $x \equiv b \ [a]$. On the other hand, $\bigcap_{\mathcal{B}}^{Cong} \{ x +^{Cong} x' \mid x \in B \} = (\bigvee_{(a\mathbb{Z}+b)\in B}(a \wedge a'))\mathbb{Z} + y$ where y is such that $\forall (a\mathbb{Z}+b) \in B$, $y \equiv b + b' \ [a \wedge a']$. Firstly, by distributivity of $\vee$ over $\wedge$, we have $((\bigvee_{(a\mathbb{Z}+b)\in B} a) \wedge a') = (\bigvee_{(a\mathbb{Z}+b)\in B}(a \wedge a'))$. Then, $\forall (a\mathbb{Z}+b) \in B$, as $x \equiv b \ [a]$, we also have $x \equiv b \ [a \wedge a']$, so, $x + b' \equiv b + b' \equiv y \ [a \wedge a']$, and so, $x + b' \equiv y \ \left[\left(\bigvee_{(a\mathbb{Z}+b)\in B} a\right) \wedge a'\right]$, which proves the equality.

○

Applications. When instancing our abstract domain family with $\mathcal{B}^{Cong}$, we obtain a relational numerical abstract domain that can infer invariants of the form $X \equiv a\ [b]$ and $X \equiv Y + a\ [b]$. We will call this domain the *zone congruence domain*. We now give a few example analyses:

Example 5.4.1. Assignment in the zone congruence domain.

Consider the following sequence of assignments:

$$\begin{array}{ll} ① & X \leftarrow [0, 10]; \\ ② & Y \leftarrow [0, 10]; \\ ③ & Z \leftarrow (4 \boldsymbol{\times} X) \boldsymbol{+} Y \boldsymbol{+} 3 \end{array}$$

Because the zone congruence domain is not able to represent intervals, the assignments at ① and ② will simply infer the information $X \in \mathbb{Z}$ and $Y \in \mathbb{Z}$. If we take care to use the interval linear form assignment transfer function $\llparenthesis V \leftarrow expr \rrparenthesis^{Weak}_{rel}$ of Def. 5.3.5 at ③, then we will be able to discover the relational invariant:

$$Z - Y \in [\![(4 \boldsymbol{\times} X) \boldsymbol{+} 3]\!]^{NonRel}(X^\sharp)$$

that is $Z \equiv Y + 3\ [4]$.

●

Example 5.4.2. One-dimensional random walk.

Our second example is the following program that simulates eleven steps of a one-dimensional random walk:

```
    X ← 0;
    I ← 0;
    while ❶ I ≠ 11 {
      if rand { X ← X + 1 }
      else { X ← X − 1 };
      I ← I + 1
②   }
③
```

As for Ex. 2.5.3, to get a precise value for X at ③, it is important to infer a relational loop invariant between X and I at ❶, that is $X \equiv I\ [2]$. The following analysis is able to prove that X is odd after the loop:

iteration i	label l	abstract element X^i_l
0	❶	$X = 0 \ \wedge\ I = 0 \ \wedge\ X - I = 0$
1	②	$X \equiv 1[2] \ \wedge\ I = 1 \ \wedge\ X - I \equiv 0\ [2]$
2	❶ $\triangledown$	$X - I \equiv 0\ [2]$

iteration i	label l	abstract element X_l^i
3	②	$X - I \equiv 0\ [2]$
4	❶ $\triangledown$	$X - I \equiv 0\ [2]$
5	③	$X - I \equiv 0\ [2] \ \wedge\ I = 11 \ \wedge\ X \equiv 1\ [2]$

●

Finally, an important application of congruence invariants in real-life programming languages is the analysis of pointer alignment. Indeed, at the machine level, pointers are generally assimilated to integers and the load and store instructions may require these pointers to be a multiple of some machine word size n — *e.g.*, $n = 4$ when dereferencing a plain integer on a 32-bit architecture. Pointers are often manipulated in a relative way: a new pointer q is constructed by displacing an existing pointer p by a given offset; hence the need for relational congruence information of the form $q - p \in n\mathbb{Z}$ meaning that q is well-aligned if p is.

Related Work. In their presentation of periodicity graphs, in [TCR94], Toman, Chomicki, and Rogers provide an emptiness test, a conjunction, and a projection operators, but no union or transfer functions as they are only interested in the satisfiability problem for Datalog queries and not abstract interpretation of programs. Also, they propose a normalisation algorithm based on the same local constraint propagation as ours, that is, replacing $\mathbf{m}_{ij}$ with $\mathbf{m}_{ij} \cap^\sharp_{\mathcal{B}} (\mathbf{m}_{ik} +^\sharp \mathbf{m}_{kj})$. However, the sequence of triplets i, j, k used is different from the sequence imposed by our Floyd–Warshall algorithm. As a consequence, the algorithm proposed in [TCR94] exhibits a $\mathcal{O}(n^4)$ worse-case time cost while we have given a, much better, cubic time complexity bound.

5.4.5 Rational Congruences

In [Gra97], Granger proposes an extension of the non-relational congruence analysis by considering sets of rationals of the form $a\mathbb{Z} + b$, where $a, b \in \mathbb{Q}$, called *rational congruences*. We recall here the basis $\mathcal{B}^{RCong}$ of rational congruences but, first, we need to extend some notions of integer arithmetics to rational numbers:

Theorem 5.4.6. Rational lattice.

Let us denote by $\mathbb{Q}^+$ the set of positive or null rationals.
Then $(\mathbb{Q}^+ \cup \{\infty\}, /, \vee, \wedge)$ where:

p/q	$\overset{\text{def}}{\iff}$	$p = 0$ *or* $q = \infty$ *or* $\exists k \in \mathbb{N}^*,\ q = kp$	*(p divides q)*
$\bigwedge A$	$\overset{\text{def}}{=}$	$\max_/ \{\ x \mid \forall a \in A,\ x/a\ \}$	*(greatest common divisor)*
$\bigvee A$	$\overset{\text{def}}{=}$	$\min_/ \{\ x \mid \forall a \in A,\ a/x\ \}$	*(least common multiple)*

is a complete, fully distributive, lattice.
Moreover, when their arguments are neither 0 *nor* ∞*, the binary* $\wedge$ *and* $\vee$ *operators can be defined using the classical integer least common multiple and greatest common divisor as follows:*

$$\frac{a}{b} \wedge \frac{c}{d} \stackrel{\text{def}}{=} \frac{ad \wedge bc}{bd} \quad \textit{and} \quad \frac{a}{b} \vee \frac{c}{d} \stackrel{\text{def}}{=} \frac{ad \vee bc}{bd} \ .$$

●

Proof.

1. The situation is quite similar to that of Thm. 5.4.4 except that the set of strictly positive rationals $\mathbb{Q}^{+*}$ is isomorphic to countable sequences in $\mathbb{Z}$ with a finite number of non-zero elements: strictly positive rationals are obtained by allowing negative as well as positive exponents of prime numbers. The $/$, $\wedge$, and $\vee$ operators still correspond to the point-wise $\leq$, minimum, and maximum operators, and so, we have a lattice structure on $\mathbb{Q}^{+*}$. We now prove that the lattice is complete thanks to the addition of two special elements: 0 and ∞.

   Whenever $0 \in A$, we have $\bigwedge A = 0$. Moreover, $\bigwedge A = \bigwedge(A \setminus \{\infty\})$, so, we can now consider only sets A that do not contain 0 nor ∞. Two cases arise. If the set of denominators of reduced fractions appearing in A is bounded, then the problem is similar to that of computing the greatest common divisor of a set of integers. If this set is unbounded, then $\bigwedge A = 0$.

   We can prove that $\bigvee A$ is always defined by duality: it is sufficient in the preceding proof to exchange 0 and ∞, and replace "denominator" with "numerator".

2. We now prove that $a \vee (\bigwedge A) = \bigwedge \{\ a \vee a' \mid a' \in A\ \}$. The cases $a = \infty$ and $a = 0$ are obvious because $\infty \vee a' = \infty$ and $0 \vee a' = a'$, so, we now suppose that $a \notin \{0, \infty\}$. If $0 \in A$, then $a \vee (\bigwedge A) = a \vee 0 = a$; moreover, $a \in \{\ a \vee a' \mid a' \in A\ \}$ which implies $\bigwedge \{\ a \vee a' \mid a' \in A\ \} = a$. As the ∞ element can be safely removed from A without changing $\bigwedge A$ nor $\bigwedge \{\ a \vee a' \mid a' \in A\ \}$, we can now consider that A does not contain 0 nor ∞. For every finite subset $A' \subseteq A$, we can apply our isomorphism and the fact that $\max(x, \min Y) = \min \{\ \max(x, y) \mid y \in Y\ \}$ to get $a \vee (\bigwedge A') = \bigwedge \{\ a \vee a' \mid a' \in A'\ \}$. We now consider an increasing sequence of finite subsets A'_i of A. Then, $a \vee (\bigwedge A'_i)$ is a sequence both decreasing for $/$ and bounded by a. We now remark that all decreasing and bounded sequences of integers with only a finite number of non-zero elements are *finite*. This means that for some finite i, $a \vee (\bigwedge A'_i) = a \vee (\bigwedge A)$ and $\bigwedge \{\ a \vee a' \mid a' \in A_i\ \} = \bigwedge \{\ a \vee a' \mid a' \in A\ \}$ which concludes the proof.

   The proof that $a \wedge (\bigvee A) = \bigvee \{\ a \wedge a' \mid a' \in A\ \}$ would be quite similar by duality.

3. The alternate formulas for the binary $\wedge$ and $\vee$ operators are extracted from [Gra97].

$\bigcirc$

Then, the rational congruence basis is constructed similarly to the integer congruence basis:

Definition 5.4.3. Rational congruence basis $\mathcal{B}^{RCong}$.

1. $\mathcal{B}^{RCong} \stackrel{\text{def}}{=} \{\bot_{\mathcal{B}}^{RCong}\} \cup \{\, a\mathbb{Z}+b \mid a \in \mathbb{Q}^+ \cup \{\infty\},\, b \in \mathbb{Q} \,\}$.

2. *The* congruence *relation* $\equiv$, *for* $x, x' \in \mathbb{Q}$ *and* $y \in \mathbb{Q}^+ \cup \{\infty\}$ *is defined formally as:*

$$x \equiv x'\ [y] \iff^{\text{def}} x = x' \text{ or } (x \neq x' \text{ and } y/|x-x'|)\ .$$

In particular, $\forall x, x',\ x \equiv x'\ [0]$ *and* $x \equiv x'\ [\infty] \iff x = x'$.

3. *The preorder* $\sqsubseteq_{\mathcal{B}}^{RCong}$ *is defined as:*

$$(a\mathbb{Z}+b) \sqsubseteq_{\mathcal{B}}^{RCong} (a'\mathbb{Z}+b') \iff^{\text{def}} a'/a \text{ and } b \equiv b'\ [a']\ .$$

This preorder is made into a partial order by stating that, if $b \equiv b'\ [a]$, *then* $(a\mathbb{Z}+b) \sqsubseteq_{\mathcal{B}}^{RCong} (a\mathbb{Z}+b') \iff^{\text{def}} |b| < |b'|$ *or* $(|b| = |b'|$ *and* $b > 0)$.

4. *The* $(\alpha_{\mathcal{B}}^{RCong}, \gamma_{\mathcal{B}}^{RCong})$ *function pair defined as follows forms a partial Galois connection:*

$$\gamma_{\mathcal{B}}^{RCong}(X^\sharp) \stackrel{\text{def}}{=} \begin{cases} \emptyset & \text{if } X^\sharp = \bot_{\mathcal{B}}^{RCong} \\ \{\, b \,\} & \text{if } X^\sharp = (\infty\mathbb{Z}+b) \\ \mathbb{Q} & \text{if } X^\sharp = (0\mathbb{Z}+b) \\ \{\, ak+b \mid k \in \mathbb{Z} \,\} & \text{if } X^\sharp = (a\mathbb{Z}+b),\ a \in \mathbb{Q}^{+*} \end{cases}$$

$$\alpha_{\mathcal{B}}^{RCong}(R) \stackrel{\text{def}}{=} \bigsqcup_{\mathcal{B}}^{RCong} \{\, (\infty\mathbb{Z}+b) \mid b \in R \,\}$$

Note that $\mathbb{Q}$ *is represented by* $(0\mathbb{Z}+b)$ *and singletons by* $(\infty\mathbb{Z}+b)$*: the bigger* a *is in* $(a\mathbb{Z}+b)$, *the more distance there is between its points. As before,* $\gamma_{\mathcal{B}}^{RCong}$ *is not injective.*

5. *The best* $\cup_{\mathcal{B}}^{RCong}$ *and* $\cap_{\mathcal{B}}^{RCong}$ *operators, up to* $\gamma_{\mathcal{B}}^{RCong}$, *are defined by the following equations, naturally extended to* $\bot_{\mathcal{B}}^{RCong}$*:*

$$(a\mathbb{Z}+b) \cup_{\mathcal{B}}^{RCong} (a'\mathbb{Z}+b') \stackrel{\text{def}}{=} (a \wedge a' \wedge |b-b'|)\mathbb{Z}+b$$

$$(a\mathbb{Z}+b) \cap_{\mathcal{B}}^{RCong} (a'\mathbb{Z}+b') \stackrel{\text{def}}{=} \begin{cases} (a \vee a')\mathbb{Z}+b'' & \text{if } b \equiv b'\ [a \wedge a'], \\ \bot_{\mathcal{B}}^{RCong} & \text{otherwise,} \end{cases}$$

where b'' is such that $b'' \equiv b\,[a \vee a']$ and $b'' \equiv b'\,[a \vee a']$.

When $a, a' \notin \{0, \infty\}$, such a b'' can be computed using Bézout's theorem on the numerators of the fractions a and a' put to the same denominator.

The operators $\sqcup_{\mathcal{B}}^{RCong}$ and $\sqcap_{\mathcal{B}}^{RCong}$ are equivalent to $\cup_{\mathcal{B}}^{RCong}$ and $\cap_{\mathcal{B}}^{RCong}$, except that we must take care to choose the right element among all those with the same concretisation $\gamma_{\mathcal{B}}^{RCong}$. An important consequence of Thm. 5.4.6 is that $\mathcal{B}^{Cong}$ is a complete lattice.

6. *We recall here only the $[a, b]^{RCong}$, $+^{RCong}$, and $-^{RCong}$ operators and refer the reader to [Gra97] for a more exhaustive presentation of the non-relational domain of rational congruences:*

$$[a, b]^{RCong} \stackrel{\text{def}}{=} \begin{cases} (\infty\mathbb{Z} + a) & \text{if } a = b \\ (0\mathbb{Z} + 0) & \text{if } a \neq b \end{cases}$$

$$(a\mathbb{Z} + b) +^{RCong} (a'\mathbb{Z} + b') \stackrel{\text{def}}{=} (a \wedge a')\mathbb{Z} + (b + b')$$

$$-^{RCong}(a\mathbb{Z} + b) \stackrel{\text{def}}{=} a\mathbb{Z} + (-b)$$

7. *$\mathcal{B}^{RCong}$ has strictly increasing* and *decreasing infinite chains. We recall here Granger's simple widening which amounts to setting to $\mathbb{Q}$ unstable unions. We also use the same narrowing as for integer congruences:*

$$X^\sharp \triangledown_{\mathcal{B}}^{RCong} Y^\sharp \stackrel{\text{def}}{=} \begin{cases} X^\sharp & \text{if } Y^\sharp \sqsubseteq_{\mathcal{B}}^{RCong} X^\sharp \\ (0\mathbb{Z} + 0) & \text{otherwise} \end{cases}$$

$$X^\sharp \triangle_{\mathcal{B}}^{RCong} Y^\sharp \stackrel{\text{def}}{=} \begin{cases} Y^\sharp & \text{if } X^\sharp = (0\mathbb{Z} + b) \\ X^\sharp & \text{otherwise} \end{cases}$$

●

When applying the congruence basis $\mathcal{B}^{RCong}$ to the non-relational abstract domain construction of Sect. 2.4.4, we obtain Granger's rational congruence domain described in [Gra97]. As $\mathcal{B}^{RCong}$ is also an acceptable basis, we can use it as a parameter to our generic weakly relational domain construction:

Theorem 5.4.7. Acceptability of the rational congruence basis.

$\mathcal{B}^{RCong}$ is an acceptable basis for Def. 5.2.5.

●

Proof.

1. As a consequence of Def. 5.4.3, $\mathcal{B}^{RCong}$ is a complete lattice that can represent each singleton $\{b\}$ as $(\infty\mathbb{Z}+b)$ and enjoys exact abstractions $\cap_{\mathcal{B}}^{RCong}$, $+^{RCong}$, and $-^{RCong}$.
2. The proof of Defs. 5.2.5.5–6 in the integer congruence case, in Thm. 5.4.5, only relied on the definition of $+^{Cong}$ and on the distributivity of the lattice $(\mathbb{N}^* \cup \{\infty\}, /, 1, \infty, \vee, \wedge)$. As $+^{RCong}$ is defined the same way and the lattice $(\mathbb{Q}^+ \cup \{\infty\}, /, 0, \infty, \vee, \wedge)$ is also distributive, the very same proof holds here.

○

We then obtain a domain that is able to infer constraints of the form $X - Y \in a\mathbb{Z} + b$, where $a, b \in \mathbb{Q}$.

5.4.6 Unacceptable Bases

Many classical bases that are used in non-relational abstract domains are not acceptable bases. This does not always mean that no relational domain for invariants of the form $X - Y \in \gamma(C^\sharp)$, $C^\sharp \in \mathcal{B}$ can be constructed with a cubic worst-case time cost per abstract operation, but merely that the generic construction presented here does not apply.

Sign Basis. The sign basis $\mathcal{B}^{Sign} = \{\, \emptyset,]-\infty, 0], [0, +\infty], \{0\}, \mathbb{I} \,\}$ is not acceptable because it cannot abstract singletons exactly. However, invariants of the form $X \leq Y$ can be discovered using the more precise interval basis.

Combined Intervals and Congruences. The *reduced product* of the interval and integer congruence bases is a basis that can represent subsets of $\mathbb{Z}$ of the form $[a, b] \times c + d$. This basis does not satisfy Def. 5.2.5.5. Consider, for instance, the following three sets representable in the basis: $\{1, 2\}$, $\{2, 3\}$, and $\{1, 3\}$; their intersection is empty while no two of them have an empty intersection. Our construction does not work on this basis, which does not come much as a surprise as the satisfiability problem for conjunctions containing constraints of the form $X \equiv Y + a\ [b]$, $X + c < Y$, and $l \leq X \leq h$ is known to be NP-complete if all the c are non-negative and undecidable if we allow negative values for c [TCR94].

5.5 Conclusion

We have proposed, in this chapter, a generalisation of the zone abstract domain: constraints of the form $X - Y \leq c$ are replaced with constraints of the form $X - Y \in C$, where C lives in a generic non-relational abstract basis which is a parameter of our generic construction. All the zone abstract domain algorithms adapt quite nicely and we obtain a similar quadratic

memory consumption per abstract element and cubic worst-case time cost per abstract operation. Unfortunately, the restrictive hypotheses required on the non-relational basis parameter limit the scope of this domain family.

The primary goal of this research was to try and adapt the zone abstract domain to integer congruences, $X \equiv Y + a\ [b]$, and we were quite successful in this respect. Some variations presented here, such as the abstract domain of strict zones or, to a lesser extent, the abstract domain of rational congruence zones, may also be of interest. However, it is unlikely that this construction will spawn many more interesting instances that those ones.

Future Work. The difficult part in the design of our family of weakly relational domains was to find sufficient algebraic conditions so that a low-cost constraint propagation scheme, such as the Floyd–Warshall shortest-path closure, can be used to find a normal form with a saturation property — and this was also quite difficult for the octagon domain of Chap. 4. From a theoretical point of view, it is interesting to see how far we can *stretch* such algorithms by providing stronger or alternate algebraic framework for constraints of the form $X - Y \in C$ and maybe generalise further these forms of constraint. For instance, the so-called $\mathbb{R}^2-$approximations, introduced — without any normal form or saturation result — by Bagnara in is Ph.D [Bag97, § 5.7] correspond to conjunctions of constraints of the form $(V_i, V_j) \in S$, were each S lives in a relational abstract domain for two variables. This form of constraints generalises our family of constraints $X - Y \in C$, but also the octagonal constraints $\pm X \pm Y \leq c$, and two variables per linear inequality constraints $\alpha X + \beta Y \leq c$. Each one of these three instances enjoys a different closure-based normal form and it would be most useful to try and unify them in a more general framework.

Chapter 6

Symbolic Enhancement Methods

Ce chapitre présente deux techniques pouvant être utilisées pour améliorer la précision de tout domaine numérique abstrait. Nous proposons d'abord un algorithme, dit de linéarisation*, capable d'abstraire une expression quelconque en une forme linéaire avec coefficients intervalles, ou une forme quasi-linéaire, et ainsi étendre l'utilité des fonctions de transfert abstraites des domaines des zones, des octogones, et de notre famille de domaines faiblement relationnels. Ensuite, nous proposons une méthode, dite de* propagation de constantes symboliques*, pour rendre nos analyses insensibles aux transformations de programmes relativement simples. Ces méthodes permettent également d'améliorer la précision des domaines non relationnels, tel que le domaine des intervalles. Nous proposons enfin quelques stratégies permettant d'ajuster finement ces techniques.*

This chapter presents two techniques that can be used to improve the precision of any numerical abstract domain. Firstly, we propose a so-called linearisation *algorithm able to abstract arbitrary expressions into interval linear forms, or quasi-linear forms, and extend the scope of the zone, octagon, and weakly relational abstract transfer functions based on interval linear forms. Secondly, we propose a so-called* symbolic constant propagation *mechanism to make our analyses more robust against simple program transformations. These methods are also useful to improve the precision of non-relational abstract domains, such as the interval domain. Finally, we present a few strategies to help tuning these techniques.*

6.1 Introduction

The design of abstract transfer functions for the whole set of numerical and boolean expressions allowed by a realistic programming language is not an easy task, especially for relational abstract domains. Generally, abstract domain designers only bother designing precise transfer functions for expressions that match more or less the form of invariants expressible exactly in their domain. The zone and octagon abstract domains we designed in Chaps. 3 and 4, as well as the zone-like domains of Chap. 5, perform a little better in this respect: we proposed exact abstract transfer functions for expressions involving at most two variables and unit coefficients, but also rather precise abstractions when considering arbitrary interval linear forms. However, we have no way to abstract generic expressions — except by momentarily switching into an abstract domain enjoying the desired transfer function.

Our purpose here is to widen the scope of the abstract transfer functions for interval linear forms and let them abstract *arbitrary* expressions. Our solution is quite simple: we first abstract the expression into an interval linear form, performing a simple expression simplification on-the-fly, and then apply an existing interval linear form transfer function. We also propose a *symbolic constant propagation* technique that allows enhancing the simplification feature of this *linearisation* to gain even more precision. These techniques are quite generic: we will show how they can improve the precision of relational abstract domains featuring interval linear form transfer functions, such as the zone and octagon domains, or quasi-linear form transfer functions, such as the polyhedron abstract domain, but also non-relational abstract domains, such as the interval domain. We will see, however, that the improvements vary greatly from example to example as there does not exist a best way to perform the linearisation and the symbolic constant propagation. These techniques depend upon *strategies*, some of which will be proposed in the chapter.

Previous Work on Affine Arithmetics. In the scientific computation community, much work has been done to try and improve the precision of interval arithmetics. In particular, *affine arithmetics*, a technique reminiscent of Hansen's generalised interval arithmetics [Han75], has been introduced by Vinícius, Andrade, Comba, and Stolfi in [VACS94] to refine interval arithmetics by taking into account existing correlations between computed quantities. In affine arithmetics, symbolic expressions of the form $e = x_0 + \sum_i x_i\epsilon_i$ are manipulated, where the $x_i \in \mathbb{R}$ are constants and the ϵ_i are synthetic variables, ranging in $[-1, 1]$: e represents to the set $\{\ x_0 + \sum_i x_i\epsilon_i \mid \epsilon_i \in [-1, 1]\ \}$. The set of *affine forms* forms a linear space: one can add two affine forms, or multiply one by a constant. The affine form corresponding to an expression is constructed as follows. To each variable X_i is associated a synthetic variable ϵ_i. If X_i has range $[a_i, b_i]$, then the expression X_i can be modeled as the affine form $(a_i + b_i)/2 + ((b_i - a_i)/2)\epsilon_i$. A new synthetic variable is introduced for each non-singleton interval appearing in an expression, as these choices are uncorrelated.

Also, each non-linear operation, such as the multiplication of two affine forms, introduces some over-approximation in the form a non-deterministic interval, and thus requires the introduction of a new ϵ_i variable.

The benefit of affine arithmetics comes from the fact that all occurrences of the same variable X_i use the same ϵ_i, allowing some simplification to take place. For instance, $X - (0.5 \times X)$ when $X \in [0, 10]$ will give back the precise interval $[0, 5]$ using affine arithmetics, instead of $[0, 10] - [0, 5] = [-5, 10]$ for regular interval arithmetics. Although affine arithmetics could be used *locally* to compute a better abstract expression evaluation than $[\![expr]\!]^{Int}$ in the interval domain, it would be tricky to use in relational domains because it does not express a function of the program variables, but a function of some synthetic variables ϵ_i. Remark also that the affine arithmetics community focuses only on abstracting expressions and does not consider the problem of control flow joins or loops. In particular, a program with loops may generate an arbitrary large number of synthetic variables — each new execution of a non-deterministic choice at a given program point being independent from its other executions, it requires a new synthetic variable — which prevents us from manipulating affine forms globally.

Our "linearisation" technique, presented in the first part of the chapter, is related to affine arithmetics in its principle: we use a symbolic form to allow basic algebraic simplifications. However, our symbolic forms are of a different nature. In particular, they are expressed directly using the program variables so that they can be fed directly to relational abstract domains as well as non-relational ones.

Previous Work on Symbolic Constant Propagation. Constant propagation, as proposed by Kildall in [Kil73] and recalled formally in Sect. 5.4.1, has been used for more than thirty years in the field of optimising compilers: whenever a program expression is proved constant by static analysis, it enables the compiler to evaluate it at compile-time, thus saving execution time at run-time. The second part of this chapter will be devoted to a kind of constant propagation technique based on similar principles, but using *symbolic* expressions, to improve the amount of simplification performed by the linearisation. Our goal here is not time consumption, but analysis precision. Indeed, numerical abstract domains are very sensitive to even the simplest program transformations, such as breaking assignments using intermediate variables; our symbolic constant propagation tries to undo such transformations by gluing bits of expressions together in the hope of making abstract domains more robust.

A somewhat related approach is that of Colby. In his Ph. D. thesis [Col96], Colby pushes this idea very far by proposing a language of transfer relations that allows propagating, combining, and simplifying, in a fully symbolic way, any sequence of transfer functions before actually executing the combination in an abstract domain. Our symbolic propagation technique is much more modest as we only propagate numerical expression trees, which are

side-effect free and do not contain any control-flow join. Unlike [Col96], we do not handle disjunction symbolically, which limits the combinatorial explosion of symbolic transfer relations and suppresses the need for symbolic invariant generation. A fundamental difference is that our symbolic domain has tight interactions with a numerical abstract domain which performs most of the semantics job, including loop invariant generation using iterations with widening and narrowing. The symbolic domain is only here to help an existing analysis based on a numerical abstract domain. Thus, while Colby's framework statically transforms the abstract equation system to be solved by the analyser, our framework performs this transformation on-the-fly to benefit from the information dynamically inferred by the analyser.

6.2 Linearisation

We will call "linearisation" the sound abstraction of an arbitrary expression into an interval linear form or a quasi-linear form.

6.2.1 Interval Linear Forms

Given a finite set of variables $\mathcal{V} \stackrel{\text{def}}{=} \{V_1, \ldots, V_n\}$ with values in $\mathbb{I}$, where $\mathbb{I} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$, we consider *interval linear forms*, that is, symbolic expressions l of the form:

$$l \stackrel{\text{def}}{=} [a_0, b_0] \mathbin{+} ([a_1, b_1] \mathbin{\times} V_1) \mathbin{+} \cdots \mathbin{+} ([a_n, b_n] \mathbin{\times} V_n)$$

where the $[a_i, b_i]$ are bounded or unbounded intervals in $\mathbb{I}$: $a_i \in \mathbb{I} \cup \{-\infty\}$, $b_i \in \mathbb{I} \cup \{+\infty\}$, and $a_i \leq b_i$. We will also use the following more compact notation:

$$l = i \mathbin{+} \sum_{k=1}^{n} i_k \mathbin{\times} V_k$$

where i and all i_k denote intervals. Recall that, as the semantics of $\mathbin{+}$ and $\mathbin{\times}$ enjoy the standard associativity and distributivity properties of $+$ and $\times$, the evaluation order of our interval linear form is irrelevant.

Semantics of Interval Linear Forms. Interval linear forms are just a subset of the expressions available in our Simple programming language. Using the semantics of Def. 2.2, an interval linear form can be seen as a function that maps a concrete environment $\rho : \mathcal{V} \to \mathbb{I}$ to a set of values in $\mathcal{P}(\mathbb{I})$:

$$[\![l]\!]\rho = \left\{ c_0 + \sum_{k=1}^{n} (c_k \times \rho(V_k)) \mid c_i \in [a_i, b_i] \right\} .$$

Note that, when $\mathbb{I} \in \{\mathbb{R}, \mathbb{Q}\}$, $[\![l]\!]\rho$ is an interval, by density. However, when $\mathbb{I} = \mathbb{Z}$, $[\![l]\!]\rho$ may contain "holes".

Partial Ordering. It is possible to compare interval linear forms, as well as Simple language expressions, using the point-wise extension to the set of environments $\mathcal{V} \to \mathbb{I}$ of the $\subseteq$ order in $\mathcal{P}(\mathbb{I})$: $e_1 \sqsubseteq e_2 \stackrel{\text{def}}{\iff} \forall \rho, [\![e_1]\!]\rho \subseteq [\![e_2]\!]\rho$. However, this order is a little too restrictive. We will define later some operators that are only sound *with respect to a set of environments* R, so, we define an ordering family $\sqsubseteq_R$ as follows:

$$e_1 \sqsubseteq_R e_2 \stackrel{\text{def}}{\iff} \forall \rho \in R, [\![e_1]\!]\rho \subseteq [\![e_2]\!]\rho \ .$$

Obviously, if $R \subseteq R'$, then $e_1 \sqsubseteq_{R'} e_2 \Longrightarrow e_1 \sqsubseteq_R e_2$, and $\sqsubseteq_{\mathcal{P}(\mathcal{V}\to\mathbb{I})}$ is equivalent to the plain point-wise ordering. We will denote by $=_R$ the associated equality relation.

We now suppose that we are given an abstract domain $\mathcal{D}^\sharp$ with concretisation $\gamma : \mathcal{D}^\sharp \to \mathcal{P}(\mathcal{V} \to \mathbb{I})$. Given an abstract element $X^\sharp \in \mathcal{D}^\sharp$, we can safely replace any abstract assignment transfer function $\{\!| V \leftarrow e_1 |\!\}^\sharp X^\sharp$ with $\{\!| V \leftarrow e_2 |\!\}^\sharp X^\sharp$ whenever $e_1 \sqsubseteq_{\gamma(X^\sharp)} e_2$: over-approximating the set of possible values of *expr* for each environment in $\gamma(X^\sharp)$ leads to over-approximating, in the concrete, the set of environments after the assignment, which is sound. This is particularly useful if there exists an efficient and precise abstract assignment for e_2 on $\mathcal{D}^\sharp$ while there is none for e_1. Likewise, test transfer functions $\{\!| e_1 \bowtie 0 \ ?|\!\}^\sharp X^\sharp$ can be safely replaced with $\{\!| e_2 \bowtie 0 \ ?|\!\}^\sharp X^\sharp$: all environments in $\gamma(X^\sharp)$ that pass the test $(e_1 \bowtie 0 \ ?)$ will also pass the test $(e_2 \bowtie 0 \ ?)$ in the concrete world. The case of backward assignments is a little more subtle: in order to safely replace $\{\!| V \to e_1 |\!\}^\sharp X^\sharp$ with $\{\!| V \to e_2 |\!\}^\sharp X^\sharp$, e_2 needs to over-approximate the value of e_1 on environments *before* the assignment, that is $e_1 \sqsubseteq_{\{V\to e_1\}(\gamma(X^\sharp))} e_2$. One sufficient condition would be $e_1 \sqsubseteq_{\gamma(\{V\to e_1\}^\sharp X^\sharp)} e_2$. Another condition, which is more restrictive but does not require the knowledge of $\{\!| V \to e_1 |\!\}^\sharp$, is $e_1 \sqsubseteq_{\gamma(\{V\leftarrow ?\}^\sharp X^\sharp)} e_2$. Remark that backward assignments are generally used in a backward pass to refine the result of a preceding forward analysis pass, and so, an abstraction of the environments before the assignment may be already available. In the following, we will use $\{\!| V \leftarrow ? |\!\}^\sharp X^\sharp$ and leave implicit the fact that it can be replaced by any such abstract element.

6.2.2 Interval Linear Form Operators

Linear Operators. Using the interval abstract operators $+^{Int}$, $-^{Int}$, and $\times^{Int}$ defined in Sect. 2.4.6, as well as a slightly modified interval division $/^{Int}_{alt}$, we can define the symbolic addition $\boxplus$ and subtraction $\boxminus$ of two linear forms, as well as the multiplication $\boxtimes$ and division $\boxslash$ of a linear form by a constant interval:

Definition 6.2.1. Interval linear form linear operators.

1. $(i + \sum_{k=1}^n i_k \times V_k) \boxplus (i' + \sum_{k=1}^n i'_k \times V_k) \stackrel{\text{def}}{=} (i +^{Int} i') + \sum_{k=1}^n (i_k +^{Int} i'_k) \times V_k$
2. $(i + \sum_{k=1}^n i_k \times V_k) \boxminus (i' + \sum_{k=1}^n i'_k \times V_k) \stackrel{\text{def}}{=} (i -^{Int} i') + \sum_{k=1}^n (i_k -^{Int} i'_k) \times V_k$

3. $i \boxtimes (i' + \sum_{k=1}^{n} i'_k \times V_k) \stackrel{\text{def}}{=} (i \times^{Int} i') + \sum_{k=1}^{n} (i \times^{Int} i'_k) \times V_k$
4. $(i + \sum_{k=1}^{n} i_k \times V_k) \boxslash i' \stackrel{\text{def}}{=} (i /^{Int}_{alt} i') + \sum_{k=1}^{n} (i_k /^{Int}_{alt} i') \times V_k$

   where the alternate division $/^{Int}_{alt}$ *is defined to be* $/^{Int}$ *when* $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$*, and as follows when* $\mathbb{I} = \mathbb{Z}$*:*

   - $[a,b] /^{Int}_{alt} [a',b'] \stackrel{\text{def}}{=} [\, \lfloor \min(a/a',\, a/b',\, b/a',\, b/b') \rfloor,\ \lceil \max(a/a',\, a/b',\, b/a',\, b/b') \rceil\,]$ *when* $0 \leq a'$
   - $X /^{Int}_{alt} Y \stackrel{\text{def}}{=} (X /^{Int}_{alt} (Y \cap^{Int}_{\mathcal{B}} [0, +\infty])) \cup^{Int}_{\mathcal{B}} ((-^{Int} X) /^{Int}_{alt} ((-^{Int} Y) \cap^{Int}_{\mathcal{B}} [0, +\infty]))$ *otherwise*

●

Recall that, in the original integer interval division, the bounds were rounded towards 0 using the *adj* function to match the concrete semantics. Here, however, we need to use an alternate division operator $/^{Int}_{alt}$ that rounds lower bounds towards $-\infty$ and upper bounds towards $+\infty$ to get a sound $\boxslash$ operator in $\mathbb{Z}$, as explained in the proof of Thm. 6.2.1.

The interval linear form operators of Def. 6.2.1 are exact when $\mathbb{I} \in \{\mathbb{R}, \mathbb{Q}\}$, but, when $\mathbb{I} = \mathbb{Z}$, the multiplication and division operators can result in some loss of precision:

Theorem 6.2.1. Interval linear form linear operators soundness.

Let us take two interval linear forms l_1*,* l_2 *and an interval* i*. We have:*

- $l_1 + l_2 \quad =_{\mathcal{P}(\mathcal{V} \to \mathbb{I})} \quad l_1 \boxplus l_2$
- $l_1 - l_2 \quad =_{\mathcal{P}(\mathcal{V} \to \mathbb{I})} \quad l_1 \boxminus l_2$
- $i \times l_2 \quad =_{\mathcal{P}(\mathcal{V} \to \mathbb{I})} \quad i \boxtimes l_2 \quad$ *when* $\mathbb{I} \in \{\mathbb{R}, \mathbb{Q}\}$
  $i \times l_2 \quad \sqsubseteq_{\mathcal{P}(\mathcal{V} \to \mathbb{I})} \quad i \boxtimes l_2 \quad$ *when* $\mathbb{I} = \mathbb{Z}$
- $l_1 / i \quad =_{\mathcal{P}(\mathcal{V} \to \mathbb{I})} \quad l_1 \boxslash i \quad$ *when* $\mathbb{I} \in \{\mathbb{R}, \mathbb{Q}\}$
  $l_1 / i \quad \sqsubseteq_{\mathcal{P}(\mathcal{V} \to \mathbb{I})} \quad l_1 \boxslash i \quad$ *when* $\mathbb{I} = \mathbb{Z}$

●

Proof.

The exactness results when $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$ come from two facts. Firstly, the classical algebraic properties of the $+$, $-$, $\times$, $/$ operators, such as associativity and distributivity. And, secondly, the exactness of the corresponding interval abstractions $+^{Int}$, $-^{Int}$, $\times^{Int}$, and $/^{Int}$.

When $\mathbb{I} = \mathbb{Z}$, the associativity and distributivity of $+$, $-$, and $\times$ are still valid, and $+^{Int}$ and $-^{Int}$ are still exact, hence the exactness of $\boxplus$ and $\boxminus$. However, as $\times^{Int}$ is sound but not exact, so is $\boxtimes$. Finally, the soundness of $\boxslash$ comes from the fact that,

given any three intervals i, j, k, we have $(i + j) / k \quad \sqsubseteq_{\mathcal{P}(\mathcal{V}\to\mathbb{I})} \quad (i /^{Int}_{alt} k) +^{Int} (j /^{Int}_{alt} k)$ and $(i \times j) /^{Int}_{alt} k \quad \sqsubseteq_{\mathcal{P}(\mathcal{V}\to\mathbb{I})} \quad (i /^{Int}_{alt} k) \times^{Int} j$. These two properties are a consequence of the following inequalities applied to interval bounds: $\forall x, y \in \mathbb{R},\ \lfloor x \rfloor + \lfloor y \rfloor \leq \lfloor x + y \rfloor \leq adj(x + y) \leq \lceil x + y \rceil \leq \lceil x \rceil + \lceil y \rceil$ and $\forall x \in \mathbb{Z}, y \in \mathbb{R},\ \lceil xy \rceil \leq x \max(\lfloor y \rfloor, \lceil y \rceil)$, $\lfloor xy \rfloor \geq x \min(\lfloor y \rfloor, \lceil y \rceil)$. Note that these properties are not true if we replace $/^{Int}_{alt}$ by $/^{Int}$ — consider, in the first property, $i = j = [1, 1]$ and $k = [2, 2]$.

○

When $\mathbb{I} = \mathbb{Z}$, the fact that the division and multiplication are not exact can lead to much precision degradation. Consider, for instance, $(X \boxslash 2) \boxtimes 2$ that evaluates to the interval linear form $[0, 2] \times X$ while one might expect $X + [-1, 1]$ — due to truncation, dividing by two and multiplying the result by two will decrement any positive odd number and increment any negative odd number. In this example, we have composed the inexact — yet optimal by themselves — operators $X \boxslash 2 = [0, 1] \times X$ and $([0, 1] \times X) \boxtimes 2 = [0, 2] \times X$, and got a very imprecise result. The problem is that the intermediate computation $X \boxslash 2$ forces its result to have only integer interval bounds. A solution to this problem is to perform all computations using interval linear forms with real or rational interval coefficients, even when $\mathbb{I} = \mathbb{Z}$. In order to be sound, it is necessary to translate our integer expressions into expressions on reals by making the division-induced truncation explicit:

$$l \ \boxslash_{\mathbb{Z}} [a, b] \stackrel{\text{def}}{=} (l \boxslash [a, b]) \boxplus [-1 + 1/\min(|a|, |b|), 1 - 1/\min(|a|, |b|)] \ .$$

For instance, $(X \boxslash 2) \boxtimes 2$ in $\mathbb{Z}$ can be abstracted into $((X \boxslash 2) \boxplus [-0.5, 0.5]) \boxtimes 2$ in $\mathbb{R}$, which evaluates to $X + [-1, 1]$. Similar techniques will be studied in more details later for the implementation of the `floor` operator, in Sect. 6.2.6, and the abstraction of floating-point computations with rounding, in Sect. 7.4.

Intervalisation. In order to deal with non-linear features of expressions, we must introduce some kind of approximation. We present here an "intervalisation" operator ι that *flattens* an arbitrary interval linear form l into a single interval. In order to compute such an interval, the intervalisation operator requires some knowledge about the range of all the variables. We thus suppose that we have an abstract element $R^\sharp$ in some abstract domain $\mathcal{D}^\sharp$, as well as an abstract projection operator $\pi_i : \mathcal{D}^\sharp \to \mathcal{B}^{Int}$, for each variable $V_i \in \mathcal{V}$, such that:

$$\pi_i(R^\sharp) \supseteq \{\ v \mid \exists (v_1, \ldots, v_n) \in \gamma(R^\sharp),\ v = v_i\ \} \ .$$

Such an operator was indeed provided for the zone — Sect. 3.5.1 — and octagon — Sect. 4.4.3 — abstract domains. If $\mathcal{D}^\sharp$ is the interval abstract domain, then $\pi_i(R^\sharp)$ is simply the $i-$th component of the interval vector $R^\sharp$. The intervalisation operator $\iota(l)R^\sharp$, that takes as argument an interval linear form l and an abstract element $R^\sharp$, is defined as follows:

Definition 6.2.2. Interval linear form intervalisation ι.

$$\iota\left(i + \sum_{k=1}^{n} i_k \times V_k\right) R^\sharp \stackrel{\text{def}}{=} i \ +^{Int} \ \sum_{k=1}^{n}{}^{Int} \ (i_k \times^{Int} \pi_k(R^\sharp)) \ .$$

●

Theorem 6.2.2. Intervalisation soundness.

