Static Analysis for Detecting and Avoiding Floating-Point Run-Time Errors in Logic Programs

by

Javier Núñez Fontarnau

Submitted in accordance with the requirements for the degree of Doctor of Philosophy.

The University of Leeds
School of Computing

June 2006

The candidate confirms that the work submitted is his own and that the appropriate credit has been given where reference has been made to the work of others.

Abstract

The aim of this thesis is to provide techniques for the abstraction of floating-point expressions into the polyhedra domain as well as into the finite powerset of polyhedra domain. Moreover, this thesis aims at presenting a forward and a backward analysis for the detection and inference of floating-point errors such as overflow and division by zero. These techniques are based on abstract interpretation, which is a theory for the sound approximation of the semantics of programs. These abstractions and analyses have important applications for instance in engineering, mechanics and computer aided graphics design.

The abstraction of floating-point expressions into polyhedra includes two stages. In the first, we present an approximation of floating-point expression by a polynomial with interval coefficients that includes the possible floating-point errors. In the second stage we present techniques for abstracting polynomials with interval coefficients into polyhedra and polyhedra powersets. Moreover, we present a technique for abstracting expressions in which a division by zero may occur.

A forward analysis for the detection of overflow and division by zero is then presented. This analysis is particular in that it always reaches the output of a programs. This is an important requirement since the backward analysis needs information on the output to infer which input to a program could cause an overflow or a division by zero. Such analyses are important especially in the design of systems with high contents of nonlinear floating-point expressions. Experimental results show the usability of these analyses.

Acknowledgements

My supervisor Dr Patricia M. Hill, or Pat for short, deserves much gratitude for her support, for her critical spirit, for her tireless enthusiasm for perfection, and especially for her immeasurable patience over these years. Feedback and comments on research have been helpful throughout these years. For this reason, mention is also owned to the rest of the group in Leeds, Parma and Verona: Vajirapan Panumong, Katy Dobson, Matthew Mundell, Dr Fausto Spoto, Dr Enea Zaffanella and Prof Roberto Bagnara.

Many thanks are owned to The School of Computing, University of Leeds, for partly funding this research work, and to Dr Raymond Kwan for his patient consideration on my research proposal corrections. Gratitude is owned to Netsanet Mebrate for his intuitive and stimulating discussions, as well as his encouragement and optimism. Thanks also to Dr Iain Stalker is mentioned with gratitude for his support and helpful advices — and none the less for his good sense of humour.

The examiners of this thesis, Dr John Stell and Dr Andrew King, are owned a very honest and deep gratitude for their effort and for providing pertinent comments and suggestions for improving this work.

Dr Dolores Rexachs del Rosario is kindly mentioned for her mentoring and support in my undergraduate studies. Dr Nieves Díaz García-Mauriño is mentioned for her optimism and dedication to education.

Close friends are owned much acknowledge as well. Miguel Ángel Sánchez Monedero is kindly mentioned for his friendship and good laughing times. Sami Kilpeläinen is kindly mentioned for his friendship, and the good times in our photographic safaris around Europe.

I would like to express my appreciation and profound and everlasting gratitude to my parents Jesús and Antonia, to my sister Olalla and to my wife Minna, who have supported me during my studies. I could have not written this thesis without their persistent support and dedication. I am thankful to them for that and therefore I would like to dedicate this thesis to them.

June 2006
Leeds

Contents

List of Figures

"And now tell me" — in the end I could not restrain myself — " how did you manage to know ? "
"My good Adso," my master said, "during our whole journey I have been teaching you to recognize the evidence through which the world speaks to us like a great book. Alanus de Insulis said that

OMNIS MUNDI CREATURA
QUASI LIBER ET PICTURA
NOBIS EST IN SPECULUM

and he was thinking of the endless array of symbols with which God, through His creatures, speaks to us of the eternal life. But the universe is even more talkative than Alanus thought, and it speaks not only of the ultimate things (which it does always in an obscure fashion) but also of closer things, and then it speaks quite clearly."

The Name of the Rose
Umberto Eco

Devotedly to my Parents, my Sister and my Wife

Chapter 1

Introduction

1.1 The Cost of Finiteness

Computer systems are more than ever omnipresent in our daily lives without even noticing it: air traffic control systems, on-board avionics systems, microprocessors, stock market systems, traffic lights systems, train and underground systems. Due to the increasing sophistication of systems, verification of software has become paramount. Numerical errors, for instance, may compromise safety of lives, lead to drastic financial consequences or even endanger the military security. In 1982, the Vancouver stock exchange introduced an index with a nominal value of $1,000,000$ [1] [MV99]. After almost two years of recomputing the index and truncating it to 3 decimal digits at each exchange, the index had a value of $524,881$, whereas its *true* value should have been $1,009,811$.

Another case is the well-known Patriot missile error in the Gulf War (1991) [Ske92, Off92]. The Patriot missile system radar failed to track and intercept an incoming Scud missile leading to loss of lives. This problem originated from the tracking frequency in the radar. Radar pulse bounces were measured in intervals of 0.1 seconds, which had an inaccurate representation in the floating point system employed. After 100 hours of operation, the accumulated error due to the inaccurate representation caused the system to lose track of its target. An automated *compile-time* verification of the numerical properties of the system software could have prevented this error.

Large economic and infrastructural losses are included amongst the consequences of numerical errors. A famous case is that of Ariane launching. On June 4^{th} 1996, the

[1] News reported both in *The Toronto Star*, November 19, 1983; and in *The Wall Street Journal*, page 37, November 8, 1983.

Ariane 501 launching [Age96] failed after 37 seconds of the start of the flight sequence, at an altitude of 3700 meters, due to an overflow in a register: the horizontal bias of the flight was measured in a 64 bit floating point register, which was cast to a 16 bit integer. The overflow was not caught and the exception handler concluded an impossible situation, which resulted in the rocket veering abruptly and self-destroying. Though, the *real* trajectory was as expected. The cost of software development was estimated in 7 billion dollars, and hardware components, including the rocket, in 500 million dollars.

Security may be also imperilled. In 1998, a crew member of the guided-missile cruiser USS Yorktown mistakenly entered a zero for a data value. This resulted in a division by zero: such an error propagated and eventually shut down the ship's propulsion system. The ship was out of operation for several hours [2]. It seems clear that the software was not protected against division by zero; namely, no conditions were included to prevent the input of values that could cause a division by zero in the system.

Thus, tools for detecting and preventing numerical errors in software can be of great utility.

1.2 Motivation

Consider the problem of specifying and then verifying embedded systems, where external elements in the *physical* world interact with program interfaces. This has been referred by Patrick Cousot as the *Grand Challenge* [Cou05]. Cousot advocates that a full abstraction of both the program and a reactive (physical) environment is possible despite the dichotomy between continuous domains, used for the abstraction of physical systems, and discrete domains, used in the implementations. This integrated abstraction, both of the program and a reactive environment, is fundamental to achieve a *full system verification*.

We approach Cousot's *Grand Challenge* from a much narrower point of view. We do not verify an embedded system together with its software interfaces. Instead we implement a model for a physical system in a prototype, and verify numerical properties related to the errors inherent in floating-point arithmetic, in particular floating-point overflow. Moreover, we introduce the issue as to whether the *supervised* development of embedded systems, namely *supervised* by verification methods from early development stages such as the design stage, reduces the verification effort, compared to the verification of a whole system once fully implemented.

1.2.1 Semantics of Floating-Point Arithmetic

In general, the correctness of programs cannot be fully guaranteed using testing techniques. Especially in large programs, it is impractical to generate the set of all possible input to a program and then verify that the resulting output satisfies a set of requirements. For

[2] Scientific American, November 1998.

instance, an expression such as $\sqrt{x^2 + y^2}$ defined for an 80 bit floating-point system has $2^{80} + 2^{80}$ possible input to verify, and some of these inputs could cause errors. Floating-point numerical systems are in fact a coarse approximation to the set of *mathematical* reals $\mathbb{R}$. A finite representation is used to approximate reals, which incurs in an error in precision. For instance, a value such as $\frac{1}{3} = 0.333\ldots \in \mathbb{R}$ is approximated by a floating point with a finite number of decimal digits; namely, for a precision of, say, 5 decimal digits, $\frac{1}{3} \approx 0.33334$ or $\frac{1}{3} \approx 0.33332$, depending on the *rounding* method applied. For successive computations, this rounding error may lead to unintended semantics in a program. Yet worse, it may lead the execution of a program to an unexpected halt. Tools for verifying the correctness with respect to floating point arithmetic in programs before execution are required.

Static analyses can verify a property of interest in a program without actually generating all the possible states of an instruction (or program), hence not requiring the verification of *infinitely* many possible execution states. Amongst the existing *compile-time* analysis techniques for verification, abstract interpretation is unusual in that it has sound foundations in lattice and domain theory. Abstract interpretation was introduced in [CC77] and comprehensively surveyed for logic programs in [CC92a]. The basic idea behind abstract interpretation is that the (concrete) semantics of a program may be approximated by an abstract semantics. Properties of the abstract semantics such as precision depends on the notion of abstraction, which is specified by the selected abstract domain [CC79]. Thus the concept of abstract domains is central in the construction of abstract interpretations.

Central in abstract interpretation is the abstract domain; namely, a set of properties of interest and a set of operators on these properties. The purpose of abstract domains is that of approximation of concrete properties of programs. Numerical abstract domains, for instance, are adequate for the abstraction of numerical properties of variables in a program. These domains are in fact useful for the abstraction and reasoning of properties of floating-point arithmetic in programs and in particular, for the analysis of overflow and division by zero. When choosing a numerical abstract domain for analysing properties of programs, we must considered issues such as precision and cost. In this way, a generic criteria to classify numerical abstract domains is that of relational, weakly relational and nonrelational domains. The first class includes the polyhedra domain [CH78], which describes relationships between variables. Though computationally expensive, this domain leads to precise abstractions. The last class includes the interval domain [CC76]. Despite the low cost of processing, no relation between variables is represented, which leads to imprecise abstractions. However, the class of weakly relational domains aim at achieving a compromise between both precision and cost. This class includes for instance the octagon domain [BK89, Min00, Min01, Min04a, Min06a] and the octahedron domain [CC04].

1.2.2 Verification of Programs at Implementation Stage

Static analyses based on abstract interpretation for large amounts of code of critical embedded systems has been developed in [BCC+03, Min04b, Min04a]. These analyses verify that for instance no floating-point run-time error due to overflow occurs. For this purpose, Antoine Miné presented a relational analysis for floating-point arithmetic based on the octagon domain. This was the first time a relational domain had been adapted and used for analysing large amounts of code.

Miné presents a novel technique to adapt symbolic expressions with interval coefficients into relational domains. Symbolic expressions hold algebraic properties such as associativity and distributivity that make them amenable for abstracting floating-point expressions. Moreover, Miné develops symbolic manipulation techniques for processing abstractions into the octagon domain [Min04a, Min06b]. This abstraction can represent relations between variables, which can deliver more precise analyses. However, for highly nonlinear floating-point expressions, Miné's symbolic techniques may not provide precise abstractions. This loss of information due to nonlinearity may be crucial in numerical static analyses for instance for the detection and inference of conditions for division by zero.

For the purpose of analysing programs with nonlinear floating-point expressions, we present new abstraction techniques that deliver sets of polyhedra. These techniques are based on interval valued polynomials. We apply an Approximation Theory method called *Bernstein approximation* to abstract such polynomials into the polyhedra domain. Using these techniques, we can attain any necessary precision for nonlinear expressions. From these abstractions into the polyhedra domain, we develop relational analyses for the detection and inference of floating-point overflow.

1.2.3 Verification of Prototypes at Design Stage

Another approach to ensure the absence of floating-point run-time errors in a program is by verifying prototypes at the earlier software development stage of design. Instead of analysing large programs, we can analyse small software components, which once guaranteed to be absent of numerical errors, can be included in a larger system and reused in further software developments. The benefit of verifying small components at the design stage is that we can specialise the analyses to examine the nonlinearity of expressions, which can be of some importance for detecting and avoiding run-time errors.

Though floating point rounding errors are common to any programming language that includes floating-point systems, some languages may prove more suitable for certain purposes than others. For example, imperative and object-oriented programming languages are widely used in *real-world* software applications. Logic programming languages, on the contrary, are delegated to the prototyping part of the implementation. A distinctive feature of logic programming languages over other prototyping languages is that of describing

or declaring a problem. Thus a language based on logic has the capability of verification that is required in the development of software systems. Logic programming is founded in a sound mathematical basis. Introduced in the 70's (see for instance [vEA76]), it has been evolved throughout the years to improve several issues related to computational efficiency in addition to adapting logic programming interpreters to particular hardware, and the introduction of compilers, a considerable effort has been devoted to developing *compile-time* automatic analyses for the verification of programs' properties.

We construct logic programming prototypes from mathematical models for (complex) physical systems. Note that such models are commonly defined in $\mathbb{R}$, which provides sound arithmetic. However, the computational version of such systems is based on floating-point arithmetic, which is unsound, and where algebraic properties of $\mathbb{R}$ such as distributivity and associativity do not hold in general in floating-point arithmetic. Thus, a model which is specified in $\mathbb{R}$ and prototyped in floating-point arithmetic must undergo a static analysis before developing the prototype toward the final implementation.

In this thesis we present precise abstractions of nonlinear expressions into the polyhedra domain and develop specialist analyses for the detection of floating-point overflow. In addition, we develop a backward analysis for the inference of conditions to avoid overflow, such as for instance overflow caused by division by zero.

1.3 Related Work

Numerical Abstract Domains. Central in analyses based on abstract interpretation is the abstract domain, by which program invariants are represented. The first numerical abstract domain used in the abstract interpretation framework was the interval domain [CC76] [3]. This domain has a simple representation and has defined efficient operators. However, it does not capture relations between variables. Relational domains, on the other hand, allow for more precise abstractions, as relations between variables can be captured. A full range of relational domains are nowadays available, spawning from the highly precise polyhedra domain [CH78, Hal79, BRZH02, BHRZ03b], to the class of weakly-relational domains, which includes for instance the octagon domain [BK89, Min00, Min01, Min04a, Min06a], the *two variables per inequality* domain [SKH02], as well as the octahedron domain [CC04]. Operators on weakly-relational domains can be implemented in an efficient way. This is achieved at the expense of precision. Weakly-relational domains, though, achieve a compromise between precision and efficiency. Nonlinear domains include for instance the ellipsoids [Fer04].

Nonrelational Analyses. Analyses based on abstract interpretation where the abstract domain cannot represent relations between variables are called nonrelational anal-

[3] Interval analysis was already considered by Turing and Wilkinson in the 1940s [Wil80, p. 104]. The first publications appeared in the 1960s and 70s [Moo66, GL70, Moo79].

yses. The nonrelational domain of intervals has led to several works on precision analyses. Precision analyses aim at determining the difference between the expected result assuming sound arithmetic and the actual result obtained by floating-point computation, with unsound arithmetic. Abstract interpretation-based precision analyses can be found in [Gou01, GMP01, Mar02, Mar, GMP, GMP06]. Another work that uses the interval domain to study the influence of *noise* in the numerical input in embedded systems can be found in [ABB+03]. Moreover, in [Mar02], an analysis for stability of looping in programs with floating-point arithmetic is presented.

Relational Analyses. These analyses are characterised by the gain in precision compared to nonrelational analyses, as they can represent relations between variables. Common relational domains were already cited in the motivation section, and included for instance the octagon domain, the *two variables per inequality* domain or the polyhedra domain. The first relational analysis for numerical expressions based on the polyhedra domain was presented in [CH78]. This analysis defined invariants calculation, widening, narrowing and transfer functions for inequality tests and assignments as well as operations for merging execution paths. However, the problem of nonlinearity of expressions is not fully addressed. An important extension of this analysis can be found in [Min04b], in which a relational analysis for detecting floating-point overflow is presented. In this work, Antoine Miné bases the analysis in abstract interpretation and abstracts linear and also nonlinear expressions into the octagon domain, which achieves a compromise between cost and efficiency. In order to abstract floating-point expressions into octagons, he introduces a symbolic form based on intervals. We will propose an analytical version of symbolic forms, based on interval valued polynomial (that is to say, polynomials with interval coefficients), that will ease the abstraction of floating-point expressions into the polyhedra domain as well as the finite powerset of polyhedra domain.

Backward Analyses. Backward analysis in the context of abstract interpretation was presented in [CC79]. and studied for functional programs for instance in [WH98, HW89], for imperative programs in [CC79, CC82], and for logic programs in [DGB92, FHW00, GC01, KL02, KL03, HKL04]. Of particular interest is the analysis presented in [KL02], where Andrew M King and Lunjin Lu developed a backward analysis based on condensing domains (condensing domains are those equipped with a pseudo-complement operator, which allows for tracing data-flow information backward). That work presents a novel approach to inferring preconditions in the moding of logic programs for avoiding grounding errors. Further development of that work led to [KL03], in which the authors used abstract interpretation to show the relationship between forward and backward analysis verification power. Both of these works will prove central in the development of the backward analysis introduced in this thesis. On the other hand, backward analyses that do not require condensing domains include for instance type inference analysis [LK02] and backward

sharing analysis [LK04]. A more recent work on backward sharing analysis can be found in [LL05], which computes maximal sets of preconditions by reformulating the problem to be that of finding the set of all maximal models of a boolean formula.

1.4 Structure of the Thesis

The present chapter motivates this thesis and lists the main contributions. Chapter 2 presents preliminary definitions used throughout.

The rest of the thesis has two main parts. The first part includes chapters 3, 4 and 5, and presents a series of techniques for the abstraction of floating-point arithmetic into polyhedra and the powerset of polyhedra. Chapter 3 formulates the abstraction by means of abstract interpretation, a theory for sound approximations of program semantics. Chapter 4 introduces the core techniques for polyhedra abstraction, which are based on Bernstein expansion. Numerical expressions with possible divisions by zero are treated in chapter 5, in which a *branch and prune* algorithm is presented.

The second part includes chapters 6 and 7, and presents a series of analyses for detecting and avoiding floating-point run-time errors in logic programs. These analyses are based on the abstractions presented in the first part. Chapter 6 presents a forward analysis for the detection of floating-point overflow. Chapter 7 presents a backward analysis for the inference of conditions to avoid overflow, such as for instance overflow caused by possible divisions by zero. To infer these conditions, the backward analysis requires the result of the forward analysis. Chapter 8 presents experimental results for both the first and second parts. Chapter 9 concludes.

1.5 Contributions

The main contributions in this thesis are as follows.

- The introduction of a nonlinear abstract semantics based on interval valued polynomials that preserves nonlinear information, which proves important to avoid loss of precision.
- The abstraction of floating-point polynomial expressions into the polyhedra domain and floating-point polynomial quotients into the polyhedra powerset.
- The introduction of a forward analysis based on the polyhedra domain for the detection of floating-point overflow and overflow due to division by zero.
- The introduction of a backward analysis based on the powerset of polyhedra for the inference of numerical properties in logic programs that perform floating-point arithmetic operations. The analysis is specialised to the inference of conditions to avoid floating-point overflow, including overflow due to division by zero.

Chapter 2

Preliminaries

2.1 Notation

For a set S, $|S|$ denotes the cardinality of S. Moreover, $\wp(S)$ is the powerset of S; $\wp_{\text{fn}}(S)$ is the set of all *finite* subsets of S. For a syntactic object s, $\mathsf{vars}(s)$ denotes the set of variables occurring in s. The set of natural numbers is denoted by $\mathbb{N}$ and the set of natural numbers greater than 0 by $\mathbb{N}^+$. A sequence of n objects is denoted by $\tilde{o} = \langle\, o_1, \ldots, o_n \,\rangle$. A k vector in $\mathbb{N}^k$ is denoted by $I = (i_1, \ldots, i_k)$. Let $I, J, N \in \mathbb{N}^k$ be k vectors where $I = (i_1, \ldots, i_k)$, $J = (j_1, \ldots, j_k)$ and $N = (n_1, \ldots, n_k)$. We say that $I \leq N$ if $i_j \leq n_j$, for $j \in [1, k]$. We denote by $\mathbf{0}$ the *zero* vector, with all the components equal to zero, and by $\mathbf{1}$ the *unit* vector, with all the components equal to one. Note that $\mathbf{0} \leq I$ for all $I \in \mathbb{N}^k$. Given $\ell \in [1, k]$, the ℓ-th unit vector is the vector with 1 in the ℓ-th position and zeroes in every other position. Vector addition is defined as $I + J = (i_1 + j_1, \ldots, i_k + j_k)$. The k-dimensional vector space on the field of real numbers $\mathbb{R}$ is denoted by $\mathbb{R}^k$. A vector in $\mathbb{R}^k$ is denoted by $\overline{\mathbf{x}} = (x_1, \ldots, x_k)$; the scalar product of its components by $\mathbf{x} = x_1 \cdot \ldots \cdot x_k \in \mathbb{R}$; and, for any $\lambda \in \mathbb{R}$, we let $\lambda\overline{\mathbf{x}} = (\lambda \cdot x_1, \ldots, \lambda \cdot x_k)$. Given $I \in \mathbb{N}^k$, we let $\mathbf{x}^I = x_1^{i_1} \cdot \ldots \cdot x_k^{i_k}$.

2.2 Polynomials

A univariate polynomial of degree n is a mapping $p : \mathbb{R} \to \mathbb{R}$ defined as

$$p(x) = \sum_{i=0}^{n} a_i \cdot x^i \ , \ a_i \in \mathbb{R} \ ,$$

where $a_n \neq 0$ and where each a_i and each $a_i \cdot x^i$ are called respectively a *polynomial coefficient* and a *monomial*. The set of all univariate polynomial functors of degree less than or equal to n is denoted $Poly^{n,1}$. A k-variate polynomial of degree $N \in \mathbb{N}^k$, $k \geq 0$, is a mapping $p : \mathbb{R}^k \to \mathbb{R}$ defined for $S \subseteq \{I \in \mathbb{N}^k \mid I \leq N\}$, by

$$p(\overline{\mathbf{x}}) = \sum_{I \in S} a_I \cdot \mathbf{x}^I \ , \ a_I \in \mathbb{R} \ .$$

with polynomial coefficients a_I and where each $a_I \cdot \mathbf{x}^I$ is called a term. We refer to a term of a polynomial as a *polynomial term*. We call a k-variate polynomial a multivariate polynomial. The set of all multivariate polynomial functors of k variables of degree less than or equal to N is denoted $Poly^{N,k}$.

Example 2.1. *Let* $\overline{\mathbf{x}} = (x_1, x_2, x_3)$. *Then, a polynomial* $p \in Poly^{N,3}$ *such as* $p(\overline{\mathbf{x}}) = 2 \cdot x_2^3 \cdot x_1^7 \cdot x_3 + 0.5 \cdot x_3$ *may be denoted, for* $S \subseteq \{(7,3,1),(0,0,1)\}$, $N = (7,3,1)$, *by* $p(\overline{\mathbf{x}}) = 2 \cdot \mathbf{x}^{(7,3,1)} + 0.5 \cdot \mathbf{x}^{(0,0,1)}$. *Likewise,* $q(\overline{\mathbf{x}}) = 3.17$ *may be denoted, for* $S \subseteq \{\mathbf{0}\}$, *by* $q(\overline{\mathbf{x}}) = 3.17 \cdot \mathbf{x}^{(0,0,0)}$.

We say that $p(\overline{\mathbf{x}}) = \sum_{I \in S} \ a_I \cdot \mathbf{x}^I$ and $q(\overline{\mathbf{x}}) = \sum_{J \in S'} \ b_J \cdot \mathbf{x}^J$ are equal, denoted by $p(\overline{\mathbf{x}}) = q(\overline{\mathbf{x}})$, if $S = S'$ and $\forall I \in S$, $a_I = b_I$.