The intervalisation is sound: $\forall l, R^\sharp, \ l \ \sqsubseteq_{\gamma(R^\sharp)} \ \iota(l)R^\sharp$.

●

Proof. This is a consequence of the soundness of the $+^{Int}$ and $\times^{Int}$ operators, as well as the soundness of each π_i. ○

Generally, ι is not exact as it performs a *non-relational abstraction* before evaluating the interval linear form. Consider, for instance, the linear form $l \stackrel{\text{def}}{=} X - Y$ and the abstract element $R^\sharp$ such that $\gamma(R^\sharp) = \{ \ (x, x) \mid 0 \leq x \leq 1 \ \}$. We have $[\![l]\!]\rho = \{0\}$ for all $\rho \in \gamma(R^\sharp)$ while $\iota(l)R^\sharp = [0, 1] -^{Int} [0, 1] = [-1, 1] \supset \{0\}$. If we denote by $Int(R^\sharp) \stackrel{\text{def}}{=} \lambda i.\pi_i(R^\sharp)$ the conversion from $\mathcal{D}^\sharp$ to the interval abstract domain, then $\iota(l)R^\sharp$ is really a synonym for $[\![l]\!]^{Int}(Int(R^\sharp))$.

6.2.3 From Expressions to Interval Linear Forms

We now present a method to abstract an arbitrary expression into an interval linear form. Given an expression *expr* and an abstract environment $R^\sharp$, the linearisation $(\!| \mathit{expr} |\!)R^\sharp$ is defined inductively as follows:

Definition 6.2.3. Linearisation $(\!| \mathit{expr} |\!)R^\sharp$.

We first consider the linear cases:

- $(\!| V_i |\!)R^\sharp \stackrel{\text{def}}{=} [1, 1] \times V_i$
- $(\!| [a, b] |\!)R^\sharp \stackrel{\text{def}}{=} [a, b]$
- $(\!| -e |\!)R^\sharp \stackrel{\text{def}}{=} \boxminus (\!| e |\!)R^\sharp$
- $(\!| e_1 + e_2 |\!)R^\sharp \stackrel{\text{def}}{=} (\!| e_1 |\!)R^\sharp \ \boxplus \ (\!| e_2 |\!)R^\sharp$
- $(\!| e_1 - e_2 |\!)R^\sharp \stackrel{\text{def}}{=} (\!| e_1 |\!)R^\sharp \ \boxminus \ (\!| e_2 |\!)R^\sharp$
- $(\!| e_1 \times e_2 |\!)R^\sharp \stackrel{\text{def}}{=} [a, b] \ \boxtimes \ (\!| e_2 |\!)R^\sharp \ \textit{when} \ (\!| e_1 |\!)R^\sharp = [a, b]$
- $(\!| e_1 \times e_2 |\!)R^\sharp \stackrel{\text{def}}{=} [a, b] \ \boxtimes \ (\!| e_1 |\!)R^\sharp \ \textit{when} \ (\!| e_2 |\!)R^\sharp = [a, b]$

- $(\!|\, e_1 \,/\, e_2 \,|\!)R^\sharp \stackrel{\text{def}}{=} (\!|\, e_1 \,|\!)R^\sharp \boxslash [a,b]$ *when* $(\!|\, e_2 \,|\!)R^\sharp = [a,b]$

In the non-linear multiplication and division cases, one of the argument is flattened into an interval using the ι function so as to fall back into a linear case:

- $(\!|\, e_1 \times e_2 \,|\!)R^\sharp \stackrel{\text{def}}{=} \iota((\!|\, e_1 \,|\!)R^\sharp) \boxtimes (\!|\, e_2 \,|\!)R^\sharp$ *or*
  $(\!|\, e_1 \times e_2 \,|\!)R^\sharp \stackrel{\text{def}}{=} (\!|\, e_1 \,|\!)R^\sharp \boxtimes \iota((\!|\, e_2 \,|\!)R^\sharp)R^\sharp$
  (these two choices will be discussed in the following section)

- $(\!|\, e_1 \,/\, e_2 \,|\!)R^\sharp \stackrel{\text{def}}{=} (\!|\, e_1 \,|\!)R^\sharp \boxslash \iota((\!|\, e_2 \,|\!)R^\sharp)R^\sharp$

●

The following theorem proves that the linearised expression indeed over-approximates the behavior of the original expression:

Theorem 6.2.3. Soundness of the linearisation.

$\forall\, expr, R^\sharp,\ expr \sqsubseteq_{\gamma(R^\sharp)} (\!|\, expr \,|\!)R^\sharp$.

●

Proof.

The proof is easy by structural induction on *expr*. We need to use the soundness of the $\boxplus$, $\boxminus$, $\boxtimes$, $\boxslash$, and ι operators, but also the monotonicity of $+$, $-$, $\times$, and $/$ with respect to $\sqsubseteq_{\mathcal{P}(\mathcal{V}\to\mathbb{I})}$ for both their arguments.

Let us consider, as an example, the case $expr = e_1 + e_2$. By induction hypothesis, we have $e_1 \sqsubseteq_{\gamma(R^\sharp)} (\!|\, e_1 \,|\!)R^\sharp$ and $e_2 \sqsubseteq_{\gamma(R^\sharp)} (\!|\, e_2 \,|\!)R^\sharp$. By monotonicity of $+$, we have $e_1 + e_2 \sqsubseteq_{\gamma(R^\sharp)} (\!|\, e_1 \,|\!)R^\sharp + (\!|\, e_2 \,|\!)R^\sharp$. By soundness of $\boxplus$, $(\!|\, e_1 \,|\!)R^\sharp + (\!|\, e_2 \,|\!)R^\sharp \sqsubseteq_{\gamma(R^\sharp)} (\!|\, e_1 \,|\!)R^\sharp \boxplus (\!|\, e_2 \,|\!)R^\sharp = (\!|\, e_1 + e_2 \,|\!)R^\sharp$.

○

When performing the linearisation, we can only obtain soundness results as, in general, there is no best interval linear form, for $\sqsubseteq_{\gamma(R^\sharp)}$, that abstracts an expression. Moreover, $(\!|\, expr \,|\!)R^\sharp$ is not monotonic in its *expr* argument. Consider, for instance, the two expressions $X \,/\, X$ and $[1,1]$, and some abstract element $R^\sharp$ such that $\pi_X(R^\sharp) = [1, +\infty]$. Then, $(\!|\, X \,/\, X \,|\!)R^\sharp = [0,1] \times X \not\sqsupseteq_{\gamma(R^\sharp)} [1,1] = (\!|\, [1,1] \,|\!)R^\sharp$ even though $X \,/\, X \sqsubseteq_{\gamma(R^\sharp)} [1,1]$.

Applications. One of the nice properties enjoyed by interval linear forms is our ability to define rather precise yet efficient abstract transfer functions for such expressions in our zone, octagon, and zone-like family of abstract domains. Even though they are not optimal, they allow deriving new relational constraints. Given an arbitrary expression *expr*, we can

use the following sound abstract transfer functions:

$$\{\!|\, V \leftarrow \mathit{expr} \,|\!\}^\sharp_{\mathit{lin}} R^\sharp \quad \stackrel{\text{def}}{=} \quad \{\!|\, V \leftarrow (\!|\, \mathit{expr} \,|\!) R^\sharp \,|\!\}^\sharp_{\mathit{rel}} R^\sharp$$

$$\{\!|\, V \rightarrow \mathit{expr} \,|\!\}^\sharp_{\mathit{lin}} R^\sharp \quad \stackrel{\text{def}}{=} \quad \{\!|\, V \rightarrow (\!|\, \mathit{expr} \,|\!)(\{\!|\, V \rightarrow ? \,|\!\} R^\sharp) \,|\!\}^\sharp_{\mathit{rel}} R^\sharp$$

$$\{\!|\, \mathit{expr} \bowtie 0 \,?\, |\!\}^\sharp_{\mathit{lin}} R^\sharp \quad \stackrel{\text{def}}{=} \quad \{\!|\, (\!|\, \mathit{expr} \,|\!) R^\sharp \bowtie 0 \,?\, |\!\}^\sharp_{\mathit{rel}} R^\sharp$$

where some definitions for $\{\!|\, \cdot \,|\!\}^\sharp_{\mathit{rel}}$ are presented in Def. 3.6.4, 3.6.6, 4.4.7, and 5.3.5 for the zone and octagon abstract domains, as well as our zone-like family of relational domains. Recall that, up to now, our only option when considering non interval linear expressions on these domains was to use the poor $\{\!|\, \cdot \,|\!\}^\sharp_{\mathit{nonrel}}$ transfer functions implemented as a conversion to the interval domain, followed by an interval abstract transfer function, and a conversion back to our relational domain.

We can also apply our linearisation to *non-relational* domains to improve their precision. Indeed, as a side effect, the linearisation procedure is able to simplify symbolically expressions. For instance $(\!|\, X - (0.5 \times X) \,|\!) R^\sharp$ gives $0.5 \times X$. Suppose that $X \in [0, 1]$ in the interval abstract domain. We will get, without linearisation, $[\![\, X - (0.5 \times X) \,]\!]^{\mathit{Int}} = [0, 1] -^{\mathit{Int}} (0.5 \times^{\mathit{Int}} [0, 1]) = [0, 1] -^{\mathit{Int}} [0, 0.5] = [-0.5, 1]$ while we get, using linearisation, $[\![\, 0.5 \times X \,]\!]^{\mathit{Int}} = [0, 0.5]$ which is much more precise. By the exactness of the $\boxplus$, $\boxminus$, $\boxtimes$, and $\boxslash$ operators when $\mathbb{I} \in \{\, \mathbb{R}, \mathbb{Q} \,\}$ and the fact that ι mimics an interval abstract evaluation, we can see that $\{\!|\, V \leftarrow (\!|\, \mathit{expr} \,|\!) R^\sharp \,|\!\}^{\mathit{Int}} R^\sharp$ will always be more precise than $\{\!|\, V \leftarrow \mathit{expr} \,|\!\}^{\mathit{Int}} R^\sharp$. This is not always the case when $\mathbb{I} = \mathbb{Z}$, especially when a division forces us to use $/^{\mathit{Int}}_{\mathit{alt}}$ instead of $/^{\mathit{Int}}$. Moreover, it is difficult to compare formally the precision of the test and backward assignment transfer functions with and without linearisation. In doubt, we can always compute both and take their intersection as the result.

It is important to remark that the numerical abstract domain and our linearisation technique interact one with another to improve the analysis in a dynamic way: the abstract domain provides interval information to the linearisation that, in turn, refines the abstract transfer functions on-the-fly, yielding more precise invariants that will feed the linearisation with tighter bounds in the following of the analysis, etc. We already saw that the interval information was useful in the intervalisation part of the linearisation; we will see shortly that it can also be used to drive a linearisation strategy in non-linear cases.

6.2.4 Multiplication Strategies

Def. 6.2.3 is completely deterministic except in the case of a multiplication $e_1 \times e_2$ when neither $(\!|\, e_1 \,|\!) R^\sharp$ nor $(\!|\, e_2 \,|\!) R^\sharp$ is a simple interval. Indeed, when multiplying two interval linear forms not reduced to an interval, we have the choice of intervalising either the first argument or the second one, and this choice may greatly influence the overall linearised result. We now propose a few motivated strategies to choose which argument to intervalise.

All-Cases Strategy. A first strategy is to always try both choices and make Def. 6.2.3 return a *set* of interval linear forms that all over-approximate the given expression, instead of a single one. Then, the abstract transfer function can be evaluated independently on each returned interval linear form and, finally, the intersection $\cap^\sharp$ in $\mathcal{D}^\sharp$ of all the computed abstract elements is returned. This strategy has a cost which is exponential in the number of multiplications in the original expression, in the worst case, and so, may not be practical. We now propose deterministic strategies that always select *one* interval linear form. Of course, one can always construct a *compound* strategy by intersecting the abstract elements yielded by a few carefully selected deterministic strategies.

Interval-Size Local Strategy. A simple local strategy is to evaluates both $\iota((\!| e_1 |\!)R^\sharp)R^\sharp$ and $\iota((\!| e_2 |\!)R^\sharp)R^\sharp$ and choose to intervalise the expression e_i leading to the smallest *interval amplitude* $\max(\iota((\!| e_i |\!)R^\sharp)R^\sharp) - (\min \iota((\!| e_i |\!)R^\sharp)R^\sharp)$. For instance, consider the assignment $X \leftarrow Y \times Z$ in the zone abstract domain where $Y \in [0,1]$ and $Z \in [0,100]$. If we intervalise Y, we can infer the invariant $0 \leq X \leq Z$ while, if we intervalise Z, we get $0 \leq X \leq 100$, which is less precise. The extreme case holds when the amplitude of one interval is zero, meaning that the sub-expression is, semantically, a constant. Consider, for instance $e \stackrel{\text{def}}{=} Y \times e'$ where $R^\sharp$ implies that $Y = 10$; even though the expression Y is not syntactically a constant, it is reasonable to linearise e as $[10,10] \boxtimes (\!| e' |\!)R^\sharp$.

Relative-Size Local Strategy. We will see, in the following chapter, that floating-point computations incur some rounding, and so, expressions that would evaluate to a single value v if computed with reals turn out to evaluate to an interval $[v(1-\epsilon), v(1+\epsilon)]$ with a small relative error ϵ. Consider, for instance, the following expression $e \times ([0.9, 1.1] \times X)$ where the expression e ranges in $[0,1]$ and $X = 100$. The expression $[0.9, 1.1] \times X$ corresponds to X, up to a relative non-deterministic rounding error of amplitude 0.2. It seems reasonable to intervalise $[0.9, 1.1] \times X$ that still represents a constant value, rather than the complex expression e, even though the former has an amplitude of 20. Unfortunately, the preceding strategy will not give the expected result. A solution, proposed by J. Feret,[1] is to compare the *relative* amplitude $(\max(\iota((\!| e_i |\!)R^\sharp)R^\sharp) - \min(\iota((\!| e_i |\!)R^\sharp)R^\sharp))/(|\max(\iota((\!| e_i |\!)R^\sharp)R^\sharp)| + |\min(\iota((\!| e_i |\!)R^\sharp)R^\sharp)|)$ instead of the absolute amplitude $\max(\iota((\!| e_i |\!)R^\sharp)R^\sharp) - \min(\iota((\!| e_i |\!)R^\sharp)R^\sharp)$.

Simplification-Driven Global Strategy. A nice property of the linearisation is that it automatically performs simplification. In order to give the linearisation more opportunity to simplify statements in an expression *expr*, we prefer to keep in symbolic form multiplication arguments containing variables that also appear in other sub-expressions of *expr*. Consider, for instance, the assignment $Z \leftarrow X - (Y \times X)$ where $Y \in [0,1]$ and $X \in [0, 0.5]$. We

[1] Private communication during the Astrée project. Unpublished.

may prefer to intervalise Y in $Y \times X$ to get $Z \leftarrow [0,1] \times X$ instead of intervalising X to get $Z \leftarrow X + [0,0.5] \times Y$ even though $\iota(Y)R^\sharp$ is larger than $\iota(X)R^\sharp$. Indeed, in the interval domain, we would get $Z \in [0,0.5]$ in the first case instead of $Z \in [0,1]$ for the second case. Moreover, in the zone abstract domain, the assignment $Z \leftarrow [0,1] \times X$ can be modeled precisely — by inferring the constraint $0 \leq Z \leq X$ — while the assignment $Z \leftarrow X + [0,0.5] \times Y$ cannot — because three distinct variables are involved.

Homogeneity Global Strategy. We now provide a refinement of the preceding strategy to deal with the following common example:

Example 6.2.1. Linear interpolation computation.

Consider the complex assignment at line ④ in the following code fragment:

$$\begin{array}{ll} ① & X \leftarrow [0,1]; \\ ② & Y \leftarrow [0,10]; \\ ③ & Z \leftarrow [0,20]; \\ ④ & T \leftarrow X \times Y - X \times Z + Z \end{array}$$

As both X and Z appear in several branches of the expression, the previous strategy does not provide any criterion for the linearisation of $X \times Z$. By rewriting the assignment as $T \leftarrow X \times Y + (1 - X) \times Z$ it becomes clear that we are computing a linear interpolation between variables Y and Z. We now explain why X should be linearised. If we choose to intervalise X, we get $T \leftarrow [0,1] \times Y + [0,1] \times Z$ and we are able to prove in the interval domain that, after the assignment, $\min Y + \min Z \leq T \leq \max Y + \max Z$. For instance, we are able to prove that a linear combination of two positive numbers is always positive. If, however, we intervalise Y and Z, we get the assignment $T \leftarrow [\min Y - \max Z, \max Y - \min Z] \times X + Z$, that is, $T \leftarrow [-20, 10] \times X + Z$, and we have lost too much information to be able to prove that T is always positive.

On the original expression, $T \leftarrow X \times Y - X \times Z + Z$, the previous strategy does not provide any criterion for the linearisation of $X \times Z$. Worse, on the second expression, $T \leftarrow X \times Y + (1 - X) \times Z$, the previous strategy will insist on intervalising Y and Z to keep the variable X that appears in two sub-expressions. Our solution is to try and intervalise a set of variables, as small as possible, that makes the expression *homogenate*, meaning that all monomials have the same degree after developing. This strategy will successfully choose to intervalise X. Moreover, it will still work in more complex cases, such as the following linear interpolation with re-normalisation:

$$T \leftarrow (((X - a) \times Y) \,/\, (b - a)) \;-\; (((X - a) \times Z) /\,/\, (b - a)) \;+\; Z$$

where $X \in [a,b]$. A final example is the following bi-linear interpolation that requires the intervalisation of *two* variables, X and X', to get an homogenate expression:

$$\begin{array}{rl} T \leftarrow & X \times X' \times A \;+\; (1 - X) \times X' \times B \;+ \\ & X \times (1 - X') \times C \;+\; (1 - X) \times (1 - X') \times D \end{array}$$

where $X, X' \in [0,1]$ are the interpolation coordinates.

●

6.2.5 From Expressions to Quasi-Linear Forms

When considering abstract transfer functions in the polyhedron domain, interval linear forms are still too generic to be modeled explicitly: we only have transfer functions for quasi-linear forms, that is, interval linear forms $[a_0, b_0] + \sum_k a_k \times V_k$ where only the constant coefficient $[a_0, b_0]$ is allowed to be a non-singleton interval.

We now present an operator $\mu(l)R^\sharp$ that can be used to abstract an interval linear form l into a quasi-linear form in a given abstract environment $R^\sharp$:

Definition 6.2.4. μ operator for removing non-singleton coefficients.

We define the μ operator as follows:

$$\mu\left([a,b] + \sum_{k=1}^{n} [a_k, b_k] \times V_k\right) R^\sharp \stackrel{\text{def}}{=}$$

$$\left([a,b] +^{Int} \sum_{k=1}^{n}{}^{Int} ([(a_k - b_k)/2, (b_k - a_k)/2] \times^{Int} \pi_k(R^\sharp))\right) \quad +$$

$$\sum_{k=1}^{n} [(a_k + b_k)/2, (a_k + b_k)/2] \times V_k$$

with the convention that $\forall a \in \mathbb{I} \cup \{+\infty, -\infty\}$, $a + (+\infty) \stackrel{\text{def}}{=} (-\infty) + a \stackrel{\text{def}}{=} 0$ when computing $a_k + b_k$. However, $(+\infty) - (-\infty) \stackrel{\text{def}}{=} (+\infty)$ and $(-\infty) - (+\infty) \stackrel{\text{def}}{=} (-\infty)$ when computing $b_k - a_k$ and $a_k - k_k$. When considering interval linear forms in $\mathbb{Z}$, $(a_k - b_k)/2$ should be rounded towards $-\infty$, $(b_k - a_k)/2$ towards $+\infty$, and $(a_k + b_k)/2$ either way.

●

This operation is implemented by "distributing" the weight $b_k - a_k$ of each variable coefficient into the constant component of the interval linear form. It returns a sound quasi-linear form:

Theorem 6.2.4. Soundness of the μ operator.

$\mu(l)R^\sharp$ soundly over-approximates l: $\forall l, R^\sharp$, $l \sqsubseteq_{\gamma(R^\sharp)} \mu(l)R^\sharp$.

●

Proof.

This is a consequence of $[a,b] \times V_k \sqsubseteq_{\mathcal{P}(\mathcal{V}\to\mathbb{I})} ([(a+b)/2, (a+b)/2] \times V_k) + ([(a-b)/2, (b-a)/2] \times V_k) \sqsubseteq_{\gamma(R^\sharp)} ([(a+b)/2, (a+b)/2] \times V_k) + ([(a-b)/2, (b-a)/2] \times \pi_k(R^\sharp))$.

Whenever one bound of an interval variable coefficient is infinite, our adapted + and − operators transfer all the weight into the constant coefficient and cancel out the variable coefficient.

○

The converse inequality does not hold in general as μ loses some relational information. Consider, for instance, the expression $l \stackrel{\text{def}}{=} [0,1] \times V$ where V ranges in $[0,1]$. If ρ maps V to 0, then $[\![l]\!]\rho = \{0\}$ while $[\![\mu(l)R^\sharp]\!]\rho = [\![[0.5,0.5] \times V + [-0.5,0.5]]\!]\rho = [-0.5,0.5] \supset \{0\}$.

Application. In order to design an abstract transfer function for arbitrary expressions in the polyhedron domain, we first abstract the expression into an interval linear form using $(\!| \cdot |\!)$, and then abstract further the interval linear form into a quasi-linear form using μ. For instance, $\{\!\{ V \leftarrow expr \}\!\}^{Poly}_{lin} R^\sharp$ can be implemented as:

$$\{\!\{ V \leftarrow \mu((\!| expr |\!)R^\sharp)R^\sharp \}\!\}^{Poly}_{exact} R^\sharp \ .$$

Note that, even though the polyhedron domain is more precise than the octagon domain, because μ induces some extra abstraction for expressions, the overall analysis result may be actually more precise with octagons than with polyhedra.

6.2.6 Extending Numerical Expressions

Our Simple language syntax is quite limited; a realistic programming language will include many more constructs and, in particular, more operators in numerical expressions. In a non-relational domain, it is sufficient to add a forward and a backward abstract operator for each new concrete numerical operator introduced. For relational domains, however, the abstract transfer functions must be totally redesigned with each new concrete operator as transfer functions act globally on expressions. We explain here how the linearisation technique allows extending easily our transfer functions compositionally in relational abstract domains, using interval abstractions locally.

Generic Extensions. Suppose that we have a fully featured abstract domain $\mathcal{D}^\sharp$, and then add a new $k-$ary numerical operator $F : \mathbb{I}^k \to \mathcal{P}(\mathbb{I})$. We have already seen two generic ways to extend $\mathcal{D}^\sharp$. The first, quite imprecise, solution is to use the generic fall-back transfer functions of Sect. 2.4.3 whenever an expression uses F. A second, more precise, solution is to use an abstract domain that allows F in its abstract transfer functions. For instance, if F has a forward F^{Int} and a backward $\overleftarrow{F}^{Int}$ abstraction in the interval domain, we can design abstract transfer functions for expressions containing F, as explained in Sect. 2.4.4. Then, for expressions containing F in the octagon domain, we can use the interval-based abstractions of Def. 4.4.5.

We now propose a linearisation-based abstraction technique. We only need a forward abstraction F^{Int} of F in the interval abstract domain. The linearisation of Def. 6.2.3 is then extended as follows:

$$(\!| F(\ldots, e_i, \ldots) |\!) R^\sharp \stackrel{\text{def}}{=} F^{Int}(\ldots,\ \iota((\!| e_i |\!) R^\sharp) R^\sharp,\ \ldots) \ .$$

By feeding this linearised expression to an interval linear form transfer function, we can obtain more precise results than when using the interval-based transfer function. One reason is that F^{Int} is fed with $\iota((\!| e_i |\!) R^\sharp) R^\sharp$ intervals which can be smaller than $[\![e_i]\!]^{Int}(Int(R^\sharp))$ because of the simplifications performed by our linearisation in each $(\!| e_i |\!)$. Another reason is that expression parts that do not use the F operator may be linearised as non-trivial interval linear forms that a relational domain $\mathcal{D}^\sharp$ can exploit. However, we lose any relationship between the different arguments of F, and between F's arguments and all the other expression parts.

The `floor` Operator. As an example, we propose to extend our Simple language with the `floor` numerical operator. The syntax and concrete semantics are extended as follows:

$$\begin{array}{lcl} \mathit{expr} & ::= & \texttt{floor}\,(\mathit{expr}) \\ [\![\, \texttt{floor}\,(\mathit{expr}) \,]\!]\rho & \stackrel{\text{def}}{=} & \{\ \lfloor x \rfloor \mid x \in [\![\, \mathit{expr} \,]\!]\rho\ \} \end{array}$$

As $\lambda x.\lfloor x \rfloor$ is monotonic, the best abstraction in the interval domain is simply:

$$\texttt{floor}^{Int}([a, b]) \stackrel{\text{def}}{=} [\ \lfloor a \rfloor,\ \lfloor b \rfloor\ ]$$

which can be used directly to linearise any expression containing the *floor* operator as follows:

$$(1) \quad (\!| \texttt{floor}\,(e) |\!) R^\sharp \stackrel{\text{def}}{=} \texttt{floor}^{Int}\,(\iota((\!| e_1 |\!) R^\sharp) R^\sharp) \ .$$

Another, non-generic, way to linearise the `floor` operator, due to D. Monniaux,[2] is based on the fact that $x - 1 < \lfloor x \rfloor \leq x$. Def. 6.2.3 is extended as follows:

$$(2) \quad (\!| \texttt{floor}\,(e) |\!) R^\sharp \stackrel{\text{def}}{=} (\!| e |\!) R^\sharp \ \boxplus\ [-1, 0] \ .$$

The benefit of (2) is that it keeps its argument in symbolic form instead of intervalising it, which can lead in some cases to a more precise analysis.

[2] Private communication during the Astrée project. Unpublished.

Example 6.2.2. Modulo computation.

Consider the following assignment, in $\mathbb{R}$:

$$Y \leftarrow X - (M - m) \times (\texttt{floor}\,((X - m) / (M - m)))$$

where M and m are constants. It computes a value in $[m, M]$ equal to X modulo $M - m$ and put it into Y.
Suppose that $X \in [0, 100]$, $m = 10$, and $M = 20$, then, in the interval domain without linearisation, we assign to Y the interval:

$$\begin{array}{rl} & [0,100] -^{Int} (10 \times^{Int} \texttt{floor}^{Int} ([-10,90] /^{Int} 10)) \\ = & [0,100] -^{Int} (10 \times^{Int} \texttt{floor}^{Int} ([-1,9])) \\ = & [0,100] -^{Int} (10 \times [-1,9]) \\ = & [-90,110] \end{array}$$

which is not very precise. Using the interval domain with linearisation and (1), we would get exactly the same result. However, using (2), we linearise the expression as:

$$X - ((X - 10) / 10 + [-1, 0]) \times 10 = X - (X - 10 + [-10, 0]) = [10, 20]$$

which is the exact range of Y.

●

6.3 Symbolic Constant Propagation

6.3.1 Motivation

As seen in the preceding section, the automatic symbolic simplification implied by our linearisation procedure turns out to be quite an interesting feature. It enables us to gain some precision on complex, linear and non-linear, expressions without the burden of using an abstract domain able to manipulate these kinds of expressions directly. Our linearisation, however, is very sensitive to even the simplest program transformations. Consider, for instance, the statement $V \leftarrow X - (0.5 \times X)$ which is linearised into $V \leftarrow 0.5 \times X$. If this statement is broken into two assignments $Y \leftarrow 0.5 \times X$; $V \leftarrow X - Y$, using Y as intermediate variable, no simplification in $X - Y$ can occur. The classical solution to this problem is to use an abstract domain that is able to represent the information $Y = 0.5 \times X$, and has transfer functions able to use this information in the assignment $V \leftarrow X - Y$. This would imply using the polyhedron abstract domain.

We propose here a much lighter solution to make our linearisation, and so, the interval, zone, and octagon domains, robust against such simple program transformations.

6.3.2 Symbolic Constant Propagation Domain

The gist of our method is a modification of the well-known constant propagation domain, $\mathcal{D}^{Cst}$, proposed by Kildall [Kil73] and recalled in Def. 5.4.1. However, instead of a constant in $\mathbb{I}$, we associate to each variable a symbolic information, that is, an expression tree.

Expressions. We first define a structure on the set of expression trees $\mathcal{B}^{Symb}$, enriched with a least and greatest elements:

Definition 6.3.1. Symbolic basis $\mathcal{B}^{Symb}$.

1. *The symbolic basis $\mathcal{B}^{Symb}$ contains:*
   - *all syntactic numerical expressions expr of our* Simple *language as described in Fig. 2.1: they are formed over variables $V \in \mathcal{V}$, constant intervals $[a, b]$ where $a \in \mathbb{I} \cup \{-\infty\}$, $b \in \mathbb{I} \cup \{+\infty\}$, and $a \leq b$, and the arithmetic operators $+$, $-$, $\times$, and $/$,*
   - *a bottom element $\bot^{Symb}_{\mathcal{B}}$ representing "no value",*
   - *and a top element $\top^{Symb}_{\mathcal{B}}$ representing "any value".*
2. *As for the constant basis $\mathcal{B}^{Cst}$, we use a flat ordering:*
$$\forall X^\sharp \in \mathcal{B}^{Symb},\ \bot^{Symb}_{\mathcal{B}} \sqsubseteq^{Symb}_{\mathcal{B}} X^\sharp \sqsubseteq^{Symb}_{\mathcal{B}} \top^{Symb}_{\mathcal{B}} \ .$$
   The least upper bound $\sqcup^{Symb}_{\mathcal{B}}$ and the greatest lower bound $\sqcap^{Symb}_{\mathcal{B}}$ are defined easily — see Def. 5.4.1.4.
3. *Each element $X^\sharp \in \mathcal{B}^{Symb}$ represents a function that associates to each environment $\mathcal{V} \to \mathbb{I}$ a set of values in $\mathcal{P}(\mathbb{I})$:*
$$\gamma^{Symb}_{\mathcal{B}}(X^\sharp) \stackrel{\text{def}}{=} \begin{cases} \lambda\rho.\emptyset & \text{if } X^\sharp = \bot^{Symb}_{\mathcal{B}} \\ \lambda\rho.\mathbb{I} & \text{if } X^\sharp = \top^{Symb}_{\mathcal{B}} \\ \lambda\rho.[\![X^\sharp]\!]\rho & \text{otherwise} \end{cases}$$
   $\gamma^{Symb}_{\mathcal{B}}$ is obviously monotonic, however it is not a $\sqcap-$morphism and there is generally no best abstraction for an arbitrary function $(\mathcal{V} \to \mathbb{I}) \to \mathcal{P}(\mathbb{I})$.

●

We now present two utility functions on symbolic expressions. Firstly, the *occurrence function occ* : $\mathcal{B}^{Symb} \to \mathcal{P}(\mathcal{V})$ that returns the set of variables appearing in an expression and can be defined by structural induction as follows:

Definition 6.3.2. Variable occurrence function *occ* **.**

$$\begin{array}{llll} occ\,(\bot_{\mathcal{B}}^{Symb}) & \stackrel{\text{def}}{=} \emptyset & occ\,(\top_{\mathcal{B}}^{Symb}) & \stackrel{\text{def}}{=} \emptyset \\ occ\,(V) & \stackrel{\text{def}}{=} \{V\} & occ\,([a,b]) & \stackrel{\text{def}}{=} \emptyset \\ occ\,(-expr) & \stackrel{\text{def}}{=} occ\,(expr) & occ\,(expr_1 \diamond expr_2) & \stackrel{\text{def}}{=} occ\,(expr_1) \cup occ\,(expr_2) \end{array}$$

●

Secondly, the *substitution function* $subst\,(e_1, V, e2) : (\mathcal{B}^{Symb} \times \mathcal{V} \times \mathcal{B}^{Symb}) \to \mathcal{B}^{Symb}$ that substitutes *all* occurrences of a variable V in an expression e_1 with the expression e_2 and can be defined by structural induction as follows:

Definition 6.3.3. Substitution function *subst* **.**

The following substitution rules should be tried, in order:

$$\begin{array}{lll} subst\,(\cdot,\ \cdot,\ \bot_{\mathcal{B}}^{Symb}) & \stackrel{\text{def}}{=} & \bot_{\mathcal{B}}^{Symb} \\ subst\,(\bot_{\mathcal{B}}^{Symb},\ \cdot,\ \cdot) & \stackrel{\text{def}}{=} & \bot_{\mathcal{B}}^{Symb} \\ subst\,(\top_{\mathcal{B}}^{Symb},\ \cdot,\ \cdot) & \stackrel{\text{def}}{=} & \top_{\mathcal{B}}^{Symb} \\ subst\,(expr,\ V,\ \top_{\mathcal{B}}^{Symb}) & \stackrel{\text{def}}{=} & \begin{cases} \top_{\mathcal{B}}^{Symb} & \text{if } V \in occ\,(expr) \\ expr & \text{otherwise} \end{cases} \\ subst\,([a,b],\ V,\ expr) & \stackrel{\text{def}}{=} & [a,b] \\ subst\,(V,\ W,\ expr) & \stackrel{\text{def}}{=} & \begin{cases} expr & \text{if } V = W \\ V & \text{otherwise} \end{cases} \\ subst\,(-expr_1,\ V,\ expr) & \stackrel{\text{def}}{=} & -subst\,(expr_1,\ V,\ expr) \\ subst\,(expr_1 \diamond expr_2,\ V,\ expr) & \stackrel{\text{def}}{=} & subst\,(expr_1,\ V,\ expr) \diamond subst\,(expr_2,\ V,\ expr) \end{array}$$

●

Note that the bottom element $\bot_{\mathcal{B}}^{Symb}$ is always absorbing while the top element $\top_{\mathcal{B}}^{Symb}$ is absorbing only when the expression *expr* contains the substituted variable V or is $\top_{\mathcal{B}}^{Symb}$ itself.

Abstract Environments. We are now ready to define the *symbolic constant propagation abstract domain* $\mathcal{D}^{Symb}$ as the set of abstract environments that associate an expression tree — or $\bot_{\mathcal{B}}^{Symb}$ or $\top_{\mathcal{B}}^{Symb}$ — to each variable:

Definition 6.3.4. Symbolic constant abstract domain $\mathcal{D}^{Symb}$.

1. $\mathcal{D}^{Symb}$ *is the set of functions* $\rho^\sharp \in \mathcal{V} \to \mathcal{B}^{Symb}$ *without cyclic dependency, that is, there does not exist pair-wise distinct variables* $V_1, \ldots, V_m \in \mathcal{V}$*, such that* $V_2 \in occ\,(\rho^\sharp(V_1))$*,* $\ldots$*,* $V_m \in occ\,(\rho^\sharp(V_{m-1}))$*, and* $V_1 \in occ\,(\rho^\sharp(V_m))$*.*

2. $\sqsubseteq^{Symb}$ *is the point-wise extension to* $\mathcal{V}$ *of the flat order* $\sqsubseteq_{\mathcal{B}}^{Symb}$.

3. *The meaning of an abstract environment* $\rho^\sharp : \mathcal{V} \to \mathcal{B}^{Symb}$ *is:*

$$\gamma^{Symb}(\rho^\sharp) \stackrel{\text{def}}{=} \{\ \rho = (v_1, \ldots, v_n) \in \mathbb{I}^n \mid \forall i,\ v_i \in (\gamma_{\mathcal{B}}^{Symb}(\rho^\sharp(v_i)))(\rho)\ \}$$

which is a monotonic concretisation, but not a $\sqcap-$*morphism. There is no best abstraction, so, we are in the concretisation-based framework of Sect. 2.2.2.*

●

Given an abstract environment $\rho^\sharp$, we can derive the following set $\mathcal{R}(\rho^\sharp)$ of *rewriting rules*:

$$\mathcal{R}(\rho^\sharp) \stackrel{\text{def}}{=} \{\ expr \to subst\,(expr,\ V_i,\ \rho^\sharp(V_i)) \mid \forall expr \text{ and } i \text{ such that } \rho^\sharp(V_i) \neq \top_{\mathcal{B}}^{Symb}\ \}\ .$$

Thanks to the non-cyclicity condition of Def. 6.3.4.1, this rewriting system terminates on any expression *expr*. Moreover, it is locally confluent as $subst\,(\ subst\,(\ subst\,(\ expr,\ V_1,\ e_1),\ V_2,\ e_2),\ V_1,\ e_1) = subst\,(\ subst\,(\ subst\,(\ expr,\ V_2,\ e_2),\ V_1,\ e_1),\ V_2,\ e_2)$ and, as a consequence, it is strongly normalising: all sequences starting from an expression *expr* terminate on the same expression, which will be denoted by $subst^*(expr,\ \rho^\sharp)$. If there is some V_i such that $\rho^\sharp(V_i) = \bot_{\mathcal{B}}^{Symb}$, then all sequences terminate on $\bot_{\mathcal{B}}^{Symb}$ due to the absorbing property of $\bot_{\mathcal{B}}^{Symb}$ on *subst*. We obtain $\top_{\mathcal{B}}^{Symb}$ only if the original argument was $\top_{\mathcal{B}}^{Symb}$. Otherwise, we get an expression were all occurring variables $V_i \in occ\,(subst^*(expr,\ \rho^\sharp))$ are such that $\rho^\sharp(V_i) = \top_{\mathcal{B}}^{Symb}$.

We now present the fundamental property of substitutions that will allow us to perform symbolic constant propagation on arbitrary expressions. Given an environment $\rho^\sharp \in \mathcal{D}^{Symb}$ and an expression *expr*, any $subst\,(expr,\ V_i,\ \rho^\sharp(V_i))$ over-approximates *expr* with respect to $\sqsubseteq_{\gamma^{Symb}(\rho^\sharp)}$, using our ordering on expressions defined in Def. 6.2.1:

Theorem 6.3.1. Substitution soundness.

$$\forall expr,\ \rho^\sharp,\ V_i, \quad expr \sqsubseteq_{\gamma^{Symb}(\rho^\sharp)} subst\,(expr,\ V_i,\ \rho^\sharp(V_i))\ .$$

●

Proof. This is an easy consequence of the definition of γ^{Symb} — Def. 6.3.4.3 — and the inductive nature of the definition of $[\![\ expr\ ]\!]$ — Def. 2.2. ○

Note that the converse inequality does not hold in general. Consider, for instance, the expression $expr \stackrel{\text{def}}{=} X - X$ and the environment $\rho^\sharp \stackrel{\text{def}}{=} [X \mapsto [0,1]]$. Then, $subst\,(expr,\ X,\ [0,1]) = [0,1] - [0,1]$ which is strictly less precise than $X - X$. There is, however, no loss of precision if $\rho^\sharp(V_i)$ is deterministic, that is, always evaluates to a singleton: $\forall \rho \in \gamma^{Symb}(\rho^\sharp),\ |[\![\ \rho^\sharp(V_i)\ ]\!]\rho| \leq 1$. This is the case whenever $\rho^\sharp(V_i)$ is not $\top_{\mathcal{B}}^{Symb}$ and has no interval leaf $[a, b]$ such that $a \neq b$.

A consequence of Thm. 6.3.1 is that it is sound to replace, in any environment $\rho^\sharp$, the value associated to a variable V_i by $subst\,(\rho^\sharp(V_i),\, V_j,\, \rho^\sharp(V_j))$, for any V_j:

$$\gamma^{Symb}(\rho^\sharp) \subseteq \gamma^{Symb}([V_i \mapsto subst\,(\rho^\sharp(V_i),\, V_j,\, \rho^\sharp(V_j))]) \ .$$

Set-Theoretic Operators. We propose the following union and intersection abstractions:

Definition 6.3.5. Abstract union and intersection of symbolic environments.

$$(\rho_1^\sharp \cup^{Symb} \rho_2^\sharp)(V_k) \stackrel{\text{def}}{=} \begin{cases} \rho_2^\sharp(V_k) & \text{if } \exists l,\, \rho_1^\sharp(V_l) = \bot_{\mathcal{B}}^{Symb} \\ \rho_1^\sharp(V_k) & \text{if } \exists l,\, \rho_2^\sharp(V_l) = \bot_{\mathcal{B}}^{Symb} \\ \rho_1^\sharp(V_k) \sqcup_{\mathcal{B}}^{Symb} \rho_2^\sharp(V_k) & \text{otherwise} \end{cases}$$

$$\rho_1^\sharp \cap^{Symb} \rho_2^\sharp \quad \stackrel{\text{def}}{=} \quad \rho_1^\sharp$$

●

The union abstraction effectively forgets the symbolic information for a variable if the expressions associated to this variable in the two arguments are not syntactically equal, except when it detects an inconsistency $\bot_{\mathcal{B}}^{Symb}$ in which case the other argument is returned unchanged. For the intersection abstraction, we simply choose the left argument. We could have chosen the right one as well. However, it is important to pick all the information in the same environment argument if we want the non-cyclicity property of Def. 6.3.4.1 to hold on the result. Trying to refine $\cap^{Symb}$ so that it associates $\rho_1^\sharp(V_k) \sqcap_{\mathcal{B}}^{Symb} \rho_2^\sharp(V_k)$ to V_k, for instance, is incorrect. For the following non-cyclic arguments: $\rho_1^\sharp \stackrel{\text{def}}{=} [X \mapsto Y,\, Y \mapsto \top_{\mathcal{B}}^{Symb}]$ and $\rho_2^\sharp \stackrel{\text{def}}{=} [X \mapsto \top_{\mathcal{B}}^{Symb},\, Y \mapsto X]$, this would return the cyclic result $(\rho_1^\sharp \cap^{Symb} \rho_2^\sharp) = [X \mapsto Y,\, Y \mapsto X]$. There is no such problem for the union as dependencies between variables can only be kept or removed, never added.

As our order is the point-wise extension of a flat one, $\mathcal{D}^{Symb}$ has a finite height and there is no need for a widening nor a narrowing operator.

Transfer Functions. The main effect of an assignment $V_i \leftarrow expr$ is to replace the symbolic value for V_i with $expr$ in $\rho^\sharp$. There are, however, two subtle points:

- First, we must take care to invalidate all the symbolic expressions in $\rho^\sharp$ where V_i occurs as they are no longer true after V_i has been modified. A straightforward solution would be to replace with $\top_{\mathcal{B}}^{Symb}$ any expression where V_i appears. A more precise option is to substitute V_i in these expressions with its symbolic value $\rho^\sharp(V_i)$ as known *before* the assignment. Thanks to this, an assignment such as $Y \leftarrow Z$ in

the abstract environment $[X \mapsto Y + 1, Y \mapsto Z \times 2]$ will result in the environment $[X \mapsto (Z \times 2) + 1, Y \mapsto Z]$ instead of $[X \mapsto \top_{\mathcal{B}}^{Symb}, Y \mapsto Z]$.

- Another problem is that we cannot associate *expr* to V_i if V_i occurs in *expr*, as in the assignment $V_i \leftarrow V_i + 1$. Our solution is also to substitute V_i with its symbolic value as known before the assignment in *expr* and associate the resulting expression to V_i in the new abstract environment. Thus, the assignment $X \leftarrow X + 1$ in the environment $[X \mapsto Y + 2, Z \mapsto X \times 2]$ will return the environment $[X \mapsto (Y + 2) + 1, Z \mapsto (Y + 2) \times 2]$.

This gives the following assignment transfer function:

Definition 6.3.6. Assignment transfer function.

$$(\{\!|\, V_i \leftarrow expr \,|\!\}^{Symb}\rho^\sharp)(V_k) \stackrel{\text{def}}{=} \begin{cases} subst\,(expr,\ V_i,\ \rho^\sharp(V_i)) & \text{if } k = i \\ subst\,(\rho^\sharp(V_k),\ V_i,\ \rho^\sharp(V_i)) & \text{if } k \neq i \end{cases}$$

●

The forget operator $\{\!|\, V_i \leftarrow ? \,|\!\}^{Symb}$ can be implemented using the same principles:

Definition 6.3.7. Forget transfer function.

$$(\{\!|\, V_i \leftarrow ? \,|\!\}^{Symb}\rho^\sharp)(V_k) \stackrel{\text{def}}{=} \begin{cases} \top_{\mathcal{B}}^{Symb} & \text{if } k = i \\ subst\,(\rho^\sharp(V_k),\ V_i,\ \rho^\sharp(V_i)) & \text{if } k \neq i \end{cases}$$

●

As remarked before, a backward assignment $X \rightarrow expr$ in a constraint system simply amounts to substituting X with *expr* in each constraint. Unfortunately, carefree substitution can lead to cyclic dependencies in an abstract environment, which breaks Def. 6.3.4.1. Consider, for instance, the environment $\rho^\sharp \stackrel{\text{def}}{=} [Y \mapsto X + 1]$ and the backward assignment $X \rightarrow Y$ that leads, by substitution, to the invalid environment $[Y \mapsto subst\,(X + 1,\ X,\ Y)] = [Y \mapsto Y + 1]$. In equation-based relational abstract domains, a satisfiability procedure exists to determine whether such equation systems correspond to an empty environment set and, when they do not, which cyclic equations can be safely removed. Such procedures for generic expressions either do not exist, or are very costly. As a consequence, we choose to abstract backward assignments as the forget operator, following the generic fall-back definitions of Sect. 2.4.3:

$$\{\!|\, V_i \rightarrow expr \,|\!\}^{Symb}\rho^\sharp \stackrel{\text{def}}{=} \{\!|\, V_i \leftarrow ? \,|\!\}^{Symb}\rho^\sharp \ .$$

Abstracting tests precisely is also quite problematic for several reasons. Firstly, only equations of the form $X = expr$ can be introduced. Some tests, such as $X \leq Y$ cannot be

converted into this form while others, such as $X = Y$, lead to several possible environments — should we associate X to Y, or Y to X ? Then, as for backward assignments, adding an arbitrary binding $X \mapsto expr$ may introduce cyclicity. As a consequence, we abstract tests as the identity, following Sect. 2.4.3:

$$\{\!\!\{ \, test \, ? \, \}\!\!\}^{Symb} \rho^\sharp \stackrel{\text{def}}{=} \rho^\sharp \ .$$

Summary. In effect, the symbolic constant abstract domain $\mathcal{D}^{Symb}$ collects the syntactic expressions assigned to all variables, at each program point, and propagates them almost unchanged until they are invalidated by new assignments. The only operation applied to abstract elements is the substitution that allows *gluing* one syntactic expression at the leafs of another one to avoid too much precision degradation. There is very little semantics embedded within this domain: it treats all mathematical operators as uninterpreted symbols.

6.3.3 Interaction With a Numerical Abstract Domain

The symbolic domain $\mathcal{D}^{Symb}$ is not of much use by itself to discover numerical invariants — indeed, there is no proper test abstraction and the union is very imprecise. It is designed to be used in tandem with a regular numerical abstract domain $\mathcal{D}^\sharp$: the symbolic domain gathers assignment information, which is then used to refine on-the-fly the linearisation procedure used in $\mathcal{D}^\sharp$. Given $\mathcal{D}^\sharp$, we now construct an analysis on the *product* domain $\mathcal{D}^\sharp \times \mathcal{D}^{Symb}$.

Abstract Operators. Following the construction on the regular product of abstract domains, presented in Sect. 2.2.6, all our operators, such as the union, intersection, widening, etc., will operate component-wise on element pairs in $\mathcal{D}^\sharp \times \mathcal{D}^{Symb}$. For instance, we define the product union as follows:

$$(R_1^\sharp, \rho_1^\sharp) \cup^{\sharp \times Symb} (R_2^\sharp, \rho_2^\sharp) \stackrel{\text{def}}{=} (R_1^\sharp \cup^\sharp R_2^\sharp, \ \rho_1^\sharp \cup^{Symb} \rho_2^\sharp) \ .$$

To gain a little more precision, we can also, as in the coalescent product construction, exploit the fact that one element represents the empty environment set, in a limited form of reduction:

$$(R_1^\sharp, \rho_1^\sharp) \cup^{\sharp \times Symb} (R_2^\sharp, \rho_2^\sharp) \stackrel{\text{def}}{=} \begin{cases} (R_1^\sharp, \rho_1^\sharp) & \text{if } R_2^\sharp = \bot^\sharp \text{ or } \exists l, \rho_2^\sharp(V_l) = \bot_{\mathcal{B}}^{Symb} \\ (R_2^\sharp, \rho_2^\sharp) & \text{if } R_1^\sharp = \bot^\sharp \text{ or } \exists l, \rho_1^\sharp(V_l) = \bot_{\mathcal{B}}^{Symb} \\ (R_1^\sharp \cup^\sharp R_2^\sharp, \ \rho_1^\sharp \cup^{Symb} \rho_2^\sharp) & \text{otherwise} \end{cases}$$

As there is no widening — resp. narrowing — on the symbolic domain, we define the widening — resp. narrowing — on the product $\mathcal{D}^\sharp \times \mathcal{D}^{Symb}$ using the symbolic union — resp. intersection:

$$(R_1^\sharp, \rho_1^\sharp) \triangledown^{\sharp\times Symb} (R_2^\sharp, \rho_2^\sharp) \stackrel{\text{def}}{=} (R_1^\sharp \triangledown^\sharp R_2^\sharp,\ \rho_1^\sharp \cup^{Symb} \rho_2^\sharp) \ .$$

Abstract Transfer Functions. Given an abstract element $(R^\sharp, \rho^\sharp)$, it represents the concrete element $\gamma^\sharp(R^\sharp) \cap \gamma^{Symb}(\rho^\sharp)$. Thanks to Thm. 6.3.1, we can safely abstract any expression in this set of concrete environments by applying any sequence of substitutions present in $\rho^\sharp$. Then, the resulting expression can be safely linearised, thanks to Thm. 6.2.3. We thus define the transfer function on $\mathcal{D}^\sharp \times \mathcal{D}^{Symb}$ by refining the $\mathcal{D}^\sharp$ component as follows:

$$\begin{aligned}
\{\!|\, V \leftarrow expr \,|\!\}^{\sharp\times Symb}(R^\sharp, \rho^\sharp) &\stackrel{\text{def}}{=} (\{\!|\, V \leftarrow (\!|\, expr' \,|\!) R^\sharp \,|\!\}^\sharp R^\sharp,\ \{\!|\, V \leftarrow expr \,|\!\}^{Symb} \rho^\sharp) \\
\{\!|\, V \rightarrow expr \,|\!\}^{\sharp\times Symb}(R^\sharp, \rho^\sharp) &\stackrel{\text{def}}{=} (\{\!|\, V \rightarrow (\!|\, expr'' \,|\!)(\{\!|\, V \leftarrow ? \,|\!\}^\sharp R^\sharp) \,|\!\}^\sharp R^\sharp,\ \{\!|\, V \leftarrow ? \,|\!\}^{Symb} \rho^\sharp) \\
\{\!|\, expr \bowtie 0 ? \,|\!\}^{\sharp\times Symb}(R^\sharp, \rho^\sharp) &\stackrel{\text{def}}{=} (\{\!|\, (\!|\, expr' \,|\!) R^\sharp \ \bowtie\ 0 ? \,|\!\}^\sharp R^\sharp,\ \rho^\sharp)
\end{aligned}$$

where $expr'$ is derived from $expr$ by arbitrary many substitutions of the form $subst\,(\cdot,\ V_i,\ \rho^\sharp(V_i))$, for any sequence of V_i's. For the backward assignment transfer function, $expr''$ is equally derived from $expr$ by substitution, but using a symbolic constant environment valid *before* the assignment, such as $\{\!|\, V \leftarrow ? \,|\!\}^{Symb} \rho^\sharp$. However, an important question remains: which substitution(s) should we perform to achieve the maximum precision ? There is unfortunately no simple answer to this question and we will provide, in Sect. 6.3.4, a few strategies with associated examples.

Relationship with the Reduced Product. The technique presented here is different from the *reduced product* presented in Sect. 2.2.6. In the reduced product, transfer functions are not changed and information transfer occurs only, by reduction, on abstract elements in-between transfer function applications while, here, we modify the transfer functions on the $\mathcal{D}^\sharp$ component to use information discovered by the symbolic domain $\mathcal{D}^{Symb}$. Note, however, that our framework also allows information transfer by reduction. For instance, if the numerical domain $\mathcal{D}^\sharp$ is able to prove that some code is unreachable, we may discard the symbolic information as well. Information flows both ways, thus, the combination of the symbolic and numeric domains is dynamic: it is more precise than an analysis pass using solely $\mathcal{D}^{Symb}$ followed by a numerical analysis in $\mathcal{D}^\sharp$ using the inferred symbolic information.

6.3.4 Substitution Strategies

Any sequence of substitutions extracted from a symbolic constant abstract environment is sound, but some sequences give more precise results than others. In particular, sometimes,

the best precision can only be achieved with no substitution at all. We now present here a few motivated strategies to help choosing which substitutions to perform.

Avoid Substituting Variable-Free Expressions. There is one case were we know that substitution is not useful: when $\mathit{occ}\,(\rho^\sharp(V_i)) = \emptyset$, which means that the symbolic expression associated to V_i is a constant one. Firstly, this information is generally already available in the numerical abstract domain $\mathcal{D}^\sharp$ which is often at least as precise as the interval domain. We now show that such propagations can be, in fact, harmful. Suppose, for instance, that $\rho^\sharp = [X \mapsto [0, 10]]$ and we wish to abstract the assignment $Y \leftarrow X - 0.5 \times X$ in the interval domain. Linearisation without substitution gives $Y \in [0, 5]$, while substitution gives $Y \leftarrow [0, 10] - 0.5 \times [0, 10]$, that is, $Y \in [-5, 10]$, which is much less precise. One solution to this problem is to prohibit substituting a variable with an expression where no variable occurs. This avoids losing correlations due to vanishing multiple occurrences of the same variable in the expression.

Avoid Substitution-Induced Precision Losses. We now propose another, more aggressive, strategy which consists in prohibiting the substitution of a variable with a non-deterministic expression, that is, an expression $\rho^\sharp(V_i)$ which may evaluate to several values in some environment: $\exists \rho \in \gamma^{\mathit{Symb}}(\rho^\sharp), |[\![\, \rho^\sharp(V_i) \,]\!]\rho| > 1$. This ensures that the substitution does not result in any loss of precision. It is sufficient to prohibit substituting variables with expressions that contain a non-singleton interval, which is a superset of all non-deterministic expressions. Both this strategy and the previous one work on the preceding example. However, their effect is difficult to compare in general.

Avoid Introducing Non-Linearity. Even if we take care to perform only substitutions that do not lose precision, performing more substitutions does not always enable more simplification in the linearisation. Consider, for instance, the following code fragment:

$$\begin{array}{ll} \textcircled{1} & Y \leftarrow U \times V\,; \\ & X \leftarrow Y + 1\,; \\ \textcircled{2} & Z \leftarrow X - Y \end{array}$$

analysed in the interval domain. Suppose, moreover, that $U, V \in [0, 10]$ at ①. At ②, the symbolic abstract environment is $\rho^\sharp = [X \mapsto Y + 1, Y \mapsto U \times V]$. If we take care to substitute only X with $Y + 1$ in $X - Y$, the subsequent linearised assignment becomes $Z \leftarrow 1$, which is optimal. If we perform all possible substitutions, we get a non-linear expression $U \times V + 1 - U \times V$ that will be linearised as $[-10, 10] \times V + 1$, if we choose to intervalise U, and $[-10, 10] \times U + 1$, if we choose V. Both will store the interval $[-99, 101]$ in Z, which is as imprecise as the plain or linearised interval abstraction without symbolic constant propagation.

A solution would be to prohibit substitutions that introduce non-linear terms as they would introduce a loss of precision in the subsequent linearisation. Unfortunately, this strategy does not always work, as demonstrated by the following example.

Example 6.3.1. Linear interpolation computation revisited.

Consider the following revisited version of the linear interpolation example of Ex. 6.2.1, to be analysed in the interval domain:

$$\begin{array}{ll} ① & X \leftarrow [0,1]; \\ ② & Y \leftarrow [0,10]; \\ ③ & Z \leftarrow [0,20]; \\ ④ & U \leftarrow X \times Y; \\ ⑤ & V \leftarrow -(X \times Z); \\ ⑥ & T \leftarrow U + V + Z \end{array}$$

The proposed strategy will pass the assignment $T \leftarrow U + V + Z$ to our linearisation procedure unmodified as U and V are non-linear. The interval for T will be $[\min Y + \min Z - \max Z, \max Y + \max Z - \min Z]$, that is, $[-20, 30]$, and we will not be able to prove that T is positive. However, if we substitute both U and V with their symbolic value at ⑥, we retrieve the linear interpolation formula $T \leftarrow X \times Y - X \times Z + Z$ of Sect. 6.2.4 that allows us, after linearisation, to prove that T is positive. The substituted expression is less linear than the original one but it has more correlations as some variables appear several times.

●

As a conclusion, we have two conflicting strategies: we can either disable substitutions to limit the non-linearity of expressions, or allow them in the hope of discovering correlations that can be exploited by a multiplication strategy during the linearisation.

Relational Domain Specificities. Finally, even linear substitutions may result in such an increase of the expression complexity that the numerical abstract domain cannot handle it precisely. Consider the following code fragment:

$$\begin{array}{ll} ① & Y \leftarrow U + V; \\ ② & X \leftarrow Y + 5 \end{array}$$

analysed in the octagon abstract domain. Substituting $U + V$ for Y in the assignment $X \leftarrow Y + 5$ gives the perfectly linear assignment $X \leftarrow U + V + 5$. Unfortunately, without any information on U and V, such an assignment results in no information on X, while the original one infers the octagonal constraint $X - Y = 5$. We could provide more complex examples where an assignment becomes exactly representable in a given domain after a specific sequence of substitutions, no more, no less.

Multiple Strategies. In general, the problem of deciding *where to stop substituting* seems inextricable. A brute force strategy would be to try the whole set of possible substitutions, which is finite but huge, and take the intersection of all results. This is costly. In our implementation embedded within the ASTRÉE static analyser, we chose to take the intersection of two special substitutions: firstly, $subst^*(expr,\ \rho^\sharp)$ which is the normal form obtained by applying all possible substitutions, and secondly, the original expression $expr$, in order to guarantee that we obtain results at least as precise as if we used linearisation without propagation. Arguably, this is a somewhat simplistic solution and we believe that there is much room for improvement, however, we will see, in Chap. 8, that the proposed technique is empirically quite successful. We now propose a full example using this strategy:

Example 6.3.2. Absolute value computation.

Consider the following code fragment that stores in Y the absolute value of X:

$X \leftarrow [-10, 20]$;
$Y \leftarrow X$;
① `if` $Y \leq 0$ `{` ② $Y \leftarrow -X$ `}`
③

We would like to infer that, at ②, $X \leq 0$, so that $Y \leftarrow -X$ assigns a positive value to Y. A way to derive this information without resorting to a relational domain is to replace, in the test $Y \leq 0$, the variable Y with X, which is possible because we have the symbolic information $\rho^\sharp = [Y \mapsto X]$ at ①. However, this is not sufficient to prove that $Y \geq 0$ at ③. We also need to infer that $Y \geq 0$ when the implicit empty `else` branch is taken, that is, when the test condition $Y \leq 0$ is false. In order to derive this information, it is crucial that we do *not* replace Y with X in the test transfer function for `not`$(Y \leq 0)$. Using the proposed strategy, involving the intersection of tests after the identity substitution and after the full substitution, the interval abstract domain will be able to prove that, at ③, $Y \in [0, 20]$.

●

6.3.5 Cost Considerations

The most expensive operation in our symbolic propagation abstract domain is the assignment transfer function which must perform n substitutions at each assignment — where n is the number of program variables. Most of the time, however, we perform substitutions $subst\,(\rho^\sharp(V_k),\ V_i,\ expr)$ in expressions $\rho^\sharp(V_k)$ where V_i does not occur, still, the expressions must be traversed fully to discover this fact. In our implementation, we chose to store, together with $\rho^\sharp$, a map $\mathcal{V} \to \mathcal{P}(\mathcal{V})$ that associates to each variable V_i the set of variables V_k such that V_i occurs in $\rho^\sharp(V_k)$. In practice, the cost of maintaining this map is largely

compensated by the fact that many useless calls to *subst* are avoided, and we achieve an effective sub-linear cost.

Another remark is that the same relatively small set of expression trees appears very frequently during an analysis. The use of *tabulation* techniques to store frequent calls to *subst* and $subst^*$ resulted in a large speed-up. As tabulation techniques require fast structural equality tests for the arguments of functions, we also implemented a *hash-consing* technique to construct the expression trees so that structurally equal trees are allocated at equal memory addresses. Hash-consing has also a good impact on memory consumption as it enforces the *sharing* of structurally equivalent sub-expressions.

Finally, in a few exceptional cases, we obtained very large expressions due to long sequences of assignments without any control flow join. Our solution was to set a bound on the depth of expressions allowed in abstract environments $\rho^\sharp$ and abstract larger expressions by $\top_{\mathcal{B}}^{Symb}$. Please note that this is just an optimisation trick and that the analysis termination is guaranteed without imposing a bound on the size of expressions: the fact that $\mathcal{D}^{Symb}$ has a finite height is sufficient. Another solution will be presented in Sect. 6.3.6. It is to be noted that such large expression trees appear very rarely as programs generally contain control flow joins, due to `if` or `while` constructs, which tend to fill abstract environments with $\top_{\mathcal{B}}^{Symb}$ values and avoid uninterrupted sequences of substitutions within the same expression.

6.3.6 Interval Linear Form Propagation Domain

A variation on our symbolic constant domain construction can be conceived by storing *linearised* expressions instead of full expression trees. Elements in $\mathcal{D}^{Symb'}$ thus associate to each variable an interval linear form. One benefit is that the size of our abstract environments is bounded by $O(n^2)$, where n is the number of program variables, independently from the program size, while expression trees could grow arbitrarily long in the symbolic propagation domain — the bound was only determined by the number of abstract transfer functions executed by the analyser, which grows with the size of the analysed program. Also, the linearisation time cost is greatly reduced: it now only depends upon the size of the argument expressions while, in the symbolic propagation domain, it depended upon the size of the expressions *after* substitution, which incurs an exponential blow-up in the worst case. Different instances of the same variable are now linearised once and for all.

Unfortunately, this can have a negative impact on precision. As the whole expanded expression tree is not available, some of our global strategies for linearising multiplications cannot be applied to their full extend. Consider, for instance, the revisited linear interpolation example of Ex. 6.3.1. In the assignment $U \leftarrow X \boldsymbol{\times} Y$, at ④, we have to choose whether to intervalise X or Y as we cannot associate to U a non-linear expression. Unfortunately, only when U is used in the assignment $T \leftarrow U \boldsymbol{+} V \boldsymbol{+} Z$ at ⑥ do we know that X should have been intervalised to be able to prove that T is positive, but it is too late!

A compromise to achieve the best of symbolic constant and interval linear form propagations is to set up a maximum bound on the size of expressions stored in the symbolic constant domain and, if this bound is reached for some expression, store the linearised expression instead to gain space at the expense of some precision loss.

6.3.7 Comparison with Relational Abstract Domains

The symbolic propagation technique performs well on the interval domain. In effect, it compensates for the lack of relationship between variables. However, our symbolic propagation is quite different from genuine relational numerical abstract domains. Unlike relational domains that store *equations*, our propagation domain stores *rewriting rules*. As tests are expressed naturally in equational form, we cannot abstract them precisely in our propagation domain. Moreover, as equations can be reversed, combined, and simplified, relational domains are able to infer new constraints and detect inconsistencies, which is impossible in a symbolic domain where the only allowed operation is the application of substitution rewriting rules. As a consequence, the symbolic propagation domain cannot discover invariants that are not syntactically present in the program — or, more precisely, composed of syntactic expression trees glued together — while relational domains can. Propagating symbolic expressions of some kind is far less precise than using a relational domain for the same kind of expressions. However, it is much less costly and its cost does not grow when considering more complex expression classes.

6.4 Conclusion

We have proposed, in this chapter, two techniques, called linearisation and symbolic constant propagation, that can be combined together to improve the precision of many numerical abstract domains. In particular, we are able to compensate for the lack of non-linear transfer functions in the polyhedron and octagon domains, and we are able to compensate for a weak or inexistent level of relationality in the octagon and interval domains. These techniques are quite fragile as they rely on program features that are more syntactic than semantic: they must be driven by strategies adapted to the analysed program. However, these techniques are very lightweight. We found out that, in many cases, it is easier and faster to design a couple of linearisation and symbolic propagation strategies to solve a local loss of precision in some program, while keeping the interval abstract domain, than to develop a robust relational abstract domain able to represent the required local property. As the linearisation and the constant propagation only slow down the underlying abstract domain by a constant factor, the solution of adding new strategies while keeping the interval domain results in an analysis that is much faster than if we added new relational domains. Practical results obtained within the Astrée static analyser confirm these statements, as

we will see in Chap. 8.

Future Work. Because the precision gain strongly depends upon the multiplication strategy used in our linearisation and the propagation strategy used in the symbolic constant domain, a natural extension of our work is to try and design new such strategies, adapted to different practical cases. We are currently pursuing this goal within the Astrée static analyser as new kinds of precision losses are uncovered. A more challenging task is to find theoretical results that guarantee the precision of some strategies. For instance, one may expect a symbolic constant propagation strategy to make a static analysis immune to a specific class of program transformations.

Chapter 7

Analysis of Machine-Integer and Floating-Point Variables

Les domaines numériques abstraits présentés précédemment font l'hypothèse que l'on travaille sur des nombres "parfaits" : entiers, rationnels, réels. En réalité, les programmes manipulent des entiers machine et des flottants qui ont une précision limitée. Dans ce chapitre, nous présentons la sémantique de ces nombres "imparfaits" et nous expliquons comment adapter les domaines numériques abstraits à leur analyse. De plus, nous présentons des méthodes pour l'implantation correcte de nos domaines abstraits dans le cas où l'analyseur utilise lui aussi des entiers machine et des flottants.

The numerical abstract domains presented so far are adapted to the analysis of "perfect" mathematical numbers: integers, rationals, reals. Real-life programs, however, use machine-integers and floating-point numbers that have only a limited precision. In this chapter, we present the semantics of these "imperfect" numbers and explain how to adapt numerical abstract domains in order to analyse them. We also explain how to implement soundly our abstract domains in an analyser that uses machine-integers and floating-point numbers.

7.1 Introduction

The Simple programming language presented in Chap. 2 manipulates perfect mathematical numbers living in $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$. This model was chosen for its simplicity. Real-life programming languages, however, generally manipulate limited, finite representations of numbers with imperfect operators that do not behave as the mathematical ones when the computa-

tion escapes the scope of the chosen representation. For instance, integer numbers are often represented using a fixed number of bits and the operators "wrap around" when there is an overflow: the correct mathematical result is thus computed only up to a modulo. When it comes to representing real or rational numbers, the limited number of bits available incurs some rounding and the computed result is correct only up to a small rounding error ϵ. Finally, some operations are simply invalid and may halt the computer.

We also supposed, in the preceding chapters, that we could compute exactly all abstract transfer functions using perfect numbers while, in practice, our analyser may be implemented using machine-integers and floating-point numbers as well — possibly with a different bit-size or semantics than that of the analysed program. Thus, there are four possibilities depending on whether the concrete semantics uses perfect or imperfect numbers and whether the abstract implementation uses perfect or imperfect numbers. Up to now, we only know how to abstract perfect concrete numbers using perfect numbers in the abstract. Applying directly our formulas in any of the three other cases would give in *unsound* results. Real-life machine-integer and floating-point computations are more difficult to abstract than perfect numbers in $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, mainly because the corresponding operators lack all the classical mathematical properties assumed when designing numerical abstract domains. Designing algorithms for computing sound transfer functions using machine-integers and floating-point numbers is equally tricky. Implementing, with imperfect numbers, abstract transfer functions that are sound with respect to the semantics of imperfect concrete numbers is even more difficult as both soundness problems tend to cumulate and not cancel each other.

Previous Works on Floating-Point Analysis. Much work is dedicated to the analysis of the *precision* of floating-point computations, that is, determining the maximal difference between a perfect computation on reals and its actual floating-point implementation, as well as the origins of this imprecision. For instance, the CESTACT method, described in [Vig96], is widely used but also much debated as it is based on probabilistic models of rounding error distributions and cannot give sound answers. Sound analyses based on abstract interpretation include a method, described in [AABB$^+$03], that uses the interval domain to bound error terms, as well as a much more advanced method, proposed by Goubault and Martel in [Gou01, Mar02b], able to relate error terms within relational, possibly non-linear, abstract domains.

In a sense our problem is simpler because we are only interested in determining the computed floating-point values, not the drift between a perfect and an imperfect model of computation. However, it seems that much less work is devoted to our problem. Sound interval abstraction of floating-point expressions is an integral part of interval arithmetics [Moo66] and can be adapted straightforwardly to obtain a full interval abstract domain — see, for instance, [GGP$^+$01] — however, there does not seem to be any work concerning

relational abstract domains for floating-point variables. The work of Goubault and Martel is not well-suited for our purpose as only synthetic error terms are related, not program variables, and tests cannot be abstracted precisely.

Our Contribution. The novelty of our approach is to slice the problem into *two* parts. Firstly, we express and abstract the *concrete* machine-integer and floating-point semantics using perfect mathematical integers and reals. For machine-integers, it is sufficient to perform a conservative *overflow detection* using an instrumented integer arithmetics to know whether the expression behaves as in $\mathbb{Z}$; if it does, we can apply our classical transfer functions; if it does not, we fall back to a sound but maybe coarse non-relational abstraction. For floating-point numbers, we rely on an adaptation of the *linearisation technique*, proposed in the previous chapter, extended to take into account the rounding introduced by each operator. As abstracted expressions are expressed using interval linear forms on real numbers $\mathbb{R}$, we can safely feed them to the interval, zone, and octagon domains presented in the preceding chapters. Secondly, we explain how to use machine-integers and floating-point numbers *in the abstract* to improve the efficiency. Combining these two steps results in abstract domains using floating-point numbers to abstract floating-point numbers — or machine-integers to abstract machine-integers. The intermediate abstract semantics can be seen as a conceptual tool allowing an easy design and much simplified soundness proofs; it is not meant to be implemented. An important feature of this technique is that it works equally well on non-relational and relational numerical abstract domains.

The first two sections present these two successive steps for the semantics of machine-integers while the following two sections focus on floating-point numbers.

7.2 Modeling Machine-Integers

Real-life programming languages do not generally manipulate unbounded mathematical integers in $\mathbb{Z}$ but use, instead, several integer *types* which differ in the *finite* range that they can represent. Unsigned integers of bit-size b, where b is generally 8, 16, 32, or 64, allows representing all integers in the range $[0, 2^b - 1]$. There are several conventions to represent signed integers; the most widespread is *two's complement* representation that can represent numbers in the range $[-2^{b-1}, 2^{b-1} - 1]$ with a bit-size of b, but the C programming language norm also permits the use of *one's complement* and *unsigned with a sign bit* representations.

These integer types share a common semantic property: all arithmetic operators behave as the mathematical ones in $\mathbb{Z}$ as long as the exact result fits in the representable range. If it does not, we have an *overflow*. In case of an overflow, the result is implementation-specific and has the ability to halt the computer with a run-time error; however, in most cases, it simply computes the only value in the representable range $[m, M]$ equal to the result modulo $M - m$.

$$
\begin{array}{lcll}
expr_{\mathbf{i}} & ::= & X & X \in \mathcal{V}_{\mathbf{i}} \\
& | & [a,b]_{\mathbf{i}} & a,b \in \mathbb{Z} \\
& | & -_{\mathbf{i}}\ expr_{\mathbf{i}} & \\
& | & expr_{\mathbf{i}}\ \diamond_{\mathbf{i}}\ expr_{\mathbf{i}} & \diamond \in \{+,-,\times,/\} \\
& | & \mathtt{cast}_{\mathbf{i}}\,(expr_{\mathbf{i}'}) & \\
\\
test & ::= & expr_{\mathbf{i}} \bowtie expr_{\mathbf{i}} & \bowtie \in \{=,\neq,<,\leq\} \\
& | & \cdots & \\
\\
inst & ::= & X \leftarrow expr_{\mathbf{i}} & X \in \mathcal{V}_{\mathbf{i}} \\
& | & \cdots &
\end{array}
$$

$\mathbf{i} \in \mathbf{I}$ is an integer format.
Each $\mathcal{V}_{\mathbf{i}}$, $\mathbf{i} \in \mathbf{I}$, is a finite set of variables.

Figure 7.1: Simple language syntax adapted to machine-integers.

7.2.1 Modified Syntax and Concrete Semantics

Adapted Syntax. We propose to modify our Simple language so that it manipulates machine-integers instead of numbers in $\mathbb{Z}$. We first denote by $\mathbf{I}$ the finite set of integer types. Each integer type $\mathbf{i} \in \mathbf{I}$ is characterised solely by the lower bound $m_{\mathbf{i}}$ and the upper bound $M_{\mathbf{i}}$ of the range of integers it can represent. The modifications to our Simple language syntax are shown in Fig. 7.1: each expression, operator, and variable now has a type $\mathbf{i} \in \mathbf{I}$ and the syntax imposes simple typing rules. We also add a `cast` operator to allow converting from one integer type to another.

Adapted Concrete Semantics. The adapted concrete semantics of numerical expressions $[\![\cdot]\!]_{mi}$, presented in Fig. 7.2, differs from the original semantics of Fig. 2.2 on two points. Firstly, an environment now associates to each variable $V \in \mathcal{V}_{\mathbf{i}}$ of type $\mathbf{i} \in \mathbf{I}$ only machine-integers that are valid for the type $\mathbf{i}$, *i.e.*, within $[m_{\mathbf{i}}, M_{\mathbf{i}}]$. Secondly, we apply the $\mathtt{check}_{\mathbf{i}}$ function after each operator to put the computed set of integers back into $[m_{\mathbf{i}}, M_{\mathbf{i}}]$. In order to account for *all* possible implementations of machine-integer arithmetics, our semantics considers that the result of a computation that does not fit in the proper interval range may be *anything* in that range or a run-time error. Consistently with the original Simple language semantics, run-time errors due to divisions by zero or overflows silently halt the program; they are not reported by the static analysis which outputs invariants that are true for error-free computation traces only. Note that the semantics of the division

$$[\![\, expr_{\mathbf{i}} \,]\!]_{mi} : (\textstyle\prod_{\mathbf{k}\in\mathbf{I}} \mathcal{V}_{\mathbf{k}} \to [m_{\mathbf{k}}, M_{\mathbf{k}}]) \to \mathcal{P}([m_{\mathbf{i}}, M_{\mathbf{i}}])$$

$$\begin{array}{lcl}
[\![\, X \,]\!]_{mi}(\rho) & \stackrel{\text{def}}{=} & \{\ \rho(X)\ \} \\
[\![\, [a,b]_{\mathbf{i}} \,]\!]_{mi}(\rho) & \stackrel{\text{def}}{=} & \mathtt{check}_{\mathbf{i}}\,(\{\ x \in \mathbb{Z} \mid a \leq x \leq b\ \}) \\
[\![\, \mathtt{cast}_{\mathbf{i}}\,(e_{\mathbf{i}'}) \,]\!]_{mi}(\rho) & \stackrel{\text{def}}{=} & \mathtt{check}_{\mathbf{i}}\,([\![\, e_{\mathbf{i}'} \,]\!]_{mi}(\rho)) \\
[\![\, -_{\mathbf{i}}\, e_{\mathbf{i}} \,]\!]_{mi}(\rho) & \stackrel{\text{def}}{=} & \mathtt{check}_{\mathbf{i}}\,(\{\ -x \mid x \in [\![\, e_{\mathbf{i}} \,]\!]_{mi}(\rho)\ \}) \\
[\![\, e_{\mathbf{i}} \diamond_{\mathbf{i}} e'_{\mathbf{i}} \,]\!]_{mi}(\rho) & \stackrel{\text{def}}{=} & \mathtt{check}_{\mathbf{i}}\,(\{\ x \diamond y \mid x \in [\![\, e_{\mathbf{i}} \,]\!]_{mi}(\rho),\ y \in [\![\, e'_{\mathbf{i}} \,]\!]_{mi}(\rho)\ \}) \\
 & & \diamond \in \{+, -, \times\} \\
[\![\, e_{\mathbf{i}} \,/_{\mathbf{i}}\, e'_{\mathbf{i}} \,]\!]_{mi}(\rho) & \stackrel{\text{def}}{=} & \mathtt{check}_{\mathbf{i}}\,(\{\ adj(x/y) \mid x \in [\![\, e_{\mathbf{i}} \,]\!]_{mi}(\rho),\ y \in [\![\, e'_{\mathbf{i}} \,]\!]_{mi}(\rho),\ y \neq 0\ \})
\end{array}$$

were $\mathtt{check}_{\mathbf{i}} : \mathcal{P}(\mathbb{Z}) \to \mathcal{P}([m_{\mathbf{i}}, M_{\mathbf{i}}])$ is defined as follows:

$$\mathtt{check}_{\mathbf{i}}\,(S) \stackrel{\text{def}}{=} \begin{cases} S & \text{if } S \subseteq [m_{\mathbf{i}}, M_{\mathbf{i}}] \\ [m_{\mathbf{i}}, M_{\mathbf{i}}] & \text{otherwise} \end{cases}$$

Figure 7.2: Concrete semantics of numerical expressions adapted to machine-integers.

/ involves truncation using the *adj* operator as in the original Simple language semantics of Fig. 2.2. The semantics of boolean expressions as well as all our concrete transfer functions are almost unchanged and not presented here: they simply use our new semantics for numerical expressions $[\![\cdot]\!]_{mi}$ instead of $[\![\cdot]\!]$.

Scope of the Semantics. Our semantics is adapted to programs that do not use precise knowledge of the behavior of overflowing integers: even though it is perfectly sound in the event of an overflow, each overflow results in a precision loss. However, as each single analysis will be able to discover the numerical properties that are valid in a wide range of machine-integer implementations at once, this semantics is very well-suited to the efficient analysis of portable programs. In order to analyse programs that perform overflows on purpose, one would need to refine our semantics by using a specific overflow policy, such as computation modulo $M-m$. For the sake of conciseness, we will not discuss these alternate semantics nor their abstractions in this thesis.

7.2.2 Adapted Abstract Semantics

We now present a simple way to adapt numerical abstract domains designed for $\mathbb{Z}$ into domains abstracting machine-integers.

Interval Domain. We first adapt the interval abstract domain by defining a new abstract semantics for numerical expressions $[\![\cdot]\!]^{Int}_{mi}$, as shown in Fig. 7.3: we reuse the original interval arithmetics operators $\diamond^{Int}$ presented in Sect. 2.4.6 and feed their output to the abstract counterpart $\texttt{check}^{Int}_{\mathbf{i}}$ of $\texttt{check}_{\mathbf{i}}$. As both interval bounds for a variable or expression of type $\mathbf{i}$ are always within $[m_{\mathbf{i}}, M_{\mathbf{i}}]$, we no longer require the special elements $+\infty$ and $-\infty$. Note also that $[\![\cdot]\!]^{Int}_{mi}$ not only outputs an interval abstract element that over-approximates the possible values of the expression, but also a flag in $\mathbb{B} \stackrel{\text{def}}{=} \{\mathbf{T}, \mathbf{F}\}$ — where $\mathbf{T}$ denotes "true" and $\mathbf{F}$ denotes "false" — telling whether or not an overflow could have occurred somewhere during the expression evaluation. When no overflow occurs, we can ignore the `check` functions, and so, the machine-integer semantics $[\![\cdot]\!]_{mi}$ is equal to the perfect integer semantics $[\![\cdot]\!]$. This is formalised as follows:

Theorem 7.2.1. Soundness of adapted intervals.

Suppose that $[\![\mathit{expr}]\!]^{Int}_{mi}(R^\sharp) = (X^\sharp, O)$ *and* $\rho \in \gamma^{Int}(R^\sharp)$*, then:*

- $[\![\mathit{expr}]\!]_{mi}(\rho) \subseteq \gamma^{Int}_{\mathcal{B}}(X^\sharp)$.
- *If* $O = \mathbf{F}$*, then no overflow occurs in* $[\![\mathit{expr}]\!]_{mi}(\rho)$ *and* $[\![\mathit{expr}]\!]_{mi}(\rho) = [\![\mathit{expr}]\!](\rho)$.

●

Thanks to the first point of Thm. 7.2.1, we can easily derive an interval abstract assignment transfer function from $[\![\mathit{expr}]\!]^{Int}_{mi}$, as in Sect. 2.4.4:

$$\{\!\!\{ V \leftarrow \mathit{expr} \}\!\!\}^{Int}_{mi}(X^\sharp) \stackrel{\text{def}}{=} X^\sharp\,[\,V \mapsto \mathit{fst}([\![\mathit{expr}]\!]^{Int}_{mi}(X^\sharp))\,]$$

where the *fst* function simply returns the first element of a pair. The overflow information is not used here but will be important in the generic adaptation of abstract domains. For the sake of conciseness, we do not present here the backward assignment and test transfer functions which resemble the generic non-relational ones. The union and intersection operators are defined as in the integer interval domain $\mathcal{D}^{Int}$. As the machine-integer interval domain has no infinite increasing nor decreasing chain, no widening or narrowing is required, in theory. However, the lattice height is quite huge, so, we strongly recommend using extrapolation operators to achieve fast analyses. Any widening and narrowing in $\mathcal{D}^{Int}$ — such as $\nabla^{Int}_{\mathcal{B}}$ and $\triangle^{Int}_{\mathcal{B}}$ defined in Sect. 2.4.6 — will do if we take care to replace $+\infty$ with $M_{\mathbf{i}}$ and $-\infty$ with $m_{\mathbf{i}}$.

Generic Domains. In order to adapt a generic numerical abstract domain $\mathcal{D}^\sharp$ to machine-integers, we use the second point of Thm. 7.2.1: in the absence of overflow, the semantics of a machine-integer expression is the same as if it was evaluated using the regular integer arithmetics for which $\mathcal{D}^\sharp$ was designed. In the event of an overflow, we fall back to an interval-based abstraction, using the sound interval returned by $[\![\mathit{expr}]\!]^{Int}_{mi}$. We now

suppose that we have a *conversion* operator Int from $\mathcal{D}^\sharp$ to $\mathcal{D}^{Int}$ and define our modified transfer functions on $\mathcal{D}^\sharp$ as follows:

Definition 7.2.1. Generic machine-integer abstract transfer functions.

Let us denote by $([a,b],O)$ *the value of* $[\![\, expr \,]\!]^{Int}_{mi}(Int(R^\sharp))$*, and by* $([a',b'],O')$ *the value of* $[\![\, expr \,]\!]^{Int}_{mi}(Int(\{\!|\, V \leftarrow ?\, |\!\}^\sharp R^\sharp))$*. We can define the following machine-integer transfer functions:*

$$\{\!|\, V \leftarrow expr \,|\!\}^\sharp_{mi}(R^\sharp) \stackrel{\text{def}}{=} \begin{cases} \{\!|\, V \leftarrow expr \,|\!\}^\sharp R^\sharp & \textit{if } O = \mathbf{F} \\ \{\!|\, V \leftarrow [a,b] \,|\!\}^\sharp R^\sharp & \textit{otherwise} \end{cases}$$

$$\{\!|\, V \rightarrow expr \,|\!\}^\sharp_{mi}(R^\sharp) \stackrel{\text{def}}{=} \begin{cases} \{\!|\, V \rightarrow expr \,|\!\}^\sharp R^\sharp & \textit{if } O' = \mathbf{F} \\ \{\!|\, V \rightarrow [a',b'] \,|\!\}^\sharp R^\sharp & \textit{otherwise} \end{cases}$$

$$\{\!|\, expr \bowtie 0\ ?\, |\!\}^\sharp_{mi}(R^\sharp) \stackrel{\text{def}}{=} \begin{cases} \{\!|\, expr \bowtie 0\ ?\, |\!\}^\sharp R^\sharp & \textit{if } O = \mathbf{F} \\ \{\!|\, [a,b] \bowtie 0\ ?\, |\!\}^\sharp R^\sharp & \textit{otherwise} \end{cases}$$

●

Note that, when it comes to backward assignments, we need to test for overflows when evaluating the expression in environments *before* the assignments, which are over-approximated here by $\{\!|\, V \leftarrow ?\, |\!\}^\sharp R^\sharp$. This technique can be applied to *all* abstract domains, even relational ones, such as, for instance, the octagon domain presented in Chap. 4. Also, the linearisation and constant propagation techniques of Chap. 6 can be applied to abstract the expression *expr* whenever there is no overflow.