2.3 Orders and Lattices

We follow standard notation as presented in [Bir67] and [DP90]. Let S be a set. For any $a, b, c \in S$, a binary operation $\diamond : S \times S \to S$ is commutative if $a \ \diamond \ b = b \ \diamond \ a$, and associative if $(a \diamond b) \diamond c = a \diamond (b \diamond c)$. A *partial ordering* is a relation $\leq : S \times S \to \{0,1\}$ that is *reflexive* (i.e. $\forall a \in S, \ a \leq a$), *transitive* (i.e. $\forall a, b, c \in S, a \leq b \ \wedge \ b \leq c \ \Rightarrow \ a \leq c$), and *anti-symmetric* (i.e. $\forall a, b \in S, a \leq b \ \wedge \ b \leq a \ \Rightarrow \ a = b$). A partially ordered set (or poset) is a set P equipped with a partial ordering $\leq$ and denoted $\langle P, \leq \rangle$. We say that $a, b \in P$ are comparable if and only if $a \leq b$ or $b \leq a$. If $\langle P, \leq \rangle$ is a poset, then $C \subseteq P$ is a chain if and only if $a \leq b$ or $b \leq a$ for all $a, b \in C$. A poset $\langle P, \leq \rangle$ with a binary operation meet, denoted $\wedge$, where $x \wedge y$ is the greatest lower bound of $x, y \in P$, is called a *meet-semilattice* and is represented by $\langle P, \leq, \wedge \rangle$. If m and m' are both meets of x and y, then $m \leq m'$ and $m' \leq m$, and so $m = m'$. A poset $\langle S, \preceq \rangle$ is said to be well-ordered if $\forall S' \subseteq S \setminus \{\emptyset\}$, $\exists a \in S'$ where $\forall b \in S'$, $a \preceq b$. A complete lattice $\langle L, \leq, \vee, \wedge, \top, \bot \rangle$, with partial ordering $\leq$, lub $\vee$, glb $\wedge$, greatest element (top) $\top$, and least element (bottom) $\bot$, is denoted $\widehat{L}$. A complete lattice $\widehat{L}$ is *completely meet-distributive* if, for each $x \in L$ and $Y \subseteq L$, $x \wedge \bigvee Y = \bigvee_{y \in Y} (x \wedge y)$ holds.

2.4 Interval Arithmetic

The concept of interval arithmetics was presented by R. E. Moore in [Moo66]. A *closed compact interval* $[a^-, a^+]$ is defined as the closed set of real values of the form

$$[a^-, a^+] = \{c \in \mathbb{R} \mid a^- \leq c \leq a^+\},$$

where a^- is called the *lower bound* and a^+ the *upper bound*. The width of an interval $[a^-, a^+]$ is defined as the difference $a^+ - a^-$. We say that $c \in \mathbb{R}$ is in $[a^-, a^+]$, and denote it $c \in [a^-, a^+]$, if $a^- \leq c \leq a^+$. Moreover, we say that $[b^-, b^+]$ is included in $[a^-, a^+]$ (or that $[b^-, b^+]$ is a subinterval of $[a^-, a^+]$), and denote it $[b^-, b^+] \subseteq [a^-, a^+]$, if $a^- \leq b^-$ and $b^+ \leq a^+$. Further, $[a^-, a^+]$ and $[b^-, b^+]$ are equal, denoted $[a^-, a^+] = [b^-, b^+]$, if $[a^-, a^+] \subseteq [b^-, b^+]$ and $[b^-, b^+] \subseteq [a^-, a^+]$.

The set union of $[a^-, a^+]$ and $[b^-, b^+]$, denoted by $\cup$, is defined as $[a^-, a^+] \cup [b^-, b^+] = \{c \in \mathbb{R} \mid \ c \in [a^-, a^+] \text{ or } c \in [b^-, b^+]\}$. Moreover, the interval union (or *interval hull*) of $[a^-, a^+]$ and $[b^-, b^+]$, denoted, $\sqcup$, is defined as $[a^-, a^+] \sqcup [b^-, b^+] = [\min(a^-, b^-), \max(a^+, b^+)]$. The set of all closed intervals $[a^-, a^+]$ for each $a^-, a^+ \in \mathbb{R}$ where $a^- \leq a^+$ is denoted $\mathbb{IR}$. An interval $[a^-, a^+]$ in which $a^- = a^+$, is called a *degenerate* interval, and an interval $[a^-, a^+]$ where $a^- < a^+$ is called a *proper* interval.

Unary and binary arithmetic operators on intervals, for $n \in \mathbb{N}$, are defined as follows:

$$\begin{aligned}
[a^-, a^+] \ \boxplus \ [b^-, b^+] &= [a^- + b^-, a^+ + b^+] \\
[a^-, a^+] \ \boxminus \ [b^-, b^+] &= [a^- - b^+, a^+ - b^-] \\
[a^-, a^+] \ \boxtimes \ [b^-, b^+] &= [\min((a^+ \cdot b^+), (a^- \cdot b^+), (a^+ \cdot b^-), (a^- \cdot b^-)), \\
&\qquad \max((a^+ \cdot b^+), (a^- \cdot b^+), (a^+ \cdot b^-), (a^- \cdot b^-))] \\
[a^-, a^+] \ \boxslash \ [b^-, b^+] &= \begin{cases} [a^-, a^+] \ \boxtimes \ [1/b^+, 1/b^-] & \text{if } b^- \cdot b^+ > 0 \\ undefined & \text{if } 0 \in [b^-, b^+] \end{cases}
\end{aligned}$$

and

$$[a^-, a^+] \ \boxdot \ n = \begin{cases} [1, 1] & \text{if } n = 0 \\ [0, \max\left((a^-)^n, (a^+)^n\right)] & \text{if } n \neq 0 \text{ and } n \equiv 0 \bmod 2 \\ [\min\left((a^-)^n, (a^+)^n\right), \max\left((a^-)^n, (a^+)^n\right)] & \text{if } n \equiv 1 \bmod 2. \end{cases}$$

To simplify the notation, a degenerate interval $[c, c]$ may be denoted by c, and the interval product $[a^-, a^+] \boxtimes [c, c]$ may be denoted in the compact form $[a^-, a^+]\ c$.

The unit interval is denoted by $\mathbf{U} = [0, 1]$. A k-dimensional box $\mathbf{B} \in \mathbb{IR}^k$ (or k-box for short) is defined by $\mathbf{B} = [a_1^-, a_1^+] \times \ldots \times [a_k^-, a_k^+]$. A box $\mathbf{B'} \in \mathbb{IR}^k$ where $\mathbf{B'} = [b_1^-, b_1^+] \times \ldots \times [b_k^-, b_k^+]$ is a subbox of $\mathbf{B}$, denoted $\mathbf{B'} \subseteq \mathbf{B}$ if for each $i \in [1, k]$, $[b_i^-, b_i^+] \subseteq [a_i^-, a_i^+]$. The set union of $\mathbf{B}_1, \mathbf{B}_2 \in \mathbb{IR}^k$ where $\mathbf{B}_1 = [a_1^-, a_1^+] \times \ldots \times [a_k^-, a_k^+]$ and $\mathbf{B}_2 = [b_1^-, b_1^+] \times \ldots \times [b_k^-, b_k^+]$,

denoted $\mathbf{B}_1 \cup \mathbf{B}_2$, is defined by $\mathbf{B}_1 \cup \mathbf{B}_2 = ([a_1^-, a_1^+] \cup [b_1^-, b_1^+]) \times \ldots \times ([a_k^-, a_k^+] \cup [b_k^-, b_k^+])$. The k-dimensional unit box is denoted by $\mathbf{U}^k$.

The set $\mathbb{IR}$ is a semi-group under addition and multiplication since interval addition and multiplication are associative and commutative, namely, for $\mathbf{B}, \mathbf{C}, \mathbf{D} \in \mathbb{IR}$,

$$\begin{array}{lcl} \mathbf{B} \boxplus (\mathbf{C} \boxplus \mathbf{D}) & = & (\mathbf{B} \boxplus \mathbf{C}) \boxplus \mathbf{D}, \\ \mathbf{B} \boxtimes (\mathbf{C} \boxtimes \mathbf{D}) & = & (\mathbf{B} \boxtimes \mathbf{C}) \boxtimes \mathbf{D}, \\ \mathbf{B} \boxplus \mathbf{C} & = & \mathbf{C} \boxplus \mathbf{B}, \\ \mathbf{B} \boxtimes \mathbf{C} & = & \mathbf{C} \boxtimes \mathbf{B}, \end{array}$$

and the zero and unit (degenerated) intervals are defined respectively as $[0, 0]$ and $[1, 1]$. However, additive and multiplicative inverses are not defined for proper intervals. Moreover, the distributivity law does not hold in general. However, the multiplication is sub-distributive with respect to addition; namely, for $\mathbf{B}, \mathbf{C}, \mathbf{D} \in \mathbb{IR}$,

$$\mathbf{B} \boxtimes (\mathbf{C} \boxplus \mathbf{D}) \subseteq (\mathbf{B} \boxtimes \mathbf{C}) \boxplus (\mathbf{B} \boxtimes \mathbf{D}).$$

The operations over intervals are monotone with respect to interval inclusion; namely, for $\mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E} \in \mathbb{IR}$, if $\mathbf{B} \subset \mathbf{D}$ and $\mathbf{C} \subset \mathbf{E}$, then

$$\begin{array}{lcl} \mathbf{B} \boxplus \mathbf{C} & \subset & \mathbf{D} \boxplus \mathbf{E}, \\ \mathbf{B} \boxminus \mathbf{C} & \subset & \mathbf{D} \boxminus \mathbf{E}, \\ \mathbf{B} \boxtimes \mathbf{C} & \subset & \mathbf{D} \boxtimes \mathbf{E}, \\ \mathbf{B} \boxslash \mathbf{C} & \subset & \mathbf{D} \boxslash \mathbf{E}. \end{array}$$

2.5 Functions and Fixpoints

We follow the notation used in [KL02]. For sets A and B, let $f : A \to B$. The domain and codomain (also referred as range) of f are respectively denoted by $\mathsf{dom}(f)$ and $\mathsf{cod}(f)$. Given two complete lattices $\langle L, \sqsubseteq, \sqcup, \sqcap \rangle$ and $\langle L', \sqsubseteq', \sqcup', \sqcap' \rangle$, let the map $f : L \to L'$. Then

- for each $X \subseteq L$, f is said to be additive if and only if $f(\sqcup X) = \sqcup' f(X)$;
- for each $X \subseteq L$, f is said to be co-additive if and only if $f(\sqcap X) = \sqcap' f(X)$;
- for each chain $C \subseteq L$, f is said to be continuous if and only if $f(\sqcup C) = \sqcup' f(C)$;
- for each chain $C \subseteq L$, f is said to be co-continuous if and only if $f(\sqcap C) = \sqcap' f(C)$;
- for each $x \sqsubseteq y$, f is said to be monotonic if and only if $f(x) \sqsubseteq' f(y)$.

Monotonicity of f is implied by continuity and also by co-continuity. Let $g : B \to C$. Then the composition of f and g, denoted by $(g \circ f)$, is defined as: $\forall x \in A : (g \circ f)(x) = g(f(x))$. Moreover, if $f : L \to L$, then

- for each $x \in L$, f is said to be idempotent if and only if $f(x) = f(f(x))$;
- for each $x \in L$, f is said to be extensive if and only if $x \sqsubseteq f(x)$;
- for each $x \in L$, f is said to be reductive if and only if $f(x) \sqsubseteq x$.

Given a monotone operator $f : L \to L$ on a complete lattice $\langle L, \sqsubseteq, \cup, \cap, \top, \bot \rangle$, the least and greatest fixpoints (denoted respectively by lfp and gfp) are defined as

$$\mathsf{lfp}(f) = \sqcap\{x \in L \mid f(x) \sqsubseteq x\} \qquad \text{and} \qquad \mathsf{gfp}(f) = \sqcup\{x \in L \mid x \sqsubseteq f(x)\}.$$

Moreover, for f continuous, $\mathsf{lfp}(f) = \sqcup_{n \in \mathbb{N}} f^n(\bot)$, and for f co-continuous, $\mathsf{gfp}(f) = \sqcap_{n \in \mathbb{N}} f^n(\top)$.

2.6 Abstract Interpretation

Abstract interpretation [CC77, CC92a] is a general theory for specifying and then validating program analyses. In abstract interpretation, if $\langle D, \sqsubseteq \rangle$ and $\langle D^\sharp, \sqsubseteq^\sharp \rangle$ are two complete lattices (called respectively the concrete and the abstract domains) where for a pair of monotonic maps $\alpha : D \mapsto D^\sharp$ and $\gamma : D^\sharp \mapsto D$, $\gamma\alpha$ is extensive ($c \sqsubseteq \gamma\alpha(c)$) and $\alpha\gamma$ is reductive ($\alpha\gamma(a) \sqsubseteq^\sharp a$), then the tuple $\langle D, \alpha, D^\sharp, \gamma \rangle$ is called a *Galois connection* between D and $D^\sharp$. In such a case, α is *additive* (i.e. it preserves least upper bounds) and γ is co-additive. If in addition, for any $y \in D^\sharp$ where $\alpha(\gamma(y)) = y$, then the tuple $\langle D, \alpha, D^\sharp, \gamma \rangle$ is called a *Galois insertion* of $D^\sharp$ in D. We call α and γ respectively the abstraction and concretisation functions. The conditions for defining a Galois connection can be weakened in order to work either with a concretisation or abstraction function [CC92b]. For the posets D and $D^\sharp$, a concretisation is a monotonic map $\gamma : D^\sharp \to D$. We say that $a \in D^\sharp$ is an abstraction for $c \in D$ if $c \sqsubseteq \gamma(a)$. For an operator $F^\sharp$ on $D^\sharp$ is said to be a *sound abstraction* for an operator F on D with respect to $\langle \alpha, D, \gamma, D^\sharp \rangle$ if and only if for all $a \in D^\sharp$, $(\alpha \circ F \circ \gamma)(a) \sqsubseteq^\sharp F^\sharp$. An abstract operator $F^\sharp$ is a sound abstraction for a concrete operator F if and only if for all $a \in A$, $(F \circ \gamma)(a) \sqsubseteq (\gamma \circ F^\sharp)(a)$. For a concrete domain D of objects, a semantic function $[\![.]\!] : \mathit{Program} \to D$ associates with each program $P \in \mathit{Program}$ its semantics: $[\![P]\!] \in C$. An abstract domain $D^\sharp$ and an abstract semantics function $[\![.]\!]^\sharp \in D^\sharp$ approximates the concrete program semantics.

2.7 Polyhedra Domain

The polyhedra domain [CH78] was introduced by Cousot and Halbwachs for the purpose of analysing numerical properties of programs at compile-time. Let $\mathbb{R}^n$, for $n > 0$, be the space of n-dimensional real vectors. A set $P \subseteq \mathbb{R}^n$ is a *closed* and *convex polyhedron* (in what follows we may refer to it as *polyhedron*) if and only if it is possible to express P as the intersection of a finite number of closed half-spaces of $\mathbb{R}^n$ [BRZH02, BHZ04]. We let $\mathbb{CP}_n$ denote the set of all closed convex polyhedra in $\mathbb{R}^n$.

Double Description. A polyhedron $P \in \mathbb{R}^n$ has two representations [MRTT53]. One representation involves an $m \times n$ matrix A and an m-vector $\overline{\mathbf{b}}$, where

$$P \;=\; \{\overline{\mathbf{x}} \in \mathbb{R}^n \mid A\overline{\mathbf{x}} \;\leq\; \overline{\mathbf{b}}\},$$

The tuple $(A, \overline{\mathbf{b}})$ is called the *constraint description* of P. A polyhedron P can also be represented by the finite sets V and R of n-vectors, where $V = \{\overline{\mathbf{v}}_i \in \mathbb{R}^n\}$ is the set of *vertices* and $R = \{\overline{\mathbf{r}}_j \in \mathbb{R}^n\}$ is the set of *rays*, where

$$P \;= \Big\{ \sum_{i=1}^{|V|} \lambda_i \overline{\mathbf{v}}_i + \sum_{j=1}^{|R|} \mu_j \overline{\mathbf{r}}_j \mid \lambda_i \geq 0,\; \mu_j \geq 0, \sum_{i=1}^{|V|} \lambda_i = 1 \;\Big\}.$$

The tuple (V, R) is called the *system of generators* of P. Chernikova's algorithm [Che68] performs a conversion from one representation to another, and delivers a minimal description. This algorithm was later improved by LeVerge [LeV92].

Affine Transformations. Let $S \subseteq \mathbb{R}^n$ be a set of vectors and $\psi : \mathbb{R}^n \to \mathbb{R}^m$ a function. We denote by $\psi(S) \subseteq \mathbb{R}^m$ the *codomain* of the set S under ψ, defined by

$$\psi(S) \;=\; \{\psi(\overline{\mathbf{x}}) \in \mathbb{R}^m \mid \overline{\mathbf{x}} \in S\}.$$

Moreover, let $S' \in \mathbb{R}^m$ be a set of vectors. We denote by $\psi^{-1}(S') \subseteq \mathbb{R}^n$ the *pre-image* of the set S' under ψ, defined by

$$\psi^{-1}(S') \;=\; \{\overline{\mathbf{x}} \in \mathbb{R}^n \mid \psi(\overline{\mathbf{x}}) \in S'\}.$$

Note that $\psi^{-1}(S')$ delivers the largest set $S \in \mathbb{R}^n$ where $\psi(S) \subseteq S'$. We say that $\psi : \mathbb{R}^n \to \mathbb{R}^m$ is *space-dimension preserving* if $n = m$, and an *affine transformation* if there exists an $m \times n$ matrix A with values in $\mathbb{R}$ and a vector $\overline{\mathbf{b}} \in \mathbb{R}^m$ where for each $\overline{\mathbf{x}} \in \mathbb{R}^n$, $\psi(\overline{\mathbf{x}}) \;=\; A\overline{\mathbf{x}} + \overline{\mathbf{b}}$. The set $\mathbb{CP}_n$ is closed under any affine image and preimage operators that are space-dimension preserving.

Existential Quantifier. This operator eliminates the j-th variable in P by the *Fourier-Motzkin* procedure, and it is denoted $\exists x_j(P)$. The existential quantifier of x_j can be calculated by adding the j-unit vector $\overline{\mathbf{u}}_j \in \mathbb{R}^n$ and $-\overline{\mathbf{u}}_j$ to the set of rays.

Test Inclusion. This operator verifies that every point of a polyhedron is included in another polyhedron. Let $P, P' \in \mathbb{CP}_n$ be represented respectively by a system of generators (V, R) and a system of constraints $(A, \overline{\mathbf{b}})$. Then, $P \subseteq P'$ if and only if for each $\overline{\mathbf{v}} \in V$ and each $\overline{\mathbf{r}} \in R$, $A\overline{\mathbf{v}} \leq \overline{\mathbf{b}}$ and $A\overline{\mathbf{r}} \leq \mathbf{0}$.

Polyhedra Lattice. When $\mathbb{CP}_n$ is partially ordered by subset inclusion, we can define a lattice, in which the empty set and $\mathbb{R}^n$ are the bottom and top elements respectively, and in which the binary meet operator $\cap$ of set-intersection and a binary join operator $\uplus$ called *poly-hull*, are included. In this way, we have the abstract domain

$$\widehat{\mathbb{CP}_n} := \langle \mathbb{CP}_n, \subseteq, \emptyset, \mathbb{R}^n, \uplus, \cap \rangle.$$

Note that given $P_1, P_2 \in \mathbb{CP}_n$, $P_1 \uplus P_2$ delivers the least closed convex polyhedron P where $P_1, P_2 \subseteq P$. Moreover, the poly-hull of a finite set of polyhedra, denoted $\uplus\{P_i \in \mathbb{CP}_n \mid i \in [0, m-1]\}$ is defined by $P_1 \uplus \ldots \uplus P_{m-1}$.

2.8 Finite Powerset Domain of Polyhedra

This section is based on [BHZ04] and [Bag98].

Definitions The powerset domain is built from the base-level domain $\mathbb{CP}_n$, $n \in \mathbb{N}$, which includes the entailment relation of set inclusion, the meet operation $\cap$, a top element $\mathbb{R}^k$, and a bottom element $\emptyset$.

The elements of the powerset of $\mathbb{CP}_n$ are *reduced* to nonredundant elements; namely, a set P is called *nonredundant* with respect to $\subseteq$ if and only if $\emptyset \notin P$ and $\forall d_1, d_2 \in P$, if $d_1 \subseteq d_2$ then $d_1 = d_2$. The set of finite nonredundant subsets of $\mathbb{CP}_n$ with respect to $\subseteq$ is denoted by $\wp_{\mathrm{fn}}^{\subseteq}(\mathbb{CP}_n)$. The reduction function $\Omega_{\mathbb{CP}_n}^{\subseteq} : \wp_{\mathrm{fn}}(\mathbb{CP}_n) \to \wp_{\mathrm{fn}}^{\subseteq}(\mathbb{CP}_n)$ that maps a finite set into its nonredundant counterpart is called *Omega-reduction*, and is defined for each $P \in \wp_{\mathrm{fn}}(\mathbb{CP}_n)$ as

$$\Omega_{\mathbb{CP}_n}^{\subseteq}(P) \;=\; P \;\setminus\; \{d \in P \;\mid\; d \;=\; \emptyset \text{ or } \exists d' \in P.\; d \subset d'\},$$

For any $P \in \wp_{\mathrm{fn}}(\mathbb{CP}_n)$ where $P \neq \{\emptyset\}$, $\Omega_{\mathbb{CP}_n}^{\subseteq}(P)$ is the finite set of the maximal elements of P.

The *finite powerset domain* over a domain $\widehat{\mathbb{CP}_n}$ is the set of all finite reduced sets of $\mathbb{CP}_n$ and denoted by $\wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n})$. The domain includes an approximation ordering $\subseteq^\star$ defined as

$$S_1 \subseteq^\star S_2 \;\Leftrightarrow\; \forall d_1 \in S_1 :\; \exists d_2 \in S_2.\; d_1 \subseteq d_2.$$

The top element is $\{\mathbb{R}^n\}$ and the bottom element is the empty set. Thus, we denote the lattice of finite powerset domain over a domain $\widehat{\mathbb{CP}_n}$ by

$$\wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n}) \;=\; \langle\; \wp_{\mathrm{fn}}(\mathbb{CP}_n), \subseteq^\star, \cup^\star, \cap^\star, \emptyset, \{\mathbb{R}^n\} \;\rangle.$$

Meet and Upper Bound Operators. Given the sets $S_1, S_2 \in \wp_{\mathrm{fn}}(\mathbb{CP}_n)$, the *powerset meet* operator $\cap^\star$ and the *upper powerset bound* operator $\cup^\star$ are defined respectively by

$$S_1 \ \cap^\star \ S_2 \ = \ \Omega^{\subseteq}_{\mathbb{CP}_n}\big(\{d_1 \cap d_2 \ \mid \ d_1 \in S_1,\ d_2 \in S_2\}\big),$$

and by the reduced set union $S_1 \cup^\star S_2 = \Omega^{\subseteq}_{\mathbb{CP}_n}(S_1 \cup S_2)$.

Add Disjunct Operators. Given the powerset element $S \in \wp_{\mathrm{fn}}(\mathbb{CP}_n)_{\mathrm{p}}$ and the base-level element $d \in \mathbb{CP}_n$, the *add disjunct* operator returns the powerset element $\Omega^{\subseteq}_{\mathbb{CP}_n}(S \cup \{d\})$.

2.9 Logic Programming

For a standard introduction to logic programming, see for instance [Apt90]. Another standard text on the topic is [Llo87]. The language considered is first order predicate logic. The following definitions in logic programming can be found at [HS02] and [BZGH01]. Let Σ denote a possibly infinite set of function symbols, ranked over the set of natural numbers and *Vars* a denumerable set of variable symbols, disjoint from Σ. Given $V \in \wp_{\mathrm{fn}}(\mathit{Vars})$, Σ and $k \in \mathbb{N}$, we define

$$\begin{array}{rcl}
\mathsf{terms}^0(\Sigma, V) & = & V \\
\mathsf{terms}^{k+1}(\Sigma, V) & = & \mathsf{terms}^k(\Sigma, V) \\
& & \cup \left\{ \ f(t_1, \ldots, t_n) \ \middle| \ \begin{array}{l} f^n \in \Sigma \text{ and} \\ \{\, t_1, \ldots, t_n \,\} \subseteq \mathsf{terms}^k(\Sigma, V) \end{array} \right\} \\
\mathsf{terms}(\Sigma, V) & = & \bigcup_{d \geq 0} \mathsf{terms}^d(\Sigma, V) \ .
\end{array}$$

It is assumed that Σ contains at least two distinct function symbols, one having rank 0 and one with rank greater than 0 (so that there exist an infinite number of terms both with and without variables).