Application to the Detection of Run-Time Errors. Even though errors do not appear explicitly in our semantics, we can still use the derived static analysis to detect run-time errors as follows:

- In a first step, we perform our static analysis to infer the maximal range of each variable $v \in \mathcal{V}$ at each program point $l \in \mathcal{L}$ as an abstract interval environment $X^\sharp : \mathcal{L} \to (\mathcal{V} \to \mathcal{D}^{Int})$.
- In a second step, for each operator at each program point l, we check the following pre-conditions that imply the absence of overflow and division by zero:
  - for each sub-expression $e_{\mathbf{i}} \diamond_{\mathbf{i}} e'_{\mathbf{i}}$, we check that:
$$fst([\![\, e_{\mathbf{i}} \,]\!]^{Int}_{mi}(X^\sharp)(l)) \diamond^{Int} fst([\![\, e'_{\mathbf{i}} \,]\!]^{Int}_{mi}(X^\sharp)(l)) \subseteq [m_{\mathbf{i}}, M_{\mathbf{i}}]$$
  - for each sub-expression $-_{\mathbf{i}}\, e_{\mathbf{i}}$, we check that:
$$-^{Int}(fst([\![\, e_{\mathbf{i}} \,]\!]^{Int}_{mi}(X^\sharp)(l))) \subseteq [m_{\mathbf{i}}, M_{\mathbf{i}}]$$

- for each sub-expression $\mathtt{cast}_{\mathbf{i}}\,(e_{\mathbf{i}'})$, we check that:

$$fst([\![\, e_{\mathbf{i}'} \,]\!]^{Int}_{mi}(X^\sharp)(l)) \subseteq [m_{\mathbf{i}}, M_{\mathbf{i}}]$$

- for each sub-expression $e_{\mathbf{i}} \,/_{\mathbf{i}}\, e'_{\mathbf{i}}$, we check that $0 \notin fst([\![\, e'_{\mathbf{i}} \,]\!]^{Int}_{mi}(X^\sharp)(l))$.

The fact that only interval information are used in our second step must not mislead us into thinking that an interval analysis is always sufficient: Exs. 2.5.2, 2.5.3, 2.5.4, 4.6.2, and 4.6.3, as well as Sect. 2.5, give examples where a relational domain is needed to infer good variable bounds. Note also that there was no need to enrich the concrete and abstract semantics with a special "error" state or "undefined" value. Such enrichment would free us from performing a second step but would result in more complex abstract transfer functions. Moreover, run-time errors should not be propagated because we are only interested in the location of errors, and not in the code unreachable after an error occurred.

7.2.3 Analysis Example

As an illustration, we restate the simple loop analysis of Ex. 3.7.1 in the zone abstract domain, but using machine-integers of type **i** corresponding to the range $[0, 255]$ instead of perfect integers.

Example 7.2.1. Machine-integer loop analysis.

Consider the following Simple program that iterates from 0 to N:

```
    X ← 0;
    N ← [0, 255];
    while ❶ X < N {
②     X ← X +_i 1
③   }
④
```

We get the following iteration sequence:

iteration i	label l	zone X^i_l
0	❶	$X = 0 \;\wedge\; 0 \leq N \leq 255 \;\wedge\; -255 \leq X - N \leq 0$
1	②	$X = 0 \;\wedge\; 1 \leq N \leq 255 \;\wedge\; -255 \leq X - N \leq -1$
2	③	$X = 1 \;\wedge\; 1 \leq N \leq 255 \;\wedge\; -254 \leq X - N \leq 0$
3	❶ $\triangledown$	$0 \leq X \;\wedge\; 0 \leq N \leq 255 \;\wedge\; -255 \leq X - N \leq 0$
4	②	$0 \leq X \leq 254 \;\wedge\; 1 \leq N \leq 255 \;\wedge\; -255 \leq X - N \leq -1$
5	③	$1 \leq X \leq 255 \;\wedge\; 1 \leq N \leq 255 \;\wedge\; -254 \leq X - N \leq 0$
6	❶ $\triangledown$	$0 \leq X \;\wedge\; 0 \leq N \leq 255 \;\wedge\; -255 \leq X - N \leq 0$
7	④	$0 \leq X \leq 255 \;\wedge\; 0 \leq N \leq 255 \;\wedge\; X - N = 0$

Note that, at iteration 3, the first widening application destroys the unstable upper bound for X — recall that widened iterates must not be closed — but a bound is restored by closure at iteration 4, just after the loop condition. Likewise, an upper bound on X is restored at iteration 7 just after the loop exit condition. So, not only are we able to prove that $X = N$ after the loop has finished, but also that the incrementation at line ② cannot result in an overflow as we always have $X \leq 254$ at this point. The first property cannot be inferred by the interval abstract domain at all, while the second can, but only if we take care to use either a narrowing or a widening with 255 as a threshold.
If we now suppose that X has type $\mathbf{i}'$ corresponding to the range $[-128, 127]$ and that line ② is replaced with $X \leftarrow X +_{\mathbf{i}'} 1$, then the run-time error checking embedded in the analyse will warn us that it is not able to prove that no overflow can occur during the evaluation of the $+_{\mathbf{i}'}$ operator. Effectively, an overflow *will* occur and we have discovered an actual programming error.

●

7.3 Using Machine-Integers in the Abstract

Even though we are now able to abstract machine-integers, all our semantic functions use mathematical integers in $\mathbb{Z}$ which means that a practical analyser implementation must use some kind of arbitrary precision integer computation package. As these are generally quite costly, we may want to prefer an implementation based purely on machine-integers.

7.3.1 Using Regular Arithmetics

Sometimes, it can be proved that the abstract computation never overflow the chosen analyser's machine-integer range; in this case, it is perfectly safe to use such machine-integers in place of $\mathbb{Z}$. For instance, in the adapted interval domain of the previous section, lower and upper bounds never exceed the minimum $m \stackrel{\text{def}}{=} \min_{\mathbf{i} \in \mathbf{I}} m_{\mathbf{i}}$ and maximum $M \stackrel{\text{def}}{=} \max_{\mathbf{i} \in \mathbf{I}} M_{\mathbf{i}}$ of all analysed integer types, so, one can use, if it exists on the analyser's platform, any integer type $\mathbf{i}$ such that $[m_{\mathbf{i}}, M_{\mathbf{i}}] \supseteq [m, M]$ to *represent* abstract intervals. In order to *implement* abstract transfer functions as described in Def. 7.3, however, we need to represent some extra integers as we can overflow $[m, M]$ locally, between a regular interval abstract operator $\diamond^{Int}$ and the subsequent $\texttt{check}_{\mathbf{i}}$ application that puts the result back within $[m, M]$. Because we compute the opposite and the product of numbers in $[m, M]$, we must be able to represent all numbers in $[-\max(|m|, |M|)^2, \max(|m|, |M|)^2]$. Thus, a $n-$bit architecture must be analysed using arithmetics on $2n + 1$ bits. Efficient implementation on most processors is possible provided that we use assembly language. This is because

high-level programming languages tend to hide the access to a $n+1-$th bit — so-called *carry* bit — as well as the natural ability of processors to "chain" $n-$bit instructions to easily perform arithmetic operations on $k \cdot n-$bit numbers.

7.3.2 Using Saturated Arithmetics

The maximum range of the constants involved in relational numerical domains is much more difficult to determine *a priori* and, sometimes, there is no upper bound at all. It can also happen that this maximum bound is too large to be practicable — such as when analysing a 64-bit program using the interval domain on a 32-bit computer. In all these cases, we may wish to trade precision for efficiency.

Contrary to the preceding approach, we start from two bounds, m and M, corresponding to the desired integer range on the *analyser's* platform and show how to modify our numerical abstract domains so that they only require manipulating numbers within $[m, M]$, even when abstracting much larger concrete integer ranges. We will need to give a special semantics to m and M: M used as an upper bound will denote the greatest abstract upper-bound considered — it may be $M_{\mathbf{i}}$ or $+\infty$, depending on the context — while m used as a lower bound will denote the smallest abstract lower-bound.

Interval Domain. Let us consider an interval with bounds in $[m, M]$ that abstracts a set of values, in $\mathcal{P}([m_{\mathbf{i}}, M_{\mathbf{i}}])$, of an expression of type $\mathbf{i} \in \mathbf{I}$. Whenever, $M < M_{\mathbf{i}}$, a right bound of M actually denotes an upper bound equal to $M_{\mathbf{i}}$, and likewise for a left bound equal to m:

$$\gamma^{Int}_{s,\mathbf{i}}([a,b]) \stackrel{\text{def}}{=} \begin{cases} [a,b] & \text{if } m < a,\ b < M \\ [m_{\mathbf{i}},b] & \text{if } m = a,\ b < M \\ [a,M_{\mathbf{i}}] & \text{if } m < a,\ b = M \\ [m_{\mathbf{i}},M_{\mathbf{i}}] & \text{if } m = a,\ b = M \end{cases}$$

We do not present the adapted abstract transfer operators on the interval domain as this would be quite long and boring due to the many cases one needs to consider. We only give a few — hopefully sufficient — guidelines to designing them:

- Firstly, the result of each bound computation should be *clamped* to $[m, M]$.
- Special care must be taken for additions $+_s$ and subtractions $-_s$ whenever some argument bounds are M, or m. When computing upper bounds, we should have $\forall x,\ M +_s x = x +_s M = M$, including when x equals m. Dually, lower bounds should *stick* to m.
- Special care must be taken with equality and disequality tests as intervals such as $[M, M]$ and $[m, m]$ may actually represent non-singleton sets.

The effect is that we can no longer distinguish interval upper bounds within $[M, M_{\mathbf{i}}]$, as well as lower bounds within $[m_{\mathbf{i}}, m]$. They are conservatively approximated as, respectively, $M_{\mathbf{i}}$ and $m_{\mathbf{i}}$, which results in some loss of precision.

Zone and Octagon Domains. For the potential set, zone, and octagon domains, we consider now DBMs where all coefficients are within $[m, M]$. As matrix coefficients always represent upper bounds of constraints, M will correspond to a $+\infty$ upper bound, while m will correspond to an upper bound equal to m. The concretisation of potential sets, Def. 3.2.1, is modified as follows:

$$\gamma_s^{Pot}(\mathbf{m}) \stackrel{\text{def}}{=} \{ (v_1, \ldots, v_n) \in \mathbb{I}^n \mid \forall i, j,\ \mathbf{m}_{ij} = M \text{ or } v_j - v_i \leq \mathbf{m}_{ij} \} \ .$$

The adapted γ^{Zone} and γ^{Oct} are derived from our new γ^{Pot} as in Defs. 3.2.2 and 4.2.2.

As all our algorithms on potential sets, zones, and octagons are defined using solely the $\leq$, $+$, and divide-by-two mathematical operators, it is sufficient to adapt these three operators to our new semantics of DBMs to construct our adapted domains. The $\leq$ operator will be unchanged, while $+$ and $/2$ are adapted as follows:

$$a +_s b \stackrel{\text{def}}{=} \begin{cases} M & \text{if } a = M \text{ or } b = M \text{ or } a + b \geq M \\ m & \text{if } a + b \leq m \\ a + b & \text{otherwise} \end{cases}$$

$$a /_s 2 \stackrel{\text{def}}{=} \begin{cases} M & \text{if } a = M \\ m & \text{if } a/2 \leq m \\ a/2 & \text{otherwise} \end{cases}$$

The loss of precision with respect to DBMs in $\mathbb{Z} \cup \{+\infty\}$ comes from the fact that all constraints of the form $expr \leq c$, when $c \geq M$, are forgotten while all constraints of the form $expr \leq c$, when $c < m$, are replaced with $expr \leq m$.

This technique was implemented in our freely available octagon abstract domain library [Mina] using native integers, and compared to a perfect implementation in $\mathbb{Z}$ based on the `GMP` multi-precision integer library [GMP]. Our experience shows that the speed gain is consequence while, in most cases, the matrix coefficients do not exceed twice the maximal bounds of all variables, which is often much smaller than M, and so, the precision loss is negligible.

Linearisation. We can safely implement the linearisation procedure of Chap. 6 using the same principles. The interval semantics used within the linearisation are similar to the one used in the interval abstract domain, except that we must be able to represent unbounded intervals, so, an upper bound M will actually mean that the interval has no upper bound — instead of an upper bound equal to $M_{\mathbf{i}}$ — while a lower bound m denotes an interval that

has no lower bound. The concretisation of such an interval is similar to $\gamma^{Int}_{s,\mathbf{i}}$, except that $M_\mathbf{i}$ and $m_\mathbf{i}$ bounds are respectively replaced with $+\infty$ and $-\infty$. Actually, such a semantics for intervals allows abstracting arbitrary, bounded or unbounded, intervals of integers, and so, can be used to perform an interval analysis of our original Simple language with perfect integers.

In all these three examples, the classical integer arithmetic operators have be altered so that the bounds m and M are more or less "sticky", hence the generic name "saturated arithmetics". It is important to note that the exact definition of each operator is not generic but varies from one abstract domain to the other. In particular, such definitions are quite straightforward for the zone, octagon, and interval domains, even if we perform linearisation, but would require a lot more work to keep the soundness in the case of the polyhedron domain. Moreover, as the values used as polyhedron coefficients tend to grow very quickly even when the variables have small bounds, limiting all the coefficients within a small range is likely to result in much more precision degradation than for the integer, zone, and octagon domains.

Saturated Arithmetics Implementation. We are now left with the problem of choosing the bounds m and M such that, not only all numbers in $[m, M]$ are representable using the analyser's machine-integers, but also the modified operators $+_s$, $-_s$, $/_s 2$ are easy to implement. Such an implementation requires two basic kinds of operations: testing whether a value is m or M, and computing the result of a mathematical operator $+$, $-$, $/$ clamped to $[m, M]$. Testing for m or M is not a problem. Following the discussion in Sect. 7.3.1, we can tell that clamped operators can be implemented as long as $[-\max(|m|, |M|)^2, \max(|m|, |M|)^2]$ fits within the range of some machine-integer type. For instance, on a 32−bit architecture, we can easily implement clamped operators on 15−bit integers, while 16−bit or even 32−bit integers may be reachable only using assembly language tricks. Finally, it is worth remarking that modern processors enjoy so-called *multimedia instruction sets* — such as MMX and SSE on Intel, and AltiVec on PowerPC processors — that directly implement clamped arithmetics and might be used to greatly enhance the speed of our saturated arithmetics.

7.4 Modeling IEEE 754 Floating-Point Numbers

We now apply the same methodology as in the first part of the chapter, but focus on floating-point numbers instead of machine-integers. The floating-point representation is a convenient way to compactly represent a wide range of numbers. It is widely used in scientific applications. Moreover, many applications that used to prefer fixed-point arithmetics — such as embedded or multimedia applications — now turn to floating-point arithmetics.

We focus here on the floating-point number representation and operators described in the IEEE 754-1985 norm [CS85] which has become, since quite a few years, a standard implemented in hardware — such as INTEL and POWERPC processors — and supported by most programming languages — such as the C programming language. Nevertheless, most of the ideas presented here may be adapted to more exotic floating-point formats used in legacy processors dating from before the IEEE norm — such as Cray — or modern Digital Signal Processors — that implement only a limited subset of the norm to save transistors.

7.4.1 IEEE 754 Representation

The binary representation of a IEEE 754 number is composed of three fields:

- a 1-bit sign s;
- an exponent $e - \mathbf{bias}$, represented by a biased $\mathbf{e}$-bit unsigned integer e;
- a fraction $f = .b_1 \dots b_{\mathbf{p}}$, represented by a $\mathbf{p}$-bit unsigned integer.

The values $\mathbf{e}$, $\mathbf{bias}$, and $\mathbf{p}$ are format-specific. We will denote by $\mathbf{F}$ the set of all available formats. Existing formats $\mathbf{f} \in \mathbf{F}$ include:

- $\mathbf{f} = \mathbf{32}$, the 32-bit *single precision* format required by the norm: $\mathbf{e} = 8$, $\mathbf{bias} = 127$, and $\mathbf{p} = 23$;
- $\mathbf{f} = \mathbf{64}$, the 64-bit *double precision* format required by the norm: $\mathbf{e} = 11$, $\mathbf{bias} = 1023$, and $\mathbf{p} = 52$;
- $\mathbf{f} = \mathbf{80}$, the 80-bit *long double* format on INTEL processors: $\mathbf{e} = 15$, $\mathbf{bias} = 16383$, and $\mathbf{p} = 63$;[1]
- $\mathbf{f} = \mathbf{128}$, the 128-bit *quadruple precision* format on POWERPC processors: $\mathbf{e} = 15$, $\mathbf{bias} = 16383$, and $\mathbf{p} = 112$;
- $\mathbf{f} = \mathbf{16}$, the 16-bit *half precision* format supported by high-end accelerated graphics cards by nVidia: $\mathbf{e} = 5$, $\mathbf{bias} = 15$, and $\mathbf{p} = 10$.

Each floating-point number belongs to one of the following categories:

- *normalised* numbers $(-1)^s \times 2^{e-\mathbf{bias}} \times 1.f$, when $1 \leq e \leq 2^{\mathbf{e}} - 2$;

[1] Note that, unlike the other presented formats, $\mathbf{e} + \mathbf{p} + 1$ is not the bit-size of the representation, but the bit-size minus one. Indeed, INTEL's long double format explicitly represents the high-order bit b_0 of the mantissa $b_0.b_1 \dots b_p$ while other formats exploit the "hidden bit" feature of the norm to represent only the fraction $.b_1 \dots b_p$ and recycle the b_0 bit.

- *denormalised* numbers $(-1)^s \times 2^{1-\mathbf{bias}} \times 0.f$, when $e = 0$ and $f \neq 0$;
- $+0$ or -0 (depending on s), when $e = 0$ and $f = 0$;
- $+\infty$ or $-\infty$ (depending on s), when $e = 2^{\mathbf{e}} - 1$ and $f = 0$;
- error codes (so-called *NaN*), when $e = 2^{\mathbf{e}} - 1$ and $f \neq 0$.

For each format $\mathbf{f} \in \mathbf{F}$, we define in particular:

- $mf_{\mathbf{f}} \stackrel{\text{def}}{=} 2^{1-\mathbf{bias}-\mathbf{p}}$, the smallest non-zero positive number;
- $Mf_{\mathbf{f}} \stackrel{\text{def}}{=} (2 - 2^{-\mathbf{p}})2^{2^{\mathbf{e}}-\mathbf{bias}-2}$, the largest non-infinity number.

Fig. 7.4 presents the positive, non-*NAN*, floating-point numbers in a graphical way.

The special values $+\infty$ and $-\infty$ may be generated as a result of operations undefined on $\mathbb{R}$ (such as $1/+0$) or when a result overflows $Mf_{\mathbf{f}}$ in absolute value. Other undefined operations (such as $+0/+0$) result in a *NaN* (that stands for *Not A Number*). The sign of ± 0 serves only to distinguish $1/+0 = +\infty$ and $1/-0 = -\infty$; $+0$ and -0 are indistinguishable in all other contexts (even comparison).

Due to the limited number of digits, the result of a floating-point operation needs to be rounded. IEEE 754 provides four rounding modes: towards 0, towards $+\infty$, towards $-\infty$, and to nearest. Depending on the chosen rounding mode and the unrounded result, either the floating-point number directly before or directly after the unrounded result is chosen (possibly $+\infty$ or $-\infty$). Rounding can build infinities from finite operands (this is called *overflow*) and may return zero when the absolute value of the result is too small to be represented (this is called *underflow*). Because of this rounding phase, most algebraic properties of the operators on $\mathbb{R}$, such as associativity and distributivity, are lost. However, the opposite of a number is always exactly represented (unlike what happens in two's complement integer arithmetics), and comparison operators are exact.

The IEEE 754-1985 norm is described in full details in [CS85].

7.4.2 IEEE 754 Computation Model

IEEE 754 arithmetics is quite complex as it is designed to be usable in various contexts and different computation models. Complex features include:

- arithmetics on $+\infty$ and $-\infty$;
- the distinction between $+0$ and -0 in certain operations;
- the ability to install trap handlers when exceptional behaviors, such as underflows, overflows, divisions by zero, inexact rounding, or invalid operations — that would otherwise return a *NaN* — occur;

$$[\![\, expr \,]\!]^{Int}_{mi} : (\mathcal{V} \to \mathcal{B}^{Int}) \to (\mathcal{B}^{Int} \times \mathbb{B})$$

$$[\![\, V \,]\!]^{Int}_{mi} R^\sharp \stackrel{\text{def}}{=} (R^\sharp(V),\, \mathbf{F})$$

$$[\![\, [a,b]_{\mathbf{i}} \,]\!]^{Int}_{mi} R^\sharp \stackrel{\text{def}}{=} \mathtt{check}_{\mathbf{i}}\,([a,b])$$

$$[\![\, \mathtt{cast}_{\mathbf{i}}\,(e_{\mathbf{i}}) \,]\!]^{Int} R^\sharp \stackrel{\text{def}}{=} (Y^\sharp,\, O \vee O') \text{ where } (X^\sharp, O) = [\![\, e_{\mathbf{i}} \,]\!]^{Int}_{mi} R^\sharp,\ (Y^\sharp, O') = \mathtt{check}_{\mathbf{i}}\,(X^\sharp)$$

$$[\![\, -_{\mathbf{i}}\, e_{\mathbf{i}} \,]\!]^{Int}_{mi} R^\sharp \stackrel{\text{def}}{=} (Y^\sharp,\, O \vee O') \text{ where } (X^\sharp, O) = [\![\, e_{\mathbf{i}} \,]\!]^{Int}_{mi} R^\sharp,\ (Y^\sharp, O') = \mathtt{check}_{\mathbf{i}}\,(-^{Int} X^\sharp)$$

$$[\![\, e_{\mathbf{i}} \diamond_{\mathbf{i}} e'_{\mathbf{i}} \,]\!]^{Int}_{mi} R^\sharp \stackrel{\text{def}}{=} (Z^\sharp,\, O \vee O' \vee O'') \text{ where } (X^\sharp, O) = [\![\, e_{\mathbf{i}} \,]\!]^{Int}_{mi} R^\sharp,\ (Y^\sharp, O') = [\![\, e'_{\mathbf{i}} \,]\!]^{Int}_{mi} R^\sharp,\ (Z^\sharp, O'') = \mathtt{check}_{\mathbf{i}}\,(X^\sharp \ \diamond^{Int}\ Y^\sharp),\ \diamond \in \{+, -, \times, /\}$$

$$\mathtt{check}^{Int}_{\mathbf{i}}\,(X^\sharp) \stackrel{\text{def}}{=} \begin{cases} (\bot^{Int}, \mathbf{F}) & \text{if } X^\sharp = \bot^{Int} \\ (X^\sharp, \mathbf{F}) & \text{if } X^\sharp = [a,b] \text{ and } a \geq m_{\mathbf{i}},\ b \leq M_{\mathbf{i}} \\ ([m_{\mathbf{i}}, M_{\mathbf{i}}], \mathbf{T}) & \text{otherwise} \end{cases}$$

where the $\diamond^{Int}$ operators are defined in Sect. 2.4.6.

Figure 7.3: Interval abstract semantics adapted to machine-integers.

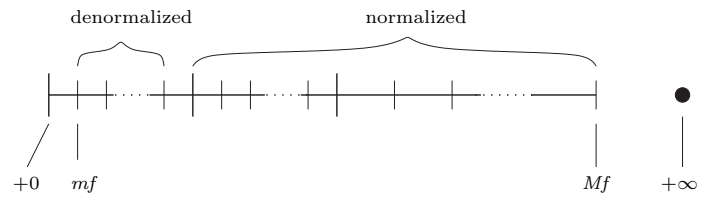


Figure 7.4: Positive, non-*NAN*, floating-point numbers.

$$
\begin{array}{lcll}
expr_{\mathbf{f}} & ::= & X & X \in \mathcal{V}_{\mathbf{f}} \\
& | & [a,b]_{\mathbf{f},r} & a,b \in \mathbb{R} \\
& | & \ominus\, expr_{\mathbf{f}} & \\
& | & expr_{\mathbf{f}} \odot_{\mathbf{f},r} expr_{\mathbf{f}} & \odot \in \{\oplus,\ominus,\otimes,\oslash\} \\
& | & \mathtt{cast}_{\mathbf{f},r}\,(expr_{\mathbf{f}'}) & \\
\\
test & ::= & expr_{\mathbf{f}} \bowtie expr_{\mathbf{f}} & \bowtie \in \{=,\neq,<,\leq\} \\
& | & \cdots & \\
\\
inst & ::= & X \leftarrow expr_{\mathbf{f}} & X \in \mathcal{V}_{\mathbf{f}} \\
& | & \cdots &
\end{array}
$$

$\mathbf{f} \in \mathbf{F}$ is a floating-point format.
$r \in \{n, 0, +\infty, -\infty\}$ is a rounding mode.
Each $\mathcal{V}_{\mathbf{f}}$, $\mathbf{f} \in \mathbf{F}$, is a finite set of variables.

Figure 7.5: Simple language syntax adapted to floating-point numbers.

- the distinction between quiet and signaling *NaN*s;
- the ability to dynamically adjust the rounding mode.

We focus here on the large class of programs that treat floating-point arithmetics as a practical approximation to the mathematical reals $\mathbb{R}$. Thus, rounding and underflows are tolerated, but not overflows, divisions by zero, or invalid operations, which are considered run-time errors and halt the program. In this context, $+\infty$, $-\infty$, and *NaN*s can never be created. As a consequence, the difference between $+0$ and -0 becomes irrelevant. In our concrete semantics, the set of floating-point numbers of format $\mathbf{f} \in \mathbf{F}$ will be assimilated to a finite subset of $\mathbb{R}$, denoted by $\mathbb{F}_{\mathbf{f}}$.

Adapted Syntax. Fig. 7.5 presents some modifications to the syntax of our Simple language described in Fig. 2.1 so as to replace expressions in $\mathbb{I}$ with floating-point expressions. As for the machine-integer semantics, we type each variable and expression with a format $\mathbf{f} \in \mathbf{F}$ and enforce simple typing rules. The arithmetic operators have been circled to distinguish them from the corresponding operators on perfect reals, and we have added a `cast` operator to convert from a floating-point format to another — possibly introducing some rounding. The binary $\oplus$, $\ominus$, $\otimes$, $\oslash$, and unary `cast` operators are tagged with a floating-point format $\mathbf{f} \in \mathbf{F}$ but also with a rounding mode $r \in \{n, 0, +\infty, -\infty\}$ (n representing rounding to nearest) as both information affect the rounded result. As all the $\mathbb{F}_{\mathbf{f}}$'s are

$$R_{\mathbf{f},r} : \mathbb{R} \to \mathbb{F}_{\mathbf{f}} \cup \{\Omega\}$$

$$R_{\mathbf{f},+\infty}(x) \stackrel{\text{def}}{=} \begin{cases} \Omega & \text{if } x > Mf_{\mathbf{f}} \\ \min\{y \in \mathbb{F}_{\mathbf{f}} \mid y \geq x\} & \text{otherwise} \end{cases}$$

$$R_{\mathbf{f},-\infty}(x) \stackrel{\text{def}}{=} \begin{cases} \Omega & \text{if } x < -Mf_{\mathbf{f}} \\ \max\{y \in \mathbb{F}_{\mathbf{f}} \mid y \leq x\} & \text{otherwise} \end{cases}$$

$$R_{\mathbf{f},0}(x) \stackrel{\text{def}}{=} \begin{cases} \min\{y \in \mathbb{F}_{\mathbf{f}} \mid y \geq x\} & \text{if } x \leq 0 \\ \max\{y \in \mathbb{F}_{\mathbf{f}} \mid y \leq x\} & \text{if } x \geq 0 \end{cases}$$

$$R_{\mathbf{f},n}(x) \stackrel{\text{def}}{=} \begin{cases} \Omega & \text{if } |x| \geq (2 - 2^{-\mathbf{p}-1})2^{2^{\mathbf{e}}-\mathbf{bias}-2} \\ Mf_{\mathbf{f}} & \text{otherwise if } x \geq Mf_{\mathbf{f}} \\ -Mf_{\mathbf{f}} & \text{otherwise if } x \leq -Mf_{\mathbf{f}} \\ R_{\mathbf{f},-\infty}(x) & \text{otherwise if } |R_{\mathbf{f},-\infty}(x) - x| < |R_{\mathbf{f},+\infty}(x) - x| \\ R_{\mathbf{f},+\infty}(x) & \text{otherwise if } |R_{\mathbf{f},+\infty}(x) - x| < |R_{\mathbf{f},-\infty}(x) - x| \\ R_{\mathbf{f},-\infty}(x) & \text{otherwise if } R_{\mathbf{f},-\infty}(x)\text{'s least significant bit is 0} \\ R_{\mathbf{f},+\infty}(x) & \text{otherwise if } R_{\mathbf{f},+\infty}(x)\text{'s least significant bit is 0} \end{cases}$$

Figure 7.6: Rounding functions, following [CS85].

perfectly symmetric, the unary minus $\ominus$ operator is always exact independently from the chosen format and rounding mode, and so, it is not tagged.

Adapted Concrete Semantics. An environment $\rho \in \prod_{\mathbf{f} \in \mathbf{F}} (\mathcal{V}_{\mathbf{f}} \to \mathbb{F}_{\mathbf{f}})$ is a function that associates to each variable a floating-point value in the corresponding format. Fig. 7.7 gives the concrete semantics $[\![expr_{\mathbf{f}}]\!]_{fl}(\rho) \in \mathcal{P}(\mathbb{F}_{\mathbf{f}})$ of a numerical expression $expr_{\mathbf{f}}$ in an environment ρ as a set of possible floating-point values. Following the IEEE 754 norm [CS85], the semantics of each floating-point operator can be decomposed into: firstly, a computation using the corresponding real operator and, secondly, the application of one of the four *rounding functions* $R_{\mathbf{f},r} : \mathbb{R} \to \mathbb{F}_{\mathbf{f}} \cup \{\Omega\}$, presented in Fig. 7.6, that return either a floating-point number or the run-time error Ω — corresponding to an overflow or a division by zero. Note that all rounding functions are fully deterministic, but they can be quite complex — especially rounding to nearest. Following our convention in Simple, all the computations that trigger a run-time error are dropped, and so, Ω is not propagated further in $[\![\cdot]\!]_{fl}$. It is used only locally to simplify the definitions of Figs. 7.6–7.7.

Each comparison operator in $\mathbb{F}_{\mathbf{f}}$ is a restriction of the corresponding operator on reals; no rounding is involved. The semantics of boolean expressions as well as all our concrete

$$\llbracket\, expr_{\mathbf{f}} \,\rrbracket_{fl} : (\prod_{\mathbf{g}\in\mathbf{F}} (\mathcal{V}_{\mathbf{g}} \to \mathbb{F}_{\mathbf{g}})) \to \mathcal{P}(\mathbb{F}_{\mathbf{f}})$$

$$\begin{array}{lcl}
\llbracket\, [a,b]_{\mathbf{f},r} \,\rrbracket_{fl}(\rho) & \stackrel{\text{def}}{=} & \{\ R_{\mathbf{f},r}(c) \mid a \leq c \leq b,\ R_{\mathbf{f},r}(c) \neq \Omega\ \} \\
\llbracket\, X \,\rrbracket_{fl}(\rho) & \stackrel{\text{def}}{=} & \{\ \rho(X)\ \} \\
\llbracket\, \ominus e_{\mathbf{f}} \,\rrbracket_{fl}(\rho) & \stackrel{\text{def}}{=} & \{\ -c \mid c \in \llbracket\, e_{\mathbf{f}} \,\rrbracket_{fl}(\rho)\ \} \\
\llbracket\, \mathtt{cast}_{\mathbf{f},r}\,(e_{\mathbf{f}'}) \,\rrbracket_{fl}(\rho) & \stackrel{\text{def}}{=} & \{\ R_{\mathbf{f},r}(c) \mid c \in \llbracket\, e_{\mathbf{f}'} \,\rrbracket_{fl}(\rho),\ R_{\mathbf{f},r}(c) \neq \Omega\ \} \\
\\
\llbracket\, e_{\mathbf{f}}\ \odot_{\mathbf{f},r}\ e'_{\mathbf{f}} \,\rrbracket_{fl}(\rho) & \stackrel{\text{def}}{=} & \{\ R_{\mathbf{f},r}(c \cdot c') \mid c \in \llbracket\, e_{\mathbf{f}} \,\rrbracket_{fl}(\rho),\ c' \in \llbracket\, e'_{\mathbf{f}} \,\rrbracket_{fl}(\rho), \\
 & & \qquad R_{\mathbf{f},r}(c \cdot c') \neq \Omega\ \} \\
 & & \cdot \in \{+,-,\times\} \\
\\
\llbracket\, e_{\mathbf{f}}\ \oslash_{\mathbf{f},r}\ e'_{\mathbf{f}} \,\rrbracket_{fl}(\rho) & \stackrel{\text{def}}{=} & \{\ R_{\mathbf{f},r}(c/c') \mid c \in \llbracket\, e_{\mathbf{f}} \,\rrbracket_{fl}(\rho),\ c' \in \llbracket\, e'_{\mathbf{f}} \,\rrbracket_{fl}(\rho), \\
 & & \qquad c' \neq 0,\ R_{\mathbf{f},r}(c/c') \neq \Omega\ \}
\end{array}$$

Figure 7.7: Concrete semantics of numerical expressions adapted to floating-point numbers.

transfer functions are almost unchanged and not presented here: they simply use our new semantics for numerical expressions $\llbracket\, \cdot \,\rrbracket_{fl}$ instead of $\llbracket\, \cdot \,\rrbracket$.

Our concrete semantics faithfully corresponds to the IEEE 754 norm [CS85] where the overflow, division by zero, and invalid operation exception traps abort the system with a run-time error.

Floating-Point Pitfalls. Due to rounding, the floating-point arithmetics behaves quite differently than the perfect real arithmetics. Small rounding errors can sum up rapidly to a large imprecision which can lead to catastrophic behaviors, such as the Patriot missile story related in [Ske92].[2] Another example, given in [GGP+01], is the following code fragment:

$$\mathtt{if}\ \ X > 0\ \ \mathtt{\{}\ \ Y \leftarrow 1 \oslash_{\mathbf{f},n} (X \otimes_{\mathbf{f},n} X)\ \ \mathtt{\}}$$

that can result in a division by zero as $X > 0$ does not implies $X \otimes_{\mathbf{f},n} X \neq 0$ when X is so small that its square underflows to 0. More generally, provably stable mathematical computations may become unstable once implemented using floating-point numbers. The reader may find the description of many more floating-point arithmetics properties and pitfalls in the classical paper [Gol91] by Goldberg.

[2] The specific Patriot missile error was due to cumulated rounding in *fixed*-point computations but similar problems arise when using floating-point arithmetics.

Programming with floating-point arithmetics may seem difficult, but abstracting floating-point numbers is even more difficult. Due to the rounding phase, all the distributivity and associativity properties of the arithmetic operators are no longer true. For instance, $X \ominus_{\mathbf{f},r} X = 0$ holds, while $X \oplus_{\mathbf{f},r} (Y \ominus_{\mathbf{f},r} X) = Y$ may not. As another example, using 32−bit single precision numbers we get:

$$(10^{22} \oplus_{\mathbf{32},n} 1.000000019 \cdot 10^{38}) \oplus_{\mathbf{32},n} (10^{22} \ominus_{\mathbf{32},n} 1.000000019 \cdot 10^{38}) = 0 \ .$$

Thus, most algebraic expression manipulations and simplifications become illegal. A consequence is that it is almost impossible to express invariants as floating-point expressions and manipulate them soundly in a relational abstract domain. Consider, for instance, a naive adaptation of the zone domain of Chap. 3 that would consists in expressing conjunctions of invariants of the form $V_j \ominus_{\mathbf{f},r} V_i \leq \mathbf{m}_{ij}$ using a matrix $\mathbf{m}$. Then, as $X \ominus_{\mathbf{f},r} Y \leq c$ and $Y \ominus_{\mathbf{f},r} Z \leq d$ do not imply $X \ominus_{\mathbf{f},r} Z \leq c + d$, the core closure algorithm is no longer sound.

Our solution is to keep expressing invariants as real expressions: we will abstract sets of floating-point numbers using numerical domains in $\mathbb{I} = \mathbb{R}$. There is an inherent loss of precision when abstracting sets of floating-point environments as sets of real environments because $\mathbb{R}$ is much more dense. The situation is similar to the abstraction of integer polyhedra using rational polyhedra, discussed in Sect. 2.4.7: operators can construct spurious floating-point numbers by combining real numbers in-between floating-point numbers. As a consequence, an operator that is exact or best when $\mathbb{I} = \mathbb{R}$ will no longer be exact or best with respect to a concretisation that keeps only representable floating-point numbers.

7.4.3 Linearising Floating-Point Expressions

On the one hand, floating-point expressions cannot be fed directly to numerical abstract domain whose transfer functions are only sound for expressions in the real field $\mathbb{R}$. On the other hand, expressing floating-point computations accurately using classical operators on reals — which is done, for instance, in our concrete semantics of Fig. 7.7 — leads to highly non-linear expressions, due to the complexity of the rounding operators. Our solution is to apply a *linearisation* function, in the spirit of Sect. 6.2.3, to abstract a floating-point expression into an interval linear form *on reals*. The abstract operators on interval of real numbers introduced in Def. 2.4.6, $+^{Int}$, $-^{Int}$, $\times^{Int}$, and $/^{Int}$, will be quite useful, as well as the interval linear form operators ι, $\boxplus$, $\boxminus$, $\boxtimes$, and $\boxslash$, proposed in Defs. 6.2.1 and 6.2.2. Note that, although the floating-point addition is not associative, we do not need to specify an evaluation order for our interval linear forms as they still use the perfectly associative $+$ operator on reals — however, changing the evaluation order of the original floating-point expression will affect the output of the linearisation.

Rounding. A first step is to bound the amount of rounding introduced by the $R_{\mathbf{f},r}$ function. Two cases arise:

- If x is *normalised*, then the gap between $R_{\mathbf{f},r}(x)$ and x is less than $2^{-\mathbf{p}} \times |x|$, where $\mathbf{p}$ is the fraction bit-size of the floating-point format $\mathbf{f}$. We thus consider a relative rounding error of amplitude $2^{-\mathbf{p}}$.
- When x is *denormalised*, $R_{\mathbf{f},r}(x) - x$ is more easily expressed as an absolute rounding error of amplitude $mf_{\mathbf{f}}$ which is the gap between two successive denormalised numbers.

We sum these two causes of rounding to get a simple approximation of the actual rounding induced by an operator as a function of its arguments:

Theorem 7.4.1. Rounding abstraction.

Suppose that $|a \cdot b| \leq Mf_{\mathbf{f}}$*, then:*

- $|(a \ \oplus_{\mathbf{f},r} \ b) - (a + b)| \leq 2^{-\mathbf{p}}(|a| + |b|) + mf_{\mathbf{f}}$
- $|(a \ \ominus_{\mathbf{f},r} \ b) - (a - b)| \leq 2^{-\mathbf{p}}(|a| + |b|) + mf_{\mathbf{f}}$
- $|(a \ \otimes_{\mathbf{f},r} \ b) - (a \times b)| \leq 2^{-\mathbf{p}}(|a| \times |b|) + mf_{\mathbf{f}}$
- $|(a \ \oslash_{\mathbf{f},r} \ b) - (a/b)| \leq 2^{-\mathbf{p}}(|a|/|b|) + mf_{\mathbf{f}}$

●

Note that these bounds are valid for all rounding modes: they correspond to an over-approximation of the gap between the two successive floating-point numbers that bracket the real result. If we know that rounding is to nearest — as it is often the case — then the maximum rounding error is only half this gap. For instance, it is safe the consider a more precise bound of the form:

$$|(a \ \oplus_{\mathbf{f},r} \ b) - (a + b)| \leq (2^{-\mathbf{p}}(|a| + |b|) + mf_{\mathbf{f}})/2 \ .$$

Given an interval linear form l, adding an absolute rounding error of amplitude $mf_{\mathbf{f}}$ simply corresponds to adding the constant interval $[-mf_{\mathbf{f}}, mf_{\mathbf{f}}]$. The relative rounding error of amplitude $2^{-\mathbf{p}}$ on l is a little more complex to compute but can be expressed as an interval linear form using the following $\epsilon_{\mathbf{f}}$ operator:

Definition 7.4.1. Relative rounding $\epsilon_{\mathbf{f}}$ on an interval linear form.

$$\epsilon_{\mathbf{f}}\left([a,b] \boldsymbol{+} \sum_{k=1}^{n} [a_k, b_k] \boldsymbol{\times} V_k\right) \stackrel{\text{def}}{=}$$
$$(\max(|a|,|b|) \boldsymbol{\times}^{Int} [-2^{-\mathbf{p}}, 2^{-\mathbf{p}}]) \ \boldsymbol{+} \ \sum_{k=1}^{n} (\max(|a_k|,|b_k|) \boldsymbol{\times}^{Int} [-2^{-\mathbf{p}}, 2^{-\mathbf{p}}]) \boldsymbol{\times} V_k \ .$$

●

Linearisation. We are now ready to describe our linearisation procedure $(\!| \mathit{expr} |\!)_{fl} R^\sharp$ for a floating-point expression *expr* in the abstract environment $R^\sharp$ as follows:

Definition 7.4.2. Floating-point linearisation.

1. $(\!| V_i |\!)_{fl} R^\sharp \stackrel{\text{def}}{=} [1,1] \boldsymbol{\times} V_i$
2. $(\!| [a,b]_{\mathbf{f},r} |\!)_{fl} R^\sharp \stackrel{\text{def}}{=} [R_{\mathbf{f},r}(a),\ R_{\mathbf{f},r}(b)]$
3. $(\!| \ominus e_{\mathbf{f}} |\!)_{fl} R^\sharp \stackrel{\text{def}}{=} \boxminus (\!| e_{\mathbf{f}} |\!)_{fl} R^\sharp$
4. $(\!| e_{\mathbf{f}} \oplus_{\mathbf{f},r} e'_{\mathbf{f}} |\!)_{fl} R^\sharp \stackrel{\text{def}}{=} (\!| e_{\mathbf{f}} |\!)_{fl} R^\sharp \boxplus (\!| e'_{\mathbf{f}} |\!)_{fl} R^\sharp \boxplus \epsilon_{\mathbf{f}}((\!| e_{\mathbf{f}} |\!)_{fl} R^\sharp) \boxplus \epsilon_{\mathbf{f}}((\!| e'_{\mathbf{f}} |\!)_{fl} R^\sharp) \boxplus mf_{\mathbf{f}}[-1,1]$
5. $(\!| e_{\mathbf{f}} \ominus_{\mathbf{f},r} e'_{\mathbf{f}} |\!)_{fl} R^\sharp \stackrel{\text{def}}{=} (\!| e_{\mathbf{f}} |\!)_{fl} R^\sharp \boxminus (\!| e'_{\mathbf{f}} |\!)_{fl} R^\sharp \boxplus \epsilon_{\mathbf{f}}((\!| e_{\mathbf{f}} |\!)_{fl} R^\sharp) \boxplus \epsilon_{\mathbf{f}}((\!| e'_{\mathbf{f}} |\!)_{fl} R^\sharp) \boxplus mf_{\mathbf{f}}[-1,1]$
6. $(\!| [a,b] \otimes_{\mathbf{f},r} e'_{\mathbf{f}} |\!)_{fl} R^\sharp \stackrel{\text{def}}{=} ([a,b] \boxtimes (\!| e'_{\mathbf{f}} |\!)_{fl} R^\sharp) \boxplus ([a,b] \boxtimes \epsilon_{\mathbf{f}}((\!| e'_{\mathbf{f}} |\!)_{fl} R^\sharp)) \boxplus mf_{\mathbf{f}}[-1,1]$
7. $(\!| e_{\mathbf{f}} \oslash_{\mathbf{f},r} [a,b] |\!)_{fl} R^\sharp \stackrel{\text{def}}{=} ((\!| e'_{\mathbf{f}} |\!)_{fl} R^\sharp \boxslash [a,b]) \boxplus (\epsilon_{\mathbf{f}}((\!| e'_{\mathbf{f}} |\!)_{fl} R^\sharp) \boxslash [a,b]) \boxplus mf_{\mathbf{f}}[-1,1]$
8. $(\!| \mathtt{cast}_{\mathbf{f},r}(e_{\mathbf{f}'}) |\!)_{fl} R^\sharp \stackrel{\text{def}}{=} \begin{cases} (\!| e_{\mathbf{f}'} |\!)_{fl} R^\sharp & \text{if } \mathbf{F}_{\mathbf{f}'} \subseteq \mathbf{F}_{\mathbf{f}} \\ (\!| e_{\mathbf{f}'} |\!)_{fl} R^\sharp \boxplus \epsilon_{\mathbf{f}}((\!| e |\!)_{fl} R^\sharp) \boxplus mf_{\mathbf{f}}[-1,1] & \text{otherwise} \end{cases}$
9. $(\!| e_{\mathbf{f}} \otimes_{\mathbf{f},r} e'_{\mathbf{f}} |\!)_{fl} R^\sharp \stackrel{\text{def}}{=} (\!| \iota((\!| e_{\mathbf{f}} |\!)_{fl} R^\sharp) R^\sharp \otimes_{\mathbf{f},r} e'_{\mathbf{f}} |\!)_{fl} R^\sharp$
   or
   $(\!| e_{\mathbf{f}} \otimes_{\mathbf{f},r} e'_{\mathbf{f}} |\!)_{fl} R^\sharp \stackrel{\text{def}}{=} (\!| \iota((\!| e'_{\mathbf{f}} |\!)_{fl} R^\sharp) R^\sharp \otimes_{\mathbf{f},r} e_{\mathbf{f}} |\!)_{fl} R^\sharp$
10. $(\!| e_{\mathbf{f}} \oslash_{\mathbf{f},r} e'_{\mathbf{f}} |\!)_{fl} R^\sharp \stackrel{\text{def}}{=} (\!| e_{\mathbf{f}} \oslash_{\mathbf{f},r} \iota((\!| e'_{\mathbf{f}} |\!)_{fl} R^\sharp) R^\sharp |\!)_{fl} R^\sharp$

●

As in the previous chapter, an abstract element $R^\sharp$ is required to use the intervalisation operator ι when encountering non-linear terms, and so, our linearisation dynamically interacts with a numerical abstract domain. The following theorem proves that the linearised expression, evaluated in the real field, indeed over-approximates the behavior of the original floating-point expression on the environments $\gamma(R^\sharp)$:

Theorem 7.4.2. Soundness of the floating-point linearisation.

$$\forall \rho \in \gamma(R^\sharp), \quad [\![\mathit{expr}]\!]_{fl}(\rho) \subseteq [\![(\!| \mathit{expr} |\!)_{fl}(R^\sharp)]\!](\rho) \ .$$

●

Proof. This is a consequence of Thm. 7.4.1 on rounding, as well as Thm. 6.2.1 on the properties of interval linear form operators. ○

Applications. The main consequence of Thm. 7.4.2 is that we can soundly abstract any floating-point transfer function using legacy transfer functions on real environments — such as the ones presented for intervals in Sect. 2.4.6, zones in Chap. 3, or octagons in Chap. 4 — if we take care to replace floating-point expressions by their linearised versions first. For instance, the floating-point assignment transfer function on the octagon abstract domain would be:

$$\{\!|\, V \leftarrow \mathit{expr} \,|\!\}^{Oct}_{fl}(X^\sharp) \stackrel{\text{def}}{=} \{\!|\, V \leftarrow (\!|\, \mathit{expr} \,|\!)_{fl}(X^\sharp) \,|\!\}^{Oct}_{rel}(X^\sharp)$$

where $\{\!|\, \cdot \,|\!\}^{Oct}_{rel}$ is defined in Def. 4.4.7.

Any of the strategies that were presented in Sect. 6.2.3 to choose which sub-expression to intervalise when encountering a multiplication can be applied here. Likewise, we can easily extend the linearisation to expressions with new, non-linear, operators as long we have a sound interval abstraction that takes rounding into account, following what was proposed in Sect. 6.2.6. Finally, the floating-point linearisation can benefit from the symbolic constant propagation technique described in Sect. 6.3.

Application to the Detection of Run-Time Errors. As for machine-integers, in Sect. 7.2.2, our abstract semantics can be used to detect run-time errors statically. As errors do not appear explicitly in our semantics, we must use two steps:

- In a first step, we perform our static analysis to infer the maximal range of each variable at each program point, using possibly a relational abstract domain for increased precision.

- In a second step, for each operator at each program point l, we check the following preconditions that imply the absence of overflow, division by zero, and invalid operation:

  - for each sub-expression $e_{\mathbf{f}} \odot_{\mathbf{f},r} e'_{\mathbf{f}}$, we check that:

    $$[\![\, e_{\mathbf{f}} \,]\!]^{Int}_{fl}(X^\sharp)(l) \diamond^{Int} [\![\, e'_{\mathbf{f}} \,]\!]^{Int}_{fl}(X^\sharp)(l) \subseteq [-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}]$$

    using the regular interval abstract operators $\diamond^{Int}$ on reals.

  - for each sub-expression $\mathtt{cast}_{\mathbf{f},r}\,(e_{\mathbf{f}'})$, we check that:

    $$[\![\, e_{\mathbf{f}'} \,]\!]^{Int}_{fl}(X^\sharp)(l) \subseteq [-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}]$$

  - for each sub-expression $e_{\mathbf{f}} \oslash_{\mathbf{f},r} e'_{\mathbf{f}}$, we check that $0 \notin [\![\, e'_{\mathbf{f}} \,]\!]^{Int}_{fl}(X^\sharp)(l)$

  where the floating-point interval expression evaluation $[\![\, \cdot \,]\!]^{Int}_{fl}$ is defined in Sect. 7.5.1.

Precision Loss. The precision loss due to our linearisation is twofold. The first cause is due, as for the vanilla linearisation of Sect. 6.2.3, to the "intervalisation" that can occur in multiplications and divisions. The second one, which is new, is due to our treatment of rounding. Indeed, all the rounding functions $R_{\mathbf{f},r}$ are perfectly deterministic but we abstract them as a non-deterministic choice within an interval. This is generally not a problem as the correctness of many numerical programs does not depend on exactly which value within the error interval is picked at each computation step. For numerical algorithms relying on the rounding being accurate "to the bit" to the definition of Fig. 7.6 — such as the accurate summation algorithm by Malcom [Mal71] — our non-deterministic abstraction may not be able to prove the full correctness but might still be sufficiently precise to prove that no overflow or division by zero can occur.

Note that, due to relative error rounding terms, linearised expressions are seldom quasi-linear forms: variable coefficients are not singletons, even if the original expression had no interval coefficient and no intervalisation was used. We recall that, if our abstract domain only supports quasi-linear forms — such as the polyhedron abstract domain — we can use the μ operator, presented in Sect. 6.2.5, to abstract interval linear forms into quasi-linear forms. However, the μ operator induces some precision loss because all the relative rounding error terms are transformed into an absolute rounding error and incorporated into the constant coefficient of the quasi-linear form: we forget which part of the rounding comes from which variable.

Rounding Mode Influence. Our definition is independent from the rounding mode r chosen for each operator. It is sound by considering always the worst case: upper bounds are rounded towards $+\infty$ and lower bounds towards $-\infty$. This is quite a nice feature because it frees the analyser from the task of discovering which rounding mode is currently in use — which may be a complex task as it can be changed dynamically using system calls.

Double Rounding Problem. An important advantage of using non-deterministic intervals combined with worst-case rounding is that the linearised result remains sound even if the concrete semantics actually computes some sub-expressions in a floating-point format *more* precise than what is required by the operator types. This can happen, for instance, on the Intel processor: actual computations on double precision expressions are performed in 80-bit registers but, when a register is spilled or stored into memory, it is converted into its correct, 64-bit, format. $X \oplus_{\mathbf{64},n} Y$ can be silently compiled into $\mathtt{cast}_{\mathbf{64},n}(X \oplus_{\mathbf{80},n} Y)$ which gives a different concrete result — this is known as the *double rounding* problem. Determining the exact concrete result would require a deep knowledge on how expressions are compiled while it is safe to consider, for our abstract linearisation, the *minimal* precision at which each operator is supposed to be executed — 64−bit here. This minimal

precision is generally available as a type information in high-level languages.

7.4.4 Analysis Example

We restate the rate limiter example of Ex. 4.6.3 in the context of floating-point computations in the single precision format, as follows:

Example 7.4.1. Floating-point rate limiter analysis.

$Y \leftarrow 0$;
`while` ❶ `rand {`
  $X \leftarrow [-128, 128]$;
  $D \leftarrow [0, 16]$;
  $S \leftarrow Y$;
② $R \leftarrow X \ominus_{\mathbf{32},n} S$;
  $Y \leftarrow X$;
  `if` $R \leq \ominus D$ `{` ③ $Y \leftarrow S \ominus_{\mathbf{32},n} D$ ④ `} else`
  `if` $D \leq R$ `{` ⑤ $Y \leftarrow S \oplus_{\mathbf{32},n} D$ ⑥ `}`
⑦ `}`

●

Recall that, in the case of real computations, the invariant $Y \in [-128, 128]$ holds at ⑦. The octagon abstract domain was able to prove that any interval $[-M, M]$ is stable for Y provided that M is a widening threshold greater than 144. This analysis was not optimal but still much more precise than the interval-based one that cannot prove that Y is bounded at all.

In the floating-point version, it is difficult to find the smallest stable interval for Y due to rounding introduced at lines ②, ③, and ⑤. All we can say is that all the $[-128-\epsilon, 128+\epsilon]$ intervals, where $\epsilon > \epsilon_0$, are stable, for some very small positive ϵ_0.

We now show that the octagon abstract domain is able to find a rather good stable interval automatically. We first suppose that, at a given abstract loop iteration, $Y \in [-M, M]$ at ❶. Then, at ②, the assignment $R \leftarrow X \ominus_{\mathbf{32},n} S$ is linearised into:

$$R \leftarrow [1 - 2^{-\mathbf{p}}, 1 + 2^{-\mathbf{p}}] \times X - [1 - 2^{-\mathbf{p}}, 1 + 2^{-\mathbf{p}}] \times S + [-mf_{\mathbf{32}}, mf_{\mathbf{32}}]$$

with $\mathbf{p} = 23$ and $mf_{\mathbf{32}} = 2^{-149}$. Hence, the assignment transfer function $\llparenthesis V \leftarrow expr \rrparenthesis^{Oct}_{rel}$ of Def. 4.4.7 is able to infer the constraint:

$$|R + S| \leq 128(1 + 2^{-23}) + 2^{-23}M + 2^{-149} \ .$$

Then, at ③, the test implies $R + D \leq 0$, which implies $-R \geq 0$. By closure, as $S = (S + R) - R \geq S + R$, the octagon domain is able to infer that:

$$S \in [-(128(1 + 2^{-23}) + 2^{-23}M + 2^{-149}),\ M] \cap [-M, M] \ .$$

The subsequent assignment is modeled as:

$$Y \leftarrow [1-2^{-\mathbf{p}}, 1+2^{-\mathbf{p}}] \times S - [1-2^{-\mathbf{p}}, 1+2^{-\mathbf{p}}] \times D + [-mf_{\mathbf{32}}, mf_{\mathbf{32}}]$$

hence the following constraints at ④, using $S \in [-M, M]$ and $D \in [0, 16]$:

$$Y - S \in [-16(1+2^{-23}) - 2^{-23}M - 2^{-149},\ 16 \times 2^{-23} + 2^{-23}M + 2^{-149}] \ .$$

By closure, as $Y = (Y - S) + S$, we can infer the constraint:

$$Y \in [-144(1+2^{-23}) - 2^{-22}M - 2^{-148},\ 16 \times 2^{-23} + (1+2^{-23})M + 2^{-149}] \ .$$

Likewise, we can prove that, at ⑥, we have:

$$Y \in [-16 \times 2^{-23} - (1+2^{-23})M - 2^{-149},\ 144(1+2^{-23}) + 2^{-22}M + 2^{-148}]$$

and so, at ⑦, we have the following bound for Y:

$$|Y| \leq 144(1+2^{-23}) + 2^{-22}M + 2^{-148} \ .$$

Thus, any bound M larger than $M_0 \stackrel{\text{def}}{=} (144(1+2^{-23}) + 2^{-148})/(1-2^{-22}) \approx 144.00005$ is stable. Using a widening with thresholds, the octagon abstract domain will be able to prove that Y is bounded by the smallest threshold larger than M_0 and, as a consequence, that the program is safe: it cannot perform any overflow at lines ②, ③, nor ⑤.

7.5 Using Floating-Point Numbers in the Abstract

The floating-point abstract semantics presented in the previous section makes use of computations in the real field $\mathbb{R}$, which may be costly to implement. We now propose to modify our linearisation technique and adapt our abstract domains so that all abstract computations are done in the floating-point world. This will induce some more loss of precision, but allow a tremendous performance boost.

7.5.1 Floating-Point Interval Analysis

We recall here the classical adaptation of interval arithmetics [Moo66] to floating-point expressions. It allows designing an interval abstract domain that is sound with respect to the floating-point semantics, but also uses only floating-point computations internally.

Rounding Mode Sensitive Abstraction. The key remark that allows easily designing interval arithmetics on floating-point numbers is the fact that all the rounding functions $R_{\mathbf{f},r}$, presented in Fig. 7.6, are *monotonic*. Thus, it is sufficient to compute each abstract bound using the same floating-point format and rounding mode as in the concrete world:

- $\mathtt{cast}^{Int}_{\mathbf{f},r}([a,b]) \stackrel{\text{def}}{=} [\mathtt{cast}_{\mathbf{f},r}(a), \mathtt{cast}_{\mathbf{f},r}(b)]$
- $\ominus^{Int}[a,b] \stackrel{\text{def}}{=} [\ominus b, \ominus a]$
- $[a,b] \oplus^{Int}_{\mathbf{f},r} [a',b'] \stackrel{\text{def}}{=} [a \oplus_{\mathbf{f},r} a',\ b \oplus_{\mathbf{f},r} b']$
- $[a,b] \ominus^{Int}_{\mathbf{f},r} [a',b'] \stackrel{\text{def}}{=} [a \ominus_{\mathbf{f},r} b',\ b \ominus_{\mathbf{f},r} a']$
- $[a,b] \otimes^{Int}_{\mathbf{f},r} [a',b'] \stackrel{\text{def}}{=} [\min(a \otimes_{\mathbf{f},r} a', a \otimes_{\mathbf{f},r} b', b \otimes_{\mathbf{f},r} a', b \otimes_{\mathbf{f},r} b'),\ \max(a \otimes_{\mathbf{f},r} a', a \otimes_{\mathbf{f},r} b', b \otimes_{\mathbf{f},r} a', b \otimes_{\mathbf{f},r} b')]$
- $[a,b] \oslash^{Int}_{\mathbf{f},r} [a',b'] \stackrel{\text{def}}{=} [\min(a \oslash_{\mathbf{f},r} a', a \oslash_{\mathbf{f},r} b', b \oslash_{\mathbf{f},r} a', b \oslash_{\mathbf{f},r} b'),\ \max(a \oslash_{\mathbf{f},r} a', a \oslash_{\mathbf{f},r} b', b \oslash_{\mathbf{f},r} a', b \oslash_{\mathbf{f},r} b')]$ when $0 \leq a'$

  $[a,b] \oslash^{Int}_{\mathbf{f},r} [a',b'] \stackrel{\text{def}}{=} ([a,b] \oslash^{Int}_{\mathbf{f},r} ([a',b'] \cap^{Int} [0,+\infty])) \cup^{Int} ([-b,-a] \oslash^{Int}_{\mathbf{f},r} ([-b',-a'] \cap^{Int} [0,+\infty]))$ otherwise

In case a bound evaluates to $+\infty$, $-\infty$, or *NaN*, it is safe to return the interval $[-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}]$.

Rounding Mode Insensitive Abstraction. A drawback of the previous abstraction is that the exact rounding mode and floating-point format must be known by the analyser. As mentioned before, the rounding mode can be changed dynamically, and so, some kind of pre-analysis is required to infer the set of possible rounding modes at each program point. Likewise, the exact floating-point format is not always known from the program source — it depends upon compiler choices — and, due to the double rounding problem mentioned in Sect. 7.4.3, it is not safe to compute the interval bounds using a different floating-point format than the actual concrete one $\mathbf{f}$ — even a less precise one.

In order to design an abstract interval semantics that does not require us to know the rounding mode and is free from the double rounding problem, it is sufficient to always round upper-bounds towards $+\infty$ and lower-bounds towards $-\infty$, as follows:

- $\mathtt{cast}^{Int}_{\mathbf{f},r}([a,b]) \stackrel{\text{def}}{=} [\mathtt{cast}_{\mathbf{f}',-\infty}(a), \mathtt{cast}_{\mathbf{f}',+\infty}(b)]$
- $\ominus^{Int}[a,b] \stackrel{\text{def}}{=} [\ominus b, \ominus a]$
- $[a,b] \oplus^{Int}_{\mathbf{f},r} [a',b'] \stackrel{\text{def}}{=} [a \oplus_{\mathbf{f}',-\infty} a', b \oplus_{\mathbf{f}',+\infty} b']$

- $[a,b] \ominus^{Int}_{\mathbf{f},r} [a',b'] \stackrel{\text{def}}{=} [a \ominus_{\mathbf{f}',-\infty} b', b \ominus_{\mathbf{f}',+\infty} a']$
- $[a,b] \otimes^{Int}_{\mathbf{f},r} [a',b'] \stackrel{\text{def}}{=} [\min(a \otimes_{\mathbf{f}',-\infty} a', a \otimes_{\mathbf{f}',-\infty} b', b \otimes_{\mathbf{f}',-\infty} a', b \otimes_{\mathbf{f}',-\infty} b'),$
  $\max(a \otimes_{\mathbf{f}',+\infty} a', a \otimes_{\mathbf{f}',+\infty} b', b \otimes_{\mathbf{f}',+\infty} a', b \otimes_{\mathbf{f}',+\infty} b')]$
- $[a,b] \oslash^{Int}_{\mathbf{f},r} [a',b'] \stackrel{\text{def}}{=} [\min(a \oslash_{\mathbf{f}',-\infty} a', a \oslash_{\mathbf{f}',-\infty} b', b \oslash_{\mathbf{f}',-\infty} a', b \oslash_{\mathbf{f}',-\infty} b'),$
  $\max(a \oslash_{\mathbf{f}',+\infty} a', a \oslash_{\mathbf{f}',+\infty} b', b \oslash_{\mathbf{f}',+\infty} a', b \oslash_{\mathbf{f}',+\infty} b')]$
  when $0 \leq a'$

  $[a,b] \oslash^{Int}_{\mathbf{f},r} [a',b'] \stackrel{\text{def}}{=} ([a,b] \oslash^{Int}_{\mathbf{f},r} ([a',b'] \cap^{Int} [0,+\infty])) \cup^{Int}$
  $([-b,-a] \oslash^{Int}_{\mathbf{f},r} ([-b',-a'] \cap^{Int} [0,+\infty]))$
  otherwise

where $\mathbf{f}'$ may be as precise or less precise than $\mathbf{f}$. It is generally costly to switch between the $+\infty$ and $-\infty$ rounding modes; fortunately, we can, in an actual analyser implementation, switch once and for all to rounding towards $+\infty$ and use the following identities to simulate rounding towards $-\infty$:

$$\begin{array}{lcllcl} a \oplus_{\mathbf{f},-\infty} b & = & \ominus((\ominus a) \ominus_{\mathbf{f},+\infty} b) & a \ominus_{\mathbf{f},-\infty} b & = & \ominus(b \ominus_{\mathbf{f},+\infty} a) \\ a \otimes_{\mathbf{f},-\infty} b & = & \ominus((\ominus a) \otimes_{\mathbf{f},+\infty} b) & a \oslash_{\mathbf{f},-\infty} b & = & \ominus((\ominus a) \oslash_{\mathbf{f},+\infty} b) \\ \mathtt{cast}_{\mathbf{f},-\infty}(a) & = & (\ominus \mathtt{cast}_{\mathbf{f},+\infty}(\ominus a)) & & & \end{array}$$

Application. From these operators we can design, by structural induction, an interval-based abstraction $[\![expr_{\mathbf{f}}]\!]^{Int}_{fl}$ for the semantics of floating-point expressions $[\![expr_{\mathbf{f}}]\!]_{fl}$ presented in Fig. 7.7. If we take care to design backwards operators, then the generic non-relational abstract domain construction of Sect. 2.4.4 can be applied here to design an interval abstract domain that is sound with respect to the floating-point semantics. Rounding effects are taken care of automatically and we do not need to add relative or absolute rounding error terms, unlike the linearisation $(\!| \mathbf{expr} |\!)_{fl}$ of Sect. 7.4.3. The backwards interval comparison operators $\overleftarrow{\bowtie}^{Int}_{fl}$ are identical to the regular interval ones $\overleftarrow{\bowtie}^{Int}$ presented in Sect. 2.4.6. For the sake of conciseness, we do not present the backwards interval arithmetic operators $\overleftarrow{\diamond}^{Int}_{fl}$ but simply note that the generic fall-back operators of Sect. 2.4.4 cannot be used here because they rely on identities that are no longer true in the floating-point world — such as $a = b \oplus_{\mathbf{f},r} c \not\Longrightarrow b = a \ominus_{\mathbf{f},r} c$.

7.5.2 Floating-Point Linearisation

Recall that the linearisation technique $(\!| \cdot |\!)$, introduced in Chap. 6, and its adaptation $(\!| \cdot |\!)_{fl}$ to floating-point expressions, presented in Sect. 7.4.3, rely on the operators $\boxplus$, $\boxminus$, $\boxtimes$, $\boxslash$, and ι that are themselves implemented using interval arithmetics on reals. We would like to trade precision for efficiency and perform these computations using floating-point arithmetics instead.

A classical result of interval arithmetics — see [Moo66] — shows that it is sufficient to use the rounding-insensitive interval abstraction in any floating-point format; because it rounds upper bounds towards $+\infty$ and lower bounds towards $-\infty$, it is always sound with respect to real arithmetics, as long as no bound evaluates to $+\infty$, $-\infty$, or *NaN*. The more precise the chosen floating-point is, the more accurate the abstraction will be. We have replaced associative operators on real intervals by non-associative ones on floating-point intervals, so, the result of some operators — such as the intervalisation ι of Def. 6.2.2 — may depend upon the chosen evaluation order. It is important to note that, in that case, all evaluation orders are sound, although they may give slightly different results.

Whenever a $+\infty$, $-\infty$, or *NaN* occurs during the linearisation procedure, the result cannot be used safely. In this case we say that the linearisation *fails*, and we need to fall back to another way of abstracting the floating-point expression, such as the non-relational interval abstraction $[\![expr_{\mathbf{f}}]\!]^{Int}_{fl}$, for instance. Note that this does not mean that the expression must or can trigger a run-time error in the concrete world, or that $[\![expr_{\mathbf{f}}]\!]^{Int}_{fl}$ will return the top element $[-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}]$, but only that using imprecise floating-point numbers in the abstract prevented us from computing a meaningful linearised result. In practice, this seldom occurs, as the experience with the ASTRÉE analyser described of Chap. 8 taught us.

Applications. As our floating-point linearisation abstracts floating-point expressions as interval linear forms on the real field, linearised expressions can be soundly fed to any numerical abstract domain that abstracts real numbers. This includes the original interval abstract domain of Sect. 2.4.6 and its floating-point implementation $[\![expr_{\mathbf{f}}]\!]^{Int}_{fl}$ presented in the previous section. Note that, to be sound, the interval domain without linearisation requires that each abstract operator is evaluated using a floating-point format equally or less precise than the concrete one. We do not have this limitation when using the interval domain with linearisation as the linearisation already includes the rounding phase and we are left to abstract a semantics on real intervals. Thanks to its simplification features, the linearisation may improve the precision of the interval domain; however, in some cases, the non-deterministic treatment of rounding in the linearisation gives less precise results than a plain interval evaluation $[\![expr_{\mathbf{f}}]\!]^{Int}_{fl}$ which rounds each interval bound tightly. For instance, the expression $[0,1] \oplus_{\mathbf{32},n} [0,1]$ is linearised as $[-2^{-148}, 2 + 2^{-22} + 2^{-148}]$ while $[\![[0,1] \oplus_{\mathbf{32},n} [0,1]]\!]^{Int}_{fl} = [0,2]$. Thus, it is advisable to compute both the linearised and non-linearised semantics, and take their intersection.

7.5.3 Floating-Point Zones and Octagons

The last pending step in order to obtain an analyser fully implemented using floating-point numbers and that discovers invariants on the floating-point variables of a program is to

adapt the zone and octagon domains so that they use floating-point arithmetics internally instead of perfect mathematical reals.

Adapting the Representation. Let us choose a floating-point format $\mathbf{f}$ on the analyser platform. Potential sets, zones, and octagons will be represented by DBMs with coefficients in $\mathbb{F}_{\mathbf{f}} \cup \{+\infty\}$. We benefit here from the fact that the IEEE 754 norm includes a representation for $+\infty$. All concretisations γ^{Pot}, γ^{Zone}, and γ^{Oct} are left unchanged: we still abstract sets of real environments — even though they abstract themselves sets of floating-point environments.

Adapting the Operators. Recall that all our algorithms on potential sets, zones, and octagons are defined using only the $\leq$, $+$, and divide-by-two mathematical operators. $\leq$ can be implemented exactly in $\mathbb{F}_{\mathbf{f}} \cup \{+\infty\}$. When designing all our DBM operators in Chaps. 3 and 4 we took care to use only expressions on $\bar{\mathbb{I}}$ representing *upper bounds* so that they could be safely over-approximated. Thus, it is sufficient to perform all $+$ and $/2$ computations with rounding towards $+\infty$ in the floating-point world to be sound. Unlike what happened for saturated integer arithmetics — see Sect. 7.3.2 — the special semantics of $+\infty$ is directly handled by the low-level floating-point operators.[3]

Precision Loss. Rounding in the abstract incurs some precision loss with respect to a perfect real implementation. An important consequence is that the closure algorithms do not compute normal forms any more, and so, our equality and inclusion tests become incomplete — they can return either a definitive "yes" or "I don't know" — and many operators are no longer best or exact abstractions. Nevertheless, this imperfect closure still propagates invariants in a useful way and is essential for a good analysis precision. The floating-point format used in the analyser can be chosen independently from that of the analysed program, but the more precise this format is, the less precision degradation occurs — unlike what happened for the interval domain adaptation, we can choose a more precise format than that of the analysed program. Finally, note that, as in the machine-integer case, using floating-point numbers in the abstract domain is orthogonal to abstracting floating-point computation: the adapted zone and octagon domains presented here are suitable to abstract Simple programs with $\mathbb{I} = \mathbb{R}$ as well as floating-point programs. It is even possible to use a floating-point implementation to abstract machine-integer programs using the semantics of Sect. 7.2: the machine-integer semantics is abstracted using perfect integers, which are abstracted using perfect reals, which are finally abstracted using a floating-point implementation of an abstract domain for real numbers. One benefit of this is the ability

[3] Ironically, such an abstract domain implementation cannot be analysed using the restriction of the IEEE 754 semantics proposed in this chapter as we make an explicit use of $+\infty$ instead of considering each occurring $+\infty$ as a run-time error.

to design a unified abstraction for expressions that mix integer and floating-point operators — as many real-life programs do — at the expense of a little extra loss of precision when rounding over-approximates integers that cannot be exactly represented as floating-point numbers but can be represented using saturated arithmetics.

Adapting Other Domains. Our DBM-based numerical domains were straightforward to implement using floating-point numbers because we only manipulate upper bounds. It is likely that the *octahedra abstract domain*, proposed in [CC04] by Clarisó and Cortadella to manipulate conjunctions of constraints of the form $\sum_i \epsilon_i X_i \leq c$, $\epsilon_i \in \{-1, 0, 1\}$, can also be adapted by simply choosing the right rounding direction. Other examples of relational numerical abstract domains abstracting invariants on reals but implemented using floating-point numbers are proposed by Feret, in [Fer04b], to analyse *digital filters*. As the octagon abstract domain, the filter domains were implemented within the Astrée static analyser on top of the adapted linearisation procedure of Sect. 7.5.2. However, other relational domains may not be as easy to adapt. If we consider, for instance, the polyhedron abstract domain, it is quite difficult to determine, for each arithmetic operation used internally, whether we should round its result towards $+\infty$ or towards $-\infty$ to achieve soundness.

7.5.4 Convergence Acceleration

As all our interval bounds and DBM coefficients live in a floating-point set $\mathbf{F_f}$ that is finite, all our abstract domains have a finite height and no widening nor narrowing is required to ensure the termination of the analysis, in theory. However, this height is quite huge, and thus, it is necessary to design widenings and narrowings to make our loop invariants stabilise within decent time. All the widenings and narrowings on $\mathbb{I} = \mathbb{R}$ — such as those presented in Sect. 2.4.6, Defs. 3.7.1, 3.7.2, 3.7.3, and 4.4.8 — can be soundly used within floating-point implementations of numerical abstract domains, but they can do with a little tweaking. Indeed, convergence acceleration operators are generally designed with a specific sub-class of loop invariants to be discovered in mind. If a widening was tailored for a specific invariant on the perfect real semantics, then the analyser may miss it because:

- due to rounding in the floating-point concrete semantics, the invariant on reals may not be true on the program, or,
- the invariant may be stable under the concrete floating-point loop transfer function F, but not under the computed abstraction $F^\sharp$ as it incurs some extra rounding with respect to F.

Perturbing Increasing Iterations. Given a widening $\nabla^\sharp$ tailored for $\mathbb{I} = \mathbb{R}$, an idea proposed by Feret[4] is to systematically derive a widening $\nabla^\sharp_\varepsilon$ by *enlarging* the result a little bit in the direction of unstable constraints. An example definition for the derived widening ∇^{Int}_ε on the interval domain is:

Definition 7.5.1. Interval widening with perturbation ∇^{Int}_ε.

$$[a,b] \nabla^{Int}_\varepsilon [a',b'] \stackrel{\text{def}}{=} \left[\begin{cases} a'' & \text{if } a'' = a \\ a'' - \varepsilon|a''| & \text{otherwise} \end{cases} , \begin{cases} b'' & \text{if } b'' = b \\ b'' + \varepsilon|b''| & \text{otherwise} \end{cases} \right]$$

where $[a'',b''] \stackrel{\text{def}}{=} [a,b] \nabla^{Int} [a',b']$.

●

where ε is a user-chosen constant corresponding to an amount of *relative* error perturbation. Stable constraints are left untouched.

Similarly, we can define an adapted widening $\nabla^{\mathsf{DBM}}_\varepsilon$ on DBMs by point-wise perturbation of unstable constraints. However, because of the closure algorithm, a floating-point implementation incurs some rounding on each DBM coefficient that depends upon *all* other matrix coefficients. Thus, it seems more reasonable to perturb each unstable coefficient by an amount proportional to the *largest* finite coefficient in the DBM. Such a derived widening $\nabla^{\mathsf{DBM}}_\varepsilon$ on DBMs can be defined as:

Definition 7.5.2. DBM widening with perturbation $\nabla^{\mathsf{DBM}}_\varepsilon$.

$$(\mathbf{m} \nabla^{\mathsf{DBM}}_\varepsilon \mathbf{n})_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{o}_{ij} & \text{if } \mathbf{o}_{ij} = \mathbf{m}_{ij} \\ \mathbf{o}_{ij} + a \times \varepsilon & \text{otherwise} \end{cases}$$

where $\mathbf{o} \stackrel{\text{def}}{=} \mathbf{m} \nabla^{\mathsf{DBM}} \mathbf{n}$,
and $a \stackrel{\text{def}}{=} \max_{i,j} \{ |\mathbf{o}_{ij}| \mid \mathbf{o}_{ij} \neq +\infty \}$.

●

Chap. 8 will give some experimental evidence that the widening with thresholds ∇^{Oct}_{th} of Def. 4.4.8 greatly benefits from the $\varepsilon-$perturbation by reducing the number of iterations required to stabilise our abstract invariants.

Perturbing Decreasing Iterations. By construction, the limit of increasing iterations with widening $X^\sharp_{n+1} = X^\sharp_n \nabla^\sharp F^\sharp(X^\sharp_n)$ is an abstract post-fixpoint. However, when performing decreasing iterations with narrowing from this limit, $Y^\sharp_{n+1} = Y^\sharp_n \vartriangle^\sharp F^\sharp(Y^\sharp_n)$, we get a sound approximation of the concrete fixpoint which may not be an abstract post-fixpoint. Yet, this would be quite a desirable property as it allows the invariant discovered by the analyser

[4]Private communication during the Astrée project. See also [BCC+03, § 7.1.4].

to be checked *a posteriori* for consistency instead of relying on the correctness of the fixpoint engine which may be buggy. The invariant abstraction does not need to be an abstract post-fixpoint to be sound but it is definitely sound if it is an abstract post-fixpoint. Our experiments showed that, often, the iterations with widening and narrowing converge towards an abstract post-fixpoint when using perfect reals, while the corresponding floating-point implementation does not because of rounding. Between its definite pre-fixpoints and its definitive post-fixpoints, there exists a chaotic region for a floating-point operator where a slight adjustment to its argument may result in its result being either larger or smaller, in a way that is hard to predict — especially when the abstract computation involves complex steps, such as transitive closure applications. We need a way to ensure that the narrowing will keep our iterates in the stable region and not drive them downwards too much. A solution is, as for widenings, to derive from $\triangle^\sharp$ a new narrowing $\triangle^\sharp_\varepsilon$ that enlarges refined constraints in $X^\sharp \triangle^\sharp Y^\sharp$ by a relative amount ε. The definition of $\triangle^{Int}_\varepsilon$ and $\triangle^{\mathsf{DBM}}_\varepsilon$ are similar to that of Defs. 7.5.1 and 7.5.1 except that $\nabla^\sharp$ is replaced with $\triangle^\sharp$. Experimental evidence of this approach will be presented in Chap. 8.

Choosing ε. In theory, a good ε should be estimated by the relative amount of rounding performed in the computation of the abstract function we wish to stabilise. Thus, it is a parameter of both the complexity of the analysed program loop and of the actual abstract domain implementation, including which floating-point format $\mathbf{f}$ is chosen to implement the analyser. In practice, experiments with the Astrée analyser — reported in the next chapter — showed that a good ε could be chosen by trial and error for a particular program to be analysed and, then, seldom had to be adjusted when we modified our analyser, the analysed program, or even analysed a different program written using the same coding practices.

7.6 Conclusion

We have extended, in this chapter, our analysis to programs that use machine-integers with limited bit-size and floating-point numbers following the IEEE 754 norm: each operation can produce an overflow and each floating-point operation induces some rounding. We focused on programs that do not perform integer overflows on purpose as we are sound but quite conservative in this event. We also focused on programs that do not compute using the special floating-point values $+\infty$, $-\infty$, and NaN but consider their generation as a run-time error, which is the case for the vast majority of numerical programs. We showed how we can infer invariants that are sound with respect to these real-life machine-integer and floating-point semantics, but also use our analysis to possible detect run-time errors.

Our main technique consisted in abstracting these complex and irregular semantics into perfect integer and real arithmetics so that all the numerical abstract domains as well as

the linearisation and symbolic constant propagation techniques, presented in the previous chapters, could be used soundly. In particular, a modified linearisation procedure was used to abstract the highly non-linear behavior of floating-point rounding using interval linear computations. Thus, this semantics is particularly well-adapted towards abstraction using the zone and octagon domains and, to a lesser extent, the polyhedron domain.

We also explained how to adapt the algorithms in our zone and octagon abstract domains so that they use machine-integers and floating-point numbers internally, for a reasonable precision loss. These two aspects can be combined together — in the sense of the combination of sound abstractions — to implement an efficient analyser for programs written in real-life programming languages. These techniques were indeed implemented and tested within the Astrée analyser presented in the following chapter.

Chapter 8

Application to the Astrée Static Analyser

Nous présentons maintenant l'analyseur statique de qualité industrielle Astrée *développé à l'ÉNS et l'École Polytechnique pour la vérification statique d'erreurs à l'exécution de programmes embarqués réactifs critiques. Cet analyseur met en pratique le domaine abstrait des octogones, les techniques de linéarisation et de propagation symbolique des constantes, ainsi que les méthodes d'abstraction d'entiers machines et de flottants. Nous pourrons ainsi démontrer l'utilité pratique des résultats théoriques de cette thèse.*

We present in this chapter the Astrée *industrial-quality static analyser developed at the ÉNS and the École Polytechnique, aimed at statically checking for the absence of run-time errors in critical embedded reactive programs. This analyser uses the octagon abstract domain, the linearisation and symbolic constant propagation techniques, as well as our framework for abstracting machine-integers and floating-point numbers. We will thus prove the practical interest of the theoretical work presented in this thesis.*

8.1 Introduction

Along with the theoretical work presented in the previous chapters, we participated in the design and implementation of the Astrée static analyser. In particular, the octagon abstract domain of Chap. 4, the linearisation and symbolic constant propagation techniques of Chap. 6, and the framework for the relational analysis of machine-integer and floating-point semantics of Chap. 7 were incorporated into Astrée. This provided us with the unique

opportunity to validate the developed techniques in an analyser for real-life industrial programs of several hundred thousand lines. We must also say that the work on ASTRÉE influenced our thesis for it provided new challenges, such as the scalability of the octagon domain or the relational analysis of floating-point variables, that were not originally our primary concerns. Several of the program examples presented in the preceding chapters, as well as the way we analyse them, crystallised from difficult points in our experimentation with ASTRÉE.

ASTRÉE is a large software: several people and many different techniques were involved in its design. It is not our goal to present ASTRÉE in details — the reader is refereed to co-authored papers [BCC+02, BCC+03] — but its overall design principles will be sketched in the first two sections to provide the reader with some context. Then, we will present, in more details, the integration in ASTRÉE of the techniques developed in the preceding chapters and provide experimental results.

8.2 Presentation of ASTRÉE

The ASTRÉE static analyser [Asta] is a join work by the Abstract Interpretation teams at the École Normale Supérieure (ÉNS), in Paris, and the École Polytechnique, in Palaiseau. Team members comprise Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, David Monniaux, Xavier Rival, and myself. It has been supported by a French exploratory RNTL — *Réseau National de recherche d'innovation en Technologies Logicielles* — project, also named ASTRÉE [Astb]. This project involves the two cited academic partners (ÉNS and École Polytechnique) as well the industrial partner Airbus [Air].

8.2.1 Scope of the Analyser

ASTRÉE is an efficient static analyser focusing on the detection of run-time errors in programs written in a subset of the `C` programming language. Due to abstractions, the issued warnings may be either real bugs or spurious behaviors called "false alarms". The goal of ASTRÉE is to *prove* the absence of run-time errors meaning that, when analysing a correct program, the analyser should issue very few or no alarm at all, at least when the program belongs to a certain program family. Up to a dozen alarms can be inspected by hand while a so-called *selectivity* of even 99% may require thousands of manual inspections which is far too prohibitive. It is aimed towards automated code *certification* unlike other analysers that only focus on finding as many bugs as possible but may miss some. It should also be fast enough to be usable during the program development so that bugs and false alarms are removed as soon as possible and the learning curve for the programmers and the testers is not too steep. To achieve these goals, the analysis is specialised for an infinite family

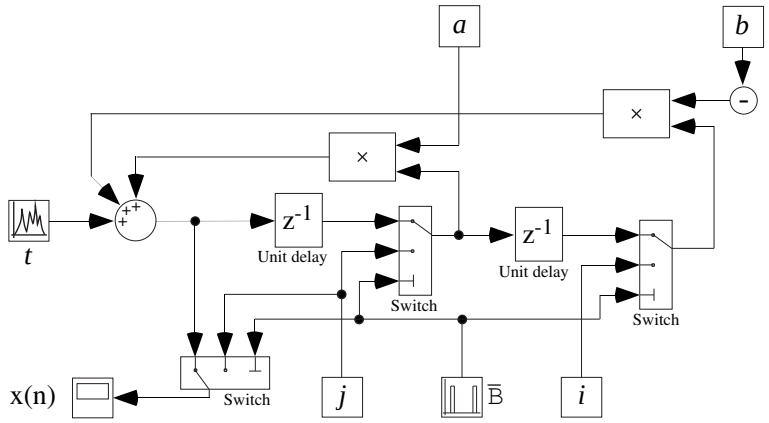


Figure 8.1: Synchronous block-diagram example. It corresponds to a simplified second order digital filtering system.

of programs constructed using roughly the same coding practices. It is expected that the analyser should adapt to other programs within the considered family — including new or corrected versions of already analysed programs — without much intervention, and those can be carried out directly by end-users.