The free algebra of all (possibly infinite) terms in the signature Σ having a set of variables $V \in \wp_f(\mathit{Vars})$ is denoted by $\mathsf{terms}(\Sigma, V)$. If $t \in \mathsf{terms}(\Sigma, V)$ then $\mathsf{vars}(t)$ denotes the set of variables occurring in t. If $\mathsf{vars}(t) = \emptyset$ then t is said to be *ground*; t is a *finite term* (or *Herbrand term*) if it contains a finite number of occurrences of function symbols. The sets of all ground and finite terms are denoted by *GTerms* and *HTerms*, respectively.

A *substitution* is a total function $\sigma : \mathit{Vars} \to \mathit{HTerms}$ that is the identity almost everywhere. That is, the *domain of* σ, defined as

$$\mathsf{dom}(\sigma) \stackrel{\mathrm{def}}{=} \{\, x \in \mathit{Vars} \,|\, \sigma(x) \neq x \,\} \ ,$$

is a finite set of variables. If $x \in \mathit{Vars}$ and $t \in \mathit{HTerms} \backslash \{x\}$, then $x \mapsto t$ is called a *binding*. The set of all bindings is denoted by *Bind*. Substitutions are conveniently denoted by the

set of their bindings. Thus, a substitution σ is identified with the (finite) set

$$\left\{ x \mapsto \sigma(x) \;\middle|\; x \in \mathsf{dom}(\sigma) \right\} .$$

We denote by $\mathsf{vars}(\sigma)$ the set of all variables occurring in the bindings of σ. For a substitution θ and a term t, the *instance* of t by θ, denoted $t\theta$, is the term obtained from t by simultaneously replacing each occurrence of x_i, by t_i, where $x_i \mapsto t_i \in \theta$, $i \in [1, n]$ and $|\theta| = n$. Moreover, $t\theta$ is a ground instance if $t\theta$ is ground.

An *atom* is of the form $p(t_1, \ldots, t_n)$, for t_i, $i \in [1, n]$, terms. A *literal* is an positive atom or a negated atom. A *clause* is a sequence of literals. A clause is called *definite* if it has at most one positive atom. In a definite clause, the positive atom is called *head atom* and the rest of (negated) atoms are called *body atoms*. A clause with all negative atoms is called a *query*. A clause with one and only one atom is called *atomic clause*. A *logic program* consists of a finite set of clauses.

A mode for a predicate p/n is an atom $p(m_1, \ldots, m_n)$ where $m_i \in I \cup O$, $i \in [1, n]$, and $I \cap O = \emptyset$. Every $m_i \in I$ is called an input argument and every $m_j \in O$, $i \neq j$, $i, j \in [1, n]$, is called an output argument. Each $i \in [1, n]$ denotes an input or output position in p/n. A mode of a program is a set of modes, one for each predicate declared in the program.

Chapter 3

Concrete and Abstract Semantics of Floating-Point Arithmetic

In this chapter we review the IEEE-754 standard for floating-point systems and existing work on floating-point arithmetic abstractions. Then we propose a generalisation of the so-called linear forms [Min04b, Min04a]. This generalisation is founded in interval valued polynomials and holds in the abstract interpretation framework of [Min04a]. As this general form preserves nonlinear information of the concrete semantics, we obtain a more expressive abstract semantics. An implication of this expressiveness is that for instance we avoid linearisation in the abstract product and division. Moreover, this generalisation will prove useful for abstracting floating-point expressions into the polyhedra domain.

Although this may seem a paradox, all exact science is dominated by the idea of approximation.

Bertrand Russell

3.1 Introduction

Programming languages often manipulate finite representations of numerical values, equipped with unsound arithmetic that do not enjoy most algebraic properties assumed in the design of software with numerical processing. For instance, real numbers have a finite representation, which often incurs in rounding errors that lead to inaccurate results.

Traditionally, the analysis for properties of floating-point arithmetic has been carried out using interval arithmetic [Moo66]. Though robust, simple to handle and efficient, analyses based on this arithmetic do not guarantee narrow enclosures; namely, the resulting intervals might be too wide so that the analysis is inconclusive. One reason for this is the fact that dependencies between variables in an arithmetic calculation are not represented, since, for instance, there exist no additive or multiplicative inverses. That is, if a variable x ranges over an interval $[a, b]$, namely, $x \in [a, b]$, then $x - x = [a - b, b - a]$ and $x/x = [a/b, b/a]$, though the optimal results are $[0, 0]$ and $[1, 1]$, respectively. To overcome this, Hansen [Han75] proposed a generalisation of interval arithmetic that included information on variables.

A further refinement was introduced in [VACS94], called *affine arithmetic*, that used synthetic variables. Two affine forms can be added, and an affine form can be multiplied by a constant. However, as noted in [Min04b, Min04a], linearisation is required when two affine forms are multiplied. Moreover, synthetic variables become a disadvantage when it comes to constructing abstractions in relational domains, since these synthetic variables do not correspond to variables in a program. In order to overcome the problem of synthetic variables, Miné [Min04b, Min04a] presented a symbolic form, called *interval linear form*. This form allows for affine arithmetics, and most importantly, the variables used correspond to those in a program, thus allowing for a direct use of such variables in the abstraction into relational (and nonrelational) domains. Moreover, based on interval linear forms, Miné provides a nondeterministic model of floating-point rounding errors. However, when multiplying two linear forms, a linearisation is required. This may represent a loss of precision in the abstraction of floating-point expressions of a program.

By preserving nonlinear information of a floating-point expression, we can avoid linearisation, for instance, in the abstract product. For this purpose, we present a generalisation of linear forms based on interval valued polynomials (that is, polynomials with interval coefficients). These polynomials are expressed with variables of the program, as in the linear forms. However, the product of two polynomial forms delivers a (possibly) nonlinear polynomial form. This is important in order to preserve nonlinear information of a floating-point expression. Polynomial forms are not exempt from problems, though. In the division of two polynomial forms, we can choose between a symbolic division, which delivers a complex expression, or else make use of linearisation. We show that there exist precise boundaries for linearised polynomial forms. The following chapter will present a series of techniques for efficient and precise linearisations of polynomial forms.

Our contribution in this chapter is the introduction of a nonlinear abstract semantics based on interval valued polynomials that has increased precision in comparison with other abstractions for floating-point arithmetic. This semantics preserves nonlinear information which proves important to avoid linearisation and thus, loss of precision. Our abstraction holds in the abstract interpretation framework of [Min04a]. This abstract semantics will

Format†	Size	Precision **p**	Exponent Size **e**	**bias**	Range‡
Single	32 bits	23 bits	8 bits	127	$3.4028 \cdot 10^{\pm 38}$
Double	64 bits	52 bits	11 bits	1023	$1.7976 \cdot 10^{\pm 308}$
Double Extended	80 bits	64 bits	15 bits	16383	$1.1897 \cdot 10^{\pm 4932}$

† Each format includes one bit to represent the sign.
‡ Maximum Range Magnitude.

Figure 3.1: *IEEE-754 single and double precision floating-point formats.*

prove useful for abstracting floating-point expressions into the polyhedra domain.

3.2 The IEEE-754 Floating-Point System Standard

In 1985, the IEEE Standards Board and the American National Standards Institute approved the IEEE 754-1985 Standard for floating-point binary arithmetic [oEE85]. Thereafter, computers and also compilers were designed following this standard. In this section we describe the IEEE 754-1985 floating-point arithmetic. For this purpose we adopt the formulation used in [Min04b, Min04a] as standard.

3.2.1 Representation

The IEEE-754 standard binary representation of a floating-point number includes three fields:

- a 1-bit sign s;
- an exponent $e = e_1 \dots e_{\mathbf{e}}$, represented by **e** bits;
- a mantissa $m = m_1 \dots m_{\mathbf{p}}$, represented by **p** bits.

The number **p** of bits is called the precision. The exponent e is biased by a value $-\mathbf{bias}$ and bounded by $e_{\min} \leq e \leq e_{\max}$, where $e_{\min} = -\mathbf{bias} + 1$ and $e_{\max} = \mathbf{bias}$. The purpose of biasing the exponent is to represent both positive and negative exponents. In the mantissa, bits m_1 and $m_{\mathbf{p}}$ are respectively the most and less significant bits.

A floating-point format is denoted by a tuple $\mathbf{f} = \langle \mathbf{e}, \mathbf{bias}, \mathbf{p} \rangle$. The set of floating-point values of format **f** is denoted $\mathbb{F}_{\mathbf{f}}$. The set of all floating-point formats is denoted **F**. For instance, the IEEE-754 *single precision* format is denoted by $\mathbf{f} = \langle 8, 127, 23 \rangle$ (or $\mathbf{f} = \mathbf{32}$ for short) and the *double precision* format by $\mathbf{f} = \langle 11, 1023, 53 \rangle$ (or $\mathbf{f} = \mathbf{64}$ for short), as detailed in figure 3.1. Note that for any format **f** of finite precision, $\mathbb{F}_{\mathbf{f}} \subset \mathbb{R}$.

3.2.2 Characteristic Values

The largest and smallest positive representable values in a format as well as the machine epsilon value are important for defining further concepts related to floating-point system and analyses. For each format **f**, the smallest and largest nonzero positive number in $\mathbb{F}_{\mathbf{f}}$

Category	Bit Pattern	Value
Normalised	$1 \leq e \leq 2^{\mathbf{e}} - 2$	$(-1)^s \cdot 2^{e-\mathbf{bias}} \cdot 1.m$
Denormalised	$e = 0$, $0 < m < 2^{\mathbf{p}-1}$	$(-1)^s \cdot 2^{1-\mathbf{bias}} \cdot 0.m$
Signed Zeroes	$e = 0$, $m = 0$	$(-1)^s \cdot e \cdot m$
Signed Infinities	$e = 2^{\mathbf{e}} - 1$, $m = 0$	$(-1)^s \cdot e \cdot m$
NaN (Not a Number)	$e = 2^{\mathbf{e}} - 1$, $m \neq 0$	$(-1)^s \cdot e \cdot m$

Figure 3.2: *IEEE-754 Categories for a format* $\mathbf{f} = \langle \mathbf{e}, \mathbf{bias}, \mathbf{p} \rangle$ *([Min04b]).*

is defined respectively by $mf_{\mathbf{f}} = 2^{1-\mathbf{bias}-\mathbf{p}}$ and $Mf_{\mathbf{f}} = (2 - 2^{-\mathbf{p}})2^{2^{\mathbf{e}}-\mathbf{bias}-2}$. Moreover, we call $eps_{\mathbf{f}} = 2^{-\mathbf{p}}$ the *machine epsilon.*

3.2.3 Categories

For a given $\mathbf{f}$, the set $\mathbb{F}_{\mathbf{f}}$ may be separated into five distinct categories, as shown in figure 3.2. Denormalised numbers are used [1] to represent smaller values than those obtained by normalised numbers. Values $+0$ and -0 validate the equality relational operator. Moreover, $+\infty$, $-\infty$ and NaN are used to represent results of floating-point arithmetic that are not mathematically defined. For instance, an operation such as $1/\pm 0$ produces a division by zero and it is represented as $\pm\infty$. Also, when the result of a floating-point operation overflows the maximum or minimum representable value $+Mf_{\mathbf{f}}$ or $-Mf_{\mathbf{f}}$, respectively, it is represented as $\pm\infty$. The result of operations such as $\pm 0/\pm 0$ are represented by NaN.

We denote by $\Omega \in \{-\infty, +\infty, NaN\}$ a floating-point value that has either no representation in $\mathbb{F}_{\mathbf{f}}$ or no mathematical definition. For instance, values greater than $Mf_{\mathbf{f}}$ or smaller than $-Mf_{\mathbf{f}}$ cannot be represented in format $\mathbf{f}$ and are denoted by Ω. In the context of rounding functions or evaluation of floating-point expressions, symbol Ω denotes a run-time error (in which case, the execution of a program may raise an exception or halt).

The result of an operation underflows if it is smaller than the absolute value of $mf_{\mathbf{f}}$ and greater than zero. For instance, the result of dividing $mf_{\mathbf{f}}$ by 10 is clearly not representable and it is set to 0. Gradual underflow delivers a denormalised number rather than rounding a normalised underflowed value to zero. The purpose of denormalised numbers is not simply to avoid underflow in all the cases, but to eliminate underflow in sensible cases such as in a multiplication followed by an addition [Dem84, Hig02].

Example 3.1. *Some examples of normalised and denormalised floating-point values in single precision as well as examples of special values for the IEEE-754 single precision format may be found in figure 3.3. The decimal representation of such values are as*

[1] In this thesis, we refer to a number as a representation (namely, a sequence of bits that comply with a format), and a value as the element in the set $\mathbb{R}$ represented by a number.

Exact Value	Representation		
	Sign	Exponent	Mantissa
		(*normalised*)	
1	0	01111111	00000000000000000000000
2	0	10000000	00000000000000000000000
-5.125	1	10000001	01001000000000000000000
0.75	0	01111110	10000000000000000000000
0.1	0	01111110	00011001100110011001100
		(*denormalised*)	
2^{-127}	0	00000000	10000000000000000000000
2^{-149}	0	00000000	00000000000000000000001
		(*special values*)	
$+0$	0	00000000	00000000000000000000000
-0	1	00000000	00000000000000000000000
$+\infty$	0	11111111	00000000000000000000000
$-\infty$	1	11111111	00000000000000000000000
NaN	0	11111111	(At least one bit set to 1.)

Figure 3.3: *Binary representation of values in each category of IEEE-754 single precision format.*

follows:

$$\begin{aligned} 1 &= (-1)^0 \cdot 2^{127-127} \cdot 1 \\ 2 &= (-1)^0 \cdot 2^{128-127} \cdot 1 \\ -5.125 &= (-1)^1 \cdot 2^{129-127} \cdot (1 + 2^{-2} + 2^{-5}) \\ 0.75 &= (-1)^0 \cdot 2^{126-127} \cdot (1 + 2^{-1}) \ . \end{aligned}$$

The decimal value 0.1 *has no finite binary representation for any finite precision. For instance, in single precision,* 0.1 *is represented as shown in figure 3.3. However, its binary representation does not denote the exact decimal value, namely*

$$\begin{aligned} 0.1 &\approx (-1)^0 \cdot 2^{126-127} \cdot (1 + 2^4 + 2^5 + 2^8 + 2^9 + 2^{12} + 2^{13} + 2^{16} + 2^{17} + 2^{20} + 2^{21}) \\ &= \frac{3355440}{33554432} \\ &\approx 0.09999990463256836. \end{aligned}$$

$$
\begin{aligned}
R_{\mathbf{f},+\infty}(x) &= \begin{cases} \Omega & \text{if } x > Mf_{\mathbf{f}} \\ \min\{y \in \mathbb{F}_{\mathbf{f}} \mid y \geq x\} & \text{otherwise} \end{cases} \\
R_{\mathbf{f},-\infty}(x) &= \begin{cases} \Omega & \text{if } x < -Mf_{\mathbf{f}} \\ \max\{y \in \mathbb{F}_{\mathbf{f}} \mid y \leq x\} & \text{otherwise} \end{cases} \\
R_{\mathbf{f},0}(x) &= \begin{cases} R_{\mathbf{f},+\infty}(x) & \text{if } x \leq 0 \\ R_{\mathbf{f},-\infty}(x) & \text{if } x \geq 0 \end{cases} \\
R_{\mathbf{f},n}(x) &= \begin{cases} \Omega & \text{if } |x| > Mf_{\mathbf{f}} \\ Mf_{\mathbf{f}} & \text{else if } x \geq Mf_{\mathbf{f}} \\ -Mf_{\mathbf{f}} & \text{else if } x \leq -Mf_{\mathbf{f}} \\ R_{\mathbf{f},-\infty}(x) & \text{else if } |R_{\mathbf{f},-\infty}(x) - x| < |R_{\mathbf{f},+\infty}(x) - x| \\ R_{\mathbf{f},+\infty}(x) & \text{else if } |R_{\mathbf{f},+\infty}(x) - x| < |R_{\mathbf{f},-\infty}(x) - x| \\ R_{\mathbf{f},-\infty}(x) & \text{else if least significant bit of } R_{\mathbf{f},-\infty}(x) \text{ is 0} \\ R_{\mathbf{f},+\infty}(x) & \text{else if least significant bit of } R_{\mathbf{f},+\infty}(x) \text{ is 0} \end{cases}
\end{aligned}
$$

Figure 3.4: *Rounding functions [Min04b].*

3.2.4 Rounding Methods

Rounding is used to approximate proper subsets of $\mathbb{R}$ by a finite set of floating-point numbers. A rounding function is a mapping $R_{\mathbf{f},\mathbf{r}} : \mathbb{R} \to \mathbb{F}_{\mathbf{f}}$, where $\mathbf{r}$ denotes a rounding method and $\mathbf{f}$ a floating-point format. The IEEE-754 standard defines four rounding methods: toward 0, toward $-\infty$, toward $+\infty$ and to the nearest (denoted by n). Each rounding method is defined by a rounding function as shown in figure 3.4.

Function $R_{\mathbf{f},+\infty}(x)$ rounds x to the next larger representable value in $\mathbf{f}$. Function $R_{\mathbf{f},+\infty}(x)$ rounds x to the next smaller representable value in $\mathbf{f}$. Function $R_{\mathbf{f},0}(x)$ applies to either of the first two functions depending on the sign of x. Note $R_{\mathbf{f},0}(0) = 0$. Function $R_{\mathbf{f},n}$ minimises the distance between the exact and the representable values. When the distance between x and the next larger representable value is the same as the distance between x and the next smaller representable value, then x is rounded to the value whose least significant digit is 0 (also called *round to even*). When $|\ x\ |$ is larger than the largest representable value $Mf_{\mathbf{f}}$, the rounding delivers Ω. A value $|x|$ that is larger than $Mf_{\mathbf{f}}$ but smaller than the first nonrepresentable value in $\mathbf{f}$, is rounded to $Mf_{\mathbf{f}}$; namely, $R_{\mathbf{f},n}(x) \in \{-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}\}$ if

$$(2 - 2^{-\mathbf{p}-1}) \cdot 2^{2^{\mathbf{e}}-\mathbf{bias}-2} > |x| \geq (2 - 2^{-\mathbf{p}}) \cdot 2^{2^{\mathbf{e}}-\mathbf{bias}-2},$$

where $Mf_{\mathbf{f}} = (2 - 2^{-\mathbf{p}}) \cdot 2^{2^{\mathbf{e}}-\mathbf{bias}-2}$.

3.3 Concrete Semantics of Floating-Point Arithmetic

Expressions in floating-point arithmetic are syntactically equivalent to those in real arithmetic. However, the evaluation of a floating-point expression does not deliver the same value as the evaluation of a real expression. This is due to the finite precision of floating-point numbers. In this section we recall the concepts of absolute and relative floating-point rounding errors, as well as a concrete semantics for floating-point expressions.

3.3.1 Rounding Errors

A rounding error refers to the error introduced by rounding an exact value to an approximate value. In such a case, the difference between the exact and represented value is called rounding error. This error is dependent on the underlying floating-point format. The following definition formalises this concept.

Definition 3.2. (Absolute rounding error.) *Given $x \in \mathbb{R}$, a format $\mathbf{f}$ and a rounding method $\mathbf{r}$, the absolute rounding error for x is defined as $\varepsilon_{\mathbf{f},\mathbf{r}}(x) = |R_{\mathbf{f},\mathbf{r}}(x) - x|$.*

Note that the rounding error is bounded by

$$|R_{\mathbf{f},\mathbf{r}}(x) - x| \leq |R_{\mathbf{f},-\infty}(x) - R_{\mathbf{f},+\infty}(x)| \leq 2^{-\mathbf{p}} \cdot 2^{e-\mathbf{bias}}.$$

Definition 3.3. (Relative rounding error.) *Given $x \in \mathbb{R}$, $x \neq 0$, a format $\mathbf{f}$ and a rounding method $\mathbf{r}$, the relative rounding error for x is defined as*

$$\varepsilon'_{\mathbf{f},\mathbf{r}}(x) = \frac{|R_{\mathbf{f},\mathbf{r}}(x) - x|}{|x|}.$$

The relative error for x can be bounded as follows. For $\mathbf{f} = \langle \mathbf{e}, \mathbf{bias}, \mathbf{p} \rangle$, let

$$(-1)^s(1.m_1 \ldots m_{\mathbf{p}}, m_{\mathbf{p}-1}, \ldots)_2 \cdot 2^{e-\mathbf{bias}} \qquad \text{and} \qquad (-1)^s(1.m_1 \ldots m_{\mathbf{p}})_2 \cdot 2^{e-\mathbf{bias}}$$

be the (possibly infinite) binary representation of $x \in \mathbb{R}$ and its (finite) floating-point binary representation, respectively. Thus

$$|x| \geq 2^{e-\mathbf{bias}} \Leftrightarrow \frac{1}{|x|} \leq \frac{1}{2^{e-\mathbf{bias}}}$$

and therefore,

$$\frac{|R_{\mathbf{f},\mathbf{r}}(x) - x|}{|x|} \leq \frac{2^{-\mathbf{p}} \cdot 2^{e-\mathbf{bias}}}{2^{e-\mathbf{bias}}} = 2^{-\mathbf{p}},$$

```
recover(N, K, X, Z) :-
        p(N, K, X, Y),
        Kinv is 1 / K,
        p(N, Kinv, Y, Z),
            (X == Z, write('Exact result.');
                     write('Precision loss.')).

p(N, K, X, Y) :-
        0 == N,
        Y is  X * K.

p(N, K, X, Y) :-
        1 =< N,
        N1 is N - 1,
        X1 is X * K,
        p(N1, K, X1, Y).
```

Figure 3.5: *Innocuous program (Prolog SICStus syntax).*

and

$$|R_{\mathbf{f},\mathbf{r}}(x) - x| \leq |x| \cdot 2^{-\mathbf{p}}. \tag{3.1}$$

This boundary proves useful for the abstraction of errors of floating-point values.

As an example that illustrates how the final result of a computation can be changed by the accumulated (absolute) rounding errors, consider the program in figure 3.3.1. For each of the N recursive calls, the program multiplies a constant value K by another initial value X. Then, the inverse operation over the result of multiplying N times K by an initial X is performed. Variable Z is instantiated in the answer substitution. In real arithmetic, Z should be equal to the value of X in the query. More precisely, a recurrence equation of the form $X_{n+1} = K \cdot X_n$ is recurred a finite number N of times for an initial value X_0. The result X_N is then equated to Y_0, and a recurrence equation of the form $Y_{n+1} = \frac{1}{K} \cdot Y_n$ is recurred N times. Thus, $X_0 = Y_N$. However, in floating-point arithmetic, for a large number N of recursive calls, the accumulated absolute rounding error may be nonzero. Thus the value that instantiates Z is not equal to the value of X in the query. Moreover, for sufficiently large N, underflow may occur; namely, for a sufficiently large i, X_i is rounded to 0. In such a case, the initial value of X cannot be obtained.

Note in figure 3.6 that the absolute rounding error increases with the number of recursive calls, and the growth is considerable for some $N > 310$. The source of errors in this case is due to the inexact binary representation of the decimal value 0.1.

Further examples related to the unsoundness of floating-point arithmetic may be found in [Gol91].