Considered Program Family. The considered subset of the `C` programming language for the current version of ASTRÉE excludes recursion, union data-types, dynamic memory allocations, calls to the C library, and multi-threaded programs; however, it includes arrays, loops, and floating-point computations. Run-time errors include integer and floating-point divisions by zero, integer overflows, generations of floating-point $+\infty$, $-\infty$, or NaN, and out of bound array accesses. The initially targeted program family — for which the analyser should issue no false alarm — is the set of `C` programs of a *few hundred thousand lines* automatically generated, using several proprietary tools, from high-level specifications in the form of synchronous data-flow block-diagrams, such as the one presented in Fig. 8.1. This scenario is quite common in the industrial world of real-time safety-critical control systems, ranging from letter-sorting machines to nuclear plants and "fly-by-wire" aeronautic systems. As most run-time errors do not appear at the specification level, which considers perfect integer and real semantics, it is important that we check the generated `C` code according to the precise `C` language and IEEE 754 floating-point semantics. The considered program family has some features that make the analysis quite difficult:

- There is a very large loop that runs for a long time (3.6×10^6 iterations) and executes most of the program instructions at each iteration (some parts get executed up to

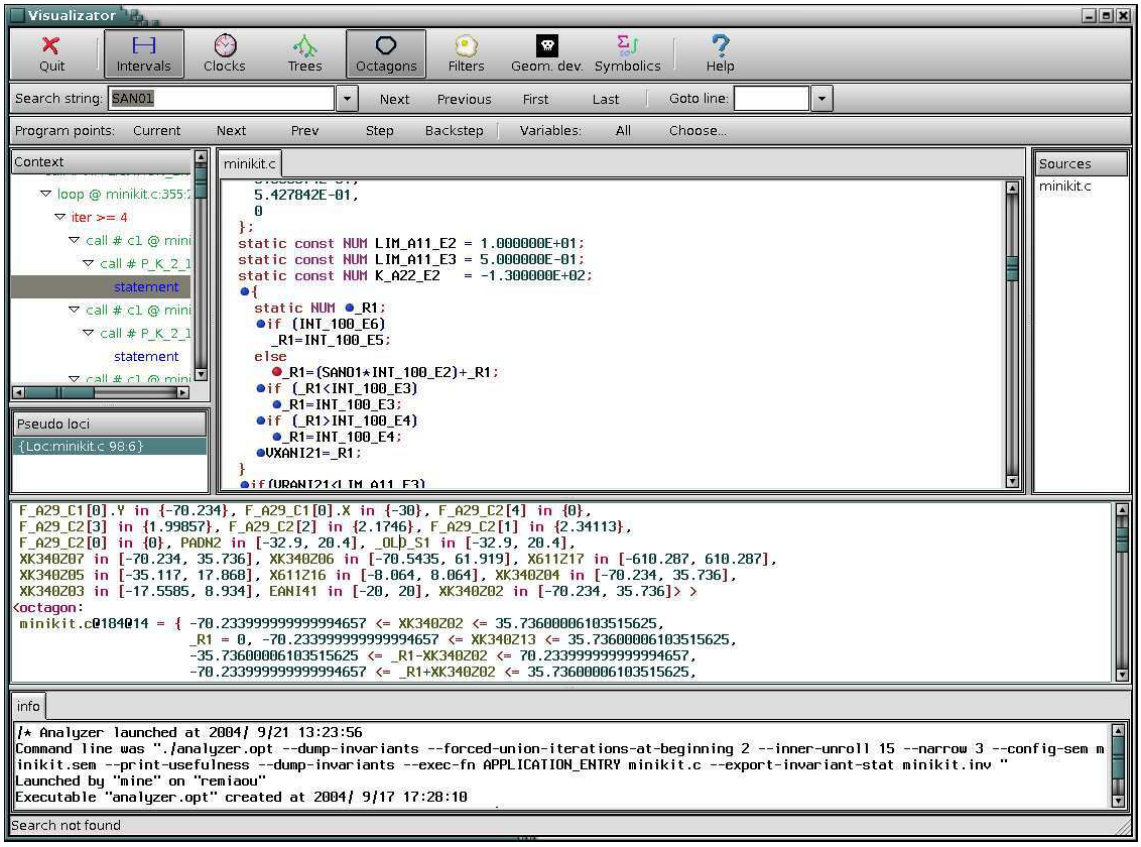


Figure 8.2: Graphical user interface for ASTRÉE.

twelve times per iteration). We will call this loop the *main loop* to distinguish it from inner loops that only execute few instructions and for a small number of iterations.

- There is a very large number of global variables (2 000 variables for each 10 000 lines of code) representing a current state. They are read and written a least once per iteration.

- The control part of the program is encoded in boolean variables.

- A large part of the program is devoted to floating-point computations — half the global variables are floating-point.

- The computation paths from an input value with known bounds can very long and may spread across many loop iterations, and so, rounding-errors cumulate easily.

8.2.2 History of the Analyser

The Astrée analyser started from scratch in November 2001. In November 2003, it was able to prove completely automatically the absence of run-time errors in the primary flight control software for the Airbus A340 fly-by-wire system. Astrée achieved this result in less than one hour of computation time on a PC-class computer, using only a few hundred megabytes of memory. Since January 2004, it is used to analyse the successive development versions of the electric flight control code for the Airbus A380.

We have been involved in the design and programming of the Astrée analyser since its beginning as it provided a unique opportunity of experimenting the theoretical tools developed during our thesis. False alarms due to the lack of relationally of the interval abstract domain motivated the design of our adaptation of relational domains to machine-integer and floating-point numbers, presented in Chap. 7. It enabled us to incorporate into Astrée the octagon abstract domain of Chap. 4, using our general-purpose octagon library implementation [Mina] developed earlier during our thesis. Likewise, several techniques of general interest presented in this thesis, such as the linearisation and the symbolic constant propagation of Chap. 6, but also some of the transfer functions and extrapolation operators on octagons, were designed during the refinement process of Astrée. Additionally to the specific work on numerical abstract domains, we participated in the general programming of Astrée to some extent and developed entirely some other features — such as the graphical interface to display the results of an analysis in a user-friendly way, shown in Fig. 8.2. The evolution of Astrée, its techniques and results, has been the subject of several papers [BCC+02, BCC+03, Mau04], some of which we co-authored by the whole Astrée team.

Perspectives. Currently, more and more programs in our original program family are checked using Astrée. Also, Astrée is being extended along several orthogonal directions. First, we plan to consider programs in the same family but written in other programming languages. Many abstract domains specialised towards control systems can be reused directly, but some more domains may be needed as abstractions are quite sensitive to language idioms. Then, we plan to consider more complex, user-defined, properties to be checked additionally to run-time errors. Finally, several new program families are being considered: the *auto-test* program family that checks the hardware for faults when the embedded system is switched on, as well as the *initialisation* program family which runs just after. Some program families comprising drivers, such as an USB controller, will require us to design new memory models and subsequent abstractions.

8.2.3 Design by Refinement

The idea of a static analyser specialised for a family of programs is the natural compromise between two extremes. On the one hand, given one specific program and one property

to check, it is always possible to design a static analyser that discovers the property fully automatically. On the other hand, it is undecidable to check this property for all programs, by Rice's theorem [Ric53]. We now present how the idea a static analyser specialised for a family of programs has been made practical in the ASTRÉE project using a *design by refinement* principle.

Development Cycle. The development cycle for our analyser is as follows. We start with a simple and imprecise but fast static analyser — based on the plain interval abstract domain, for instance — and a modest-sized program which has been used for a long time without any run-time error — a fragment of the A340 flight control system, in our case. Then, the analyser is refined step by step: each step begins with a static analysis yielding false alarms; then, we manually trace backwards some alarms towards the origin of the analysis imprecision; finally, the imprecision is corrected by either incorporating a new abstract domain or refining a transfer function, a linearisation or constant propagation strategy, or a reduction between existing abstract domains. When there is no false alarm left in our simple program, we continue this process using more and more complex programs in our family. After a while, most semantic features in our program family which are important to prove the absence of run-time errors have been abstracted sufficiently precisely and new programs become easier and easier to analyse, that is, require less and less refinement. We have obtained an analyser targeted towards a whole family of programs instead of a single one. As we started from a simple and fast analyser and only refined it to remove false alarms, the refined analyser is neither very large nor very costly. It is important to note than, when designing ASTRÉE, no development time was wasted on implementing transfer functions or reductions that would improve globally the precision but are not useful to remove false alarms.

Parametrisation. A desirable feature for an analyser designed for a whole family of programs is the ability to tune the analysis specifically for one program or another easily without requiring a new refinement pass. In ASTRÉE, this is done using *parameters* that can be specified in the command-line or as analyser-specific commands that can be either integrated in the `C` code or provided as an external file. A first parameter is the set of abstract domains that will be enabled among the many ones present in the analyser. Then, several parameters can tune the extrapolation strategy; for instance, it is possible to change the set of thresholds used in a widening with thresholds — see Def. 3.7.2 — or the perturbation ϵ in the perturbed widenings and narrowings — see Defs. 7.5.1 and 7.5.2. Finally, our last parametrisation is a more *local* one: experience shows that extra precision — such as relational invariants — is only required for small localised parts of the program and it would be a huge waste of resource to use this extra precision for the whole program. Thus, it is possible to *tag* program parts or variables that need to be analysed using our more

costly domains. Following our design by refinement principle, after a few experiments with manual insertions of such tags, we designed strategies to insert them automatically using simple syntactic rules. We call this technique *automatic parameterisation*. An important parameter tag example is the *octagon packing* that will be presented in Sect. 8.4.2 and allows relating only few variables at a time together in an octagon abstract element and achieve an effective linear cost in the number of variables, instead of a cubic one.

8.3 Brief Description

We give here a short description of the Astrée analyser. Our goal is to give some context for the following sections that will present in more details the analyser parts that are most relevant to this thesis. The interested reader is refereed to [BCC+03] for a more comprehensive discussion about other components of Astrée that can only be briefly mentioned here. We do not distinguish, only for this section, our work and the work of other members in the Astrée team; our work will be detailed in the following section.

8.3.1 Implementation

The Astrée analyser comprises approximately 50 000 lines of OCaml code. To this, we must add approximately 9 000 lines of `C` code for the octagon abstract domain library [Mina] used by Astrée. The graphical user interface is written using the Gtk+ 2 toolkit [Prob] and its OCaml interface LablGTK 2 [Ga]. The code is fully portable and has been tested on the following 32-bit and 64-bit platforms:

- UltraSparc Sun stations running Solaris (32-bit mode);
- Pentium-class computers running Linux, FreeBSD, and Windows;
- AMD-64 computers running Linux (64-bit mode);
- PowerPC G4 Apple computers running MacOS and Linux.

8.3.2 Analysis Steps

Astrée accepts `C` programs as input and performs the following successive very fast *pre-processing* passes:

- each `C` source is preprocessed using the `cpp` standard `C` preprocessor from the freely available Gnu `gcc` compiler suite [Proa];
- each `C` source is parsed using D. Monniaux's freely available C99-compatible `C` parser for OCaml [Mona];

- all the `C` syntax trees are merged into a single tree by an internal linker;
- the syntax tree is then compiled into an intermediate representation corresponding to a simplified `C` language where all type conversions have been made explicit;
- constant expressions are detected using a simple intra-procedural analysis *à la Kildall* [Kil73] and evaluated once and for all;
- dead statements and variables — mostly introduced by the previous pass — are pruned;
- the resulting syntax tree is scanned for parts that may require more precise and costly abstract domains or reductions, using techniques such as the octagon packing described in Sect. 8.4.2, and the corresponding parameter tags are inserted.

Then, the proper analysis is performed on the tagged internal representation. Astrée performs an *abstract execution* that follows the control flow of an actual execution but uses abstract transfer functions in the enabled abstract domains as well as widenings and narrowings for loops. This can be seen as a special instance of the chaotic iterations, described in Sect. 2.2.5, allowing much memory optimisation. It would be impossible, for large programs, to maintain in memory a map associating an abstract environment to each program point but, for the iteration order induced by our abstract execution, it is only necessary to keep one abstract environment for each level of nested loop and `if` statement enclosing the current statement; unlike the code size, the maximum amount of nesting is generally very small. This also allows programming the iterator as a simple recursive function on the syntax tree, in a way similar to *denotational semantics*. Finally, function calls are in-lined, which is possible because there is no recursive functions. It gives a fully context-sensitive inter-procedural analysis: we compute, at each program point, a distinct abstract value for each sequence of call-sites leading to this point.

After loop invariants are stabilised thanks to iterations with widening and narrowing, the program tree is traversed once more in a *checking pass* and a warning is output for each operation that may trigger a run-time error. It is also possible to dump invariants at all or selected program points — such as loop invariants — for further inspection using our graphical user interface — see Fig. 8.2. Presently, no backward analysis is performed by Astrée.

8.3.3 Abstract Domains

Several abstract domains are used during the analysis. They are written as modules using a common interface and can be easily plugged in and out of the analyser.

Memory Abstract Domain. The memory abstract domain takes care of pointers and aggregate data-structures in `C` expressions. Each abstract element contains a mapping from arrays and structures to a flat collection of scalar variables, as well as abstract points-to information. It is thus able to transform a complex `C` expression into a purely arithmetic expression manipulating only machine-integer and floating-point scalar variables. The memory abstract domain does not abstract such purely numerical environments by itself but relies on a numerical abstract domain or, more probably, the partially reduced product of several numerical abstract domains.

Interval Abstract Domains. The first and simplest numerical abstract domain used is the interval domain, introduced in [CC76] and recalled in Sect. 2.4.6. The interval domain is quite important for two reasons. Firstly, we use only variable bound information in the checking pass to prove that no run-time error can occur. Secondly, bound information is used when linearising expressions using the technique of Sect. 7.4.3 to enable the sound use of relational abstract domains designed for perfect real numbers as well as symbolic expression simplification. The interval domain is always enabled so that bound information is always available.

Clock Abstract Domain. A first kind of false alarms to be removed were due to counters that are incremented or decremented at most once per main loop iteration. Such counters are bounded because the main loop does not iterate forever. We saw, in Exs. 3.7.4 and 4.6.1, that such a proof is not in the scope of the interval abstract domain because we need to relate the modified variables to the loop counter in a relational invariant. Thus, we implemented in Astrée a simple relational domain, called the *clocked interval abstract domain*, that infers invariants of the form $V + \texttt{clock} \in [a, b]$ and $V - \texttt{clock} \in [a, b]$ for each program variable V, `clock` being a phantom variable incremented exactly once per main loop iteration. The clocked interval abstract domain can be seen as the octagon domain restricted to only two variables.

Linearisation. It soon became obvious that the remaining false alarms could only be removed by inferring more complex numerical invariants and, in particular, invariants on floating-point variables. In order to provide a "glue" between numerical `C` expressions, manipulating machine-integers and floating-point values, and some of our abstract domains designed for perfect reals numbers, we implemented the linearisation of Chaps. 6 and 7. Machine-integer and floating-point expressions are both abstracted in terms of interval linear forms with a real semantics but, internally, we compute all bounds using floating-point values rounded in a sound way. Expressions are linearised using the relative size strategy: when encountering a non-linear multiplication, the argument yielding to the smallest relative interval amplitude is intervalised, as described in Sect. 6.2.4. When encountering a

highly non-linear assignment, such as one using a bit-wise `C` operator, or in the event of a machine-integer overflow, we fall back to the interval abstract evaluation $[\![expr]\!]^{Int}$.

Octagon Abstract Domain. False alarms due to imprecise interval abstractions in program parts similar to the absolute value computation of Ex. 4.6.2 or the rate limiter of Ex. 4.6.3 motivated the inclusion of numerical abstract domains for linear inequalities. The most widespread domain is the polyhedron domain but we chose the octagon abstract domain — described in Chap. 4 — instead for two reasons. Firstly, its theoretical memory and time cost is much lighter than that of the polyhedron domain — respectively quadratic and cubic in the number of variables instead of exponential. In practice, the performance gain is even larger because the octagon domain can be implemented using floating-point numbers, as described in Sect. 7.5, instead of costly arbitrary precision rational packages. Secondly, it supports easily rather precise transfer functions involving interval linear forms that appear naturally when linearising floating-point expressions. Our octagon domain implementation is described in more details in Sect. 8.4.

Symbolic Constant Propagation. In some cases where it is necessary to retain relational information locally, the symbolic constant propagation of Sect. 6.3 can be combined with the linearisation and the interval domain to get the desired result. A false alarm example that could be removed this way was due to an interpolation computation similar to that of Ex. 6.3.1. Thus, we implemented a symbolic constant domain that can perform symbolic constant propagation on expressions. This propagation does not occur by default in all expressions as this may lead to precision degradation in some cases. Instead, it is provided as a service for those abstract domains that ask for it. In particular, the interval abstract domain has been modified to use the symbolic constant propagation, as explained in more details in Sect. 8.5.

Digital Filter Abstract Domains. As many real-time embedded control systems, the considered program family makes a great use of *digital filters*, that is, recurrence of the form $X_n = F(X_{n-1}, X_{n-2}, \ldots, Y_n, Y_{n-1}, \ldots)$ where, at each main loop iteration, a new value for X is computed using its values at preceding iterations, as well as the current and past values of some input Y. One example is the second order filter: $X_n = (\alpha \otimes_{\mathbf{32},n} X_{n-1}) \oplus_{\mathbf{32},n} (\beta \otimes_{\mathbf{32},n} X_{n-2}) \oplus_{\mathbf{32},n} Y_n$, which corresponds to the block diagram of Fig. 8.1. Proving that X is bounded is out of the scope of the interval and octagon domains as it requires inferring a so-called *ellipsoid* invariant which has the following form: $aX_n^2 + bX_{n-1}^2 + cX_nX_{n-1} \leq d$. An abstract domain for such invariants is described in [BCC+03]. Another example is the following first order filter: $X_n = (\alpha \otimes_{\mathbf{32},n} X_{n-1}) \oplus_{\mathbf{32},n} (Y_n \ominus_{\mathbf{32},n} Y_{n-1})$. In order to find a precise bound for X, it is important to note that, after development, the Y_i terms are almost cancelled. In [Fer04b], Feret proposes to seek history-sensitive invariants of

the form $|X_n| \leq a \max_{i \leq n} |Y_i| + b$. Special-purpose numerical abstract domains for such non-linear and history-sensitive invariants, as well as others spawning from Feret's generic framework for analysing digital filters, were implemented within ASTRÉE. As the octagon abstract domain, these domains naturally abstract real numbers and are adapted to the floating-point semantics using the floating-point linearisation of Sect. 7.4.3.

Arithmetic-Geometric Progression. One of the last false alarm to be removed in the analysis of the full A340 fly-by-wire system was due to a variable whose bounds grow exponentially with respect the main loop iteration, but with such a low exponent that it is in fact bounded. Removing this false alarm required the design by Feret and Mauborgne of an *arithmetic-geometric progression* abstract domain, able to discover invariants of the form $|X| \leq a^{\texttt{clock}} + b$. The interested reader can find the precise description of this domain in [Fer05]; let us simply say that this domain requires the help of both the linearisation and the symbolic constant propagation techniques of Chap. 6.

Information Flows. In order to achieve a precise analysis, our numerical abstract domains must exchange information. In ASTRÉE, the situation is a little more complex than in the partially reduced product of abstract domains presented in Sect. 2.2.6, for efficiency purposes. There exists two types of information flow: *read* and *reduce*. Using the *read* channel, an abstract domain can ask some information from the abstract environment, such as the interval bound or the symbolic constant associated to a variable; the requested information is gathered from all the abstract domains and combined by intersection. This is used, for instance, to make available the interval abstract evaluation of arbitrary expressions to the intervalisation procedure of Sect. 6.2.2. Using the *reduce* channel, an abstract domain can broadcast an invariant to all the abstract domains. This is used, for instance, when the octagon or a filter domain is able to infer a variable bound from its abstract state and wants the interval domain to take it into account. Reduction orders are not issued every time the abstract value in an abstract domain changes as it would be very costly and may prevent the iterate with widening to converge. Instead, a global strategy is used to ensure that the information required to remove our false alarms are propagated while not jeopardising iterate stabilisation by widening. It is not our purpose here to explain this strategy, so, we refer the interested reader to [Fer04b] which develops this topic in details for the case of the interval and filter domains.

8.3.4 Partitioning Techniques

Most of the numerical abstract domains used in ASTRÉE can abstract intersection exactly, but not union: these are *non-distributive* abstract domains. Unfortunately, abstract unions are very frequent as they appear when we merge control-flows after each conditional branch

and each loop iteration. Two partitioning techniques were developed in ASTRÉE to reduce this cause of imprecision without the need to design new abstract domains.

Value Partitioning. In our program family, a great part of the control is encoded in booleans. For instance, the result of a test on a numerical variable X is put into a boolean B and, a few lines — or thousand lines — later, some decision on X is taken depending on the value of B and other criteria. In order to keep the relationship between X and B, a *value partitioning* abstract domain functor based on *decision trees* was incorporated into ASTRÉE: it allows manipulating an abstract environment for each possible value of any set of integer variables. In particular, the partitioned domain is no longer convex and the union of two decision trees with distinct values for the partitioning variables is exact. As value partitioning incurs a cost that is exponential, at worse, in the number of partitioning variables, ASTRÉE uses a *packing* technique to relate the value of a few variables with respect to a few variables in a collection of small decision trees. ASTRÉE contains an automatic packing algorithm for decision tree similar to the octagon packing described in Sect. 8.4.2. Even though any abstract domain can be partitioned with respect to the value of a few boolean variables using this decision trees, we currently only partition the interval and symbolic constant abstract values.

Trace Partitioning. Trace partitioning is a technique introduced by Handjieva and Tzolovski in [HT98] to refine any abstract domain by distinguishing, for a program point, several abstract elements based on the history of the program points traversed to reach the current program point. More precisely, the authors remember the sequence of `if` branch taken and the truth of each loop condition evaluated; this sequence is called a *trace*. Handjieva and Tzolovski propose to use abstractions of sets of traces and present transfer functions and operators — including a widening operator — for the abstract domain obtained by lifting a base domain to functions associating an abstract element to each abstract trace, for a finite set of abstract traces. As this technique can be quite costly if applied as-is to large programs, it was modified as follows in ASTRÉE: firstly, we only take into account a few selected tests and loops; secondly, traces are merged at the end of each procedure using an abstract union. Thus, the effect of trace partitioning is to *delay* unions; this always results in some precision gain. We also introduced a new partitioning criterion to distinguish traces based on the value of a variable at a selected program point — this is much less costly than effective value partitioning that must track the value of the variable at all subsequent program points. Tests, loops, or variables can be selected for partitioning by manually inserting tags in the code; a fully automatic tagging strategy based on syntactic criteria has also been implemented in ASTRÉE.

Comparison with Disjunctive Completion. The partitioning techniques used in Astrée avoid the need for disjunctive completions, which was discussed in Sect. 3.4.3 for the case of the zone abstract domain. Partitioning seems a much better approach for two main reasons. Firstly, it does not require to change any data-structure or algorithm on the base domain. Secondly, it allows a fine tuning of the cost versus precision trade-off while the disjunctive completion can only be very precise and very costly. In the design by refinement of an efficient static analyser specialised for a program family, it is important to control precisely where we trade efficiency for precision. Our partitioning techniques naturally attach some extra precision to visible program variables or control-flow constructs.

8.4 Integrating the Octagon Abstract Domain

8.4.1 Implementation Choices

Our octagon domain implementation relies on the freely available octagon library [Mina] also developed during our thesis.

Underlying Numerical Set. The octagon library [Mina] allows representing invariants of the forms $\pm X \pm Y \leq c$ where c lives in a numerical set $\mathbb{I}$. Several sets $\mathbb{I}$ are proposed: integers or fractions, using either machine-integers with saturated arithmetics, as in Sect. 7.3.2, or arbitrary precision numbers, or floating-point numbers rounded soundly, as in Sect. 7.5. We chose, in Astrée, an implementation of octagons based on floating-point numbers which gives an extra performance boost with respect to known polyhedra implementations as they are all based on costly arbitrary precision rational packages. Internally, the octagon domain uniformly treats all variables as real-valued and we rely on the techniques introduced in Chap. 7 to soundly abstract machine-integer and floating-point expressions into expressions in the real field. In particular, we do not distinguish machine-integer and floating-point variables, but mix them in the same abstract environment. As said before, abstracting integers as reals causes some small loss of precision, but this fact is compensated by two important benefits: our ability to discover relationships between machine-integer and floating-point variables and the ease of implementation.

Transfer Functions. We use the assignment transfer functions of Def. 4.4.7 to precisely handle linearised assignments. Because most tests in our program family either involve octagonal constraints or are highly non-linear, we use the exact test abstractions of Def. 4.4.4 when possible and abstract tests as the identity otherwise.

Strong Closure. We perform the adapted Floyd–Warshall algorithm for the strong closure of Def. 4.3.2 on the arguments of all abstract unions and transfer functions. Even

though this does not result in strongly closed DBMs and optimal abstractions — because of floating-point rounding in the abstract — it still propagates sufficient relational information for our purpose: we are yet to find a false alarm that can be removed using an octagon implementation based on perfect reals but cannot be removed using our floating-point implementation. Following Sect. 3.8.1, we always use versions of transfer functions that preserve the closure, when they exist, and apply the incremental closure of Def. 4.3.4 to restore the strong closure after each transfer function that modifies its argument only locally; indeed, experiments showed that the result of a transfer function is likely to be fed to an operator or transfer function that requires strongly closed arguments for best precision.

8.4.2 Octagon Packing

Even though its cost is light compared to the polyhedron abstract domain, it would still be too costly to use the octagon domain to relate *all* our program variables in a large octagon as there are tens of thousands of them. We decided, instead, to break down our variable set into *packs* of a few couple variables, each pack corresponding to variables that should be related together.

Adapted Operators and Transfer Functions. Given a fixed packing, an abstract environment then associates an octagon of the correct dimension to each pack. Transfer functions and operators are applied point-wisely as in the plain domain product of Sect. 2.2.6: even though one program variable may belong to several packs, no reduction is used to transfer information from one pack to another. Inter-packing reductions could be used to gain precision by using common variables as pivots. However, we found simpler, when extra precision was needed to remove false alarms, to refine the automatic packing strategy instead. We now explain more precisely how assignments are handled:

- Assignments of expressions that cannot be made into interval linear forms (the ones containing operators that are too non-linear, such as the bit-wise and operator `&`, or that may trigger an overflow) use the interval-based abstract transfer functions — Def. 4.4.5.

- Otherwise, for each pack containing the assigned variable, we first construct an interval linear form containing only variables in this pack. This is done by replacing unwanted variables by their corresponding interval as fetched from the interval domain. Then, the interval linear form assignment of Def. 4.4.7 is used on the corresponding octagon. Finally, the interval for the assigned variable is extracted from each of the modified octagon. These intervals are intersected and used to refine the interval domain information for the assigned variable.

It can be possible to gain more precision by embedding all the octagons containing one variable appearing in the assignment into one large octagon, then applying the transfer function only once, and finally projecting back the large octagon on the smaller ones defined by the packing. As for inter-packing reductions, this technique has been rejected because it can construct intermediate octagons that are much bigger than that of the packing and can grow costly in an unpredictable way; we rely instead on a proper packing strategy. Tests are handled in a similar way.

The cost of the octagonal analysis depends on several parameters: the number of octagon packs, the size of octagon packs, but also the number of times the same variable appears in different octagons — this determines the number of octagons updated by each transfer function application.

Automatic Packing Technique. Which variables to pack together can be specified by hand in the analysed program. After a few manual experiments, we developed a packing algorithm to automate this task for our considered program family. This algorithm traverses the code after preprocessing, merging, constant propagation, and dead-code pruning, and associates a pack to each syntactic `C` block, that is, code sequence bracketed within `{` and `}` but also unbracketed if-then-else branches and loop bodies. In order to fill the pack for a given syntactic block with variables, we perform the following filtering steps:

1. We first gather all statements in that block excluding the statements in its sub-blocks.

2. From these statements, we only keep those that are reduced to a single `C` expression. This includes assignments but not `if` or `while` statements.

3. From each such expression, we extract the variables it uses but ignore a variable if there is little chance that the expression behaves linearly with respect to this variable. More precisely, we do not scan the arguments of a bit-wise `C` operator, a function call, an array lookup, or an "address-of" operator but we scan recursively both arguments of the `+`, `-`, and `*` arithmetic operators as well as the `&&` and `||` logical operators and all comparison operators; we also scan the left argument of a `/` operator.

4. For each expression, if the set of extracted variables contains at least *two* variables, we add all of them to the pack. If it contains only one variable, we do not add it. For instance, the assignment `X=Y+(Z&2)` will result in both `X` and `Y` being added to the current pack, `Z` being ignored as argument to a bit-wise operator; while `X=3` does not contribute any variable to the pack.

Additionally, steps 3 and 4 are executed on expressions appearing in an if-then-else condition but the extracted variables are added to *both* the block enclosing the `if` statement and the blocks in the then and else branches. Variables are also extracted from each loop

condition and added to both the block enclosing the loop and the loop body block. The effect of this filtering is to keep, for each assignment, only variables that have an opportunity to generate linear relational invariants. If we are to analyse the effect of a sequence of assignments and tests sharing common variables with the best precision possible, it is necessary to put *all* the variables of the involved expression in the *same* octagon pack as there is no information transfer between distinct packs. As packing all the extracted variables from all expressions in the same octagon would result in a huge octagon, we relate together only variables from expressions in the same syntactic block and from conditional expressions that relate the block to both its directly enclosing and nested blocks. This strategy can be extended by considering the expressions in nested sub-blocks up to some nesting limit; this would result in larger packs — but less of them. Additionally to variables extracted from expressions using steps 3 and 4, we add to the current octagon pack any variable that is either incremented or decremented, so that we are able to infer relationships between loop counters, as in Exs. 3.7.4 and 4.6.1. It is quite important, for our considered program family, not to rely on variable *declaration* but on variable *usage* to define packing; otherwise, this would result in all global variables being packed together in an octagon with thousand variables. Finally, a part of the analysed program family is generated from a different, newer, proprietary tool that declares a lot of local variables at the beginning of each procedure, even though each variable is used in only a few statements. For these programs, we use the same packing strategy except that we ignore the top-level syntactic block of each procedure to avoid all local variables being put in the octagon corresponding to the top-level pack.

We perform an optimisation step before passing the packing information to the static analyser: if the set of variables of a pack is included in the set of variables of a larger pack, then the smaller pack is discarded. We stress on the fact that, even though we rely on a *local* analysis of the syntax to determine which variables should be related together, the packing is considered *globally* by the subsequent static analysis: octagons are no longer attached to syntactic blocks and live throughout the abstract execution of the whole program.

A technique similar to this one, but more complex, is used to determine which variables to partition with respect to which variables in the abstract domain of decision trees. It may also be useful for other relational numerical domains such as, for instance, the polyhedron domain, to find a trade-off between cost and relationality.

Automatic Packing Example. Consider the following `C` implementation of the rate limiter of Ex. 4.6.3:

```
{ ①
  float Y,S,R;
  Y=0;
  while rand(0,1) { ②
```

```
    float X,D;
    X=rand(-128,128);
    D=rand(0,16);
    S=Y;
    R=X-S;
    Y=X;
    if (R<-D) { ③ Y=S-D; }
    if (R>D) { ④ Y=S+D; }
  }
}
```

We have four blocks, numbed ① to ④. Before the packing optimisation, ① is empty, ② contains the variables `S`, `Y`, `X`, `R`, and `D` while ③ and ④ contain the variables `S`, `R`, `Y`, and `D`. After optimisation, we have only one pack containing all variables, which permits a precise analysis. In a real-life example, the loop would contain many such rate limiter instances, each in a syntactic block such as ②, and so, we would have one octagon pack for each instance that would contain only the variables useful for the instance.

Useful Octagons. Our packing strategy has been designed so that the octagon analysis is sufficiently relational to remove selected false alarms; however, it may be too relational and some packs may be in fact useless. In order to check *a posteriori* the usefulness of each octagon pack, we implemented a simple monitoring technique in Astrée to count the number of times a variable bound was more precise in an octagon than in all other enabled abstract domains, that is, the octagonal information prevails by reduction. The set of octagons that were useful at least once is output with the result of the analysis, and so, it is possible to re-run a much faster analysis using only the octagons that were proved useful. We can also use this list of useful octagons to analyse a slightly modified program: it is likely that not many useless octagons become useful and, if some do, this results in a sound loss of precision. We experimented successfully with the following method: a long and full analysis is run at night and determines a set of useful octagons that will be used the following day for quick analyses during program development.

Another application of useful octagons is to enhance our iterations with widening strategy: indeed, when checking for invariant stability, it is only necessary consider the set of octagons useful *up to now*. This results in a large performance gain as we do not loose time stabilising octagonal invariants that are not actually used in the determination of variable bounds.

Packing Statistics. We now present, for a few programs in the considered family, statistics on the automatically generated packing. Our family has been split into two sub-families:

the three lower, more recent, `C` programs are generated using a different proprietary tool. The code size is computed as the number of indented lines of `C` code after merging all the preprocessed sources together — eliminating all useless or redundant declarations in headers — but before constant expression simplification and dead code elimination. Together with the number of variables, the number of octagons, and the average number of variables per octagon, we give the number of octagons that were proved useful at least once *a posteriori*. As the memory consumption and the time cost depend respectively on n^2 and n^3, we show not only the average number of variables, but also the square — resp. cubic — root of the average of the squared — resp. cubed — sizes.

code size in lines	number of variables	number of packs	average size	$\sqrt{\sum \text{size}^2}$	$\sqrt[3]{\sum \text{size}^3}$	useful percentage
370	100	20	3.6	4.8	6.2	85 %
9 500	1 400	200	3.1	4.6	6.6	41 %
70 000	14 000	2 470	3.5	5.2	7.8	57 %
70 000	16 000	2 546	2.9	3.4	4.4	41 %
226 000	47 500	7 429	3.5	4.5	5.8	52 %
400 000	82 000	12 964	3.3	4.1	5.3	50 %

This table shows that the average size of packs is almost constant while the number of packs grows roughly linearly with the code size. This means that the octagon domain with an adequate packing strategy has a time and memory cost that is *linear* with respect to the program size. Two other interesting information not presented in this table are that the largest packs contain only up to a *few dozen* variables and that a variable that is present in one octagon is present in fact in *two* distinct octagons in the average.

8.4.3 Analysis Results

We now compare the results of ASTRÉE on the considered program family with and without the octagon domain, all other abstract domains being enabled. We give, in both cases, the analysis time, the maximum memory consumption, and the number of false alarms. All the analyses have been carried on an 64-bit AMD Opteron 248 (2 GHz) workstation running Linux, using a single processor. We observed that, on a 32-bit architecture, the memory consumption is roughly one third smaller, which is explained by the large usage of pointers in OCAML data-structures.

	without octagon			with octagons		
code size in lines	analysis time	max. memory	false alarms	analysis time	max. memory	false alarms
370	1.7s	14 MB	0	3.1s	16 MB	0
9 500	75s	75 MB	8	160s	80 MB	8
70 000	3h 17mn	537 MB	58	1h 16mn	582 MB	0
70 000	18mn	289 MB	4	30mn	378 MB	4
226 000	7h 8mn	1.0 GB	165	6h 36mn	1.3 GB	1
400 000	20h 31mn	1.7 GB	804	13h 52mn	2.2 GB	0

This table shows that the octagon domain is useful to reduce the number of false alarms to only a few, and even to zero in some cases. Moreover, enabling the octagon domain adds roughly 30% to the total memory consumption in the worst cases, which is very reasonable considering the precision gain. The analysis time does not seem to follow a logical pattern: sometimes the analysis is longer with the octagon domain, which seems quite natural, but sometimes it is faster. In order to explain this fact, we need to take into account the number of iterations with widening and narrowing of the main loop that are needed to stabilise our invariants. This is presented in the following table:

	without octagon		with octagons	
code size in lines	number of iterations	time per iteration	number of iterations	time per iteration
370	12	0.1s	17	0.18s
9 500	23	3.2s	39	4.1s
70 000	159	74s	44	104s
70 000	36	30s	38	49s
226 000	144	178s	86	276s
400 000	172	429s	96	520s

We now see clearly that the octagon domain makes each abstract iteration up to 65% slower, which is due to the extra time spent in octagon transfer functions and operators. The octagon domain also affects the number of required iterations, but in a non-easily predictable way: sometimes, more iterations are required because we are trying to stabilise a greater amount of invariants; sometimes, the octagon information can prove the stability of some variable bound and save widening steps in the interval domain. Sometimes, the gain in iteration numbers is sufficient to reduce the total analysis time even though each iteration takes more time. This shows that using an abstract domain adapted to the program invariants can increase both the precision and the efficiency of a static analysis at the same time.

Octagon Precision. It is quite interesting to note that, even though we added the octagon domain to remove false alarms in codes such as the rate limiter of Ex. 4.6.3, several other false alarms in unrelated code parts were removed altogether. We did not have time, yet, to examine all these code parts and determine why the octagon domain successfully analyses them, but we believed this may have saved us from designing several special-purpose abstract domains. As an example, it came quite as a surprise that the octagon domain could handle absolute value computations in the spirit of Ex. 4.6.2 and this saved us from designing an abstract domain targeted towards capturing non-linear invariants of the form $X = |Y|$. Unlike the filters [Fer04b] and the arithmetic-geometric progression [Fer05] domains that are targeted towards very specific kind of invariants in Astrée, the octagon domain seems to be of general use.

Octagon Cost. In order to determine more precisely which parts of the octagon domain are responsible for the increased computation time per iteration, we performed a few analyses using profiling. Unsurprisingly, we spend most of the analysis time closing our matrices: over 6% of the total analysis time is spent in the incremental strong closure. There is only one function in which the analyser spends more time: the mark phase of OCaml's garbage collector — 10% of the total analysis time. Also, the octagon algorithm coming right after the incremental strong closure is the forget operator, and this accounts for only 0.35% of the total analysis time. The non-incremental version of the strong closure corresponds to a negligible fraction of the analysis time because it is seldom called; we prefer to use the much faster incremental closure whenever possible.

8.5 Integrating the Symbolic Constant Propagation

As explained in Chap. 6, the linearisation and symbolic constant propagation techniques can sometimes increase the precision of the interval domain.

8.5.1 Implementation Choices

In Astrée, the interval abstract domain is refined by computing, for each transfer function, the intersection of the following *three* information:

- the plain non-relational transfer functions, as described in Sect. 2.4.4;
- the non-relational transfer functions after the linearisation of expressions, following Sects. 6.2.3 and 7.4.3;
- the non-relational transfer functions after the symbolic constant propagation and the linearisation, following Sect. 6.3.

As for the octagon domain, we use the relative size strategy of Sect. 6.2.4 when linearising expressions. For the symbolic constant propagation strategy, we choose to substitute each variable by its symbolic value, unless it is variable-free, until no substitution can occur; this follows Sect. 6.3.4. In order to avoid large expressions created by cascaded substitutions, we limit the depth of the expressions stored in the abstract environments to a user-specified value, which is set to 20 in our analyses. Also, expressions containing highly non-linear features, such as bit-wise `C` operators, are not stored at all. Finally, following Sect. 6.3.5, our implementation uses hash-consing and tabulation techniques, and maintains a dependency map between symbolic expressions to improve both the time and the memory costs.

8.5.2 Analysis Results

As in the preceding section, we now compare the results of Astrée on the considered program family with plain intervals and with intervals enhanced with linearisation and constant propagation. All other abstract domains are enabled in both cases.

	plain intervals			**enhanced intervals**		
code size in lines	analysis time	max. memory	false alarms	analysis time	max. memory	false alarms
370	1.8s	16 MB	0	3.1s	16 MB	0
9 500	90s	81 MB	8	160s	80 MB	8
70 000	2h 40mn	559 MB	391	1h 16mn	582 MB	0
70 000	24mn	382 MB	10	30mn	378 MB	4
226 000	11h 16mn	1.3 GB	141	6h 36mn	1.3 GB	1
400 000	22h 9mn	2.2 GB	282	13h 52mn	2.2 GB	0

And we also provide the corresponding number of iterations of the main loop, as well as the average time spent in one iteration:

	plain intervals		**enhanced intervals**	
code size in lines	number of iterations	time per iteration	number of iterations	time per iteration
370	17	0.1s	17	0.18s
9 500	39	2.3s	39	4.1s
70 000	141	68s	44	104s
70 000	38	39s	38	49s
226 000	150	270s	86	276s
400 000	172	462s	96	520s

We see that the linearisation and symbolic constant propagation are quite efficient as they result, in our largest examples, in less than a 25% increase in time per iteration. As

for the octagon domain, a decrease in the required number of iterations may compensate this cost and give an actual lower total analysis time. The memory consumption is not noticeably increased by the symbolic constant domain. The precision gain is quite impressive as up to hundreds false alarms are removed.

Possible Precision Degradation. There are several causes of precision loss when linearising. The main cause of precision degradation in the linearisation is, in fact, due to our abstraction of floating-point rounding as a non-deterministic error, as discussed in Sect. 7.5.2: this causes an expression such as $[0,1] \oplus_{\mathbf{32},n} [0,1]$ to be linearised as $[-2^{-148}, 2+2^{-22}+2^{-148}]$ instead of $[0,2]$. Due to this precision degradation, it is important to always intersect the interval evaluation of a linearised expression with the evaluation of the non-linearised expression. Up to now, all precision degradations due to the intervalisation occurring in non-linear expressions could be limited by using a proper multiplication strategy. Likewise, precision degradations due to too deep symbolic constant propagations could always be eliminated by using a proper propagation strategy. Finally, using a sound floating-point implementation of interval linear forms instead of a perfect one on reals does not seem to result in any noticeable precision degradation.

Discussion. Even though it would be possible to use the symbolic constant propagation in the octagon domain, this was not needed to remove false alarms. Our experiments show that, even though the linearisation and constant propagation techniques on intervals are not as robust as fully relational abstract domains, they proved not only very fast, but also quite versatile thanks to their parametrisation in terms of strategies, and much simpler to implement than even a simple relational abstract domain. As the cost of the symbolic constant domain is naturally near linear, there is no need to develop a packing technique limiting the possible dependencies between symbolic expressions, unlike what happened for the octagon domain.

8.6 Extrapolation Operators

For the interval and octagon abstract domains, we use the widening with thresholds of Def. 4.4.8 and the standard narrowing of Def. 3.7.3.

Thresholds Choice. All the computation we encountered that required a widening with thresholds were in fact stable for bounds provided they are "large enough": in the abstract, they stabilise to the smallest threshold larger than the concrete bound, plus an extra abstract rounding error. Such an example is given by the rate limiter of Ex. 4.6.3. Thus, the exact value of the widening steps is of no importance to prove that our variables are bounded and they do not need to be adapted from a program to another. However, as our

programs are composed of many such computations in sequence, imprecision — that is, the difference between the stable abstract bound and the concrete fixpoint — cumulates easily and operations on stable but too large bounds may result in false alarms. This means that the set of thresholds should be sufficiently dense. It should not be much denser than required, however, as the number of thresholds directly affects the number of abstract iterations, and so, the analysis time. In Astrée, we use, as set of thresholds, a simple piecewise linear ramp with a few dozen steps. This is sufficiently precise for all the programs we analyse, and yet provides reasonable analysis times.