N	K	X	Z	$\varepsilon_{\mathbf{f},\mathbf{n}}(Z)$, $\mathbf{f} = \mathbf{64}$
1	0.1	1	1.0000000000000002	$2.22044 \cdot 10^{-16}$
5	0.1	1	1.0000000000000007	$6.66133 \cdot 10^{-16}$
10	0.1	1	1.0000000000000007	$6.66133 \cdot 10^{-16}$
100	0.1	1	1.0000000000000062	$6.21724 \cdot 10^{-15}$
200	0.1	1	1.0000000000000115	$1.15463 \cdot 10^{-14}$
300	0.1	1	1.0000000000000175	$1.75415 \cdot 10^{-14}$
310	0.1	1	0.9999999999999474	$5.26245 \cdot 10^{-14}$
315	0.1	1	0.9999999836597133	$1.63402 \cdot 10^{-8}$
316	0.1	1	1.0000002306925362	$2.30692 \cdot 10^{-7}$
317	0.1	1	0.9999987484955987	$1.25150 \cdot 10^{-6}$
318	0.1	1	0.9999888671826831	$1.11328 \cdot 10^{-5}$
319	0.1	1	0.9999888671826831	$1.11328 \cdot 10^{-5}$
320	0.1	1	0.9980126045993181	0.00198
321	0.1	1	0.9881312916824931	0.01186
322	0.1	1	0.9881312916824931	0.01186
323	0.1	1	0.0	1.0 *due to underflow*

Figure 3.6: *Rounding errors for queries to the innocuous program (Prolog SICStus 3.11.0 in double precision and rounding to the nearest).*

$$
\begin{array}{lcll}
expr_{\mathbf{f}} & ::= & [a,b]_{\mathbf{f},\mathbf{r}} & a,b \in \mathbb{R} \\
& | & v & v \in V_{\mathbf{f}} \\
& | & \ominus expr_{\mathbf{f}} & \\
& | & expr_{\mathbf{f}} \circledast_{\mathbf{f},\mathbf{r}} expr_{\mathbf{f}} & * \in \{+,-,\times,/\} \\
& | & expr_{\mathbf{f}} \circledcirc_{\mathbf{f},\mathbf{r}} n & n \in \mathbb{N}^{+}
\end{array}
$$

Figure 3.7: *Grammar for floating-point expressions (adapted from [Min04b]).*

3.3.2 Evaluation of Floating-Point Expressions

We present now a concrete semantics for floating-point expressions. In order to define the concrete semantics, we present first a grammar for floating-point expressions and the definition of concrete environment.

Figure 3.7 presents the syntax of numerical expressions for floating-point arithmetic for a format $\mathbf{f}$ and rounding mode $\mathbf{r}$. This grammar includes operators of addition, subtraction, product and division, denoted by the operator symbols $\oplus_{\mathbf{f},\mathbf{r}}$, $\ominus_{\mathbf{f},\mathbf{r}}$, $\otimes_{\mathbf{f},\mathbf{r}}$ and $\oslash_{\mathbf{f},\mathbf{r}}$. It extends the grammar presented in [Min04b] by including an exponentiation operator $\circledcirc_{\mathbf{f},\mathbf{r}}$ for the exponentiation by a natural number. It also includes constants (represented as closed intervals) and a finite set $V_{\mathbf{f}}$ of variables that can store values of format $\mathbf{f}$. Note that $[a,b]_{\mathbf{f},\mathbf{r}}$ for $a,b \in \mathbb{R}$ denotes the set of floating-point values where $a \leq c \leq b$, for $c \in \mathbb{F}_{\mathbf{f}}$ and rounding method $\mathbf{r}$. This will be useful for treating rounding errors as nondeterministic.

Definition 3.4. (Concrete environment.) *A concrete environment $e : V_{\mathbf{f}} \to \mathbb{F}_{\mathbf{f}}$ is a function that associates to each variable $v \in V_{\mathbf{f}}$ a floating-point value in $\mathbb{F}_{\mathbf{f}}$.*

The following semantics is an extension of that proposed in [Min04a] by including an exponentiation operator.

Definition 3.5. (Concrete semantics (adapted from [Min04a]).) *A concrete semantics for a floating-point expression is a function* $[\![expr_{\mathbf{f}}]\!]_{fl}(e) : (\prod_{\mathbf{f}\in\mathbf{F}}(V_{\mathbf{f}} \to \mathbb{F}_{\mathbf{f}})) \to \wp(\mathbb{F}_{\mathbf{f}})$ *defined in an environment* e *as*

$$
\begin{aligned}
[\![\ [a,b]_{\mathbf{f}}\ ]\!]_{fl}(e) &= \left\{ R_{\mathbf{f},\mathbf{r}}(c) \in \mathbb{F}_{\mathbf{f}} \;\middle|\; \begin{array}{l} const_{\mathbf{f},\mathbf{r}}(a) \leq c \leq const_{\mathbf{f},\mathbf{r}}(b), \\ R_{\mathbf{f},\mathbf{r}}(c) \neq \Omega \end{array} \right\} \\
[\![v]\!]_{fl}(e) &= \{e(v)\} \\
[\![\ominus_{\mathbf{f},\mathbf{r}}\ p_{\mathbf{f}}]\!]_{fl}(e) &= \left\{ -c \in \mathbb{F}_{\mathbf{f}} \;\middle|\; \begin{array}{l} c \in [\![p_{\mathbf{f}}]\!]_{fl}(e), \\ R_{\mathbf{f},\mathbf{r}}(c) \neq \Omega \end{array} \right\} \\
[\![p_{\mathbf{f}}\ \circledast_{\mathbf{f},\mathbf{r}}\ q_{\mathbf{f}}]\!]_{fl}(e) &= \left\{ R_{\mathbf{f},\mathbf{r}}(c * d) \in \mathbb{F}_{\mathbf{f}} \;\middle|\; \begin{array}{l} c \in [\![p_{\mathbf{f}}]\!]_{fl}(e),\ d \in [\![q_{\mathbf{f}}]\!]_{fl}(e), \\ R_{\mathbf{f},\mathbf{r}}(c * d) \neq \Omega \end{array} \right\}, \\
&\quad * \in \{+,-\} \\
[\![p_{\mathbf{f}}\ \otimes_{\mathbf{f},\mathbf{r}}\ q_{\mathbf{f}}]\!]_{fl}(e) &= \left\{ R_{\mathbf{f},\mathbf{r}}(c \cdot d) \in \mathbb{F}_{\mathbf{f}} \;\middle|\; \begin{array}{l} c \in [\![p_{\mathbf{f}}]\!]_{fl}(e),\ d \in [\![q_{\mathbf{f}}]\!]_{fl}(e), \\ R_{\mathbf{f},\mathbf{r}}(c \cdot d) \neq \Omega \end{array} \right\} \\
[\![p_{\mathbf{f}}\ \oslash_{\mathbf{f},\mathbf{r}}\ q_{\mathbf{f}}]\!]_{fl}(e) &= \left\{ R_{\mathbf{f},\mathbf{r}}(c/d) \in \mathbb{F}_{\mathbf{f}} \;\middle|\; \begin{array}{l} c \in [\![p_{\mathbf{f}}]\!]_{fl}(e),\ d \in [\![q_{\mathbf{f}}]\!]_{fl}(e), \\ R_{\mathbf{f},\mathbf{r}}(d) \neq 0,\ R_{\mathbf{f},\mathbf{r}}(c/d) \neq \Omega \end{array} \right\} \\
[\![p_{\mathbf{f}}\ \circledcirc_{\mathbf{f},\mathbf{r}}\ 0]\!]_{fl}(e) &= \left\{ 1 \in \mathbb{F}_{\mathbf{f}} \;\middle|\; [\![p_{\mathbf{f}}]\!]_{fl}(e) \neq \Omega \right\} \\
[\![p_{\mathbf{f}}\ \circledcirc_{\mathbf{f},\mathbf{r}}\ n]\!]_{fl}(e) &= \left\{ \{R_{\mathbf{f},\mathbf{r}}(c \cdot d) \in \mathbb{F}_{\mathbf{f}} \;\middle|\; \begin{array}{l} c \in [\![p_{\mathbf{f}}]\!]_{fl}(e),\ d \in [\![p_{\mathbf{f}} \circledcirc_{\mathbf{f},\mathbf{r}} (n-1)]\!]_{fl}(e), \\ R_{\mathbf{f},\mathbf{r}}(c \cdot d),\ n \in \mathbb{N}^+ \end{array} \right\}.
\end{aligned}
$$

It must be noted that this semantics delivers a set of floating-point values, which are usually singleton sets, except for the case of intervals. The purpose of this is that we can define an abstract interpretation between this concrete semantics and an abstract semantics defined over intervals. In this way, a single floating-point value that results from evaluating a floating-point expression is represented by a degenerate interval, that is a singleton set.

In the context of floating-point representations, Ω denotes either $\pm\infty$ or NaN. In the semantics for floating-point expressions, Ω is interpreted as a run-time error. This error can be caused by:

- an invalid operation, denoted NaN (e.g. $0/0$ or $0 \cdot \infty$);
- a division by zero, denoted $\pm\infty$ (e.g. $Mf_{\mathbf{f}}/0$); or

- an overflow, denoted $\pm\infty$ (e.g. $Mf_{\mathbf{f}} \cdot 10$).

When an underflow occurs, however, the result is expressed with denormalised numbers or 0. The occurrence of a run-time error in the evaluation of an expression means that the computation has failed. As a possible consequence, the execution of a program may be halted, or an exception raised. Moreover, when evaluating a floating-point expression, the associative and distributive laws of algebra do not necessarily hold. For instance, for some $a, b, c \in \mathbb{F}_{\mathbf{f}}$ and rounding mode $\mathbf{r}$,

$$\begin{aligned}
(a \oplus_{\mathbf{f},\mathbf{r}} b) \oplus_{\mathbf{f},\mathbf{r}} c &\neq a \oplus_{\mathbf{f},\mathbf{r}} (b \oplus_{\mathbf{f},\mathbf{r}} c) \\
(a \otimes_{\mathbf{f},\mathbf{r}} b) \otimes_{\mathbf{f},\mathbf{r}} c &\neq a \otimes_{\mathbf{f},\mathbf{r}} (b \otimes_{\mathbf{f},\mathbf{r}} c) \\
a \otimes_{\mathbf{f},\mathbf{r}} (b \oplus_{\mathbf{f},\mathbf{r}} c) &\neq (a \otimes_{\mathbf{f},\mathbf{r}} b) \oplus_{\mathbf{f},\mathbf{r}} (a \otimes_{\mathbf{f},\mathbf{r}} c) \ .
\end{aligned}$$

Different evaluation orders of a floating-point expression may result in different values.

3.4 Nonlinear Abstract Semantics of Floating-Point Arithmetic

This section introduces a new specialised transfer function for the purpose of achieving high precision in the abstraction of *arbitrary expressions*. The key idea is that in the abstract semantics we preserve the nonlinearity of the concrete expressions. In this way, the transfer function delivers a nonlinear expression in the form of interval valued polynomials, which includes all the possible concrete semantics, as well as an approximation to rounding errors. From this nonlinear expression, we will show that it is possible to construct polyhedra (chapter 4). Central in this nonlinear abstraction is the concept of interval valued polynomials (or polynomials with interval coefficients) [RR84, RR03]. We show that for each interval valued polynomials there exists an enclosure. Moreover, we define a partial ordering for interval valued polynomials, and a poset. This is necessary for defining an abstract interpretation scheme.

3.4.1 Interval Valued Polynomials

An interval valued polynomial is a polynomial in which the coefficients are closed intervals. We denote such polynomials by $[p](\overline{\mathbf{x}})$ in order to distinguish them from single-valued polynomials. In this way, an interval valued polynomial of degree $N \in \mathbb{N}^k$ for $S \in \wp_{\mathrm{fn}}(\mathbb{N}^k)$, where $S \subseteq \{I \in \mathbb{N}^k \mid I \leq N\}$, is a function $[p] : \mathbb{R}^k \to \mathbb{IR}$ expressed asa polynomial $r(\overline{\mathbf{x}}) = \sum_{I \in S} a_I \mathbf{x}^I$,

$$r(\overline{\mathbf{x}}) \in [p](\overline{\mathbf{x}}) \ \Leftrightarrow \ \forall I \in S \ : \ a_I \in [a_I^-, a_I^+].$$

In order to compare interval valued polynomials, we define a partial ordering relation $\sqsubseteq$ as

$$[p](\overline{\mathbf{x}}) \sqsubseteq [q](\overline{\mathbf{x}}) \;\Leftrightarrow\; \forall r(\overline{\mathbf{x}}) \in [p](\overline{\mathbf{x}}) \;:\; \exists s(\overline{\mathbf{x}}) \in [q](\overline{\mathbf{x}}) \;.\; r(\overline{\mathbf{x}}) = s(\overline{\mathbf{x}}).$$

Interval valued polynomials are comparable under variable renaming if and only if their interval coefficients are comparable. Moreover,

$$[p](\overline{\mathbf{x}}) = [q](\overline{\mathbf{x}}) \;\Leftrightarrow\; [p](\overline{\mathbf{x}}) \sqsubseteq [q](\overline{\mathbf{x}}) \;\wedge\; [q](\overline{\mathbf{x}}) \sqsubseteq [p](\overline{\mathbf{x}}).$$

From this ordering we can define a poset, which will prove useful in constructing a Galois connection between the concrete and the abstract semantics.

Proposition 3.6. *The set of all interval valued polynomials of at most k variables and degree less than or equal to $N \in \mathbb{N}^k$, equipped with the partial ordering relation $\sqsubseteq$, is a poset.*

Recall that for a floating-point format $\mathbf{f}$ of finite precision, $\mathbb{F}_{\mathbf{f}} \subset \mathbb{R}$, and thus, all the representable floating-point values can be mapped one to one onto $\mathbb{R}$. Put another way, a variable in a program that is instantiated to a floating-point value, in fact holds a real value. This observation is important in the definition of a concrete semantics for interval valued polynomials, which is defined over $\mathbb{R}$ and for real arithmetic.

Definition 3.7. (Concrete semantics of an interval valued polynomial.) *A concrete semantics for an interval valued polynomial $[p](\overline{\mathbf{x}}) = \sum_{I \in S} [a_I^-, a_I^+]\mathbf{x}^I$, where $\overline{\mathbf{x}} = (x_1, \ldots, x_k)$, is a function $[\![\ [p](\overline{\mathbf{x}})\ ]\!]_{re}(e) : (\prod_{\mathbf{f} \in \mathbf{F}}(V_{\mathbf{f}} \to \mathbb{F}_{\mathbf{f}})) \to \mathbb{IR}$ defined for a concrete environment e as*

$$[\![\ [p](\overline{\mathbf{x}})\ ]\!]_{re}(e) \;= \sum_{I \in S} [a_I^-, a_I^+](e(x_1))^{i_1} \cdot \ldots \cdot (e(x_k)^{i_k}).$$

3.4.2 Range Enclosure for Interval Valued Polynomials

Computing the boundaries of a floating-point expression is central in our analyses since the knowledge of its extreme values allows for deciding whether a program may overflow or divide by zero. Chapter 4 provides approximate methods to compute the boundaries of expressions. Here we show that every interval valued polynomial has an optimal upper and lower boundary. It is well-known that for each $\mathbf{B} \subseteq \mathbb{R}^k$, if a real function $f : \mathbf{B} \to \mathbb{R}$ is continuous then it is upper and lower-bounded by a closed interval $[f_{\min}, f_{\max}]$; namely, for all $\overline{\mathbf{x}} \in \mathbf{B}$, $f(\overline{\mathbf{x}}) \in [f_{\min}, f_{\max}]$. We say that a function is bounded if it is upper and lower-bounded. Note that a polynomial is a continuous function and therefore bounded for each $\mathbf{B} \subseteq \mathbb{R}^k$. We extend the notion of boundedness to interval valued polynomials.

Definition 3.8. (Range enclosure of an interval valued polynomial.) *We say that* $Enc([p](\overline{\mathbf{x}}), \mathbf{B}) \in \mathbb{IR}$ *is a* range enclosure *of* $[p](\overline{\mathbf{x}})$ *for a bounding box* $\mathbf{B} \in \mathbb{R}^k$ *if for all* $\overline{\mathbf{x}} \in \mathbf{B}$*,*

$$[p](\overline{\mathbf{x}}) \ \subseteq \ Enc([p](\overline{\mathbf{x}}), \mathbf{B}).$$

We show that for every interval valued polynomial defined over a bounding box, there exists a range enclosure. Let $[p](\overline{\mathbf{x}})$ be an interval valued polynomial and suppose that $[p](\overline{\mathbf{x}})$ is a single valued polynomial, that is, every interval coefficient is degenerated. Then, $[p](\overline{\mathbf{x}})$ is a continuous function and therefore bounded. Suppose now $[p](\overline{\mathbf{x}}) = \sum_{I \in S} [a_I^-, a_I^+]\mathbf{x}^I$. Note that each $r(\overline{\mathbf{x}}) \in [p](\overline{\mathbf{x}})$ is continuous and therefore bounded. The problem now is to find for each $\overline{\mathbf{x}} \in \mathbf{B}$ a polynomial $r(\overline{\mathbf{x}}) \in [p](\overline{\mathbf{x}})$ that upper or lower-bounds all polynomials in $[p](\overline{\mathbf{x}})$. Note that each polynomial is defined by a finite summation of monomials. Let $r(\overline{\mathbf{x}}) = \sum_{I \in S} a_I \cdot \mathbf{x}^I$. Then, for each $I \in S$,

$$a_I^- \cdot \mathbf{x}^I \ \leq a_I \cdot \mathbf{x} \ \leq \ a_I^+ \cdot \mathbf{x}^I \qquad \text{or} \qquad a_I^- \cdot \mathbf{x}^I \ \geq a_I \cdot \mathbf{x} \ \geq \ a_I^+ \cdot \mathbf{x}^I.$$

Note that $a_I \in [a_I^-, a_I^+]$ and thus, $\forall \overline{\mathbf{x}} \in \mathbb{R}^k$ where $\mathbf{x}^I \geq 0$,

$$a_I \cdot \mathbf{x}^I \ \in \ [a_I^- \cdot \mathbf{x}^I, a_I^+ \cdot \mathbf{x}^I] \ = \ [a_I^-, a_I^+]\mathbf{x}^I,$$

and $\forall \overline{\mathbf{x}} \in \mathbb{R}^k$ where $\mathbf{x}^I \leq 0$,

$$a_I \cdot \mathbf{x}^I \ \in \ [a_I^+ \cdot \mathbf{x}^I, a_I^- \cdot \mathbf{x}^I] \ = \ [a_I^+, a_I^-]\mathbf{x}^I.$$

Moreover, from the inequalities above we can note that the maximal values for boundaries are obtained when $a_I \in \{a_I^+, a_I^-\}$ for each $\overline{\mathbf{x}} \in \mathbb{R}^k$. Thus, for each $\overline{\mathbf{x}} \in \mathbf{B}$ there exists a single valued polynomial in $[p](\overline{\mathbf{x}})$ defined as the summation of monomials in which each coefficient is a boundary of each interval in $[p](x)$, which upper or lower-bounds $[p](\overline{\mathbf{x}})$.

Example 3.9. *Let* $[p](x) = [1, 2]x^3 + [1, 2]x^2$ *defined over* $\mathbf{B} = [-1, 1]$*. In this case, we can form up to four permutations and thus up to four single-valued polynomials with the boundaries of each interval; namely,*

$$r_0(x) = 1 \cdot x^3 + 1 \cdot x^2, \quad r_1(x) = 2 \cdot x^3 + 1 \cdot x^2,$$
$$r_2(x) = 1 \cdot x^3 + 2 \cdot x^2, \quad r_3(x) = 2 \cdot x^3 + 2 \cdot x^2.$$

Let $PR([p])$ denote the finite set of single valued polynomials defined by permuting the boundaries of each interval in $[p](\overline{\mathbf{x}})$. Note that $|\,PR([p])\,| = \ 2^{|\,S\,|}$. Moreover, let

$$M \ = \ \{\beta \ \in \ \mathbb{R} \ \mid \ \forall \overline{\mathbf{x}} \in \mathbf{B} \ . \ \beta \ = \ \max(r_i(\overline{\mathbf{x}})), \ \{r_i\} \ \subseteq \ PR([p])\} \text{ and}$$
$$m \ = \ \{\alpha \ \in \ \mathbb{R} \ \mid \ \forall \overline{\mathbf{x}} \in \mathbf{B} \ . \ \alpha \ = \ \min(r_i(\overline{\mathbf{x}})), \ \{r_i\} \ \subseteq \ PR([p])\}$$

be respectively the sets of all upper and lower-bounds of all the polynomials included in $PR([p])$. Then for all $\overline{\mathbf{x}} \in \mathbf{B}$, $[p](\overline{\mathbf{x}})$ is bounded by $[\min(m), \max(M)]$.

Definition 3.10. (Optimal range enclosure of an interval valued polynomial.) *The* optimal range *of an interval valued polynomial $[p](\overline{\mathbf{x}})$ over a box $\boldsymbol{B}$, is defined as the interval $[\min(m), \max(M)]$ and is denoted by $Opt([p](\overline{\mathbf{x}}), \mathbf{B})$.*

Thus the following proposition holds.

Proposition 3.11. *Let $Opt([p](\overline{\mathbf{x}}), \mathbf{B}) = [p_{\min}, p_{\max}]$. Then $[p](\overline{\mathbf{x}}) \in [p_{\min}, p_{\max}]$ and, for all $Enc([p](\overline{\mathbf{x}}), \mathbf{B}) = [l, u]$, $l \leq p_{\min}$ and $u \geq p_{\max}$.*

Clearly, any closed interval that includes $Opt([p](\overline{\mathbf{x}}), \mathbf{B})$ is a range enclosure of $[p](\overline{\mathbf{x}})$.

3.4.3 Operators for Interval Valued Polynomials

The following arithmetic will be used to construct the abstract semantics.

Definition 3.12. *The operators $\boxplus^\sharp$, $\boxminus^\sharp$, $\boxtimes^\sharp$, $\boxslash^\sharp$ and $\circledcirc^\sharp$ on interval valued polynomials are defined by:*

$$
\begin{array}{lcl}
\bullet i_I\mathbf{x}^I \boxplus^\sharp i'_I\mathbf{x}^I & = & (i_I \boxplus i'_I)\mathbf{x}^I \\
\bullet i_I\mathbf{x}^I \boxminus^\sharp i'_I\mathbf{x}^I & = & (i_I \boxminus i'_I)\mathbf{x}^I \\
\bullet i_I\mathbf{x}^I \ \boxtimes^\sharp \ i_J\mathbf{x}^J & = & (i_I \boxtimes i_J)\mathbf{x}^{I+J} \\
\bullet i_I\mathbf{x}^I \ \boxslash^\sharp \ i' & = & (i_I \boxslash i')\mathbf{x}^I \\
\bullet i_I\mathbf{x}^I \ \circledcirc^\sharp \ n & = & \begin{cases} [1,1] & \text{if } n = 0 \\ i_I\mathbf{x}^I \ \boxtimes^\sharp \left((i_I\mathbf{x}^I) \circledcirc^\sharp (n-1)\right) & \text{if } n \in \mathbb{N}^+. \end{cases}
\end{array}
$$

Algebraic properties such as associativity and distributivity of $\boxplus^\sharp$, $\boxminus^\sharp$, $\boxtimes^\sharp$ and $\boxslash^\sharp$ are important for proving the soundness of the abstract semantics. These properties are formalised in the following proposition.