Perturbed Widening and Narrowing Following Sect. 7.5.4, we use perturbed widenings and narrowings in the octagon domain to compensate for the non-deterministic floating-point errors committed during the analysis. By skipping above the chaotic region between unstable and stable portions of the abstract transfer function computed along one main loop iteration, we require less iterations with widening to stabilise to an abstract post-fixpoint, and so, the overall analysis time is reduced. Also, by using a perturbed narrowing, we have a better chance to stay within the region of abstract post-fixpoints. Recall that the result of the iterations with widening and narrowing is not required to be an abstract post-fixpoint to be sound, but this guarantees that the analyser is able to check that it is indeed an abstraction of the concrete fix-point using abstract computation only, and thus, it is a very desirable property. The relative amount by which are enlarged unstable constraint, ϵ, is user-defined. It is set to 0.01 in all our examples. In order to demonstrate the importance of the perturbed widening and narrowing, we now present some analysis examples on the same program but using different values for ϵ and, in particular, the value 0 corresponding to the plain widening with thresholds and standard narrowing. For each value of ϵ, we give the number of required iterations with widening and whether the result after narrowing is an abstract post-fixpoint:

ϵ	is a post-fixpoint	number of iterations with ∇	false alarms
0	no	86	0
0.001	no	49	0
0.01	yes	46	0
0.1	yes	41	0
2	yes	31	0

We see that, if ϵ is too small, we need more iterations with widening, and so, a greater analysis time, and the result after narrowing is not an abstract post-fixpoint. If ϵ is too large, there might be, in theory, a loss of precision — although not shown by this experimental results. Between these two extremes, the value of ϵ can be chosen rather loosely. In particular, a sufficiently big ϵ can compensate for a too dense set of thresholds, effectively

decreasing the number of iterations with widening required to stabilise the invariant, and thus, the total analysis time. In Astrée, the initial value 0.01 was found by trial and error after a few analysis runs. This value was kept and did not need any adjustment even when we analysed different programs in the considered family or when we changed the analyser. Perturbed widenings are also used in Feret's digital filters [Fer04b] integrated in Astrée.

8.7 Conclusion

As a conclusion, we would like to say that our experiments with the octagon abstract domain in Astrée for the static analysis of run-time errors in real-life embedded critical programs manipulating machine-integers and floating-point numbers were quite successful. They show that, if we limit the "relationality" to small sets of variables, we can achieve a notable precision gain for a time and memory cost that is effectively linear in the size of the program. In some cases, adding the octagon domain even resulted in a *reduced* total analysis time, showing that octagonal invariants adequately match the semantics of the analysed programs. Similar results also hold for the symbolic constant propagation technique applied to the interval domain. These two abstract domains contribute to remove hundreds of false alarms in Astrée while maintaining its reasonable time and memory consumption: a few hundreds megabytes of memory and a few hours of computation time for `C` programs of a few hundred thousand lines. In particular, the octagon abstract domain removed unexpected kinds of false alarms before we had the opportunity to analyse their cause and develop specific abstract domains to treat each one of them: the octagon domain proved itself of generic use.

Further work is pursued on the Astrée project at the ÉNS and the École Polytechnique to extend it to other program families and other kinds of properties to be proved. We are confident that the numerical abstract domains introduced in this thesis will be useful in the future of Astrée.

Chapter 9

Conclusion

In this thesis, we have contributed to the design of several methods for the analysis of the numerical properties of program variables. These methods can be used independently or in combination with others techniques, including existing ones — such as the interval or polyhedron abstract domains.

Our first contribution is the design of several new numerical abstract domains. Starting from the concept of potential constraints, we built the zone abstract domain to infer conjunction of invariants of the form $X - Y \leq c$, where X and Y range within a fixed set of program variables, and the constant c in $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$ is automatically inferred. We then extended the zone domain into the octagon domain ($\pm X \pm Y \leq c$), the strict zone domain ($X - Y < c$), and the zone congruence domain ($X \equiv Y + a\,[b]$), among others. All these domains share a similar representation in terms of constraint matrices and enjoy cubic-time transitive closure algorithms that compute a normal form by propagating and minimising all constraints at once. We defined all the operators required by an abstract domain: union, intersection, widening, narrowing, equality and inclusion tests. We also provided several transfer functions for the assignment, test, and backward assignment with different trade-offs between cost and precision. The closure algorithm is used pervasively in the design of these operators and transfer functions as its properties guarantee many exactness and best precision results.

Each one of these domains is in-between, in terms of cost and precision, between a non-relational domain — such as the interval or the simple congruence domains — and a relational domain — such as the polyhedron domain. On the one hand, they are relational: they can discover relationships between variables. We showed that the octagon domain was precise enough to find tight variable bounds for program fragments manipulating counters in loops, computing absolute values, and rate limiters, for instance, while this is not possible using the interval domain. On the other hand, they feature a cubic time cost per abstract operation, in the worse case, which is much cheaper than the exponential worst-case cost

of the polyhedron domain, but also much more predictable. Thus, we called these domains "weakly rationals domains".

Natural extensions of this work include the design of even more transfer functions, to offer more choices to its users between cost and precision, and new widening and narrowing operators, corresponding to different discovery strategies for inductive invariants, but also the design of domains with extended expressiveness. It is very tempting to try and extend closure-based normalisation algorithms to other forms of constraints so that exact or best operators and transfer functions can be constructed as in our weakly relational domains. However, the existence of such a closure relies on strong algebraic properties of the manipulated sets of constraints. Even though closure properties for potential constraints are well-known and easy to prove, our extensions to octagonal constraints and constraints of the form $X - Y \in C$ were quite difficult. Form a theoretical point of view, determining how generic closure-based algorithms are would be a hard but exciting problem.

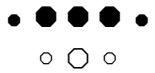
Our second contribution is a set of techniques that can be used to extend the scope and the precision of any numerical abstract domain. They work by soundly replacing expressions encountered in transfer functions by other ones that can be abstracted more precisely in the chosen domain. A first technique, called "linearisation", transforms an arbitrary expression into a linear form where constant coefficients are intervals. Indeed, such interval linear forms can be abstracted quite precisely by all our weakly relational domains. Moreover, the linearisation induces some simplifications that result in increased precision when using non-relational domains, such as the interval one. Finally, it is possible to further abstract interval linear forms into quasi-linear forms that can be fed directly to the polyhedron abstract domain. Our second technique, called "symbolic constant propagation", consists in gathering and propagating assigned expressions in a symbolical way. By "gluing" expression bits together, we are able to enhance the simplification features of the linearisation and compensate for a weak or inexistent level of relationality in the octagon or interval domains, for a small increase in cost. We gave several examples where our techniques applied to the interval domain gives precise results that would otherwise require the use of a costly relational domain. Our two techniques are parametrised by specific strategies to determine were the precision loss occurs when linearising non-linear expressions and how far to propagate symbolic expressions. We saw on selected examples that, depending on the chosen strategy, we may gain more or less precision.

A natural extension is thus to provide more strategies, adapted to specific programming idioms. A more challenging task is to provide theoretical results to guarantee the precision increase in broad contexts. One may wish, for instance, to find a strategy such that abstract computations become immune to a specific class of program transformations.

Our final contribution is the application of (weakly) relational abstract domains to the analysis of machine-integers and floating-point numbers. Floating-point numbers, in par-

ticular, are difficult to abstract as they do not satisfy the algebraic properties of arithmetic operators implicitly required by relational domain algorithms and, up to now, only non-relational domains — such as the interval one — could be used. Our solution is to abstract floating-point and machine-integer expressions into interval linear forms manipulating perfect reals or integers. We also presented techniques to implement our weakly relational domains and linearisation technique using floating-point numbers and machine-integers in the abstract, and thus, trade precision for efficiency. Combining these two aspects, it is possible to design an efficient weakly relational abstract domain, such as the octagon domain, using floating-point numbers — resp. machine-integers — to discover invariants on floating-point — resp. machine-integers — variables. Our unique approach of introducing an intermediate semantics on perfect reals and integers allows a modular construction and easy soundness proofs.

An important part of the thesis work was devoted to implementation and experimentation. A first step was the implementation of a generic and efficient octagon domain library, now freely available on the WEB [Mina]. It is hoped that this software can be used by the static analysis community. It could be, for instance, plugged into existing analysers as an alternative to widely used polyhedra libraries [Jea, PPL] to change the cost versus precision trade-off. A second important step was our participation in the Astrée project. In particular, we incorporated the octagon abstract domain as well as the linearisation and symbolic constant propagation techniques adapted to floating-point numbers into the Astrée analyser. We also designed a packing technique allowing to relate together only small sets of variables and cut the cubic cost into a cost that is linear in practice. All these techniques were instrumental in the success of Astrée, that is, its ability to prove the absence of run-time errors in the primary flight control software of the Airbus A340 fly-by-wire system. This 100 000-line long `C` program performing mainly floating-point computations is presently analysed in little more than one hour on a desktop computer. As a last minute note, a 300 000-line long early version of the corresponding software for the Airbus A380 — still in development at the time of writing — was also proved correct in November 2004 by Astrée. We hoped to demonstrate by this industrial application the practical interest of our methods in terms of both precision and scalability. As the Astrée analyser is applied to different kinds of programs and to prove different kinds of properties, we are confident that it will drive the search for efficient and precise numerical abstract domains even further.



Appendix A

Lengthy Proofs

This appendix presents the complete proofs that were postponed from the main chapters of this thesis because they are quite long.

A.1 Proof of Thm. 4.3.4: Strong Closure Algorithm for Octagons

Properties of the Floyd–Warshall algorithm for strong closure.

1. $\gamma^{Oct}(\mathbf{m}) = \emptyset \iff \exists i,\ \mathbf{m}^n_{ii} < 0$, where $\mathbf{m}^n$ is defined as in Def. 4.3.2.
2. If $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$ then $\mathbf{m}^\bullet$ computed by Def. 4.3.2 is the strong closure as defined by Def. 4.3.1 and Thm. 4.3.3.

●

Proof.

1. We can prove by induction on k that, for all $k \leq n$, $\gamma^{Oct}(\mathbf{m}^k) = \gamma^{Oct}(\mathbf{m})$. In particular, if $\exists i,\ \mathbf{m}^n_{ii} < 0$, then $\gamma^{Oct}(\mathbf{m}^n) = \emptyset$, and so, $\gamma^{Oct}(\mathbf{m})$ is empty.

   Suppose conversely that $\gamma^{Oct}(\mathbf{m}) = \emptyset$. By Thm. 4.3.1, we also have $\gamma^{Pot}(\mathbf{m}) = \emptyset$. We will denote by $\mathbf{m}'^k$ the DBM computed at the $k-$th step of the *regular* Floyd–Warshall algorithm of Def. 3.3.2. By Thm. 3.3.5, there is some x such that $\mathbf{m}'^{2n}_{xx} < 0$. Now, we can prove by induction on k that $\forall i, j,\ \mathbf{m}^k_{ij} \leq \mathbf{m}'^{2k}_{ij}$. As a consequence, $\mathbf{m}^n_{xx} < 0$, which concludes the proof.

2. Suppose that $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$ and let $\mathbf{m}^\bullet$ be the result computed by the modified Floyd–Warshall algorithm of Def. 4.3.2. We prove that $\mathbf{m}^\bullet$ verifies the three criteria of Def. 4.3.1.

By definition $\forall i,\ \mathbf{m}^{\bullet}_{ii} = 0$.

We now prove that $\forall i, j,\ \mathbf{m}^{\bullet}_{ij} \leq (\mathbf{m}^{\bullet}_{i\bar{i}} + \mathbf{m}^{\bullet}_{\bar{j}j})/2$.

First of all, we prove that for any matrix $\mathbf{n}$, $S(\mathbf{n})_{ij} \leq (S(\mathbf{n})_{i\bar{i}} + S(\mathbf{n})_{\bar{j}j})/2$. Indeed, $\forall i,\ S(\mathbf{n})_{i\bar{i}} = \min(\mathbf{n}_{i\bar{i}}, (\mathbf{n}_{i\bar{i}} + \mathbf{n}_{i\bar{i}})/2) = \mathbf{n}_{i\bar{i}}$, so $\forall i, j,\ S(\mathbf{n})_{ij} \leq (\mathbf{n}_{i\bar{i}} + \mathbf{n}_{\bar{j}j})/2 = (S(\mathbf{n})_{i\bar{i}} + S(\mathbf{n})_{\bar{j}j})/2$. Applying this property for $\mathbf{n} \stackrel{\text{def}}{=} C^{2n-1}(\mathbf{m}^{n-1})$, we get that $\forall i, j,\ \mathbf{m}^{n}_{ij} \leq (\mathbf{m}^{n}_{i\bar{i}} + \mathbf{m}^{n}_{\bar{j}j})/2$. This implies that if $i \neq j$, then $\mathbf{m}^{\bullet}_{ij} \leq (\mathbf{m}^{\bullet}_{i\bar{i}} + \mathbf{m}^{\bullet}_{\bar{j}j})/2$. Whenever $i = j$, $\mathbf{m}^{\bullet}_{ii} = 0$ which is smaller than $(\mathbf{m}^{\bullet}_{i\bar{i}} + \mathbf{m}^{\bullet}_{\bar{j}j})/2 = (\mathbf{m}^{n}_{i\bar{i}} + \mathbf{m}^{n}_{\bar{j}j})/2$, or else, there would be a cycle with strictly negative total weight in $\mathcal{G}(\mathbf{m}^n)$ implying $\gamma^{Oct}(\mathbf{m}^n) = \emptyset$, and so, $\gamma^{Oct}(\mathbf{m}) = \emptyset$, which is not true.

Finally, we prove that $\forall i, j, k,\ \mathbf{m}^{\bullet}_{ij} \leq \mathbf{m}^{\bullet}_{ik} + \mathbf{m}^{\bullet}_{kj}$. This is a hard property to prove which justifies the complexity of the modified Floyd–Warshall algorithm of Def. 4.3.2. We will use several lemmas.

- **Lemma 1**.
  Let $\mathbf{n}$ be a coherent DBM $\mathbf{n}$ such that $\gamma^{Oct}(\mathbf{n}) \neq \emptyset$ and there exists some k such that $\forall i, j,\ \mathbf{n}_{ij} \leq \mathbf{n}_{ik} + \mathbf{n}_{kj}$ and $\forall i, j,\ \mathbf{n}_{ij} \leq \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}j}$. We prove that $\forall i, j,\ S(\mathbf{n})_{ij} \leq S(\mathbf{n})_{ik} + S(\mathbf{n})_{kj}$.
  - **Case 1**: $S(\mathbf{n})_{ik} = \mathbf{n}_{ik}$ and $S(\mathbf{n})_{kj} = \mathbf{n}_{kj}$.
    We have obviously:
    $$\begin{array}{lll} S(\mathbf{n})_{ij} & \leq \mathbf{n}_{ij} & \text{(by definition of } S(\mathbf{n})) \\ & \leq \mathbf{n}_{ik} + \mathbf{n}_{kj} & \text{(by hypothesis)} \\ & = S(\mathbf{n})_{ik} + S(\mathbf{n})_{kj} & \text{(case hypothesis).} \end{array}$$
  - **Case 2**: $S(\mathbf{n})_{ik} = (\mathbf{n}_{i\bar{i}} + \mathbf{n}_{\bar{k}k})/2$ and $S(\mathbf{n})_{kj} = \mathbf{n}_{kj}$ (or the symmetric case $S(\mathbf{n})_{ik} = \mathbf{n}_{ik}$ and $S(\mathbf{n})_{kj} = (\mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{j}j})/2$.
    Using the hypothesis two times, we have $\mathbf{n}_{\bar{j}j} \leq \mathbf{n}_{\bar{j}\bar{k}} + \mathbf{n}_{\bar{k}j} \leq \mathbf{n}_{\bar{j}\bar{k}} + (\mathbf{n}_{\bar{k}k} + \mathbf{n}_{kj})$ (1), so we get:
    $$\begin{array}{lll} S(\mathbf{n})_{ij} & \leq \mathbf{n}_{i\bar{i}}/2 + \mathbf{n}_{\bar{j}j}/2 & \text{(by definition of } S(\mathbf{n})) \\ & \leq \mathbf{n}_{i\bar{i}}/2 + (\mathbf{n}_{\bar{j}\bar{k}} + \mathbf{n}_{\bar{k}k} + \mathbf{n}_{kj})/2 & \text{(by (1))} \\ & \leq \mathbf{n}_{i\bar{i}}/2 + \mathbf{n}_{\bar{k}k}/2 + \mathbf{n}_{kj} & \text{(by coherence } \mathbf{n}_{\bar{j}\bar{k}} = \mathbf{n}_{kj}) \\ & = S(\mathbf{n})_{ik} + S(\mathbf{n})_{kj} & \text{(case hypothesis).} \end{array}$$
  - **Case 3**: $S(\mathbf{n})_{ik} = (\mathbf{n}_{i\bar{i}} + \mathbf{n}_{\bar{k}k})/2$ and $S(\mathbf{n})_{kj} = (\mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{j}j})/2$.
    Now we use the fact that $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$ so that the cycle $\langle k, \bar{k}, k\rangle$ has a positive weight, so $0 \leq \mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{k}k}$ (1) and:
    $$\begin{array}{lll} S(\mathbf{n})_{ij} & \leq (\mathbf{n}_{i\bar{i}} + \mathbf{n}_{\bar{j}j})/2 & \text{(by definition of } S(\mathbf{n})) \\ & \leq (\mathbf{n}_{i\bar{i}} + (\mathbf{n}_{\bar{k}k} + \mathbf{n}_{k\bar{k}}) + \mathbf{n}_{\bar{j}j})/2 & \text{(by (1))} \\ & = S(\mathbf{n})_{ik} + S(\mathbf{n})_{kj} & \text{(case hypothesis).} \end{array}$$

- **Lemma 2**.
  Let $\mathbf{n}$ be a coherent DBM such that $\gamma^{Oct}(\mathbf{n}) \neq \emptyset$ and there exists some k such that $\forall i \neq j,\ \mathbf{n}_{ij} \leq \mathbf{n}_{ik} + \mathbf{n}_{kj}$ and $\forall i \neq j,\ \mathbf{n}_{ij} \leq \mathbf{n}_{i\overline{k}} + \mathbf{n}_{\overline{k}j}$. We prove that $\forall o, i \neq j,\ C^o(\mathbf{n})_{ij} \leq C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj}$.
  There are five different cases for the value of $C^o(\mathbf{n})_{ik}$ and five cases for the value of $C^o(\mathbf{n})_{kj}$:

1	$C^o(\mathbf{n})_{ik} = \mathbf{n}_{ik}$	1	$C^o(\mathbf{n})_{kj} = \mathbf{n}_{kj}$
2	$C^o(\mathbf{n})_{ik} = \mathbf{n}_{io} + \mathbf{n}_{ok}$	2	$C^o(\mathbf{n})_{kj} = \mathbf{n}_{ko} + \mathbf{n}_{oj}$
3	$C^o(\mathbf{n})_{ik} = \mathbf{n}_{i\overline{o}} + \mathbf{n}_{\overline{o}k}$	3	$C^o(\mathbf{n})_{kj} = \mathbf{n}_{k\overline{o}} + \mathbf{n}_{\overline{o}j}$
4	$C^o(\mathbf{n})_{ik} = \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}k}$	4	$C^o(\mathbf{n})_{kj} = \mathbf{n}_{ko} + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j}$
5	$C^o(\mathbf{n})_{ik} = \mathbf{n}_{i\overline{o}} + \mathbf{n}_{\overline{o}o} + \mathbf{n}_{ok}$	5	$C^o(\mathbf{n})_{kj} = \mathbf{n}_{k\overline{o}} + \mathbf{n}_{\overline{o}o} + \mathbf{n}_{oj}$

  In the following, we will denote by (a, b) the case where the value of $C^o(\mathbf{n})_{ik}$ is defined by the a^{th} case and the value of $C^o(\mathbf{n})_{kj}$ is defined by the b^{th} case. We then have 25 different cases to inspect.
  To reduce the number of cases really studied we use the strong symmetry of the definition of $C^o(\mathbf{n})$ with respect to o and $\overline{o}$ together with the symmetry of the hypotheses with respect to k and $\overline{k}$ and the fact that $\forall i, j,\ \mathbf{n}_{ij} = \mathbf{n}_{\overline{j}\,\overline{i}}$ by coherence of $\mathbf{n}$.
  We also use the fact that analysis of case (a, b) for $a \neq b$ is very similar to the analysis of (b, a), so, we will suppose $a \leq b$.
  We will also often use the fact that $\forall i, j,\ \mathbf{n}_{ij} + \mathbf{n}_{ji} \geq 0$, which is the consequence of the fact that $\langle i, j, i\rangle$ is a cycle in $\mathbf{n}$ with $\gamma^{Oct}(\mathbf{n}) \neq \emptyset$.

  - **Case 1**: $(1, 1)$.
    We have, by hypothesis, $\mathbf{n}_{ij} \leq \mathbf{n}_{ik} + \mathbf{n}_{kj}$ (1), so obviously:

$$\begin{array}{lll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{ij} & \text{(by definition of } C^o(\mathbf{n})) \\ & \leq \mathbf{n}_{ik} + \mathbf{n}_{kj} & \text{(by (1))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis).} \end{array}$$

  - **Case 2**: $(1, 2)$ (and $(1, 3)$ by $(o, \overline{o})$ symmetry).
    Sub-case 1: $i \neq o$.
    We have, by hypothesis, $\mathbf{n}_{io} \leq \mathbf{n}_{ik} + \mathbf{n}_{ko}$ (1), so:

$$\begin{array}{lll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{io} + \mathbf{n}_{oj} & \text{(by definition of } C^o(\mathbf{n})) \\ & \leq (\mathbf{n}_{ik} + \mathbf{n}_{ko}) + \mathbf{n}_{oj} & \text{(by (1))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis).} \end{array}$$

    Sub-case 2: $i = o$.

We know that $\mathbf{n}_{ik} + \mathbf{n}_{ko} \geq 0$ (1), so:

$$\begin{array}{lll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{oj} & \text{(by definition of } C^o(\mathbf{n})\text{)} \\ & \leq (\mathbf{n}_{ik} + \mathbf{n}_{ko}) + \mathbf{n}_{oj} & \text{(by (1))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis).} \end{array}$$

- **Case 3**: $(1, 4)$ (and $(1, 5)$ by $(o, \overline{o})$ symmetry).
  Sub-case 1: $i \neq o$.
  We have, by hypothesis, $\mathbf{n}_{io} \leq \mathbf{n}_{ik} + \mathbf{n}_{ko}$ (1), so:

$$\begin{array}{lll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j} & \text{(by definition of } C^o(\mathbf{n})\text{)} \\ & \leq (\mathbf{n}_{ik} + \mathbf{n}_{ko}) + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j} & \text{(by (1))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis).} \end{array}$$

  Sub-case 2: $i = o$.
  As in the second case, we have $\mathbf{n}_{ik} + \mathbf{n}_{ko} \geq 0$ (1), so:

$$\begin{array}{lll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{i\overline{o}} + \mathbf{n}_{\overline{o}j} & \text{(by definition of } C^o(\mathbf{n})\text{)} \\ & \leq (\mathbf{n}_{ik} + \mathbf{n}_{ko}) + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j} & \text{(by (1))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis).} \end{array}$$

- **Case 4**: $(2, 2)$ (and $(3, 3)$ by $(o, \overline{o})$ symmetry).
  We know that $\mathbf{n}_{ok} + \mathbf{n}_{ko} \geq 0$ (1), so:

$$\begin{array}{lll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{io} + \mathbf{n}_{oj} & \text{(by definition of } C^o(\mathbf{n})\text{)} \\ & \leq \mathbf{n}_{io} + (\mathbf{n}_{ok} + \mathbf{n}_{ko}) + \mathbf{n}_{oj} & \text{(by (1))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis).} \end{array}$$

- **Case 5**: $(2, 3)$.
  We have, by hypothesis, $\mathbf{n}_{o\overline{o}} \leq \mathbf{n}_{ok} + \mathbf{n}_{k\overline{o}}$ (1), so:

$$\begin{array}{lll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j} & \text{(by definition of } C^o(\mathbf{n})\text{)} \\ & \leq \mathbf{n}_{io} + (\mathbf{n}_{ok} + \mathbf{n}_{k\overline{o}}) + \mathbf{n}_{\overline{o}j} & \text{(by (1))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis).} \end{array}$$

- **Case 6**: $(2, 4)$ (and $(3, 5)$ by $(o, \overline{o})$ symmetry).
  We use, as in the fourth case $\mathbf{n}_{ok} + \mathbf{n}_{ko} \geq 0$ (1), so:

$$\begin{array}{lll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j} & \text{(by definition of } C^o(\mathbf{n})\text{)} \\ & \leq \mathbf{n}_{io} + (\mathbf{n}_{ok} + \mathbf{n}_{ko}) + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j} & \text{(by (1))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis).} \end{array}$$

◦ **Case 7**: $(2,5)$ (and $(3,4)$ by $(o,\overline{o})$ symmetry).
We use the fact that $\mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}o} \geq 0$ (1), together with the hypothesis $\mathbf{n}_{o\overline{o}} \leq \mathbf{n}_{ok} + \mathbf{n}_{k\overline{o}}$ (2), so:

$$\begin{array}{ll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{io} + \mathbf{n}_{oj} \quad \text{(by definition of } C^o(\mathbf{n})\text{)} \\ & \leq \mathbf{n}_{io} + (\mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}o}) + \mathbf{n}_{oj} \quad \text{(by (1))} \\ & \leq \mathbf{n}_{io} + ((\mathbf{n}_{ok} + \mathbf{n}_{k\overline{o}}) + \mathbf{n}_{\overline{o}o}) + \mathbf{n}_{oj} \quad \text{(by (2))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} \quad \text{(case hypothesis).} \end{array}$$

◦ **Case 8**: $(4,4)$ (and $(5,5)$ by $(o,\overline{o})$ symmetry).
We use, as in the seventh case $\mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}o} \geq 0$ (1), together with and the hypothesis $\mathbf{n}_{\overline{o}o} \leq \mathbf{n}_{\overline{o}k} + \mathbf{n}_{ko}$ (2), so:

$$\begin{array}{ll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j} \quad \text{(by def. of } C^o(\mathbf{n})\text{)} \\ & \leq \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + (\mathbf{n}_{\overline{o}o} + \mathbf{n}_{o\overline{o}}) + \mathbf{n}_{\overline{o}j} \quad \text{(by (1))} \\ & \leq \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + ((\mathbf{n}_{\overline{o}k} + \mathbf{n}_{ko}) + \mathbf{n}_{o\overline{o}}) + \mathbf{n}_{\overline{o}j} \quad \text{(by (2))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} \quad \text{(case hypothesis).} \end{array}$$

◦ **Case 9**: $(4,5)$.
We use, as in the seventh case $\mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}o} \geq 0$ (1) and $\mathbf{n}_{\overline{o}k} + \mathbf{n}_{k\overline{o}} \geq 0$ (2), so:

$$\begin{array}{ll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{io} + \mathbf{n}_{oj} \quad \text{(by def. of } C^o(\mathbf{n})\text{)} \\ & \leq \mathbf{n}_{io} + (\mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}o}) + (\mathbf{n}_{\overline{o}k} + \mathbf{n}_{k\overline{o}}) + \mathbf{n}_{oj} \quad \text{(by (1) and (2))} \\ & = \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}k} + \mathbf{n}_{k\overline{o}} + \mathbf{n}_{\overline{o}o} + \mathbf{n}_{oj} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} \quad \text{(case hypothesis).} \end{array}$$

• **Lemma 3**.
We prove now that, given a coherent DBM $\mathbf{n}$ such that $\gamma^{Oct}(\mathbf{n}) \neq \emptyset$ and an index k, we have — without any other hypothesis — $\forall i \neq j,\ C^k(\mathbf{n})_{ij} \leq C^k(\mathbf{n})_{ik} + C^k(\mathbf{n})_{kj}$.

We have the same five different cases for the value of $C^k(\mathbf{n})_{ik}$ and the same five cases for the value of $C^k(\mathbf{n})_{kj}$ as in the preceding lemma, so we have the same 25 different cases to inspect.

In order to reduce the number of cases really studied let us remark that $\mathbf{n}_{kk} \geq 0$ and $\mathbf{n}_{k\overline{k}} + \mathbf{n}_{\overline{k}k} \geq 0$ because $\mathbf{n}$ has no strictly negative cycle. This means that, in fact, $C^k(\mathbf{n})_{ik} = \min(\mathbf{n}_{ik}, \mathbf{n}_{i\overline{k}} + \mathbf{n}_{\overline{k}k})$. Cases 2, 4 and 5 are not relevant for the value of $C^k(\mathbf{n})_{ik}$. The same result holds for $C^k(\mathbf{n})_{kj}$ and we get $C^k(\mathbf{n})_{kj} = \min(\mathbf{n}_{kj}, \mathbf{n}_{k\overline{k}} + \mathbf{n}_{\overline{k}j})$.

This means we only have four different cases to study:

- **Case 1**: $(1,1)$.
  We have:

$$\begin{array}{lll} C^k(\mathbf{n})_{ij} & \leq \mathbf{n}_{ik} + \mathbf{n}_{kj} & \text{(by definition of } C^k(\mathbf{n})) \\ & = C^k(\mathbf{n})_{ik} + C^k(\mathbf{n})_{kj} & \text{(by case hypothesis).} \end{array}$$

- **Case 2**: $(1,3)$.
  We have:

$$\begin{array}{lll} C^k(\mathbf{n})_{ij} & \leq \mathbf{n}_{ik} + \mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{k}j} & \text{(by definition of } C^k(\mathbf{n})) \\ & = C^k(\mathbf{n})_{ik} + C^k(\mathbf{n})_{kj} & \text{(by case hypothesis).} \end{array}$$

- **Case 3**: $(3,1)$.
  We have:

$$\begin{array}{lll} C^k(\mathbf{n})_{ij} & \leq \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}k} + \mathbf{n}_{kj} & \text{(by definition of } C^k(\mathbf{n})) \\ & = C^k(\mathbf{n})_{ik} + C^k(\mathbf{n})_{kj} & \text{(by case hypothesis).} \end{array}$$

- **Case 4**: $(3,3)$.
  We have $\mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{k}k} \geq 0$ (1), so

$$\begin{array}{lll} C^k(\mathbf{n})_{ij} & \leq \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}j} & \text{(by definition of } C^k(\mathbf{n})) \\ & \leq \mathbf{n}_{i\bar{k}} + (\mathbf{n}_{\bar{k}k} + \mathbf{n}_{k\bar{k}}) + \mathbf{n}_{\bar{k}j} & \text{(by (1))} \\ & = C^k(\mathbf{n})_{ik} + C^k(\mathbf{n})_{kj} & \text{(by case hypothesis).} \end{array}$$

Now we use all three lemmas to prove by induction on o the following property:

$$\left\{ \begin{array}{l} \forall\, 1 \leq k \leq o,\ \forall i,j\ \mathbf{m}^o_{ij} \leq \mathbf{m}^o_{i\,(2k-1)} + \mathbf{m}^o_{(2k-1)\,j} \\ \forall\, 1 \leq k \leq o,\ \forall i,j\ \mathbf{m}^o_{ij} \leq \mathbf{m}^o_{i\,(2k)} + \mathbf{m}^o_{(2k)\,j}. \end{array} \right.$$

- The case $o = 0$ is obvious.
- Suppose that the property is true for $o - 1 \geq 0$.
  Using the second lemma with $2k-1$ and $2k$, for all $k \leq o-1$, we get $\forall i \neq j$:

$$\left\{ \begin{array}{l} (C^{2o-1}(\mathbf{m}^{o-1}))_{ij} \leq (C^{2o-1}(\mathbf{m}^{o-1}))_{i\,(2k-1)} + (C^{2o-1}(\mathbf{m}^{o-1}))_{(2k-1)\,j} \\ (C^{2o-1}(\mathbf{m}^{o-1}))_{ij} \leq (C^{2o-1}(\mathbf{m}^{o-1}))_{i\,(2k)} + (C^{2o-1}(\mathbf{m}^{o-1}))_{(2k)\,j}. \end{array} \right.$$

  Using the third lemma with $2o-1$ and $2o$ and the remark that $\forall \mathbf{m},\ o,\ C^o(\mathbf{m}) = C^{\overline{o}}(\mathbf{m})$ we get $\forall i \neq j$:

$$\left\{ \begin{array}{l} (C^{2o-1}(\mathbf{m}^{o-1}))_{ij} \leq (C^{2o-1}(\mathbf{m}^{o-1}))_{i\,(2o-1)} + (C^{2o-1}(\mathbf{m}^{o-1}))_{(2o-1)\,j} \\ (C^{2o-1}(\mathbf{m}^{o-1}))_{ij} \leq (C^{2o-1}(\mathbf{m}^{o-1}))_{i\,(2o)} + (C^{2o-1}(\mathbf{m}^{o-1}))_{(2o)\,j}. \end{array} \right.$$

Recall that, by definition, $(C^{2k-1}(\mathbf{m}^{o-1}))_{ii} = 0$.
Obviously, $\gamma^{Oct}((C^{2k-1}(\mathbf{m}^{o-1}))) = \gamma^{Oct}(\mathbf{m}) \neq \emptyset$, so for all k, the cycle $\langle i, k, i\rangle$ has a positive weight which means that:

$$\forall k,\ (C^{2k-1}(\mathbf{m}^{o-1}))_{ii} = 0 \leq (C^{2o-1}(\mathbf{m}^{o-1}))_{ik} + (C^{2o-1}(\mathbf{m}^{o-1}))_{ki}$$

and we have $\forall k \leq o,\ \forall i, j$:

$$\left\{ \begin{array}{l} (C^{2o-1}(\mathbf{m}^{o-1}))_{ij} \leq (C^{2o-1}(\mathbf{m}^{o-1}))_{i\,(2k-1)} + (C^{2o-1}(\mathbf{m}^{o-1}))_{(2k-1)\,j} \\ (C^{2o-1}(\mathbf{m}^{o-1}))_{ij} \leq (C^{2o-1}(\mathbf{m}^{o-1}))_{i\,(2k)} + (C^{2o-1}(\mathbf{m}^{o-1}))_{(2k)\,j}. \end{array} \right.$$

Now we use the first lemma to get $\forall k \leq o$ and $\forall i, j$:

$$\left\{ \begin{array}{l} (S(C^{2o-1}(\mathbf{m}^{o-1})))_{ij} \leq (S(C^{2o-1}(\mathbf{m}^{o-1})))_{i\,(2k-1)} + (S(C^{2o-1}(\mathbf{m}^{o-1})))_{(2k-1)\,j} \\ (S(C^{2o-1}(\mathbf{m}^{o-1})))_{ij} \leq (S(C^{2o-1}(\mathbf{m}^{o-1})))_{i\,(2k)} + (S(C^{2o-1}(\mathbf{m}^{o-1})))_{(2k)\,j}. \end{array} \right.$$

The property for $o = n$ is settles the proof.

A.2 Proof of Thm. 5.2.1: Closure Algorithm for Constraint Matrices

Properties of the Floyd–Warshall algorithm for constraint matrices.

1. $\gamma^{Weak}(\mathbf{m}^{\bigstar}) = \gamma^{Weak}(\mathbf{m})$.
2. $\gamma^{Weak}(\mathbf{m}) = \emptyset \iff \exists i,\ 0 \notin \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ii})$.
3. If $\gamma^{Weak}(\mathbf{m}^{\bigstar}) \neq \emptyset$, then:
   - $\mathbf{m}^{\bigstar}$ is coherent,
   - $\mathbf{m}^{\bigstar}$ is the *transitive closure* of $\mathbf{m}$, up to $\gamma_{\mathcal{B}}$:

$$\forall i, j,\ \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij}) = \gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\substack{\mathcal{B} \\ \langle i=i_1,\ldots,i_m=j\rangle}} \mathbf{m}_{i_1 i_2} +^{\sharp} \cdots +^{\sharp} \mathbf{m}_{i_{m-1} i_m} \right)$$

   - all constraints in $\mathbf{m}^{\bigstar}$ *saturate* $\gamma^{Weak}(\mathbf{m})$:

$$\begin{array}{l} \forall i, j,\ \forall c \in \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij}), \\ \exists \vec{v} \in \mathbb{I}^{n+1} \text{ such that } v_0 = 0,\ (v_1, \ldots, v_n) \in \gamma^{Weak}(\mathbf{m}),\ \text{and } v_j - v_i = c \end{array}$$

- $\mathbf{m}^\star$ is a normal form, up to $\gamma_{\mathcal{B}}$:

$$\forall i,j,\ \gamma_{\mathcal{B}}(\mathbf{m}^\star_{ij}) = \inf_{\subseteq} \{\ \gamma_{\mathcal{B}}(\mathbf{n}_{ij}) \mid \gamma^{Weak}(\mathbf{m}) = \gamma^{Weak}(\mathbf{n})\ \}$$

- the closure has the following *local characterisation*:

$$\mathbf{m} = \mathbf{m}^\star \iff \begin{cases} \forall i,j,k, & \gamma_{\mathcal{B}}(\mathbf{m}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}_{ik} \mathbin{+^\sharp} \mathbf{m}_{kj}) \\ \forall i, & \gamma_{\mathcal{B}}(\mathbf{m}_{ii}) = \{0\} \end{cases}$$

●

Proof.

- **Claim:** $\gamma^{Weak}(\mathbf{m}^{n+1}) = \gamma^{Weak}(\mathbf{m})$.

  This is an easy consequence of the fact that $\cap^\sharp_{\mathcal{B}}$ and $+^\sharp$ are sound over-approximations of $\cap$ and $+$.

- **Claim:** $\mathbf{m}^{n+1}$ and $\mathbf{m}^\star$ are coherent

  Proof. This is an easy consequence of the exactness of $-^\sharp$, $+^\sharp$, and $\cap^\sharp_{\mathcal{B}}$. One would prove by induction that $\forall i,j,k,\ \gamma_{\mathcal{B}}(-^\sharp(\mathbf{m}^k_{ij} \cap^\sharp_{\mathcal{B}} (\mathbf{m}^k_{ik} +^\sharp \mathbf{m}^k_{kj}))) = \gamma_{\mathcal{B}}((-^\sharp\mathbf{m}^k_{ij}) \cap^\sharp_{\mathcal{B}} (-^\sharp(\mathbf{m}^k_{ik}) +^\sharp -^\sharp(\mathbf{m}^k_{kj})))$. Finally, changing $\mathbf{m}^{n+1}_{ii}$ into $[0,0]^\sharp$ to get $\mathbf{m}^\star$ preserves the coherence.

- **Lemma 1:** for any fixed $0 \leq i,j \leq n$, $0 \leq k \leq n+1$ and simple path $\langle i = i_1, \ldots, i_m = j\rangle$ in $\mathbf{m}$ such that $i_l < k$ for all $1 < l < m$, we have $\gamma_{\mathcal{B}}(\mathbf{m}^k_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}_{i_1 i_2} +^\sharp \cdots +^\sharp \mathbf{m}_{i_{m-1} i_m})$. Even though $+^\sharp$ may not be associative, it is exact, and so, $\gamma_{\mathcal{B}}(\mathbf{m}_{i_1 i_2} +^\sharp \cdots +^\sharp \mathbf{m}_{i_{m-1} i_m})$ is uniquely defined.

  Corollary. For every simple path $\langle i = i_1, \ldots, i_m = j\rangle$, $\gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}_{i_1 i_2} +^\sharp \cdots +^\sharp \mathbf{m}_{i_{m-1} i_m})$.

  Proof. By induction.. The property is obvious for $k = 0$ as it is equivalent to $\gamma_{\mathcal{B}}(\mathbf{m}^0_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}_{ij})$ and we have indeed $\mathbf{m}^0 = \mathbf{m}$. Suppose that the property is true for $k \leq n$ and let $\langle i = i_1, \ldots, i_m = j\rangle$ be a path satisfying the hypotheses of the lemma at $k+1$. If $\forall l \in \{2, \ldots, m-1\},\ i_l < k$, the property is true by induction hypothesis and because $\gamma_{\mathcal{B}}(\mathbf{m}^{k+1}_{ij}) = \gamma_B(\mathbf{m}^k_{ij} \cap^\sharp_{\mathcal{B}} (\mathbf{m}^k_{ik} +^\sharp \mathbf{m}^k_{kj})) = \gamma_B(\mathbf{m}^k_{ij}) \cap \gamma_B(\mathbf{m}^k_{ik} +^\sharp \mathbf{m}^k_{kj}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}^k_{ij})$. On the contrary, if there exists a l in $2 \ldots m-1$ such that $i_l \geq k$, we know that $i_l = k$ and l is unique because the path is simple. By definition of $\mathbf{m}^{k+1}$, we have $\gamma_{\mathcal{B}}(\mathbf{m}^{k+1}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}^k_{ik} +^\sharp \mathbf{m}^k_{kj})$. We get the expected result by applying the induction hypothesis to $\langle i = i_1, \ldots, i_l = k\rangle$ in $\mathbf{m}^k_{ik}$ and to $\langle k = i_l, \ldots, i_m = j\rangle$ in $\mathbf{m}^k_{kj}$ as well as the exactness of $+^\sharp$.

- **Lemma 2:** if, for some $0 \leq i, j \leq n$, $\gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right) = \emptyset$, then $\gamma^{Weak}(\mathbf{m}) = \emptyset$.

  Proof.

  Suppose that $\gamma^{Weak}(\mathbf{m}) \neq \emptyset$. Take some $(x_1, \ldots, x_n) \in \gamma^{Weak}(\mathbf{m})$ extended to a vector of $\mathbb{I}^{n+1}$ by $x_0 = 0$. For any elements i, j, and any path $\langle i = i_1, \ldots, i_m = j\rangle$ from i to j, we have $\forall l \in \{1, \ldots, m-1\}$, $x_{i_{l+1}} - x_{i_l} \in \gamma_{\mathcal{B}}(\mathbf{m}_{i_l i_{l+1}})$, so, by summation and exactness of $\mathbin{+^{\sharp}}$, we get $x_j - x_i \in \gamma_{\mathcal{B}}(\mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m})$. As this is valid for all paths from i to j, we have $x_j - x_i \in \bigcap_{\langle i=i_1,\ldots,i_m=j\rangle} \gamma_{\mathcal{B}}(\mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m})$. Moreover, by Def. 5.2.5.2, $\gamma_{\mathcal{B}}$ is a complete $\cap-$morphism, so $x_j - x_i \in \gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right)$, which is thus not empty.

- **Lemma 3:** if $\forall\, 0 \leq i, j \leq n$, $\gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right) \neq \emptyset$, then $\forall\, 0 \leq i, j \leq n,\ 0 \leq k \leq n+1$, $\gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}^k_{ij})$.