Proposition 3.13.

$$
\begin{array}{lcl}
\bullet \left(\sum_{I\in S} \ i_I\mathbf{x}^I\right) \boxplus^\sharp \left(\sum_{I\in S} \ i'_I\mathbf{x}^I\right) & = & \sum_{I\in S} \left(i_I\mathbf{x}^I \boxplus^\sharp i'_I\mathbf{x}^I\right) \\
\bullet \left(\sum_{I\in S} \ i_I\mathbf{x}^I\right) \boxminus^\sharp \left(\sum_{I\in S} \ i'_I\mathbf{x}^I\right) & = & \sum_{I\in S} \left(i_I\mathbf{x}^I \boxminus^\sharp i'_I\mathbf{x}^I\right) \\
\\
\bullet \left(i_I\mathbf{x}^I\right) \ \boxtimes^\sharp \ \left(\sum_{J\in S} \ i_J\mathbf{x}^J\right) & = & \sum_{J\in S} (i_I\mathbf{x}^I \boxtimes^\sharp i_J\mathbf{x}^J) \text{ for some } I \in \mathbb{N}^k \\
\bullet \left(\sum_{I\in S} \ i_I\mathbf{x}^I\right) \ \boxtimes^\sharp \left(\sum_{J\in S'} \ i_J\mathbf{x}^J\right) & = & \sum_{I\in S} \sum_{J\in S'} \left(i_I\mathbf{x}^I \ \boxtimes^\sharp i_J\mathbf{x}^J\right) \\
\\
\bullet \left(\sum_{I\in S} \ i_I\mathbf{x}^I\right) \ \boxslash^\sharp \ i' & = & \sum_{I\in S} \left(i_I\mathbf{x}^I \boxslash^\sharp i'\right) \\
\\
\bullet \left(\sum_{I\in S} \ i_I\mathbf{x}^I\right) \ \circledcirc^\sharp \ n & = & \begin{cases} [1,1] & \text{if } n = 0 \\ \left(\sum_{I\in S} \ i_I\mathbf{x}^I\right) \ \boxtimes^\sharp \left(\left(\sum_{I\in S} \ i_I\mathbf{x}^I\right) \circledcirc^\sharp (n-1)\right) & \text{if } n \in \mathbb{N}^+. \end{cases}
\end{array}
$$

Proof. The proof is by induction on the cardinality of S. If $S = \emptyset$, the result is trivial. Suppose now that S has at least one element and $I_1 \in S$. Then, for the $\boxplus^\sharp$ operator we have:

$$\begin{aligned}
\big(\sum_{I\in S} i_I\mathbf{x}^I\big) \boxplus^\sharp \big(\sum_{I\in S} i'_I\mathbf{x}^I\big) &= \big(i_{I_1}\mathbf{x}^{I_1} \boxplus^\sharp \sum_{I\in S\setminus\{I_1\}} i_I\mathbf{x}^I\big) \boxplus^\sharp \big(i'_{I_1}\mathbf{x}^{I_1} \boxplus^\sharp \sum_{I\in S\setminus\{I_1\}} i'_I\mathbf{x}^I\big)\\
&= \big(i_{I_1}\mathbf{x}^{I_1} \boxplus^\sharp i'_{I_1}\mathbf{x}^{I_1}\big) \boxplus^\sharp \sum_{I\in S\setminus\{I_1\}} i_I\mathbf{x}^I \boxplus^\sharp \sum_{I\in S\setminus\{I_1\}} i'_I\mathbf{x}^I\\
&= \big(i_{I_1}\mathbf{x}^{I_1} \boxplus^\sharp i'_{I_1}\mathbf{x}^{I_1}\big) \boxplus^\sharp \sum_{I\in S\setminus\{I_1\}} \big(i_I\mathbf{x}^I \boxplus^\sharp i'_I\mathbf{x}^I\big)\\
&= \sum_{I\in S} \big(i_I\mathbf{x}^I \boxplus^\sharp i'_I\mathbf{x}^I\big),
\end{aligned}$$

where the third step follows from the induction hypothesis. A similar reasoning applies in the case of the $\boxminus^\sharp$ operator. In the case of the $\boxtimes^\sharp$ operator, for some $I \in \mathbb{N}^k$ and $J_1 \in S$,

$$\begin{aligned}
i_I\mathbf{x}^I \boxtimes^\sharp \sum_{J\in S} i_J\mathbf{x}^J &= \big(i_I\mathbf{x}^I \boxtimes^\sharp i_{J_1}\mathbf{x}^{J_1}\big) \boxplus^\sharp \big(i_I\mathbf{x}^I \boxtimes^\sharp \sum_{J\in S\setminus\{J_1\}} i_J\mathbf{x}^J\big)\\
&= \sum_{J\in S} \big(i_I\mathbf{x}^I \boxtimes^\sharp i_J\mathbf{x}^J\big).
\end{aligned}$$

For the general case of the $\boxtimes^\sharp$ operator,

$$\begin{aligned}
\big(\sum_{I\in S} i_I\mathbf{x}^I\big) \boxtimes^\sharp \big(\sum_{J\in S'} i_J\mathbf{x}^J\big) &= \big(i_{I_1}\mathbf{x}^{I_1} \boxtimes^\sharp \sum_{J\in S'} i_J\mathbf{x}^J\big) \boxplus^\sharp \big(\sum_{I\in S\setminus\{I_1\}} \big(i_I\mathbf{x}^I \boxtimes^\sharp \sum_{J\in S'} i_J\mathbf{x}^J\big)\big)\\
&= \big(i_{I_1}\mathbf{x}^{I_1} \boxtimes^\sharp \sum_{J\in S'} i_J\mathbf{x}^J\big) \boxplus^\sharp \big(\sum_{I\in S\setminus\{I_1\}} \sum_{J\in S'} (i_I\mathbf{x}^I \boxtimes^\sharp i_J\mathbf{x}^J)\big)\\
&= \sum_{I\in S} \sum_{J\in S'} \big(i_I\mathbf{x}^I \boxtimes^\sharp i_J\mathbf{x}^J\big).
\end{aligned}$$

In the case of the $\boxslash^\sharp$ operator, the proof is identical to the $\boxtimes^\sharp$ operator since for $i' = [a^-, a^+]$,

$$\big(\sum_{I\in S} i_I\mathbf{x}^I\big) \boxslash^\sharp i' = \big(\sum_{I\in S} i_I\mathbf{x}^I\big) \boxtimes^\sharp [1/a^+, 1/a^-].$$

The proof for the $\boxodot^\sharp$ operator follows the same reasoning as that for the product operator. □

3.4.4 Abstracting Rounding Errors

The abstract semantics for floating-point arithmetic must enclose the rounding errors introduced in the evaluation of a floating-point expression. In this section we summarise the results presented in [Min04a] which will serve to generalise the definition of relative

rounding error to interval valued polynomials. In this way, two classes of rounding errors must be considered: the class of errors in which normalised numbers are involved, and the class of errors in which denormalised numbers are involved. For a format $\mathbf{f} \in \mathbf{F}$, a relative rounding error of amplitude $2^{-\mathbf{p}}$ and an absolute error of amplitude $mf_{\mathbf{f}}$ are considered. Thus, for a normalised x, as noted in (3.1) (page 24),

$$\mid R_{\mathbf{f},\mathbf{r}}(x) - x \mid < 2^{-\mathbf{p}} \cdot \mid x \mid,$$

and for a denormalised x,

$$\mid R_{\mathbf{f},\mathbf{r}}(x) - x \mid \leq \ mf_{\mathbf{f}} \ .$$

Recall that the distance between two consecutive normalised floating-point values is proportional to $2^{-\mathbf{p}}$. Theorem [Min04a, cf. 7.4.1] proposes a boundary for rounding errors for $\oplus_{\mathbf{f},\mathbf{r}}$, $\ominus_{\mathbf{f},\mathbf{r}}$, $\otimes_{\mathbf{f},\mathbf{r}}$ and $\oslash_{\mathbf{f},\mathbf{r}}$ that is valid for all rounding modes. Then a relative rounding error approximation is proposed for *interval linear forms*. We generalise this definition to interval valued polynomials.

Definition 3.14. (Relative rounding error on an interval valued polynomial (adapted from [Min04a]).)

Given a floating-point format $\mathbf{f} = \langle \mathbf{e}, \mathbf{bias}, \mathbf{p} \rangle$, the *relative rounding error on the interval valued polynomial* $[p](\overline{\mathbf{x}}) = \sum_{I \in S} [a_I^-, a_I^+]\mathbf{x}^I$ is defined as

$$\varepsilon_{\mathbf{f}} \left(\sum_{I \in S} [a_I^-, a_I^+]\mathbf{x}^I \right) \ = \ \sum_{I \in S} \left[\mid a_I^- \mid \cdot(-2^{-\mathbf{p}}), \ \mid a_I^+ \mid \cdot 2^{-\mathbf{p}} \right] \mathbf{x}^I. \tag{3.2}$$

Particular in this definition is that as we raise $\mathbf{x}$ to the I-th power, the rounding error may be reduced to a smaller value if $\overline{\mathbf{x}} \in \mathbf{U}$, since any $\mathbf{x}^I$ would deliver a value less than or equal to $\mid 1 \mid$. To approximate the overall rounding error, Miné proposes to add these two types of rounding error. In this way, by adding the absolute rounding error $[-mf_{\mathbf{f}}, mf_{\mathbf{f}}]$ to the relative rounding error, we obtain the overall rounding error. This approximation is immune to those architectures that use a guard bit [2] when operating floating-point arithmetic.

3.4.5 Abstract Interpretation of Floating-Point Arithmetic

Given a polynomial in floating-point arithmetic, we construct an abstract semantics that safely over-approximates the concrete semantics and takes into account rounding errors. An abstract environment is a function that delivers the set of values that a variable can take at each program point for all the possible executions that reach each point.

[2] A guard bit is an extra bit used to store the least significant bit of the mantissa when computing a floating-point arithmetic operation in order to preserve precision.

Definition 3.15. (Abstract environment.) *An* abstract environment $e^\sharp$ *is defined by a mapping* $e^\sharp : \prod_{\mathbf{f}\in\mathbf{F}}(V_\mathbf{f} \rightarrow \mathbb{IR})$ *that associates to each variable an interval.*

The set of abstract environments for a format $\mathbf{f} \in \mathbf{F}$ is denoted by $AEnv_\mathbf{f}$.

We reformulate the abstract semantics presented in [Min04a, Def. 7.4.2], for interval valued polynomials. Although the underlying arithmetic in the following definition corresponds to interval valued polynomials, the main difference with respect to Miné's semantics can be found in the definitions of product and division. By reformulating such a semantics by means of interval valued polynomials, we can use the techniques presented in the following chapter for polyhedral abstraction.

Definition 3.16. (Abstract semantics (partly adapted from [Min04a]).) *Given an abstract environment* $e^\sharp$*, the interval valued polynomial* $[\![expr_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp)$ *for* $expr_\mathbf{f}$ *is constructed as follows:*

$$
\begin{aligned}
[\![[a,b]_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp) &= [R_{\mathbf{f},r}(a), R_{\mathbf{f},r}(b)] \\
[\![v]\!]^\sharp_{fl}(e^\sharp) &= [1,1]\ v \\
[\![\ominus p_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp) &= \boxminus^\sharp\ [\![p_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp) \\
[\![p_\mathbf{f}\ \oplus_{\mathbf{f},\mathbf{r}}\ q_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp) &= [\![p_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp)\ \boxplus^\sharp\ [\![q_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp)\ \boxplus^\sharp \\
&\quad \varepsilon_\mathbf{f}([\![p_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp))\ \boxplus^\sharp\ \varepsilon_\mathbf{f}([\![q_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp))\ \boxplus^\sharp\ [-mf_\mathbf{f}, mf_\mathbf{f}] \\
[\![p_\mathbf{f}\ \ominus_{\mathbf{f},\mathbf{r}}\ q_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp) &= [\![p_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp)\ \boxminus^\sharp\ [\![q_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp)\ \boxplus^\sharp \\
&\quad \varepsilon_\mathbf{f}([\![p_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp))\ \boxplus^\sharp\ \varepsilon_\mathbf{f}([\![q_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp))\ \boxplus^\sharp\ [-mf_\mathbf{f}, mf_\mathbf{f}] \\
[\![p_\mathbf{f}\ \otimes_{\mathbf{f},\mathbf{r}}\ q_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp) &= \big([\![p_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp)\ \boxtimes^\sharp\ [\![q_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp)\big)\ \boxplus^\sharp \\
&\quad \big(\varepsilon_\mathbf{f}([\![p_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp))\ \boxtimes^\sharp\ \varepsilon_\mathbf{f}([\![q_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp))\big)\ \boxplus^\sharp\ [-mf_\mathbf{f}, mf_\mathbf{f}] \\
[\![p_\mathbf{f}\ \oslash_{\mathbf{f},\mathbf{r}}\ q_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp) &= \big([\![p_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp)\ \boxslash^\sharp\ Opt([\![q_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp), e^\sharp(\overline{\mathbf{x}}))\big)\ \boxplus^\sharp \\
&\quad (\varepsilon_\mathbf{f}([\![p_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp))\ \boxslash^\sharp\ Opt([\![q_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp), e^\sharp(\overline{\mathbf{x}})))\ \boxplus^\sharp \\
&\quad [-mf_\mathbf{f}, mf_\mathbf{f}] \\
[\![p_\mathbf{f}\ \circledcirc_{\mathbf{f},\mathbf{r}}\ n]\!]^\sharp_{fl}(e^\sharp) &= [1,1] \\
[\![p_\mathbf{f}\ \circledcirc_{\mathbf{f},\mathbf{r}}\ n]\!]^\sharp_{fl}(e^\sharp) &= [\![p_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp)\ \boxtimes^\sharp\ \big([\![p_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp)\ \boxdot^\sharp\ (n-1)\big),
\end{aligned}
$$

where for the optimal range enclosure $Opt([\![q_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp), e^\sharp(\overline{\mathbf{x}}))$, $\overline{\mathbf{x}} = (x_1, \ldots, x_k) \in \mathbb{R}^k$ *where*

$\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(q_{\mathbf{f}})$, *and* $e^\sharp(\overline{\mathbf{x}}) \in \mathbb{IR}^k$ *where*

$$e^\sharp(\overline{\mathbf{x}}) \;=\; e^\sharp(x_1) \;\times\; \ldots \;\times\; e^\sharp(x_k).$$

In the definition above, two points must be noted. First, the product of two interval polynomial forms does not need intervalisation [Min04a, Def. 6.2.2] for any of its arguments, and subsequently no tailored strategy is needed to minimise the loss of precision. Second, in the operator division, an interval valued polynomial is divided by the range enclosure of another interval valued polynomial in an abstract environment. An alternative would be to compute a symbolic division, which would deliver a complex expression difficult to process. This range enclosure is in fact the optimal enclosure for a floating-point expression defined for an abstract environment.

An interval valued polynomial evaluated in $\mathbb{R}$ over-approximates the possible values that a floating-point expression can deliver in the environments $\gamma(e^\sharp)$. This is formalised in the following theorem, which was originally formulated for (symbolic) interval forms [Min04a]. However, the theorem also holds in our formulation for interval valued polynomials, since $\langle \mathbb{IR}, \subseteq \rangle$ is a poset, the abstract semantics is defined over sound interval arithmetic and a sound nonlinear rounding error approximation (definition 3.2), and the arithmetic operators presented in definition 3.12 for interval valued polynomials are associative and distributive. We formulate the soundness of the abstract semantics in a similar fashion to [Min04a, cf. theorem 7.4.2], but in this case for interval valued polynomials and optimal range enclosures; namely,

Theorem 3.17. *For all* $e \in \gamma(e^\sharp)$,

$$[\![expr_{\mathbf{f}}]\!]_{fl}(e) \subseteq [\![\; [\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp) \;]\!]_{re}(e).$$

Note that the theorem holds not only for the enclosure delivered by the intervalisation procedure defined in [Min04a], but for an optimal range enclosure. This is important for precision considerations, since we can define procedures that approximate the optimal enclosure to any degree of precision. However, we will require further techniques to compute a (possibly optimal) range enclosure. This is in contrast with Miné's approach that can compute the approximations without any further application of specific techniques. Thus, we trade efficiency for precision. On the other hand, for highly nonlinear floating-point expressions, the gain in precision may be crucial for the outcome of an analysis. As a final remark, note that analyses based on this abstract semantics may use a default abstract environment $e^\sharp(v) = [-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}]$ for each variable $v \in V_{\mathbf{f}}$.

3.5 Discussion and Related Work

Computational Approaches to Real Algebraic Structures. A major problem found when defining computational models for $\mathbb{R}$ is that of finiteness of representation. This in turn derives in other problems such as the rounding errors. Floating-point arithmetic is not associative or distributive. Floating-point systems [Hig02] are in fact a coarse finite approximation to real algebraic structures. Sound alternatives for specialised scientific applications, where precision is crucial, includes the systems provided in [Eda97, EP97,ES99], which are based on domain theory and fractal encoding of exact real numbers. Other alternatives include multi-precision floating point arithmetic [KJ93, Sco89], which uses floating-point numbers of variable precision. Also continued fractions [KM85, RT73, Sei83, Vui90] have been considered as an alternative for sound computational arithmetic.

Rounding Errors. Goubault [GMP01, Gou01, Mar02] provides a precise interval abstraction of floating-point expressions with no variables. Then he presents a complex (and precise) abstract semantics that soundly approximates the rounding errors of an expression. This precise semantics can be used to trace back the origin of floating-point arithmetic errors in a program. Miné [Min04b, Min04a] presents an nondeterministic abstraction of rounding errors using intervals. This brings a simpler framework for the analysis of floating-point errors in programs, such as the representation of relationships between variables. Sound abstractions of floating-point divisions must consider the truncation in the concrete division operator, in addition to rounding errors. In fact, truncation is an implicit rounding that should be considered in the abstraction for relational analyses, as remarked by Axel Simon in [Sim05]. Alternative approaches to studying roundoff errors in programs are those provided by stochastic methods. These approaches have not been fully accepted especially in the scientific community (criticism against such methods may be found for instance [Kah95]). Perturbation methods such as the CESTAC method [CV88, CV92, Vig96] are based on probabilistic error distribution and they cannot ensure sound answers.

Precision of Abstraction for Highly Nonlinear Expressions. In the following chapters we study some problems related to intervalisation when abstracting nonlinear floating-point expressions. Although intervalisation is efficient and simple, the loss in precision that it incurs in the case of nonlinear floating-point expressions may affect the outcome of an analysis, especially when analysing division by zero, where tight enclosures are fundamental.

Chapter 4

Polyhedra for Floating-Point Polynomial Expressions

In this chapter we present methods to abstract the semantics of floating-point polynomial expressions into polyhedra as well as to approximate and optimise the range enclosure of such semantics. These methods are based on interval Bernstein expansion and vary in computational cost and in precision of approximation to the optimal range enclosure. In contrast to existing methods for estimating range enclosures, such as intervalisation [Min04a], the increase in sophistication leads to higher precision as well as polyhedra abstraction.

4.1 Introduction

The nonlinearity of concrete and abstract functions can limit the scope of static analysis of program semantics. Nonlinear multivariate polynomial expressions are used for instance to abstract nested loops or subscripts in array references. Advanced features of compilers also require the manipulation of nonlinear expressions for optimisation transformations. The problem of analysing nonlinear expressions has been tackled for instance with the use of a technique from Approximation Theory called Bernstein expansion [Ber52, Ber54]. Bernstein expansion has been used in dependency analysis [MP94] and also in analyses based on integer multivariate polynomials for loop parallelisation [CT04]. In this chapter we use Bernstein expansion to soundly abstract floating-point expressions into the polyhedral domain.

Programs that use floating-point arithmetic are often analysed using numerical domains such as intervals (see for instance [HJvE01] and [Hig02]) and more recently octagons [Min04b, Min04a]. The use of the weakly relational octagon domain for analysing floating-point arithmetic, in contrast to the nonrelational interval domain, provides more precise analyses as relations between variables are captured. In [Min04b], after constructing a linear abstract semantics for floating-point expressions, based on interval linear forms, an adaptation to the octagon domain is proposed. This adaptation is algebraic in nature, thus appropriate for handling by an analyser.

However, such adaptation has two main drawbacks related to the (possible) nonlinearity of the concrete semantics. First, as the abstract semantics does not capture any nonlinear information, any subsequent adaptation into an octagon will loose considerable precision in this respect. In particular, a coarse linearisation method called intervalisation is proposed. Though sound and efficient, intervalisation may loose considerable precision when computing nonlinear expressions, and thus be inappropriate for analyses in which preserving precision in the abstract semantics is paramount. Second, the proposed adaptation technique does not allow for abstracting into polyhedra. Precise abstractions means that we can approximate the range enclosure of an expression as close to the optimal as required, and that in the analysis, we can use highly precise relational abstract domains such as the polyhedra.

In this chapter we study these two central characteristics of precise abstractions of nonlinear floating-point polynomial expressions using Bernstein expansion, which is based on Bernstein polynomials. Recall that the set of polynomials of degree less than or equal to n forms a vector space. A particular basis of interest is the Bernstein basis that allows for symbolic computation with low complexity and low resolution cost of complex program analysis. The coefficients of Bernstein polynomials enjoy two important properties for the analysis of floating-point expressions: the range enclosure property and the convexity property. The range enclosure of a polynomial defined over a box is included in the interval formed by the largest and smallest of these coefficients. Moreover, the poly-hull of all of these coefficients delivers a convex hull that includes all the values defined by a polynomial over a box.

Our contribution in this chapter is the adaptation of a series of methods based on Bernstein polynomials to abstract the nonlinear abstract semantics introduced in chapter 3, into polyhedra, and also to approximate the range enclosure of such semantics. Such methods vary in cost of computation as well as in the degree of precision attained. Methods that achieve a compromise between time and space efficiency, as well as precision are also proposed.

4.2 Bernstein Expansion

This section presents the results of [CS66], [RR84], [Far93] and [ZG98] for Bernstein expansion and its properties of interest. These results are central in the definitions of operators and further properties of interest for abstracting floating-point expressions into polyhedra.

Consider a single valued univariate polynomial, and let us represent it as

$$p(x) \;=\; a_n \cdot x^n \;+\; a_{n-1} \cdots x^{n-1} \;+\; \ldots \;+\; a_1 \cdot x \;+\; a_0.$$

This polynomial is a linear combination of the elementary polynomials $\{1, x, x^2, \ldots, x^n\}$. The set of polynomials of degree less than or equal to n form a vector space, where the set of elementary polynomials form a basis of this vector space. Any two polynomials in the vector space can be added, multiplied by a scalar, and moreover, the vector space properties hold.

A polynomial can also be represented as a linear combination of Bernstein basis elements. Bernstein expansion transforms a single valued polynomial from the power basis to the Bernstein basis form. A polynomial in Bernstein basis enjoys the convexity and range enclosure properties by which we can construct a polyhedron for a given polynomial and also obtain an estimate of the range enclosure of the polynomial when defined over a closed box.

We follow the notation of [ZG98], and in order to ease the presentation, we introduce first the case for univariate polynomials, and then the general case for multivariate polynomials.

4.2.1 Definitions

Case for Univariate Polynomials Given a polynomial $p(x)$ where

$$p(x) \;=\; \sum_{i=0}^{n} a_i \cdot x^i,$$

the *Bernstein form* of $p(x)$ is defined for $x \in \mathbf{U}$ as

$$p(x) \;=\; \sum_{i=0}^{n} b_i \cdot B_{n,i}(x),$$

where for each $i \in [0, n]$,

$$b_i \;=\; \sum_{j=0}^{n} \frac{\binom{i}{j}}{\binom{n}{j}} \cdot a_j \qquad \text{and} \qquad B_{n,i}(x) \;=\; \binom{n}{i} \cdot x^i \cdot (1-x)^{n-i}$$

are respectively the *Bernstein coefficients* and *Bernstein polynomial basis*, each a_j denotes the j-th coefficient of $p(x)$, and where each $\binom{n}{i}$ denotes a binomial coefficient defined by

$$\binom{n}{k} = \frac{n!}{(n-k)!k!},\ k \in [0,n],$$

where $n!$ denotes the factorial of n. Moreover, for each $i \in [0,n]$, $B_{n,i}(x) \geq 0$ if $x \in \mathbf{U}$.

The Bernstein coefficients enjoy important properties that will be useful for the abstraction of interval valued polynomials into the polyhedra domain and also for the computation of enclosures. In order to use these properties, the concept of *control points*, defined by means of Bernstein coefficients, proves central. The set of *control points* of $p(x)$ for $x \in \mathbf{U}$ is delivered by the mapping $Cp : (\mathbb{R} \to \mathbb{R}) \to \wp_{\mathrm{fn}}(\mathbb{R}^2)$ defined as

$$Cp(p(x)) \ = \ \left\{ \left(\tfrac{i}{n}, b_i\right) \ \in \ \mathbb{R}^2 \ \middle| \ \begin{array}{l} \forall i \in [0,n] \ : \ \forall x \in \mathbf{U} \ . \\ p(x) \ = \ \sum_{\ell=0}^{n} \ a_\ell \cdot x^\ell \ \wedge \ b_i \ = \ \sum_{j=0}^{n} \frac{\binom{i}{j}}{\binom{n}{j}} \cdot a_j \ \wedge \\ a_\ell \ = \ a_j \ \Leftrightarrow \ \ell \ = \ j \end{array} \right\},$$

where each b_i is a Bernstein coefficient.

Example 4.1. *For $x \in \mathbf{U}$ consider the polynomial*

$$p(x) \ = \ 7 \cdot x^3 - 7 \cdot x^2 + x + 5/4,$$

and let $a_0 = 5/4$, $a_1 = 1$, $a_2 = -7$, and $a_3 = 7$. Then the Bernstein coefficients for $p(x)$ are calculated as follows:

$$\begin{aligned} b_0 \ &= \ a_0 \ = \ 5/4 \\ b_1 \ &= \ a_0 \ + \ \frac{\binom{1}{1}}{\binom{3}{1}} \cdot a_1 \ = \ a_0 \ + \ 1/3 \cdot a_1 \ = \ 19/12 \\ b_2 \ &= \ a_0 \ + \ \frac{\binom{2}{1}}{\binom{3}{1}} \cdot a_1 \ + \ \frac{\binom{2}{2}}{\binom{3}{2}} \cdot a_2 \ = \ a_0 \ + \ 2/3 \cdot a_1 \ + \ 1/3 \cdot a_2 \ = \ -5/12 \\ b_3 \ &= \ a_0 \ + \ a_1 \ + \ a_2 \ + \ a_3 \ = \ 9/4. \end{aligned}$$

The set of control points is then

$$Cp(p(x)) \ = \ \{(0, 5/4),\ (1/3, 19/12),\ (2/3, -5/12),\ (1, 9/4)\}.$$

Case for Multivariate Polynomials This is the general case of Bernstein expansion. Let $\overline{\mathbf{x}} \in \mathbf{U}^k$ and $I, N \in \mathbb{N}^k$ where $I = (i_1, \ldots, i_k)$ and $N = (n_1, \ldots, n_k)$. Moreover, let the

binomial coefficient for N and I be defined as

$$\binom{N}{I} \;=\; \prod_{j=1}^{k} \binom{n_j}{i_j}.$$

Let $p(\overline{\mathbf{x}})$ be a polynomial of degree N, where for $S \subseteq \{I \in \mathbb{N}^k \mid I \leq N\}$,

$$p(\overline{\mathbf{x}}) = \sum_{I \in S} a_I \cdot \mathbf{x}^I.$$

Then the Bernstein form of $p(\overline{\mathbf{x}})$ is defined for $\overline{\mathbf{x}} \in \mathbf{U}^k$ by

$$p(\mathbf{x}) = \sum_{I \in S} b_I \cdot B_{N,I}(\mathbf{x}),$$

where for each $I \in S$,

$$b_I \;=\; \sum_{J \leq I} \frac{\binom{I}{J}}{\binom{N}{J}} \cdot a_J \qquad \text{and} \qquad B_{N,I}(\mathbf{x}) \;=\; B_{n_1,i_1}(x_1) \cdot \ldots \cdot B_{n_k,i_k}(x_k),$$

which are called the *Bernstein coefficients* and *Bernstein polynomial basis* respectively, and where each a_J denotes the J-th coefficient of $p(\overline{\mathbf{x}})$. Moreover, for each $I \in S$, $B_{N,I}(\overline{\mathbf{x}}) \geq 0$ if $\overline{\mathbf{x}} \in \mathbf{U}^k$. The set of control points of $p(\overline{\mathbf{x}})$ for $\overline{\mathbf{x}} \in \mathbf{U}^k$ is delivered by the mapping $CP : (\mathbb{R}^k \to \mathbb{R}) \to \wp_{\mathrm{fn}}(\mathbb{R}^{k+1})$ defined as

$$CP(p(\overline{\mathbf{x}})) \;=\; \left\{ \left(\tfrac{i_1}{n_1}, \ldots, \tfrac{i_k}{n_k}, b_I\right) \in \mathbb{R}^{k+1} \;\middle|\; \begin{array}{l} \forall S \;\in\; \wp_{\mathrm{fn}}(\mathbb{N}^k) \;.\; S \;=\; \{L \in \mathbb{N}^k \;\mid\; L \leq N\} \;: \\ \forall I \;\in\; S \;.\; I \;=\; (i_1, \ldots, i_k) \;: \\ \forall \overline{\mathbf{x}} \;\in\; \mathbf{U}^k \;: \\ p(\overline{\mathbf{x}}) = \sum_{L \in S} a_L \cdot \mathbf{x}^L \;\wedge\; b_I \;=\; \sum_{J \leq I} \frac{\binom{I}{J}}{\binom{N}{J}} \cdot a_J \;\wedge \\ a_L \;=\; a_J \;\Leftrightarrow\; L \;=\; J \end{array} \right\},$$

where each b_I is a Bernstein coefficient. Note that the mapping CP is a generalisation of the mapping Cp for the case of multiple variables.

4.2.2 Convex Hull Property

An important characteristic of Bernstein expansion is that the set of control points can be used to construct a polyhedron which includes a polynomial defined over a closed box.

Theorem 4.2. ([Far93].) *Let $p(\overline{\mathbf{x}})$, $\overline{\mathbf{x}} \in \mathbf{U}^k$, be a polynomial of degree N. Then,*

$$\left\{ \left(x_1, \ldots, x_k, p(\overline{\mathbf{x}})\right) \in \mathbb{R}^{k+1} \mid \overline{\mathbf{x}} \in \mathbf{U}^k \right\} \;\subseteq\; \uplus\, CP(p(\overline{\mathbf{x}})).$$

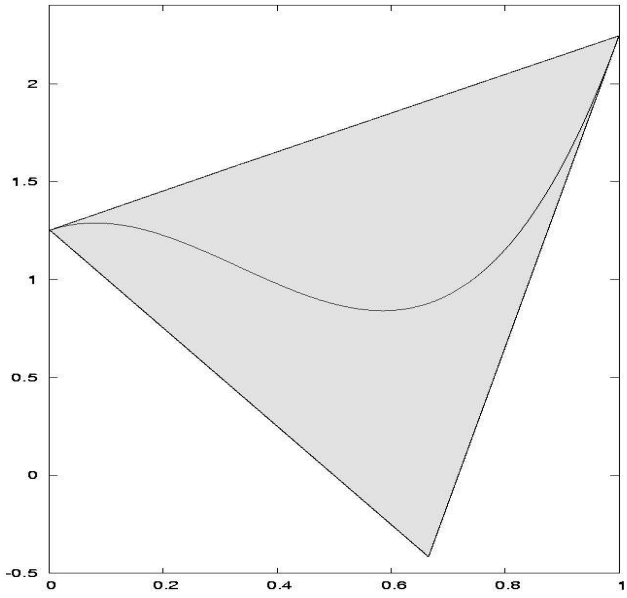


Figure 4.1: *Convex hulls for* $[p](x) = 7 \cdot x^3 - 7 \cdot x^2 + x + 5/4$, $x \in [0, 1]$.

Example 4.3. *In example 4.1, the poly-hull of the set of control points* $Cp(p(x))$ *delivers a convex closed polyhedron* $P \in \mathbb{CP}_2$ *where for all* $x \in \mathbf{U}$, $p(x) \in P$, *as shown in figure 4.1.*

4.2.3 Range Enclosure Property

A central problem in interval-based analysis is the computation of the range of a polynomial over an interval. From the set of control points of a polynomial defined over a closed intervals, it can be estimated the range enclosure of the polynomial. This is formalised in the following theorem.

Theorem 4.4. ([CS66].) *For* $N \in \mathbb{N}^k$, *let* $S \in \wp_{\mathrm{fn}}(\mathbb{N}^k)$ *where* $S \subseteq \{I \in \mathbb{N}^k \mid I \leq N\}$. *Let* $p(\overline{\mathbf{x}}) = \sum_{I \in S} \ a_I \cdot \mathbf{x}^I$, $\overline{\mathbf{x}} \in \mathbf{U}^k$, *be a polynomial of degree* N *and* $p(\overline{\mathbf{x}}) = \sum_{I \in S} \ b_I \cdot B_{N,I}(\overline{\mathbf{x}})$ *its Bernstein form. Moreover, let* $C = \{b_I \in \mathbb{R} \mid \forall I \in \mathbb{N}^k : \ I \in S\}$ *be the set of Bernstein coefficients of the Bernstein form of* $p(\overline{\mathbf{x}})$. *Then*

$$Enc(p(\overline{\mathbf{x}}), \mathbf{U}^k) \ = \ [\min(C), \max(C)].$$

When $\min(C), \max(C) \in \{p(\mathbf{0}), p(\mathbf{1})\}$ for $\overline{\mathbf{x}} \in \mathbf{U}^k$, the enclosure is optimal.

Example 4.5. *From the set of Bernstein coefficients in example 4.1,*

$$\begin{aligned} \max(b_0, b_1, b_2, b_3) \ &= \ 9/4 \\ \min(b_0, b_1, b_2, b_3) \ &= \ -5/12. \end{aligned}$$

Therefore $Enc(p(x), \mathbf{U}) = [-5/12, 9/4]$. *The upper boundary of the enclosure is optimal since* $p(1) = 9/4$.

4.2.4 The ϕ Transformation

An important property of the Bernstein coefficients obtained for a polynomial is that they are invariant under affine transformations. For $\mathbf{B} \in \mathbb{IR}^k$, $\overline{\mathbf{x}} \in \mathbf{B}$, and $\overline{\mathbf{y}} \in \mathbf{U}^k$, we define an affine transformation ϕ which transforms a polynomial $p(\overline{\mathbf{x}})$ into a polynomial $p'(\overline{\mathbf{y}})$. Then, by applying the Bernstein expansion on $p'(\overline{\mathbf{y}})$, we can obtain the Bernstein coefficients for $p(\overline{\mathbf{x}})$. For

$$\mathbf{B} \;=\; [a_1^-, a_1^+] \times \ldots \times [a_k^-, a_k^+],$$

where for each $i \in [1, k]$, $a_i^- < a_i^+$, let $\phi_{\mathbf{B}}$ be a mapping

$$\begin{array}{rrcl} \phi_{\mathbf{B}}: & \mathbf{B} & \rightarrow & \mathbf{U}^k \\ & \overline{\mathbf{x}} = (x_1, \ldots, x_k) & \rightarrow & \overline{\mathbf{y}} = (y_1, \ldots, y_k), \end{array}$$

where for $i \in [1, k]$,

$$y_i \;=\; \frac{x_i - a_i^-}{a_i^+ - a_i^-}.$$

Thus,

$$\begin{aligned} p(\phi_{\mathbf{U}}(\overline{\mathbf{x}})) \;&=\; p(\phi_{\mathbf{U}}(x_1), \ldots, \phi_{\mathbf{U}}(x_k)) \\ &=\; p'(\overline{\mathbf{y}}). \end{aligned}$$

Example 4.6. *Let* $x \in [a^-, a^+] = [-1/2, 1]$ *and consider the polynomial*

$$p(x) \;=\; 56/27 \cdot x^3 - 8/9 \cdot x + 115/108.$$

Moreover, let

$$\begin{aligned} y \;&=\; \frac{x - a^-}{a^+ - a^-} \\ &=\; 2/3 \cdot x + 1/3, \end{aligned}$$

where $x = 3/2 \cdot y - 1/2$. *Thus, by replacing each occurrence of* x *in* $p(x)$ *by* $3/2 \cdot y - 1/2$, *we obtain the new polynomial*

$$\begin{aligned} p(\phi_{\mathbf{U}}(x)) \;&=\; p'(y) \\ &=\; 7 \cdot y^3 - 7 \cdot y^2 + y + 5/4, \end{aligned}$$

where $y \in [0,1]$.

The inverse transformation $\phi_{\mathbf{B}}^{-1}$ is defined as

$$\begin{array}{rrcl} \phi_{\mathbf{B}}^{-1}: & \mathbf{U}^k & \rightarrow & \mathbf{B} \\ & \overline{\mathbf{y}} = (y_1, \ldots, y_k) & \rightarrow & \overline{\mathbf{x}} = (x_1, \ldots, x_k), \end{array}$$

where for $i \in [1, n]$,

$$x_i \;=\; (a_i^+ - a_i^-) \;\cdot\; y_i + a_i^-.$$

Thus,

$$\begin{aligned} p'(\phi_{\mathbf{B}}^{-1}(\overline{\mathbf{y}})) \;&=\; p'(\phi_{\mathbf{B}}^{-1}(y_1), \ldots, \phi_{\mathbf{B}}^{-1}(y_k)) \\ &=\; p(\overline{\mathbf{x}}). \end{aligned}$$

Example 4.7. *Given the polynomial* $p'(y)$ *of example 4.6, where* $x \in [1/2, 1]$ *and* $y \in \mathbf{U}$,

$$p'(\phi_{[1/2,1]}^{-1}(y)) \;=\; p(x),$$

by replacing each occurrence of y *in* $p'(y)$ *by* $2/3 \cdot x + 1/3$.

Note that $p'(\overline{\mathbf{y}}) = p(\phi_{\mathbf{B}}^{-1}(\overline{\mathbf{y}}))$, and therefore the range enclosure property is invariant under ϕ transformation.

4.3 Standard Method

In this section we present a method for abstracting floating-point expressions into polyhedra. Central in this method is the use of interval extensions of the Bernstein expansion, which is important for proving the soundness of our abstraction with respect to the nonlinear abstract semantics of definition 3.16. This method is the basis for developing further techniques for abstracting floating-point expressions into more precise polyhedra.

4.3.1 Definitions

We first formalise the concept of a floating-point polynomial expression which complies with the grammar presented in figure 3.7.

Definition 4.8. (Floating-point polynomial expression.) *Let* $expr_{\mathbf{f}}$ *be a floating-point expression defined for all* $e \in \gamma(e^\sharp)$. *Moreover, by definition 3.16,* $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$. *We say that* $expr_{\mathbf{f}}$ *is a floating-point* polynomial *expression if*

$$[\![expr_{\mathbf{f}}]\!]_{fl}^\sharp(e^\sharp) \;=\; [p](\overline{\mathbf{x}}),$$

for $e(\overline{\mathbf{x}}) \in e^\sharp(\overline{\mathbf{x}})$.

Let $N \in \mathbb{N}^k$ and $S \in \wp_{\text{fn}}(\mathbb{N}^k)$ where $S \subseteq \{I \in \mathbb{N}^k \mid I \leq N\}$. Then the Bernstein form of an interval valued polynomial of degree $N = (n_1, \ldots, n_k)$ such as

$$[p](\overline{\mathbf{x}}) \;=\; \sum_{I \in S} [a_I^-, a_I^+]\ \mathbf{x}^I$$

is defined for $\overline{\mathbf{x}} \in \mathbf{U}^k$ as

$$[p'](\overline{\mathbf{x}}) \;=\; \sum_{I \in S} [b_I^-, b_I^+]\ B_{N,I}(\overline{\mathbf{x}}),$$

where for each $I \in S$, $I = (i_1, \ldots, i_k)$,

$$[b_I^-, b_I^+] \;=\; \sum_{J \leq I} \frac{\binom{I}{J}}{\binom{N}{J}}\ [a_J^-, a_J^+] \qquad \text{and} \qquad B_{N,I}(\overline{\mathbf{x}}) \;=\; B_{n_1,i_1}(x_1) \cdot \ldots \cdot B_{n_k,i_k}(x_k).$$

are respectively the *interval Bernstein coefficients*and Bernstein polynomial basis. The relationship between $[p](\overline{\mathbf{x}})$ and $[p'](\overline{\mathbf{x}})$ is shown in section 4.3.4.

4.3.2 Interval ϕ Transformation

The ϕ transformation defined for single valued polynomials can be extended to interval valued polynomials by considering the product of an interval by the compact representation of a degenerate interval. Recall that the interval product $[a^-, a^+] \boxtimes [c, c] = [c, c] \boxtimes [a^-, a^+]$ is denoted by $[a^-, a^+]c$. Let $\overline{\mathbf{x}} \in \mathbf{B}$, $\mathbf{B} \in \mathbb{IR}^k$, and $\overline{\mathbf{y}} \in \mathbf{U}^k$. Then, by considering this compact representation, the definition of the ϕ transformation can be extended for an interval valued polynomial $[p](\overline{\mathbf{x}})$ as

$$\begin{aligned} [p](\phi_{\mathbf{B}}(\overline{\mathbf{x}})) \;&=\; [p](\phi_{\mathbf{B}}(x_1), \ldots, \phi_{\mathbf{B}}(x_k)) \\ &=\; [p'](\overline{\mathbf{y}}). \end{aligned}$$

This extension is illustrated in the following example.

Example 4.9. *Consider the interval valued polynomial*

$$[p](x) \;=\; [2, 2.1]x^3 + [-1, -0.9]x + [1, 1.1\ ],$$

for $x \in [-1/2, 1]$*, as plotted in figure 4.2. The interval* ϕ *transformation on* $[p'](y)$ *proceeds*

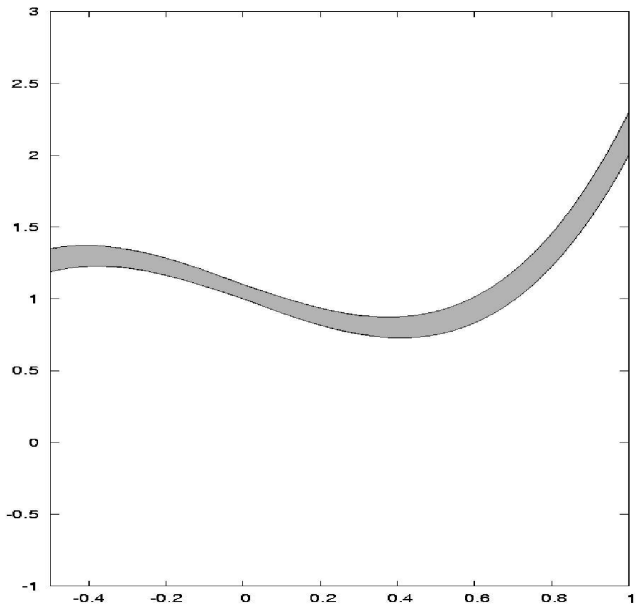


Figure 4.2: *Plot of* $[p](x) = [2, 2.1]x^3 + [-1, -0.9]x + [1, 1.1]$ *for* $x \in [-0.5, 1]$.

as follows:

$$\begin{aligned}
[p'](y) &= [p'](\phi_{\mathbf{B}}(x)) \\
&= [2, 2.1]\big(1.5 \cdot y - 0.5\big)^3 + [-1, -0.9]\big(1.5 \cdot y - 0.5\big) + [1, 1.1] \\
&= [2, 2.1]\big(3.375 \cdot y^3 - 3.375 \cdot y^2 + 1.125 \cdot y - 0.125\big) + \\
&\quad [-1, -0.9\ ]\big(1.5 \cdot y - 0.5\big) + [1, 1.1] \\
&= [6.75, 7.085]y^3 + [-7.0875, -6.75]y^2 + [0.75, 1.0125]y + [1.1875, 1.35],
\end{aligned}$$

where $y \in \mathbf{U}$. *In the polynomials* $p(x)$ *and* $p'(y)$ *in examples 4.6 and 4.7, note that* $p(x) \in [p](x)$ *and* $p'(y) \in [p'](y)$.

Let us define the interval extension of the inverse ϕ^{-1} transformation as

$$[p'](\phi_{\mathbf{B}}^{-1}(\overline{\mathbf{y}})) \;=\; [p](\phi_{\mathbf{B}}^{-1}(y_1), \ldots, \phi_{\mathbf{B}}^{-1}(y_k)).$$

Recall that the additive inverse of $\mathbb{IR}$ is not defined for proper intervals; namely, for proper intervals $[a^-, a^+]$ and $[b^-, b^+]$,

$$\begin{aligned}
[a^-, a^+] &\subseteq [a^-, a^+] \boxplus [b^-, b^+] \boxminus [b^-, b^+] \\
&= [a^- + b^- - b^+, a^+ + b^+ - b^-],
\end{aligned}$$

where $b^- - b^+ < 0$ and $b^+ - b^- > 0$. Therefore, when the coefficients of $[p](\overline{\mathbf{x}})$ are proper

intervals,

$$[p](\overline{\mathbf{x}}) \;\sqsubseteq\; [p'](\phi_{\mathbf{B}}^{-1}(\overline{\mathbf{y}})).$$

Example 4.10. *The inverse transformation for $[p'](y)$ in example 4.9, namely*

$$\begin{aligned} y &= (x + 1/2)/(3/2) \\ &= 2/3 \cdot x + 1/3, \end{aligned}$$

delivers the polynomial

$$[p'](x) \;=\; [2, 2.1]x^3 + [-0.15, 0.15]x^2 + [-1.15, -0.751]x + [0.9, 1.1995].$$

Note that $[p](x) \sqsubseteq [p'](x)$.

From this observation we can note also that the interval ϕ transformation is sound: namely, the optimal range enclosure of $[p](\overline{\mathbf{x}})$ for $\overline{\mathbf{x}} \in \mathbf{B}$ is included in the optimal range enclosure of $[p'](\overline{\mathbf{x}})$, which is computed by calculating first the interval ϕ transformation and then the inverse of the initial transformation.

4.3.3 Bernstein Mappings

We present now a series of mappings based on Bernstein expansion which deliver a set of points in $\mathbb{R}^{k+1}$ for a given interval valued polynomial. These sets of points are central in our framework for abstracting floating-point polynomial expressions into polyhedra, since they enjoy the convexity property as well as the range enclosure property. Moreover, from these mappings we can define recursive versions, with increased precision in the polyhedral abstraction and range enclosures.

Definition 4.11. (Interval Bernstein mapping.) *The interval Bernstein mapping $\pi_{\mathbf{U}} : (\mathbb{R}^k \to \mathbb{IR}) \to \wp_{\mathrm{fn}}(\mathbb{R}^{k+1})$ for a polynomial $p(\overline{\mathbf{x}})$ of degree $N \in \mathbb{R}^k$, where $N = (n_1, \ldots, n_k)$, is defined by*

$$\pi_{\mathbf{U}}([p](\overline{\mathbf{x}})) \;=\; \left\{ \left(\tfrac{i_1}{n_1}, \ldots, \tfrac{i_k}{n_k}, b_I\right) \in \mathbb{R}^{k+1} \;\middle|\; \begin{array}{l} \forall S \in \wp_{\mathrm{fn}}(\mathbb{N}^k) \,.\, S = \{L \in \mathbb{N}^k \mid L \leq N\} : \\ \forall I \in S \,.\, I = (i_1, \ldots, i_k) : \\ \forall \overline{\mathbf{x}} \in \mathbf{U}^k : \\ [p](\overline{\mathbf{x}}) = \sum_{L \in S} [a_L^-, a_L^+]\mathbf{x}^L \wedge \\ [b_I^-, b_I^+] = \sum_{J \leq I} \frac{\binom{I}{J}}{\binom{N}{J}} [a_J^-, a_J^+] \wedge b_I = b^-, b^+ \wedge \\ [a_L^-, a_L^+] = [a_J^-, a_J^+] \Leftrightarrow L = J \end{array} \right\}.$$

An interval Bernstein mapping may be referred as Bernstein mapping hereafter. Each

element contained in the set delivered by a Bernstein mapping is called a *control point*. The interval Bernstein mapping can be defined for any box $\mathbf{B} \in \mathbb{IR}^k$, where $\overline{\mathbf{x}} \in \mathbf{B}$, using the ϕ transformation. Thus,

$$\pi_{\mathbf{B}}([p](\overline{\mathbf{x}})) \;=\; \pi_{\mathbf{U}}([p](\phi_{\mathbf{B}}(\overline{\mathbf{x}})). \tag{4.1}$$

Moreover, we can define a mapping which uses the inverse ϕ transformation to map the set of control points from $\mathbf{U}^k$ onto $\mathbf{B}$:

$$\pi_{\mathbf{B}}([p](\phi_{\mathbf{B}}^{-1}(\overline{\mathbf{x}}))) \;=\; \left\{ \; \overline{\mathbf{c}} \;\in\; \mathbb{R}^{k+1} \;\middle|\; \begin{array}{l} \forall I \in \mathbb{N}^k \;: \\ \forall(\frac{i_1}{i_k},\ldots,\frac{i_k}{n_k},b_I) \;\in\; \pi_{\mathbf{U}}([p](\overline{\mathbf{y}})) \;: \\ \overline{\mathbf{c}} \;=\; (\phi_{\mathbf{B}}^{-1}(\frac{i_1}{n_1}),\ldots,\phi_{\mathbf{B}}^{-1}(\frac{i_k}{n_k}),b_I) \end{array} \right\}.$$

The following example illustrates the calculation process of sets of control points.