  By induction. If $k = 0$, then we have $\gamma_{\mathcal{B}}(\mathbf{m}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}^0_{ij})$ because $\mathbf{m}^0 = \mathbf{m}$, so *a fortiori* the lemma is true. Suppose that the property is true for $k \leq n$, we now prove the property for $k+1$. As $\mathbf{m}^{k+1}_{ij} \stackrel{\text{def}}{=} \mathbf{m}^k_{ij} \cap^{\sharp}_{\mathcal{B}} (\mathbf{m}^k_{ik} \mathbin{+^{\sharp}} \mathbf{m}^k_{kj})$ and $\cap^{\sharp}_{\mathcal{B}}$ is exact, we need to prove that $\gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right)$ is smaller than both $\gamma_{\mathcal{B}}(\mathbf{m}^k_{ij})$ and $\gamma_{\mathcal{B}}(\mathbf{m}^k_{ik} \mathbin{+^{\sharp}} \mathbf{m}^k_{kj})$. The first inequality is a direct consequence of the induction hypothesis. For the second inequality, we will prove the stronger property that the intersection of constraints gathered along all paths that pass through k is more precise than $\mathbf{m}^k_{ik} \mathbin{+^{\sharp}} \mathbf{m}^k_{kj}$. As we consider *fewer* paths than required, this is a *stronger* result which implies the desired property:

$$
\begin{aligned}
&\gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=j\rangle} (\mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m})\right)\\
&\quad \subseteq \gamma_{\mathcal{B}}(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_o=k,\ldots,i_m=j\rangle} (\ (\mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{o-1} i_o}) \mathbin{+^{\sharp}}\\
&\qquad\qquad\qquad\qquad (\mathbf{m}_{i_o i_{o+1}} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m})\ )\ )\\
&\quad = \gamma_{\mathcal{B}}\ \left(\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=k\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right) \mathbin{+^{\sharp}}\right.\\
&\qquad\qquad \left.\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle k=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right)\quad\right)\\
&\quad \subseteq \gamma_{\mathcal{B}}(\mathbf{m}^k_{ik}) + \gamma_{\mathcal{B}}(\mathbf{m}^k_{kj})\\
&\quad = \gamma_{\mathcal{B}}(\mathbf{m}^k_{ik} \mathbin{+^{\sharp}} \mathbf{m}^k_{kj})
\end{aligned}
$$

  where $+$ has been extended to sets of numbers.

  Remark: the restricted distributivity of $\cap^{\sharp}_{\mathcal{B}}$ over $\mathbin{+^{\sharp}}$, used in the first equality, is crucial in the proof of this lemma.

- **Lemma 4:** if $\exists i,\ 0 \notin \gamma_{\mathcal{B}}(\mathbf{m}_{ii}^{n+1})$, then $\gamma^{Weak}(\mathbf{m}) = \emptyset$.

  Proof. Suppose that for some i, $0 \notin \gamma_{\mathcal{B}}(\mathbf{m}_{ii}^{n+1})$. If there was an element $\vec{x} \in \mathbb{I}^{n+1}$ such that $x_0 = 0$ and $(x_1, \ldots, x_n) \in \gamma^{Weak}(\mathbf{m})$, it would be such that $0 = x_i - x_i \in \gamma_{\mathcal{B}}(\mathbf{m}_{ii}^{\bigstar})$, which is absurd, so $\gamma^{Weak}(\mathbf{m}^{n+1}) = \emptyset$. By the first point, we also get $\gamma^{Weak}(\mathbf{m}) = \emptyset$.

- **Lemma 5:** if $\forall i,\ 0 \in \gamma_{\mathcal{B}}(\mathbf{m}_{ii}^{n+1})$, then $\forall i, j$,

$$\gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right)$$
$$= \gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\substack{\mathcal{B}\ \langle i=i_1,\ldots,i_n=j\rangle \\ \text{simple path}}} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right)$$

  Proof. The $\subseteq$ part of the equality is a direct consequence of the fact that the right intersection is less restrictive than the left one because fewer paths are considered.

  For the $\supseteq$ part, we prove that, for each path with at least one cycle in it, there exists a path with one simple cycle less which leads to a more precise constraint. Let $\langle i = i_1, \ldots, i_s, \ldots, i_t = i_s, \ldots, i_m = j\rangle$ be a path and $\langle i_s, \ldots, i_t = i_s\rangle$ a simple cycle in it. By Lemma 1, $\gamma_{\mathcal{B}}(\mathbf{m}_{i_s i_{s+1}} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{t-1} i_t}) \supseteq \gamma_{\mathcal{B}}(\mathbf{m}_{ii}^{n+1})$. By hypothesis, we have $0 \in \gamma_{\mathcal{B}}(\mathbf{m}_{ii}^{n+1})$. Thus, $0 \in \gamma_{\mathcal{B}}(\mathbf{m}_{i_s i_{s+1}} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{t-1} i_t})$. We then have:

$$\begin{aligned}
&\gamma_{\mathcal{B}}(\mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}) \\
&\quad = \gamma_{\mathcal{B}}(\mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{s-1} i_s}) + \gamma_{\mathcal{B}}(\mathbf{m}_{i_s i_{s+1}} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{t-1} i_t}) + \\
&\qquad \gamma_{\mathcal{B}}(\mathbf{m}_{i_t i_{t+1}} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}) \\
&\quad \subseteq \gamma_{\mathcal{B}}(\mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{s-1} i_s}) + \{0\} + \gamma_{\mathcal{B}}(\mathbf{m}_{i_t i_{t+1}} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}) \\
&\quad = \gamma_{\mathcal{B}}(\mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{s-1} i_s}) + \gamma_{\mathcal{B}}(\mathbf{m}_{i_t i_{t+1}} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}) \\
&\quad = \gamma_{\mathcal{B}}((\mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{s-1} i_s}) \mathbin{+^{\sharp}} (\mathbf{m}_{i_t i_{t+1}} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}))
\end{aligned}$$

  where $+$ has been extended to sets of numbers.

- **Lemma 6:** if $\gamma^{Weak}(\mathbf{m}) \neq \emptyset$, then

$$\forall i, j,\ \gamma_{\mathcal{B}}(\mathbf{m}_{ij}^{n+1}) = \gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\substack{\mathcal{B} \\ \langle i=i_1,\ldots,i_m=j\rangle}} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right)$$

  and

$$\forall i, j, k,\ \gamma_{\mathcal{B}}(\mathbf{m}_{ij}^{n+1}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}_{ik}^{n+1} \mathbin{+^{\sharp}} \mathbf{m}_{kj}^{n+1})$$

  Proof. Suppose that $\gamma^{Weak}(\mathbf{m}) \neq \emptyset$. By Lemma 2, $\forall i, j$, $\gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right) \neq \emptyset$.

Thus, we can apply Lemma 1 and 3 to get $\forall i, j$;

$$\gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right)$$

$$\subseteq \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ij})$$

$$\subseteq \gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \substack{\langle i=i_1,\ldots,i_m=j\rangle \\ \text{simple path}}} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right)$$

By Lemma 4, $\forall i,\ 0 \in \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ii})$. Thus, we can apply Lemma 5 to get $\forall i, j$:

$$\begin{aligned}\gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ij}) &= \gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right) \\ &= \gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \substack{\langle i=i_1,\ldots,i_m=j\rangle \\ \text{simple path}}} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right)\end{aligned}$$

Applying a method similar to the one used in Lemma 3, we get: $\forall i, j, k$,

$$\begin{aligned}\gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ij}) &= \gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right) \\ &\subseteq \gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_o=k,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right) \\ &= \gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=k\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right) + \\ &\quad \gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle k=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right) \\ &= \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ik}) + \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{kj})\end{aligned}$$

- **Claim:** if $\gamma^{Weak}(\mathbf{m}) \neq \emptyset$, then

$$\forall i, j,\ \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij}) \subseteq \gamma_{\mathcal{B}}\left(\bigcap^{\sharp}_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} \mathbin{+^{\sharp}} \cdots \mathbin{+^{\sharp}} \mathbf{m}_{i_{m-1} i_m}\right)$$

and

$$\forall i, j, k,\ \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}^{\bigstar}_{ik} \mathbin{+^{\sharp}} \mathbf{m}^{n+1}_{kj})$$

Proof. Recall that $\mathbf{m}^\star$ is defined to be $\mathbf{m}^{n+1}$ except that $\forall i,\ \mathbf{m}^\star_{ii} = [0,0]^\sharp$.

Suppose that $i \neq j$. Then, $\gamma_{\mathcal{B}}(\mathbf{m}^\star_{ij}) = \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ij}) = \gamma_{\mathcal{B}}(\bigcap^\sharp_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} +^\sharp \cdots +^\sharp \mathbf{m}_{i_{m-1} i_m})$ by Lemma 6. Moreover, if $k \neq i, j$, then $\gamma_{\mathcal{B}}(\mathbf{m}^\star_{ij}) = \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ik} +^\sharp \mathbf{m}^{n+1}_{kj}) = \gamma_{\mathcal{B}}(\mathbf{m}^\star_{ik} +^\sharp \mathbf{m}^\star_{kj})$ also by Lemma 6. If $k = i$, $\gamma_{\mathcal{B}}(\mathbf{m}^\star_{ij}) = \gamma_{\mathcal{B}}([0,0]^\sharp +^\sharp \mathbf{m}^\star_{kj}) = \gamma_{\mathcal{B}}(\mathbf{m}^\star_{ik} +^\sharp \mathbf{m}^\star_{kj})$, and likewise when $k = j$.

Suppose that $i = j$. By Lemma 4, $0 \in \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ii})$, so by Lemma 6, $\gamma_{\mathcal{B}}(\mathbf{m}^\star_{ii}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ii}) = \gamma_{\mathcal{B}}\left(\bigcap^\sharp_{\mathcal{B}\ \langle i=i_1,\ldots,i_m=j\rangle} \mathbf{m}_{i_1 i_2} +^\sharp \cdots +^\sharp \mathbf{m}_{i_{m-1} i_m}\right)$. If $i \neq k$, by Lemma 6 we also have $\gamma_{\mathcal{B}}(\mathbf{m}^\star_{ii}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ii}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ik} +^\sharp \mathbf{m}^{n+1}_{ki}) = \gamma_{\mathcal{B}}(\mathbf{m}^\star_{ik} +^\sharp \mathbf{m}^\star_{ki})$. If $i = k$, then $\gamma_{\mathcal{B}}(\mathbf{m}^\star_{ii}) = \gamma_{\mathcal{B}}(\mathbf{m}^\star_{ii} +^\sharp \mathbf{m}^\star_{ii}) = \{0\}$.

- **Lemma 7:** if $\gamma^{Weak}(\mathbf{m}) = \emptyset$, then $\exists i,\ 0 \notin \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{ii})$.

  Corollary. Together with Lemma 4, this proves correctness of our emptiness test.

  Proof. We prove this property by induction on the size n of the matrix.

  If $n = 0$, we have obviously $\gamma^{Weak}(\mathbf{m}) = \{(0)\} \iff 0 \in \gamma_{\mathcal{B}}(\mathbf{m}_{00})$, and $\gamma^{Weak}(\mathbf{m}) = \emptyset \iff 0 \notin \gamma_{\mathcal{B}}(\mathbf{m}_{00})$. By definition, we have $\mathbf{m}^{n+1}_{00} = \mathbf{m}_{00} \cap^\sharp_{\mathcal{B}} (\mathbf{m}_{00} +^\sharp \mathbf{m}_{00})$, so $0 \in \gamma_{\mathcal{B}}(\mathbf{m}_{00}) \iff 0 \in \gamma_{\mathcal{B}}(\mathbf{m}^{n+1}_{00})$.

  Suppose the property is true for some n. Let $\mathbf{m}$ be a matrix of size $n+1$ such that $\forall i,\ 0 \in \gamma_{\mathcal{B}}(\mathbf{m}^{n+2}_{ii})$, we prove that $\gamma^{Weak}(\mathbf{m}) \neq \emptyset$. Let $\mathbf{m}'$ be the matrix of size n constructed as follows: $\forall i, j < n, \mathbf{m}'_{ij} = \mathbf{m}_{(i+1)\,(j+1)} \cap^\sharp_{\mathcal{B}} (\mathbf{m}_{(i+1)\,0} +^\sharp \mathbf{m}_{0\,(j+1)})$. We have $\forall i, j,\ \mathbf{m}'_{ij} = \mathbf{m}^1_{(i+1)\,(j+1)}$, so $\forall i, j,\ \mathbf{m}'^{n+1}_{ij} = \mathbf{m}^{n+2}_{(i+1)\,(j+1)}$. We deduce that $\forall i,\ 0 \in \gamma_{\mathcal{B}}(\mathbf{m}'^{n+1}_{ii})$ and, by induction hypothesis, $\gamma^{Weak}(\mathbf{m}') \neq \emptyset$. Let us take $(x_1, \ldots, x_n)$ such that $\forall\, 1 \leq i, j,\ x_j - x_i \in \gamma_{\mathcal{B}}(\mathbf{m}'_{i-1\,j-1}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}_{ij})$.

  Let us prove that we can choose x_0 such that $\forall i,\ x_0 - x_i \in \gamma_{\mathcal{B}}(\mathbf{m}_{i0})$, and $x_i - x_0 \in \gamma_{\mathcal{B}}(\mathbf{m}_{0i})$. This will prove that $\forall\, 0 \leq i, j,\ x_j - x_i \in \gamma_{\mathcal{B}}(\mathbf{m}_{ij})$, implying $(x_1 - x_0, \ldots, x_n - x_0) gamma^{Weak}(\mathbf{m})$ and $\gamma^{Weak}(\mathbf{m}) \neq \emptyset$.

  First remark that $x_i - x_0 \in \gamma_{\mathcal{B}}(\mathbf{m}_{0i}) \iff x_0 - x_i \in \gamma_{\mathcal{B}}(-^\sharp \mathbf{m}_{0i}) \iff x_0 - x_i \in \gamma_{\mathcal{B}}(\mathbf{m}_{i0})$. Consider the set $C = \gamma_{\mathcal{B}}(\bigcap^\sharp_{\mathcal{B}\ 1 \leq i}(\{x_i\} +^\sharp \mathbf{m}_{i0}))$. Any $x_0 \in C$ will do and we only have to prove that $C \neq \emptyset$. Then $C \neq \emptyset$, or else, by Def. 5.2.5.6 there exists $i, j \geq 1$ such that $\gamma_{\mathcal{B}}((x^\sharp_i +^\sharp \mathbf{m}_{i0}) \cap^\sharp_{\mathcal{B}} (x^\sharp_j +^\sharp \mathbf{m}_{j0})) = \emptyset$, that is to say $x_j - x_i \notin \gamma_{\mathcal{B}}(\mathbf{m}_{i0} +^\sharp (-^\sharp \mathbf{m}_{j0})) = \gamma_{\mathcal{B}}(\mathbf{m}_{i0} +^\sharp \mathbf{m}_{0j})$, which is absurd because $x_j - x_i \in \gamma_{\mathcal{B}}(\mathbf{m}'^{n+1}_{(i-1)\,(j-1)}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}'_{(i-1)\,(j-1)}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}_{i0} +^\sharp \mathbf{m}_{0j})$. So C is not empty.

  Remark: the ability to represent exactly any singleton $[c, c]^\sharp$, as well as Def. 5.2.5.6 are crucial in the proof of this lemma.

- **Claim:** if $\gamma^{Weak}(\mathbf{m}) \neq \emptyset$, then $\forall i_0 \neq j_0$ and $c \in \gamma_{\mathcal{B}}(\mathbf{m}^{\star}_{i_0 j_0})$, there exists $\vec{x} \in \mathbb{I}^{n+1}$ such that $x_0 = 0$, $(x_1, \ldots, x_n) \in \gamma^{Weak}(\mathbf{m})$ and $x_{j_0} - x_{i_0} = c$.

  Proof. By induction on n. As the case $i_0 = j_0$ is obvious, we moreover suppose that $i_0 \neq j_0$.

  When $n = 1$ and $\gamma^{Weak}(\mathbf{m}) \neq \emptyset$, $\gamma^{Weak}(\mathbf{m}) = \gamma^{Weak}(\mathbf{m}^{\star}) = \{\ x_1 \mid x_1 \in \mathbf{m}^{\star}_{01}\ \}$. We can choose, without loss of generality, $i_0 = 0$, $j_0 = 1$, that is, $c \in \gamma_{\mathcal{B}}(\mathbf{m}^{\star}_{01})$. Then, the property is obvious.

  Suppose the property is true for some $n > 1$ and let $\mathbf{m}$ be a matrix of size $n+1$ with non-empty domain. We suppose also, without loss of generality, that $i_0, j_0 > 0$ ($n + 1 > 2$, so one can easily ensure $i_0, j_0 > 0$ using a simple variable permutation). We construct $\mathbf{m}'$ of size n as in Lemma 7: $\forall i, j < n, \mathbf{m}'_{ij} = \mathbf{m}_{(i+1)\,(j+1)} \cap^{\sharp}_{\mathcal{B}} (\mathbf{m}_{(i+1)\,0} \mathbin{+^{\sharp}} \mathbf{m}_{0\,(j+1)})$. Recall that $\forall i, j,\ \mathbf{m}'^{\star}_{ij} = \mathbf{m}^{\star}_{(i+1)\,(j+1)}$, so, in particular, $c \in \gamma_{\mathcal{B}}(\mathbf{m}'^{\star}_{i_0-1\,j_0-1})$.

  Applying the induction hypothesis to $\mathbf{m}'$, there exists $(x_1, \ldots, x_n)$ such that $\forall\, 1 \leq i, j,\ x_j - x_i \in \gamma_{\mathcal{B}}(\mathbf{m}'_{i-1\,j-1}) \subseteq \gamma_{\mathcal{B}}(\mathbf{m}_{ij})$ and $x_{j_0} - x_{i_0} = c$. Then, we can find x_0, as in Lemma 7, such that $\forall\, 0 \leq i, j,\ x_j - x_i \in \gamma_{\mathcal{B}}(\mathbf{m}_{ij})$. Thus $(x_1 - x_0, \ldots, x_n - x_0) \in \gamma^{Weak}(\mathbf{m})$ and verifies $x_{j_0} - x_{i_0} = c$ which ends the proof.

- **Claim:** if $\gamma^{Weak}(\mathbf{m}) \neq \emptyset$, then $\forall i, j$, $\gamma_{\mathcal{B}}(\mathbf{m}^{\star}_{ij}) = \inf_{\subseteq} \{\ \gamma_{\mathcal{B}}(\mathbf{n}_{ij}) \mid \gamma^{Weak}(\mathbf{m}) = \gamma^{Weak}(\mathbf{n})\ \}$.

  Proof. Let us first prove that $\gamma^{Weak}(\mathbf{m}^{\star}) = \gamma^{Weak}(\mathbf{m})$. We know that $\gamma^{Weak}(\mathbf{m}^{n+1}) = \gamma^{Weak}(\mathbf{m})$. On the one hand, $(x_1, \ldots, x_n) \in \gamma^{Weak}(\mathbf{m}^{n+1}) \implies (x_1, \ldots, x_n) \in \gamma^{Weak}(\mathbf{m}^{\star})$. On the other hand, as $\gamma^{Weak}(\mathbf{m}) \neq \emptyset$, Lemma 4 gives $\forall i,\ 0 \in \mathbf{m}^{n+1}_{ii}$, which means that $\gamma^{Weak}(\mathbf{m}^{\star}) \subseteq \gamma^{Weak}(\mathbf{m}^{n+1})$.

  Let us now consider $\mathbf{n}$ such that $\gamma^{Weak}(\mathbf{m}) = \gamma^{Weak}(\mathbf{n})$, and take any i, j and $c \in \gamma_{\mathcal{B}}(\mathbf{m}^{\star}_{ij})$. By the saturation property proved by the preceding claim, we have $\vec{x} \in \mathbb{I}^{n+1}$ such that $x_0 = 0$, $(x_1, \ldots, x_n) \in \gamma^{Weak}(\mathbf{m})$ and $x_j - x_i = c$. But hypothesis, we thus have $(x_1, \ldots, x_n) \in \gamma^{Weak}(\mathbf{n})$ which implies $x_j - x_i \in \gamma_{\mathcal{B}}(\mathbf{n}_{ij})$, so $c \in \gamma_{\mathcal{B}}(\mathbf{n}_{ij})$. This means that $\gamma_{\mathcal{B}}(\mathbf{m}^{\star}_{ij}) \subseteq \gamma_{\mathcal{B}}(\mathbf{n}_{ij})$, which completes the proof.

Bibliography

[AAB01] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *CAV'00*, volume 1855 of *LNCS*, pages 419–449. Springer, 2001. http://www.liafa.jussieu.fr/~abou/Papers/pdbm-cav00.ps.gz.

[AABB+03] Y. Aït-Ameur, G. Bel, F. Boniol, S. Pairault, and V. Wiels. Robustness analysis of avionics embedded systems. In *ACM LCTES'03*, pages 123–132. ACM Press, 2003. http://www.cert.fr/francais/deri/boniol/Papiers_pdf/LCTES03.pdf.

[AACFG92] Y. Aït Ameur, P. Cros, J.-J. Falcón, and A. Gómez. An application of abstract interpretation to floating-point arithmetic. In *WSA'92*, pages 205–212. Atelier Irisa, 1992. http://www.cert.fr/francais/deri/cros/Cros/Papers/WSA92.ps.

[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[Air] Airbus. http://www.airbus.com/.

[Asta] Astrée. Analyse Statique de logiciels Temps-RÉel embarqués (static analysis of critical real-time embedded software) analyzer page. http://www.astree.ens.fr/.

[Astb] Astrée. Analyse Statique de logiciels Temps-RÉel embarqués (static analysis of critical real-time embedded software) RNTL project page. http://www.di.ens.fr/~cousot/projets/ASTREE/.

[Bag97] R. Bagnara. *Data-Flow Analysis for Constraint Logic-Based Languages*. PhD thesis, Dipartimento di Informatica, Università di Pisa, Corso Italia 40, I-56125 Pisa, Italy, 1997. http://www.cs.unipr.it/~bagnara/Papers/Abstracts/PhDthesis.

[BCC+02] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software, invited chapter. In *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, LNCS, pages 85–108. Springer, 2002. http://www.di.ens.fr/~cousot/publications.www/BlanchetCousotEtAl-LNCS-v2566-p85-108-2002.pdf.

[BCC+03] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *ACM PLDI'03*, volume 548030, pages 196–207. ACM Press, 2003. http://www.di.ens.fr/~mine/publi/pldi045-blanchet.pdf.

[Bel58] R. Bellman. On a routing problem. In *Quarterly of Applied Mathematics*, volume 16, pages 87–90, 1958.

[BHRZ03] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. In *SAS'03*, volume 2694 of *LNCS*, pages 337–354. Springer, 2003. http://www.cs.unipr.it/~bagnara/Papers/Abstracts/SAS03.

[BHZ03] R. Bagnara, P. M. Hill, and E. Zaffanella. Widening operators for powerset domains. In *VMCAI'04*, volume 2937 of *LNCS*, pages 135–148. Springer, 2003. http://www.cs.unipr.it/~bagnara/Papers/Abstracts/Q349.

[BK89] V. Balasundaram and K. Kennedy. A technique for summarizing data access and its use in parallelism enhancing transformations. In *ACM PLDI'89*, pages 41–53. ACM Press, 1989. http://portal.acm.org/citation.cfm?id=74822&dl=ACM&coll=portal.

[Bou90] F. Bourdoncle. Interprocedural abstract interpretation of block structured languages with nested procedures, aliasing and recursivity. In Springer, editor, *PLILP'90*, volume 456 of *LNCS*, pages 307–323, 1990. http://www.exalead.com/Francois.Bourdoncle/plilp90.html.

[Bou93a] F. Bourdoncle. Abstract debugging of higher-order imperative languages. In *ACM PLDI'93*, pages 46–55. ACM Press, 1993. http://www.exalead.com/Francois.Bourdoncle/pldi93.html.

[Bou93b] F. Bourdoncle. Efficient chaotic iteration strategies with widenings. In *FMPA'93*, volume 735 of *LNCS*, pages 128–14. Springer, 1993. http://www.exalead.com/Francois.Bourdoncle/fmpa93.html.

[Bry86] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEETC*, C-35(8):677–691, 1986.

[BRZH02] R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In *SAS'02*, volume 2477 of *LNCS*, pages 213–229. Springer, 2002. `http://www.cs.unipr.it/~bagnara/Papers/Abstracts/SAS02`.

[Car97] L. Cardelli. Type systems. In *The Computer Science and Engineering Handbook*. CRC Press, 1997.

[CC76] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *ISOP'76*, pages 106–130. Dunod, Paris, France, 1976. `http://www.di.ens.fr/~cousot/COUSOTpapers/ISOP76.shtml`.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM POPL'77*, pages 238–252. ACM Press, 1977. `http://www.di.ens.fr/~cousot/COUSOTpapers/POPL77.shtml`.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *ACM POPL'79*, pages 269–282. ACM Press, 1979. `http://www.di.ens.fr/~cousot/COUSOTpapers/POPL79.shtml`.

[CC92a] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992. `http://www.di.ens.fr/~cousot/COUSOTpapers/JLP92.shtml`.

[CC92b] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992. `http://www.di.ens.fr/~cousot/COUSOTpapers/JLC92.shtml`.

[CC92c] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In *PLILP'92*, LNCS, pages 269–295. Springer, 1992. `http://www.di.ens.fr/~cousot/COUSOTpapers/PLILP92.shtml`.

[CC02] P. Cousot and R. Cousot. Modular static program analysis, invited paper. In *CC'02*, volume 2304 of *LNCS*, pages 159–178. Springer, 2002. `http://www.di.ens.fr/~cousot/COUSOTpapers/CC02.shtml`.

[CC04] R. Clarisó and J. Cortadella. The octahedron abstract domain. In *SAS'04*, volume 3148 of *LNCS*, pages 312–327. Springer, 2004. `http://www.lsi.upc.es/~jordicf/publications/pdf/sas2004.pdf`.

[Ce03] P. Černý. Vérification par interprétation abstraite de prédicats paramétriques. D.E.A. report, Univ. Paris VII & ENS-DI, Paris, France, 2003. http://www.cis.upenn.edu/~cernyp/.

[CEA] CEA. Fluctuat: a static analyzer for floating-point operations. http://www-drt.cea.fr/Pages/List/lse/LSL/Flop/index.html.

[CGP00] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 2000.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *ACM POPL'78*, pages 84–97. ACM Press, 1978. http://www.di.ens.fr/~cousot/COUSOTpapers/POPL78.shtml.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[Col96] C. Colby. *Semantics-Based Program Analysis via Symbolic Composition of Transfer Relations*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 1996. http://www-2.cs.cmu.edu/~rwh/theses/colby.pdf.

[Cou78] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. PhD thesis, Thèse d'état ès sciences mathématiques, Université scientifique et medicale de Grenoble, France, 1978.

[Cou99] P. Cousot. The calculational design of a generic abstract interpreter. In *Calculational System Design*. NATO ASI Series F. IOS Press, 1999. http://www.di.ens.fr/~cousot/COUSOTpapers/Marktoberdorf98.shtml.

[Cou02] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoretical Computer Science*, 277(1–2):47–103, 2002. http://www.di.ens.fr/~cousot/COUSOTpapers/TCS02-1.shtml.

[Cou03] P. Cousot. Verification by abstract interpretation. In *Proc. Int. Symp. on Verification – Theory & Practice – Honoring Zohar Manna's 64th Birthday*, volume 2772, pages 243–268. Springer, 2003. http://www.di.ens.fr/~cousot/COUSOTpapers/Zohar03.shtml.

[CS85] IEEE Computer Society. IEEE standard for binary floating-point arithmetic. Technical report, ANSI/IEEE Std 745-1985, 1985. http://grouper.ieee.org/groups/754/ and http://cch.loria.fr/documentation/IEEE754/.

[CS01] M. A. Colón and H. B. Sipma. Synthesis of linear ranking functions. In *TACAS'01*, volume 2031 of *LNCS*, pages 67–81, 2001. http://theory.stanford.edu/~sipma/papers/tacas01.html.

[Deu94] A. Deutsch. Interprocedural may-alias analysis for pointers: Beyond k-limiting. In *ACM PLDI'94*, pages 230–241. ACM Press, 1994. http://citeseer.ist.psu.edu/deutsch94interprocedural.html.

[DRS01] N. Dor, M. Rodeh, and M. Sagiv. Cleanness checking of string manipulations in C programs via integer analysis. In *SAS'01*, volume 2126 of *LNCS*. Springer, 2001. http://www.math.tau.ac.il/~nurr/.

[ea96] J. L. Lions et al. ARIANE 5, flight 501 failure, report by the inquiry board, 1996. http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html.

[Fer01] J. Feret. Occurrence counting analysis for the π-calculus. *GETCO'00*, 2(39), 2001. http://www.di.ens.fr/~feret/publication/getco2000.html.

[Fer04a] J. Feret. Abstract interpretation of mobile systems. *JLAP*, 2004. http://www.di.ens.fr/~feret/publication/jlap.html.

[Fer04b] J. Feret. Static analysis of digital filters. In *ESOP'04*, volume 2986 of *LNCS*. Springer, 2004. http://www.di.ens.fr/~feret/publication/esop2004.html.

[Fer05] J. Feret. The arithmetic-geometric progression abstract domain. In *VMCAI'05*, volume 3385 of *LNCS*. Springer, 2005. http://www.di.ens.fr/~feret/publication/esop2004.html.

[Ga] J. Garrigue and al. LablGTK2: an interface to the GIMP Tool Kit. http://wwwfun.kurims.kyoto-u.ac.jp/soft/olabl/lablgtk.html.

[GDD+04] D. Gopan, F. DiMaio, N. Dor, T. Reps, and M. Sagiv. Numeric domains with summarized dimensions. In *TACAS'04*, LNCS, pages 512–529. Springer, 2004. http://www.cs.wisc.edu/wpis/abstracts/tacas04.ndsd.abs.html.

[GGP+01] D. Guilbaud, É. Goubault, A. Pascalet, B. Starynkévitch, and F. Védrine. A simple abstract interpreter for threat detection and test case generation. In *WAPATV'01 in ICSE'01*, 2001. http://www.di.ens.fr/~goubault/papers/icse01.ps.gz.

[GM84] M. Gondran and M. Minoux. *Graphs and Algorithms*. Wiley, 1984.

[GMP] GMP. Gnu multiple precision library. http://www.swox.com/gmp/.

[Gol91] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5–48, 1991. http://www.wldelft.nl/soft/d3d/intro/misc/goldberg.pdf.

[Gou01] É. Goubault. Static analyses of floating-point operations. In *SAS'01*, volume 2126 of *LNCS*, pages 234–259. Springer, 2001. http://www.di.ens.fr/~goubault/papers/precision2.ps.gz.

[Gra89] P. Granger. Static analysis of arithmetical congruences. In *International Journal of Computer Mathematics*, volume 30, pages 165–190, 1989.

[Gra91] P. Granger. Static analysis of linear congruence equalities among variables of a program. In *TAPSOFT'91*, volume 493 of *LNCS*, pages 169–192. Springer, 1991.

[Gra92] P. Granger. Improving the results of static analyses programs by local decreasing iteration. In *FSTTCS*, volume 652 of *LNCS*, pages 68–79. Springer, 1992.

[Gra97] P. Granger. Static analyses of congruence properties on rational numbers. In *SAS'97*, volume 1302 of *LNCS*, pages 278–292. Springer, 1997.

[Hal79] N. Halbwachs. *Détermination automatique de relations linéaires vérifiés par les variables d'un programme*. PhD thesis, Université scientifique et medicale de Grenoble, France, 1979.

[Han75] E. R. Hansen. A generalized interval arithmetic. In *Interval Mathematics*, volume 29 of *LNCS*, pages 7–18. Springer, 1975.

[HS97] W. Harvey and P. Stuckey. A unit two variable per inequality integer constraint solver for constraint logic programming. In *ACSC'97*, volume 19, pages 102–111, 1997. http://www.icparc.ic.ac.uk/~wh/publications/ACSC97.ps.gz.

[HT98] M. Handjieva and S. Tzolovski. Refining static analyses by trace-based partitioning using control flow. In *SAS'98*, volume 1503 of *LNCS*, pages 200–214, 1998.

[Jea] B. Jeannet. New polka: A library to handle convex polyhedra in any dimension. http://www-verimag.imag.fr/~bjeannet/newpolka-english.html.

[Jea00] B. Jeannet. *Partitionnement Dynamique dans l'Analyse de Relations Linéaires et Application à la Vérification de Programmes Synchrones*. PhD thesis, Institut National Polytechnique de Grenoble, France, 2000.

[JMSY94] J. Jaffar, M. Maher, P. Stuckey, and H. Yap. Beyond finite domains. In *PPCP'94*, volume 874 of *LNCS*, pages 86–94. Springer, 1994. `http://citeseer.nj.nec.com/joxan94beyond.html`.

[Kar76] M. Karr. Affine relationships among variables of a program. *Acta Informatica*, pages 133–151, 1976.

[Kil73] G. Kildall. A unified approach to global program optimization. In *ACM POPL'73*, pages 194–206. ACM Press, 1973.

[LLPY97] K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *IEEE RTSS'97*, pages 14–24. IEEE CS Press, 1997. `http://www.docs.uu.se/docs/rtmv/papers/llpw-rtss97.ps.gz`.

[LV92] H. Le Verge. A note on Chernikova's algorithm. Technical Report 635, IRISA, France, 1992.

[LWYP99] K. Larsen, C. Weise, W. Yi, and J. Pearson. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999. `http://www.brics.dk/RS/98/46/`.

[Mal71] M. A. Malcolm. On accurate floating-point summation. *Commun. ACM*, 14(11):731–736, 1971. `http://portal.acm.org/citation.cfm?id=362889`.

[Man92] E. G. Manes. *Predicate Transformer Semantics*. Cambridge University Press, 1992.

[Mar02a] M. Martel. Propagation of rounding errors in finite precision computations: A semantics approach. In *ESOP'02*, volume 2305 of *LNCS*, pages 194–208. Springer, 2002. `http://www.enseignement.polytechnique.fr/profs/informatique/Matthieu.Martel/ESOP.ps`.

[Mar02b] M. Martel. Static analysis of the numerical stability of loops. In *SAS'02*, volume 2477 of *LNCS*, pages 133–150. Springer, 2002. `http://www.enseignement.polytechnique.fr/profs/informatique/Matthieu.Martel/sas02.ps`.

[Mas92] F. Masdupuy. Array abstraction using semantic analysis of trapezoid congruences. In *ACM ICS'92*, pages 226–235. ACM Press, 1992. `http://portal.acm.org/citation.cfm?id=143414&dl=ACM&coll=portal`.

[Mas93] F. Masdupuy. Semantic analysis of interval congruences. In *FMPTA'93*, volume 735 of *LNCS*, pages 142–155. Springer, 1993.

[Mas01] I. Mastroeni. Numerical power analysis. In *PADO II*, volume 2053 of *LNCS*, pages 117–137. Springer, 2001. http://profs.sci.univr.it/~mastroen/abstracts/pado01.abstract.html.

[Mau99] L. Mauborgne. *Representation of Sets of Trees for Abstract Interpretation.* PhD thesis, École Polytechnique, Palaiseau, France, 1999. http://www.di.ens.fr/~mauborgn/publi/these.html.

[Mau04] L. Mauborgne. ASTRÉE: verification of absence of run-time error. In *Building the Information Society (18th IFIP World Computer Congress)*, volume 156, pages 385–392. Springer, 2004. http://www.di.ens.fr/~mauborgn/publi/wcc04.html.

[MB83] M. Measche and B. Berthomieu. Time petri-nets for analyzing and verifying time dependent communication protocols. *Protocol Specification, Testing and Verification III*, pages 161–172, 1983.

[Mina] A. Miné. The octagon abstract domain library. http://www.di.ens.fr/~mine/oct/.

[Minb] A. Miné. On-line octagon abstract domain sample analyzer. http://cgi.di.ens.fr/cgi-bin/mine/octanalhtml/octanalweb/.

[Min00] A. Miné. Representation of two-variable difference or sum constraint set and application to automatic program analysis. D.E.A. report, Univ. Paris VII & ENS-DI, Paris, France, 2000. http://www.di.ens.fr/~mine/publi/report-mine-dea.pdf.

[Min01a] A. Miné. A new numerical abstract domain based on difference-bound matrices. In *PADO II*, volume 2053 of *LNCS*, pages 155–172. Springer, 2001. http://www.di.ens.fr/~mine/publi/article-mine-padoII.pdf.

[Min01b] A. Miné. The octagon abstract domain. In *AST 2001 in WCRE 2001*, IEEE, pages 310–319. IEEE CS Press, 2001. http://www.di.ens.fr/~mine/publi/article-mine-ast01.pdf.

[Min02] A. Miné. A few graph-based relational numerical abstract domains. In *SAS'02*, volume 2477 of *LNCS*, pages 117–132. Springer, 2002. http://www.di.ens.fr/~mine/publi/article-mine-sas02.pdf.

[Min04] A. Miné. Relational abstract domains for the detection of floating-point run-time errors. In *ESOP'04*, volume 2986 of *LNCS*, pages 3–17. Springer, 2004. http://www.di.ens.fr/~mine/publi/article-mine-esop04.pdf.

[MJ81] S. S. Muchnick and N. D. Jones. *Program Flow Analysis: Theory and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[MLAH99] J. Møller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard. Difference decision diagrams. In *CSL'99*, volume 1683 of *LNCS*, pages 111–125. Springer, 1999. http://www.it-c.dk/research/ddd/publications/ddd-csl-99.ps.

[Mona] D . Monniaux. C99-compatible c front-end for ocaml. http://www.di.ens.fr/~monniaux/download/c_parser.tar.gz.

[Monb] D. Monniaux. Caml-gmp, an extended precision computation library. http://caml.inria.fr/hump.html.

[Mon01] D. Monniaux. An abstract Monte-Carlo method for the analysis of probabilistic programs. In *ACM POPL'01*, volume 1824 of *ACM Press*, pages 93–101, 2001. http://www.di.ens.fr/~monniaux/biblio/Monniaux_POPL01.pdf.

[Moo66] R. E. Moore. *Interval Analysis*. Prentice Hall, 1966.

[MPF] MPFR. The multiple precision floating-point reliable library. http://www.loria.fr/projets/mpfr/.

[Nel78] C. G. Nelson. An n^{logn} algorithm for the two-variable-per-constraint linear programming satisfiability problem. Technical Report STAN-CS-78-689, Stanford University, Program Verification Group, 1978. http://www-db.stanford.edu/TR/CS-TR-78-689.html.

[OCa] OCaml. The objective caml system. http://paulliac.inria.fr/ocaml.

[Pol] PolySpace Verifier. http://www.polyspace.com/.

[PPL] PPL. The Parma Polyhedra Library. http://www.cs.unipr.it/ppl/.

[Pra77] V. Pratt. Two easy theories whose combination is hard. Technical report, Massachusetts Institute of Technology. Cambridge., 1977. http://boole.stanford.edu/pub/sefnp.pdf.

[Proa] GNU Project. Gcc: the GNU compiler collection. http://gcc.gnu.org/.

[Prob] GNU Project. GTK+: the GIMP Tool Kit, version 2. http://www.gtk.org/.

[RCK04a] E. Rodríguez-Carbonell and D. Kapur. An abstract interpretation approach for automatic generation of polynomial invariants. In *SAS'04*, volume 3148 of *LNCS*, pages 280–295. Springer, 2004. http://www.cs.unm.edu/~kapur/myabstracts/sas04.html.

[RCK04b] E. Rodríguez-Carbonell and D. Kapur. Automatic generation of polynomial loop invariants: Algebraic foundations. In *ACM ISSAC'04*, volume 505040, pages 266–273. ACM Press, 2004. http://www.cs.unm.edu/~kapur/myabstracts/issac04enric.html.

[Rey69] J. C. Reynolds. Automatic computation of data set definitions. *Information Processing'68*, pages 456–461, 1969.

[Ric53] H. G. Rice. Classes of recursively enumerable sets and their decision problems. In *Trans. Amer. Math. Soc.*, volume 74, pages 358–366, 1953.

[Rug04] R. Rugina. Quantitative shape analysis. In *SAS'04*, volume 3148 of *LNCS*, pages 228–245. Springer, 2004. http://www.cs.cornell.edu/~rugina/papers/sas04.ps.

[Sho81] R. Shostak. Deciding linear inequalities by computing loop residues. *Journal of the ACM*, 28(4):769–779, 1981. http://portal.acm.org/citation.cfm?id=322288&dl=ACM&coll=portal.

[SK02] A. Simon and A. King. Analyzing string buffers in C. In *ICAMST*, volume 2422 of *LNCS*, pages 365–379. Springer, 2002. http://www.cs.kent.ac.uk/pubs/2002/1367/index.html.

[Ske92] R. Skeel. Roundoff error and the Patriot missile. *SIAM News*, 25(4):11, 1992. http://www.siam.org/siamnews/general/patriot.htm.

[SKH02] A. Simon, A. King, and J. Howe. Two variables per linear inequality as an abstract domain. In *LOPSTR'02*, volume 2664 of *LNCS*, pages 71–89. Springer, 2002. http://www.cs.kent.ac.uk/pubs/2002/1515.

[SKS00] R. Shaham, E. K. Kolodner, and M. Sagiv. Automatic removal of array memory leaks in java. In *CC'00*, LNCS, pages 50–66. Springer, 2000. http://www.math.tau.ac.il/~rans/cc00.ps.gz.

[SW04] Z. Su and D. Wagner. A class of polynomially solvable range constraints for interval analysis without widenings and narrowings. In *TACAS'04*, LNCS, pages 280–295. Springer, 2004. http://www.cs.berkeley.edu/~daw/papers/range-tacas04.ps.

[Tar55] A. Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–310, 1955.

[TCR94] D. Toman, J. Chomicki, and D. S. Rogers. Datalog with integer periodicity constraints. In *Journal of Logic Programming*, pages 189–203. The MIT Press, 1994. `http://citeseer.nj.nec.com/toman94datalog.html`.

[VACS94] M. Vinícius, A. Andrade, J. L. D. Comba, and J. Stolfi. Affine arithmetic. In *INTERVAL'94*, 1994. `http://www.dcc.unicamp.br/~stolfi/EXPORT/papers/by-tag/and-com-sto-94-aax.ps.gz` (draft).

[Ven02] A. Venet. Nonuniform alias analysis of recursive data structures and arrays. In *SAS'02*, volume 2477 of *LNCS*, pages 36–51. Springer, 2002. `http://ase.arc.nasa.gov/people/venet/sas02.ps`.

[Ven04] A. Venet. A scalable nonuniform pointer analysis for embedded programs. In *SAS'04*, volume 3148 of *LNCS*, pages 149–164. Springer, 2004. `http://ase.arc.nasa.gov/people/venet/sas04.pdf`.

[Vig96] J. Vignes. A stochastic approach of the analysis of round-off error propagation: A survey of the CESTAC method. In *Proc. of the Second Real Numbers and Computer Conf.*, pages 233–251, 1996.

[Yov98] S. Yovine. Model-checking timed automata. In *Embedded Systems*, volume 1494 of *LNCS*. Springer, 1998. `http://www-verimag.imag.fr/~yovine/articles/embedded98.pdf`.

List of Figures

List of Definitions

List of Theorems

List of Examples

Index

Index of Symbols