Example 4.12. *Consider the interval valued polynomial $[p'](y)$ of example 4.9, and let the polynomial coefficients be denoted by*

$$\begin{aligned} [a_0^-, a_0^+] &= [1.1875, 1.35] \\ [a_1^-, a_1^+] &= [0.75, 1.0125] \\ [a_2^-, a_2^+] &= [-7.0875, -6.75] \\ [a_3^-, a_3^+] &= [6.75, 7.085]. \end{aligned}$$

From these interval coefficients we calculate now the interval Bernstein coefficients $[b_i^-, b_i^+]$, *as follows:*

$$\begin{aligned} [b_0^{0,-}, b_0^{0,+}] \;&=\; \frac{\binom{0}{0}}{\binom{3}{0}} \;\cdot\; [a_0^-, a_0^+] \\ &=\; [a_0^-, a_0^+] \\ &=\; [1.1875, 1.35] \end{aligned}$$

$$\begin{aligned} [b_1^{0,-}, b_1^{0,+}] \;&=\; \frac{\binom{1}{0}}{\binom{3}{0}} \;\cdot\; [a_0^-, a_0^+] \;\boxplus\; \frac{\binom{1}{1}}{\binom{3}{1}} \;\cdot\; [a_1^-, a_1^+] \\ &=\; [a_0^-, a_0^+] \;\boxplus\; \frac{1}{3} \;\cdot\; [a_1^-, a_1^+] \\ &=\; [1.1875, 1.35] \;\boxplus\; \frac{1}{3} \;\cdot\; [0.75, 1.0125] \\ &=\; [1.4375, 1.6875] \end{aligned}$$

$$\begin{aligned}
[b_2^{0,-}, b_2^{0,+}] &= \frac{\binom{2}{0}}{\binom{3}{0}} \cdot [a_0^-, a_0^+] \boxplus \frac{\binom{2}{1}}{\binom{3}{1}} \cdot [a_1^-, a_1^+] \boxplus \frac{\binom{2}{2}}{\binom{3}{2}} \cdot [a_2^-, a_2^+] \\
&= [a_0^-, a_0^+] \boxplus \frac{2}{3} \cdot [a_1^-, a_1^+] \boxplus \frac{1}{3} \cdot [a_2^-, a_2^+] \\
&= [1.1875, 1.35] \boxplus \frac{2}{3}[0.75, 1.0125] \boxplus \frac{1}{3}[-7.0875, -6.75] \\
&= [-0.675, -0.225]
\end{aligned}$$

$$\begin{aligned}
[b_3^{0,-}, b_3^{0,+}] &= \frac{\binom{3}{0}}{\binom{3}{0}} \cdot [a_0^-, a_0^+] \boxplus \frac{\binom{3}{1}}{\binom{3}{1}} \cdot [a_1^-, a_1^+] \boxplus \frac{\binom{3}{2}}{\binom{3}{2}} \cdot [a_2^-, a_2^+] \boxplus \frac{\binom{3}{3}}{\binom{3}{3}} \cdot [a_3^-, a_3^+] \\
&= [a_0^-, a_0^+] \boxplus [a_1^-, a_1^+] \boxplus [a_2^-, a_2^+] \boxplus [a_3^-, a_3^+] \\
&= [1.1875, 1.35] \boxplus [0.75, 1.0125] \boxplus [-7.0875, -6.75] \boxplus [6.75, 7.085] \\
&= [1.6, 2.6975].
\end{aligned}$$

For each b_i^- *and* b_i^+, $i \in [0, 3]$, *we can now define the set of control points, namely*

$$\begin{aligned}
\pi_{\mathbf{U}}([p'](y)) = &\big\{(0, 1.1875),\ (1/3, 1.4375),\ (2/3, -0.675),\ (1, 1.6)\big\} \cup \\
&\big\{(0, 1.35),\ (1/3, 1.6875),\ (2/3, -0.225),\ (1, 2.7)\big\}
\end{aligned}$$

These points can be mapped back to the original interval $[-0.5, 1]$ *by the inverse of the affine transformation. Recall that*

$$\begin{aligned}
x &= \phi^{-1}(y) \\
&= (a^+ - a^-) \cdot y + a^- \\
&= 3/2 \cdot y - 1/2.
\end{aligned}$$

Hence for $x \in [a^-, a^+] = [-0.5, 1]$,

$$\begin{aligned}
\phi^{-1}(0) &= -1/2 \\
\phi^{-1}(1/3) &= 0 \\
\phi^{-1}(2/3) &= 1/2 \\
\phi^{-1}(1) &= 1.
\end{aligned}$$

Thus,

$$\begin{aligned}
P^- &= \big\{\big(-0.5, 1.1875\big), \big(0, 1.4375\big), \big(0.5, -0.675\big), \big(1, 1.6\big)\big\} \\
P^+ &= \big\{\big(-0.5, 1.35\big), \big(0, 1.6875\big), \big(0.5, -0.235\big), \big(1, 2.7\big)\big\}\Big).
\end{aligned}$$

4.3.4 Polyhedra Abstraction

This section formalises the abstraction of floating-point polynomial expressions into polyhedra. For this purpose, we must first show the relationship between an interval valued polynomial and its (interval) Bernstein form; namely, for $N \in \mathbb{N}$ and $S \in \wp_{\mathrm{fn}}(\mathbb{N}^k)$ where $S \subseteq \{I \in \mathbb{N}^k \mid I \leq N\}$, let

$$[p](\overline{\mathbf{x}}) \;=\; \sum_{I \in S} [a_I^-, a_I^+]\mathbf{x}^I$$

be an interval valued polynomial of degree N defined for $\overline{\mathbf{x}} \in \mathbf{U}^k$, and let

$$[p'](\overline{\mathbf{x}}) \;=\; \sum_{I \in S} [b_I^-, b_I^+]B_{N,I}(\overline{\mathbf{x}})$$

be the Bernstein form of $[p](\overline{\mathbf{x}})$. Consider now a polynomial $r(\overline{\mathbf{x}}) \in [p](\overline{\mathbf{x}})$ where

$$r(\overline{\mathbf{x}}) \;=\; \sum_{I \in S} \alpha_I \cdot \overline{\mathbf{x}}^I,\; \alpha_I \in [a_I^-, a_I^+].$$

The transformation of $r(\overline{\mathbf{x}})$ into its Bernstein form is

$$r(\overline{\mathbf{x}}) \;=\; \sum_{I \in S} b_I \cdot B_{N,I}(\overline{\mathbf{x}})$$

where

$$b_I \;=\; \sum_{J \leq I} \frac{\binom{I}{J}}{\binom{N}{J}} \cdot \alpha_I.$$

Note that each $\alpha_I \in [a_I^-, a_I^+]$. Moreover,

$$b_I^- \;=\; \sum_{J \leq I} \frac{\binom{I}{J}}{\binom{N}{J}} \cdot a_I^-, \qquad b_I \;=\; \sum_{J \leq I} \frac{\binom{I}{J}}{\binom{N}{J}} \cdot \alpha_I, \qquad b_I^+ \;=\; \sum_{J \leq I} \frac{\binom{I}{J}}{\binom{N}{J}} \cdot a_I^+.$$

For

$$c \;=\; \sum_{J \leq I} \frac{\binom{I}{J}}{\binom{N}{J}}$$

we have that

$$c \cdot a_I^- \;\leq\; c \cdot a_I \;\leq\; c \cdot a_I^+.$$

Hence $b_I \in [b_I^-, b_I^+]$ and $\forall r(\overline{\mathbf{x}}) \in [p](\overline{\mathbf{x}})$,

$$r(\overline{\mathbf{x}}) \;\in\; \sum_{I \in S} [b_I^-, b_I^+]B_{N,I}(\overline{\mathbf{x}}).$$

Thus

$$[p](\overline{\mathbf{x}}) \;\sqsubseteq\; \sum_{I \in S} [b_I^-, b_I^+] B_{N,I}(\overline{\mathbf{x}}).$$

These observations are formalised in the following theorem.

Theorem 4.13. *Let $N \in \mathbb{N}$ and $S \in \wp_{\mathrm{fn}}(\mathbb{N}^k)$ where $S \subseteq \{I \in \mathbb{N}^k \mid I \leq N\}$. Moreover, let $[p](\overline{\mathbf{x}}) = \sum_{I \in S} [a_I^-, a_I^+]\mathbf{x}^I$ be an interval valued polynomial of degree N defined for $\overline{\mathbf{x}} \in \mathbf{U}^k$, and let $\sum_{I \in S} [b_I^-, b_I^+] B_{N,I}(\overline{\mathbf{x}})$ be the Bernstein form of $[p](\overline{\mathbf{x}})$. Then*

$$\sum_{I \in S} [a_I^-, a_I^+]\mathbf{x}^I \;\sqsubseteq\; \sum_{I \in S} [b_I^-, b_I^+] B_{N,I}(\overline{\mathbf{x}}).$$

The theorem above also says that the convex hull obtained by Bernstein expansion for a single valued polynomial $r(\overline{\mathbf{x}})$ for $\overline{\mathbf{x}} \in \mathbf{U}^k$ is included in the convex hull obtained for an interval valued polynomial that includes $r(\overline{\mathbf{x}})$ for $\overline{\mathbf{x}} \in \mathbf{U}^k$. That is, for some $\overline{\mathbf{x}} \in \mathbf{U}^k$ for $r(\overline{\mathbf{x}})$ we can associate a Bernstein coefficient b_I. But for the same $\overline{\mathbf{x}}$ we can associate for $[p](\overline{\mathbf{x}})$ an interval Bernstein coefficient $[b_I^-, b_I^+]$ where $b_I \in [b_I^-, b_I^+]$. Thus, by the definition of the poly-hull operator in $\mathbb{R}$,

$$\begin{aligned} b_I &= \uplus\, b_I \\ &\in \uplus\{b_I^-,\ b_I^+\} \\ &= [b_I^-,\ b_I^+]. \end{aligned}$$

This can be generalised to $\mathbb{R}^k$, as follows.

Proposition 4.14. *Let $\mathbf{B} \in \mathbb{IR}^k$ and $[p](\overline{\mathbf{x}})$ an interval valued polynomial defined for all $\overline{\mathbf{x}} \in \mathbf{B}$. Then,*

$$\uplus \left\{ (x_1, \ldots, x_k, c) \in \mathbb{R}^{k+1} \;\middle|\; \begin{array}{l} \forall \overline{\mathbf{x}} \in \mathbf{B} : \forall r(\overline{\mathbf{x}}) \in [p](\overline{\mathbf{x}}) : \\ c = r(\overline{\mathbf{x}}) \end{array} \right\} \subseteq \uplus\, \pi_{\mathbf{B}}([p](\overline{\mathbf{x}})).$$

Example 4.15. *Consider the interval valued polynomial*

$$[p](x) \;=\; [2, 2.1]x^3 + [-1, -0.9]x + [1, 1.1]$$

of example 4.12, and let a single valued polynomial

$$r(x) \;=\; 2 \cdot x^3 - x + 1,$$

for $x \in [-0.5, 1]$, where $r(x) \in [p](x)$. Thus, after ϕ transformation on $r(x)$ we have the

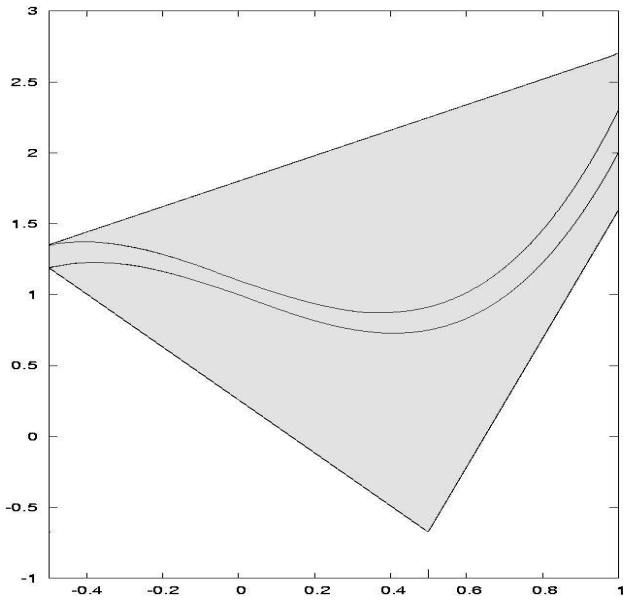


Figure 4.3: *Convex hull for* $[p](x) = [2, 2.1]x^3 + [-1, -0.9]x + [1, 1.1]$, $x \in [-0.5, 1]$.

polynomial

$$r'(y) \;=\; 6.75 \cdot x^3 - 6.75 \cdot x^2 + 0.75 \cdot x + 1.25,$$

for $y \in \mathbf{U}$. *The Bernstein coefficients for* $r'(y)$ *are* $b_0 = 1.25$, $b_1 = 1.5$, $b_2 = -0.5$ *and* $b_3 = 2$. *Note that from the interval Bernstein coefficients* $[b_i^-, b_i^+]$, $i \in [0, 3]$, *for* $[p](x)$ *(see page 47),* $b_i \in [b_i^-, b_i^+]$. *Moreover, it can be noted that for*

$$\begin{aligned} P^- &= \{(i/3, b_i^-) \in \mathbb{R}^2 \mid i \in [0, 3]\} \text{ and} \\ P^+ &= \{(i/3, b_i^+) \in \mathbb{R}^2 \mid i \in [0, 3]\} \end{aligned}$$

it holds that

$$\uplus \left(\{(i/3, b_i) \in \mathbb{R}^2 \mid i \in [0, 3]\}\right) \;\subseteq\; \uplus \left(P^- \;\cup\; P^+\right).$$

Figure 4.3 plots $[p](x)$ *for* $x \in [-0.5, 1]$ *and the convex hull delivered by the poly-hull of the set of control points obtained from the Bernstein mapping.*

From theorem 4.13, we can now present an abstraction of floating-point polynomial expressions into polyhedra. Recall from theorem 3.17 that an interval valued polynomial evaluated in $\mathbb{R}$ delivers an interval which over-approximates the possible values that a floating-point polynomial expression can deliver in the concrete environments $\gamma(e^\sharp)$. The approach we take here is to abstract an interval valued polynomial into a polyhedron using the Bernstein mappings, and to prove that the polyhedron includes the concrete semantics

a floating-point polynomial expression. The following theorem shows that the abstraction of a floating-point polynomial expression into polyhedra is sound.

Theorem 4.16. $\forall e \in \gamma(e^\sharp)$, $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_\mathbf{f})$, *where* $\overline{\mathbf{x}} = (x_1, \ldots, x_k)$ *and* $e(x_i) \subseteq e^\sharp(x_i)$, $i \in [1, k]$,

$$\uplus \left\{ \big(e(x_1), \ldots, e(x_k), c\big) \in \mathbb{F}^{k+1} \;\middle|\; \begin{array}{l} \forall c \in \mathbb{F}_\mathbf{f} : \\ c = [\![expr_\mathbf{f}]\!]_{fl}(e) \end{array} \right\} \subseteq \uplus\, \pi_{e^\sharp(\overline{\mathbf{x}})}([\![expr_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp))$$

Proof. Note first that

$$\begin{array}{rll} [\![expr_\mathbf{f}]\!]_{fl}(e) & \subseteq & [\![[\![expr_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp)]\!]_{re}(e) \qquad \text{[by theorem 3.17]} \\ & = & [\![\ [p](\overline{\mathbf{x}})\ ]\!]_{re}(e). \qquad \text{[by definition 3.7]} \end{array}$$

Moreover, note that for all $\overline{\mathbf{x}} \in \mathbb{R}^k$,

$$[\![\ [p](\overline{\mathbf{x}})\ ]\!]_{re}(e) \sqsubseteq [p](\overline{\mathbf{x}}),$$

where $\sqsubseteq$ denotes the partial ordering between interval valued polynomials. Given $\mathbf{B} \in \mathbb{IR}$ and $[p](\overline{\mathbf{x}})$, for all $\overline{\mathbf{x}} \in \mathbb{R}^k$

$$\mathbf{B} \sqsubseteq [p](\overline{\mathbf{x}}) \Leftrightarrow \mathbf{B} \subseteq [p](\overline{\mathbf{x}}).$$

Thus, for $[\![expr_\mathbf{f}]\!]_{fl}(e) \in \mathbb{IR}$, for all $\overline{\mathbf{x}} \in \mathbb{R}^k$, and for all $\overline{\mathbf{x}} \in \mathbb{R}^k$ where $e(\overline{\mathbf{x}}) \in e^\sharp(\overline{\mathbf{x}})$,

$$[\![expr_\mathbf{f}]\!]_{fl}(e) \subseteq [p](\overline{\mathbf{x}}),$$

Therefore,

$$\begin{array}{l} \uplus \left\{ \big(e(x_1), \ldots, e(x_k), c\big) \in \mathbb{F}^{k+1} \mid \forall c \in \mathbb{F}_\mathbf{f} : c = [\![expr_\mathbf{f}]\!]_{fl}(e) \right\} \\ \subseteq \uplus \left\{ [p](\overline{\mathbf{x}}) \in \mathbb{IR} \mid \forall \overline{\mathbf{x}} \in \mathbb{R}^k : e(\overline{\mathbf{x}}) \in e^\sharp(\overline{\mathbf{x}}) \right\} \\ \subseteq \uplus\, \pi_{e^\sharp(\overline{\mathbf{x}})}([\![expr_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp)). \end{array}$$

□

4.3.5 Range Enclosures

The range enclosure of a floating-point polynomial expression can be approximated by the interval Bernstein coefficients, as follows.

Theorem 4.17. (Range enclosure of a floating-point polynomial expression.) *Let* $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_\mathbf{f})$. *Moreover, let* $\pi_{e^\sharp(\overline{\mathbf{x}})}([\![expr_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp)) = \{(x_1, \ldots, x_k, b_I) \in \mathbb{R}^{k+1} \mid$

$I \in S,\ S \in \wp_{\mathrm{fn}}(\mathbb{N}^k)\}$. *Then,*

$$Enc\big([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp),\ e^\sharp(\overline{\mathbf{x}})\big) \ = \ [\min(b_I),\ \max(b_I)].$$

Proof. Let $[\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp) = [p](\overline{\mathbf{x}})$ and $r(\overline{\mathbf{x}}) \in [p](\overline{\mathbf{x}})$. By theorem 4.4,

$$Enc(r(\overline{\mathbf{x}}), \mathbf{U}^k) \ = \ [\min_{I\in S}\ b_I,\ \max_{I\in S}\ b_I],$$

where each b_I is a Bernstein coefficient of $r(\overline{\mathbf{x}})$. By proposition 3.13 that shows the soundness of interval arithmetic for interval valued polynomials, and by theorem 4.13, $\forall I \in S$,

$$b_I \ \in \ [\min_{I\in S}\ b_I,\ \max_{I\in S}\ b_I] \ \subseteq \ [b_I^-, b_I^+],$$

where each b_I^- and b_I^+ are Bernstein coefficients delivered by the Bernstein mapping for $[p](\overline{\mathbf{x}})$ defined over $e^\sharp(\overline{\mathbf{x}})$. □

As a consequence of theorems 4.16 and 4.17, we have the following result.

Corollary 4.18. $\forall e \in \gamma(e^\sharp)$, $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$, $\forall c \in [\![expr_{\mathbf{f}}]\!]_{fl}(e)$

$$c \ \in \ Enc([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp), e^\sharp(\overline{\mathbf{x}})).$$

Moreover, by theorem 4.16 of soundness of the abstraction of floating-point polynomial expressions into polyhedra, and corollary 4.18, we have that

$$Opt([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp),\ e^\sharp(\overline{\mathbf{x}})) \ \subseteq \ Enc([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp),\ e^\sharp(\overline{\mathbf{x}})).$$

Example 4.19. *In example 4.12, the range enclosure of* $[p](x)$ *for* $x \in [-0.5, 1]$ *is*

$$\begin{aligned} Enc([p'](y), \mathbf{U}) \ &= \ Enc([p](x), [-0.5, 1]) \\ &= \ [\min(\{b \in \mathbb{R} \mid (x,b) \in P^+ \cup P^-\}),\ \max(\{b \in \mathbb{R} \mid (x,b) \in P^+ \cup P^-\})] \\ &= \ [-0.675, 2.7]. \end{aligned}$$

The knowledge of the range of a floating-point polynomial expression is important for instance to determine whether such expression may overflow.

4.4 Recursive Method

Tightening the range enclosures is important for the abstraction of the abstract division: the closer to the optimal range enclosure, the more precise abstractions of abstract divisions can be achieved. Moreover, the recursive method presented here is useful for finding roots

of floating-point expressions, which in turn is important for the analysis of division by zero. In addition, the knowledge of tighter enclosures is useful for reducing the number of false positives in the analysis of floating-point overflow.

The range enclosure can be tightened by increasing the degree of the Bernstein expansion, as noted in [CS66, Rok77]. That is to say, by computing the expansion for a degree greater than that of the given polynomial. However, as shown in [Gar86, Zet91, Gar93], subdivision-based methods prove more efficient at producing tight range enclosures. The recursive method is a subdivision-based method that delivers a polyhedra powerset which provides range enclosures that are tighter than those provided in the standard method. The central idea in the recursive method is that of segmentation of an abstract environment. We can apply the standard method to subintervals of an abstract environment and hence gain in precision. The outcome of the recursive method is a finite set of polyhedra.

Central in the recursive method is the concept of segmentation of abstract environments. By segmenting an abstract environment we mean that the box that defines the application of an abstract environment to the set of variables of a floating-point expression, is segmented.

An important property of this method is that transfer functions based on it deliver a set of polyhedra that are nonredundant. Thus, no preprocessing of the abstraction is required before computing an analysis.

4.4.1 Definitions

The recursive method is based on segmenting the domain over which an expression is defined. This domain is approximated by an abstract environment which maps each variable of the expression to an interval. Thus, we define segmentations for intervals mapped by abstract environments.

Definition 4.20. (Degree of segmentation.) *The* degree of segmentation $n \in \mathbb{N}$, $n \geq 1$, *is the number of subintervals of same width in which an interval can be segmented.*

For instance, for $n = 2$, the interval $\mathbf{U}$ is segmented into $[0, 1/2]$ and $[1/2, 1]$. We define a segmentation operator for k-variate boxes, as follows.

Definition 4.21. (Segmentation operator.) *Let $n \geq 1$ be a segmentation degree. Moreover, let $I \in \mathbb{N}^k$, $I = (i_1, \ldots, i_k)$, where $I \in [0, n-1]^k$. Then, the segmentation operator $\varphi_{n,I} : \mathbb{IR}^k \to \mathbb{IR}^k$ is defined as*

$$\varphi_{n,i}([a^-, a^+]) \;=\; [\; a^- + i \cdot \frac{w}{n},\; a^- + (i+1) \cdot \frac{w}{n} \;]$$
$$\varphi_{n,I}([a_1^-, a_1^+] \times \ldots \times [a_k^-, a_k^+]) \;=\; \varphi_{n,i_1}([a_1^-, a_1^+]) \;\times\; \ldots \;\times\; \varphi_{n,i_1}([a_k^-, a_k^+]).$$

Definition 4.22. (I-th segmentation of an abstract environment.) *Let $I \in [0, n-1]^k$ where $I = (i_1, \ldots, i_k)$. The I-*th segmentation of segmentation degree $n \geq 1$ of an

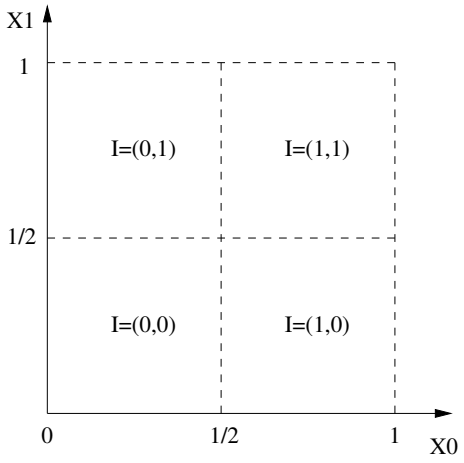


Figure 4.4: *Plot of segmented unit box for* $n = 2$.

abstract environment $e^\sharp$ *is a mapping* $e^\sharp_{n,I} : \prod_{\mathbf{f}\in\mathbf{F}}(V_\mathbf{f} \to \mathbb{IR})$ *defined as*

$$e^\sharp_{n,I}(\overline{\mathbf{x}}) = \varphi_{n,I}(e^\sharp(\overline{\mathbf{x}})),$$

where $\varphi_{n,I}(e^\sharp(\overline{\mathbf{x}})) = \varphi_{n,i_1}(e^\sharp(x_1)) \times \ldots \times \varphi_{n,i_1}(e^\sharp(x_k))$, $\overline{\mathbf{x}} = (x_1, \ldots, x_k)$.

The number of segments for an abstract environment defined for k variables and segmentation degree n is n^k.

It is important to note that for all $n \geq 1$ and some $I \in \mathbb{N}^k$, $e^\sharp_{n,I} \in AEnv_\mathbf{f}$ is an abstract environment. However, to make it noticeable that an abstract environment $e^\sharp_{n,I}$ is defined by a segmentation operator, we refer to $e^\sharp_{n,I}$ as a segmentation.

Example 4.23. *For a floating-point polynomial expression* $expr_\mathbf{f}$*, let* $x \in \mathsf{vars}(expr_\mathbf{f})$*, and* $e^\sharp(x) = \mathbf{U}$*. Then for a segmentation degree* $n = 2$ *on* $e^\sharp(x)$*,*

$$e^\sharp_{2,0}(x) = [0, \frac{1}{2}] \qquad \textit{and} \qquad e^\sharp_{2,1}(x) = [\frac{1}{2}, 1].$$

Example 4.24. *Let* $expr_\mathbf{f}$ *be a floating-point polynomial expression where* $\mathsf{vars}(\overline{\mathbf{x}}) \in \mathsf{vars}(expr_\mathbf{f})$*, and* $\overline{\mathbf{x}} = (x_0, x_1)$*. Moreover, let* $e^\sharp(\overline{\mathbf{x}}) = \mathbf{U}^2$*, where*

$$e^\sharp(x_0) \;=\; e^\sharp(x_1) \;=\; [0, 1].$$

Then for a segmentation degree $n = 2$ on $e^\sharp(\overline{\mathbf{x}})$,

$$\begin{aligned}
e^\sharp_{2,(0,0)}(\overline{\mathbf{x}}) &= e^\sharp_{2,0}(x_0) \times e^\sharp_{2,0}(x_1) = [0, \frac{1}{2}] \times [0, \frac{1}{2}] \\
e^\sharp_{2,(0,1)}(\overline{\mathbf{x}}) &= e^\sharp_{2,0}(x_0) \times e^\sharp_{2,1}(x_1) = [0, \frac{1}{2}] \times [\frac{1}{2}, 1] \\
e^\sharp_{2,(1,0)}(\overline{\mathbf{x}}) &= e^\sharp_{2,1}(x_0) \times e^\sharp_{2,0}(x_1) = [\frac{1}{2}, 1] \times [0, \frac{1}{2}] \\
e^\sharp_{2,(1,1)}(\overline{\mathbf{x}}) &= e^\sharp_{2,1}(x_0) \times e^\sharp_{2,1}(x_1) = [\frac{1}{2}, 1] \times [\frac{1}{2}, 1].
\end{aligned}$$

Figure 4.4 plots the unit box and the subboxes that result of segmenting it for $n = 2$.

From these definitions, we can now formalise the recursive method.

Definition 4.25. (Recursive Bernstein mapping.) *Let $expr_\mathbf{f}$ be a floating-point polynomial expression and $e^\sharp$ an abstract environment where $[\![expr_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp) = [p](\overline{\mathbf{x}})$ and $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_\mathbf{f})$. A* recursive Bernstein mapping $\pi^n_\mathbf{B} : (\mathbb{R}^k \to \mathbb{IR}) \to \wp_{\mathrm{fn}}(\wp_{\mathrm{fn}}(\mathbb{R}^{k+1}))$ *for $\mathbf{B} \in \mathbb{IR}^k$ and segmentation degree n is defined as*

$$\pi^n_{e^\sharp(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}})) = \left\{ P \in \wp_{\mathrm{fn}}(\mathbb{R}^{k+1}) \;\middle|\; \begin{array}{l} \forall I \in [0, n-1]^k \ : \\ e^\sharp_{n,I}(\overline{\mathbf{x}}) = \varphi_{n,I}(e^\sharp(\overline{\mathbf{x}})) \ \wedge \ P = \pi_{e^\sharp_{n,I}(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}})) \end{array} \right\}.$$

4.4.2 Polyhedra Powerset Abstraction

The abstraction presented in this section is based on the abstraction introduced in the standard method. Central is the definition of segmentations of an abstract environment for a floating-point polynomial. Then it must be applied the Bernstein mapping for each new segmentation. The following mapping delivers an abstraction of a floating-point polynomial defined for an abstract environment $e^\sharp$, into a polyhedra powerset.

Definition 4.26. (Powerset polyhedra abstraction operator.) *Let $expr_\mathbf{f}$ be a floating-point polynomial expression defined for all $e \in \gamma(e^\sharp)$, where $\mathsf{vars}(\overline{\mathbf{x}}) = \mathsf{vars}(expr_\mathbf{f})$, that $[p](\overline{\mathbf{x}}) = [\![expr_\mathbf{f}]\!]^\sharp_{fl}(e^\sharp)$, and $e(\overline{\mathbf{x}}) \in e^\sharp(\overline{\mathbf{x}})$. Moreover, let $n \geq 1$ be a segmentation degree. Then the operator $\Upsilon^n_{e^\sharp} : (\mathbb{R}^k \to \mathbb{IR}) \to \wp_{\mathrm{fn}}(\mathbb{CP}_{k+1})$ is defined as*

$$\Upsilon^n_{e^\sharp}([p](\overline{\mathbf{x}})) = \left\{ P \in \mathbb{CP}_{k+1} \;\middle|\; \begin{array}{l} \forall I \in \mathbb{N}^k \,.\, I \in [0, n-1]^k \ : \\ \forall Q \in \wp_{\mathrm{fn}}(\mathbb{R}^{k+1}) \ : \ \forall e^\sharp_{n,I}(\overline{\mathbf{x}}) \in AEnv_\mathbf{f} \ : \\ e^\sharp_{n,I}(\overline{\mathbf{x}}) = \varphi_{n,I}(e^\sharp(\overline{\mathbf{x}})) \ \wedge \ Q = \pi^n_{e_{n,I}(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}})) \ \wedge \ P = \uplus\, Q \end{array} \right\}.$$

Example 4.27. *To attain a more precise polyhedral abstraction, we can apply the recursive method to $[p](x)$ of example 4.9. For a segmentation degree $n = 2$, we first segment the original interval $[-0.5, 1]$ into $[-0, 5, 0.25]$ and $[0.25, 1]$; then we apply the standard method*

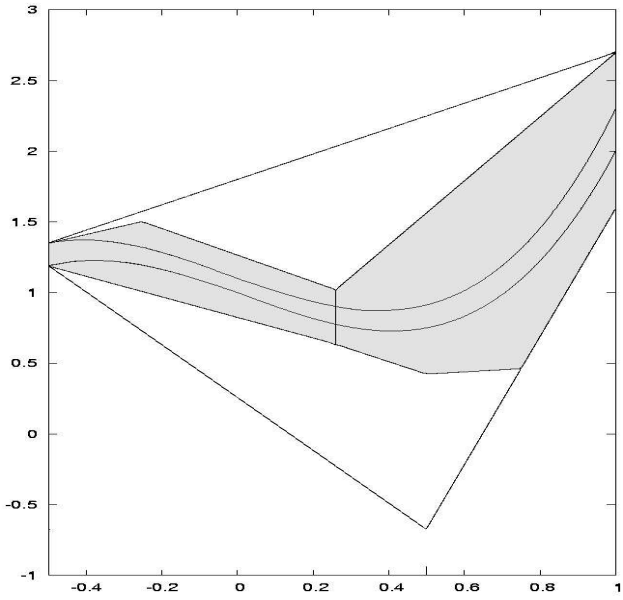


Figure 4.5: *Convex hulls for* $[p](x) = [2, 2.1]x^3 + [-1, -0.9]x + [1, 1.1]$, $x \in [-0.5, 0.25] \cup [0.25, 1]$.

to each subinterval. The result is a set of sets of control points; namely

$$\begin{aligned}\pi^2_{[-0.5,1]}([p](x)) \;=\; \Upsilon^n_{e^\sharp(x)}([p](x)) \;&=\; \big(\uplus\ \pi_{e^\sharp_{2,0}(x)}([p](\overline{\mathbf{x}}))\big) \;\cup\; \big(\uplus\ \pi_{e^\sharp_{2,0}(x)}([p](\overline{\mathbf{x}}))\big)\\ &=\; \big(\uplus\ \pi_{[-0.5,0.25]}([p](\overline{\mathbf{x}}))\big) \;\cup\; \big(\uplus\ \pi_{[0.25,1]}([p](\overline{\mathbf{x}}))\big)\\ &=\; \big(\uplus\ (P_1^- \;\cup\; P_1^+)\big) \;\cup\; \big(\uplus\ (P_2^- \;\cup\; P_2^+)\big),\end{aligned}$$

where

$$\begin{aligned}P_1^- \;&=\; \big\{(-0.5, 1.1875), (-0.25, 1.3125), (0, 0.846875), (0.25, 0.634375)\big\}\\ P_1^+ \;&=\; \big\{(-0.5, 1.35), (-0.25, 1.51), (0, 1.12), (0.25, 1.05)\big\}\end{aligned}$$

and

$$\begin{aligned}P_2^- \;&=\; \big\{(0.25, 0.634375), (0.5, 0.421875), (0.75, 0.4625), (1, 1.6)\big\}\\ P_2^+ \;&=\; \big\{(0.25, 1.05), (0.5, 0.98), (0.75, 1.23), (1, 2.7)\big\}\end{aligned}$$

By theorem 4.32, $\forall\ \overline{\mathbf{z}} \in \mathbb{R}^2$ *where*

$$\overline{\mathbf{z}} \;\in\; \Big\{\big(x, r(x)\big) \in \mathbb{R}^2 \;\mid\; \forall x \in [-0.5, 1],\ r(x) \in [p](x)\Big\}$$

$\overline{\mathbf{z}} \in \uplus P_1^-$*, and/or* $\overline{\mathbf{z}} \in \uplus P_2^-$*, and/or* $\overline{\mathbf{z}} \in \uplus P_1^+$*, and/or* $\overline{\mathbf{z}} \in \uplus P_2^+$.
Figure 4.5 shows the polyhedra obtained by the recursive method, in which the original

interval $[-0.5, 1]$ *has been segmented. Also in this figure, it can be noted that the poly-hull of these two polyhedra delivers a single polyhedron that preserves the range enclosure.*

The proof of soundness of this abstraction requires a series of results, as follows. First we show the relationship between segmentations of an abstract environment.

Proposition 4.28. *Let* $expr_{\mathbf{f}}$ *be a floating-point expression defined for an abstract environment* $e^\sharp$ *where* $x \subseteq \mathsf{vars}(expr_{\mathbf{f}})$*. Moreover, let* $n \geq 1$ *be a segmentation degree. Then* $\forall i, j \in [0, n-1]$*,*

$$e^\sharp_{n,i}(x) \;\subseteq\; e^\sharp_{n,j}(x) \;\Leftrightarrow\; e^\sharp_{n,i}(x) \;=\; e^\sharp_{n,j}(x).$$

where each $e^\sharp_{n,i}$ *is the* i*-th segmentation of* $e^\sharp$*.*

Proof. ($\Rightarrow$) Let $e^\sharp(x) = [a^-, a^+]$, $w = a^+ - a^-$, and $i, j \in [0, n-1]$. Then $e^\sharp_{n,i} = [a^- + i \cdot w/n, a^- + (i+1) \cdot w/n]$ and $e^\sharp_{n,j} = [a^- + j \cdot w/n, a^- + (j+1) \cdot w/n]$. By definition of interval inclusion, $e^\sharp_{n,i} \subseteq e^\sharp_{n,j}$ implies $a^- + i \cdot w/n \geq a^- + j \cdot w/n$ if and only if $i \geq j$ and $a^- + (i+1) \cdot w/n \leq a^- + (j+1) \cdot w/n$, if and only if $i \leq j$. Thus, $i = j$ and by definition 4.21, $e^\sharp_{n,i}(x) = e^\sharp_{n,j}(x)$.

($\Leftarrow$) Let $e^\sharp(x) = [a^-, a^+]$ and $w = a^+ - a^-$. Moreover, let $i, j \in [0, n-1]$ where $e^\sharp_{n,i}(x) = e^\sharp_{n,j}(x)$. By definition 4.21, we can write this equality as $[a^- + i \cdot w/n, a^- + (i+1) \cdot w/n] = [a^- + j \cdot w/n, a^- + (j+1) \cdot w/n]$, and by definition of interval inclusion, $[a^- + i \cdot w/n, a^- + (i+1) \cdot w/n] \subseteq [a^- + j \cdot w/n, a^- + (j+1) \cdot w/n]$. Thus the statement of the proposition holds. □

The result presented above can be generalised to multiple variables, as follows.

Proposition 4.29. *Let* $expr_{\mathbf{f}}$ *be a floating-point expression defined for an abstract environment* $e^\sharp$ *where* $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$ *and* $\overline{\mathbf{x}} = (x_1, \ldots, x_k)$*. Moreover, let* $n \geq 1$ *be a segmentation degree. Then* $\forall I, J \in [0, n-1]^k$*,*

$$e^\sharp_{n,I}(\overline{\mathbf{x}}) \;\subseteq\; e^\sharp_{n,J}(\overline{\mathbf{x}}) \;\Leftrightarrow\; e^\sharp_{n,I}(\overline{\mathbf{x}}) \;=\; e^\sharp_{n,J}(\overline{\mathbf{x}}).$$

Proof. ($\Rightarrow$) Let $I = (i_1, \ldots, i_k)$ and $J = (j_1, \ldots, j_k)$. Moreover let $e^\sharp_{n,I}(\overline{\mathbf{x}}) = e^\sharp_{n,i_1} \times \ldots \times e^\sharp_{n,i_k}$ and $e^\sharp_{n,J}(\overline{\mathbf{x}}) = e^\sharp_{n,j_1} \times \ldots \times e^\sharp_{n,j_k}$. By definition of box inclusion, $e^\sharp_{n,I} \subseteq e^\sharp_{n,J}$ if and only for each $\ell \in [1, k]$, $e^\sharp_{n,i_\ell} \subseteq e^\sharp_{n,j_\ell}$, if and only if $i_\ell = j_\ell$ (by proposition 4.28). Thus, $e^\sharp_{n,I}(\overline{\mathbf{x}}) = e^\sharp_{n,J}(\overline{\mathbf{x}})$.

($\Leftarrow$) Let $I = (i_1, \ldots, i_k)$ and $J = (j_1, \ldots, j_k)$. Recall that $e^\sharp_{n,I}(\overline{\mathbf{x}}) = e^\sharp_{n,i_1}(x_1) \times \ldots \times e^\sharp_{n,i_k}(x_k)$, and by definition of subbox inclusion, $e^\sharp_{n,I}(\overline{\mathbf{x}}) = e^\sharp_{n,J}(\overline{\mathbf{x}})$ implies $e^\sharp_{n,I}(\overline{\mathbf{x}}) \subseteq e^\sharp_{n,J}(\overline{\mathbf{x}})$ if and only if for each $\ell \in [1, k]$, $e^\sharp_{n,i_\ell}(x_\ell) \subseteq e^\sharp_{n,j_\ell}(x_\ell)$, which holds by proposition 4.28. □

Corollary 4.30. *For each* $I \in [0, n-1]^k$*,* $e^\sharp_{n,I}(\overline{\mathbf{x}}) \subseteq e^\sharp(\overline{\mathbf{x}})$*.*

Moreover, note that

$$e^\sharp(\overline{\mathbf{x}}) \;=\; \bigcup_{I\in[0,n-1]^k} e^\sharp_{n,I}(\overline{\mathbf{x}}).$$

The segmentation of an abstract environment has important implications. For instance, when applying a Bernstein mapping to segments of an abstract environment, the range enclosure for example is tighter. Consider a floating-point expression $expr_{\mathbf{f}}$ defined for an abstract environment $e^\sharp$ where $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$. For a segmentation degree $n \geq 1$ and $I \in [0, n-1]$,

$$[\![expr_{\mathbf{f}}]\!]^\sharp_{fl} e^\sharp \;=\; [p](\overline{\mathbf{x}}),\ \overline{\mathbf{x}} \in e^\sharp(\overline{\mathbf{x}}),$$

and

$$[\![expr_{\mathbf{f}}]\!]^\sharp_{fl} e^\sharp_{n,I} \;=\; [p](\overline{\mathbf{x}}),\ \overline{\mathbf{x}} \in e^\sharp_{n,I}(\overline{\mathbf{x}}),$$

where $e^\sharp_{n,I}(\overline{\mathbf{x}}) \subseteq e^\sharp(\overline{\mathbf{x}})$. This observation is important since the poly-hull of points obtained from a Bernstein mapping for some $e^\sharp_{n,I}$ delivers a smaller polyhedron than those from the mapping for $e^\sharp$. To see this, recall the concept of inclusion monotonicity of an interval function; namely, an interval function $[f] : \mathbb{IR}^k \to \mathbb{IR}$ is inclusion monotone if for all $\mathbf{B}_1, \mathbf{B}_2 \in \mathbb{IR}^k$, $[f](\mathbf{B}_1) \subseteq [f](\mathbf{B}_2)$ whenever $\mathbf{B}_1 \subseteq \mathbf{B}_2$. Moreover, recall that interval arithmetic operators such as those of addition, subtraction, multiplication and division are monotone inclusion. As shown in [HS95] and [Sta95], the range enclosures of the sets of points delivered by the Bernstein mappings for interval valued polynomials defined in section 4.3.3 are monotone inclusion. Moreover, it can be noted that the poly-hull operator is monotone inclusion with respect to polyhedron inclusion; namely, for $P_1, P_2, Q_1, Q_2 \in \mathbb{CP}_k$ where $P_1 \subseteq Q_1$ and $P_2 \subseteq Q_2$, $(P_1 \uplus P_2) \subseteq (Q_1 \uplus Q_2)$. These observations can be used to obtain more precise polyhedral abstractions and tighter range enclosures of floating-point polynomial expressions, as formalised in the following theorem.

Theorem 4.31. *For a floating-point polynomial expression $expr_{\mathbf{f}}$ where $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$, and for a segmentation degree $n \geq 1$ and $I \in [0, n-1]^k$, if $e^\sharp_{n,I}(\overline{\mathbf{x}}) \subseteq e^\sharp(\overline{\mathbf{x}})$, then*

$$\uplus\, \pi_{e^\sharp_{n,I}(\overline{\mathbf{x}})}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl} e^\sharp_{n,I}) \;\subseteq\; \uplus\, \pi_{e^\sharp(\overline{\mathbf{x}})}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp)).$$

From these results, the abstraction of floating-point polynomial expressions into polyhedra powerset is formalised as follows.

Theorem 4.32. (Soundness of polyhedra powerset abstraction.) $\forall e \in \gamma(e^\sharp)$,

$\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$, $\overline{\mathbf{x}} \in \mathbb{R}^k$, *there exists* $P \in \pi^n_{e^\sharp(\overline{\mathbf{x}})}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp))$ *where*

$$\big\{\big(e(x_1),\ldots,e(x_k),c\big) \ \in \ \mathbb{R}^{k+1} \ \mid \ c \ \in \ [\![expr_{\mathbf{f}}]\!](e)\big\} \ \subseteq \ \uplus \ P.$$

Proof. By theorem 4.16, for each $e^\sharp_{n,I}(\overline{\mathbf{x}})$. □

Moreover, we have completeness since

$$e^\sharp(\overline{\mathbf{x}}) \ = \ \bigcup_{I\in[0,n-1]^k} e^\sharp_{n,I}(\overline{\mathbf{x}}).$$

Further, by the monotone inclusion property, each concrete interpretation of a floating-point polynomial expression is included in some polyhedron of the powerset.

4.4.3 Nonredundancy Property

In this section we show that the elements of the powerset of $\mathbb{CP}_k$ delivered by the recursive method are in a reduced or nonredundant form. Recall that a set $P \in \wp_{\mathrm{fn}}\mathbb{CP}_k$ is called nonredundant with respect to $\subseteq$ if and only if $\emptyset \notin P$ and for each $d_1, d_2 \in P$, $d_1 \subseteq d_2$ implies that $d_1 = d_2$.

Definition 4.33. (Comparable polyhedra.) *Let $P, Q \in \mathbb{CP}_n$. Then P and Q are comparable if and only if $P \subseteq Q$ or $Q \subseteq P$ (or $P = Q$).*

By definition of nonredundant sets for a partial ordering, if a set of convex hulls is noncomparable, then the set is nonredundant. In the following theorem we show that the polyhedra that can be obtained by computing the recursive Bernstein mapping is noncomparable and therefore nonredundant; namely, we define a bijective mapping between the set of abstract environment segmentations and the polyhedra for the sets of control points of the recursive Bernstein mapping. We show that the injectivity of this mapping implies that two elements in the co-domain are comparable if and only if both are the same polyhedron. Moreover, surjectivity ensures soundness of the mapping with respect to the abstract environment that is segmented. Put another way, we have bijectivity since $e^\sharp_{n,I} = e^\sharp_{n,J}$ if and only if $I = J$ and a recursive Bernstein mapping is defined for each $e^\sharp_{n,I}$, $n \geq 1$, $I \in [0, n-1]^k$.

Theorem 4.34. *Let $expr_{\mathbf{f}}$ be a floating-point expression defined for an abstract environment $e^\sharp$ and $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$, where $\overline{\mathbf{x}} \in \mathbb{R}^k$. Moreover, for a segmentation degree $n \geq 1$, let*

$$\rho : \{e^\sharp_{n,I}(\overline{\mathbf{x}})\}_{I\in[0,n-1]^k} \to \{\uplus \ P_I\}_{I\in[0,n-1]^k},$$

where $P_I \in \pi^n_{e^\sharp(\overline{\mathbf{x}})}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp))$ and $e^\sharp(\overline{\mathbf{x}}) = \cup_{I\in[0,n-1]^k} e^\sharp_{n,I}$. Then ρ is a bijection.

The bijectivity of ρ ensures the elements in the co-domain are noncomparable unless the co-domain is a singleton, in which case, nonredundancy is trivial. Therefore any set of polyhedra computed from a recursive Bernstein mapping is nonredundant with each other. Thus, any abstraction of floating-point expressions into polyhedra powerset using the recursive Bernstein mappings is nonredundant.

Corollary 4.35. *Let $expr_{\mathbf{f}}$ be a floating-point polynomial expression defined for an abstract environment $e^\sharp$, where $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$, and let n be a segmentation degree for $e^\sharp$. Then the set*

$$\left\{ \uplus P \in \mathbb{CP}_{k+1} \mid \forall P \in \wp_{\mathrm{fn}}(\mathbb{R}^k) : P \in \pi^n_{e^\sharp(\overline{\mathbf{x}})}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp)) \right\} \in \wp_{\mathrm{fn}}(\mathbb{CP}_{k+1})$$

is nonredundant with respect to polyhedron inclusion.

Example 4.36. *In example 4.27, it can be verified graphically that none of the polyhedra obtained by $\uplus P_i^-$ or $\uplus P_i^+$, $i \in \{1, 2\}$ are comparable with each other, and therefore the set of all these polyhedra is nonredundant.*

4.4.4 Range Enclosures

This section shows that the recursive method is sound with respect to the optimal range enclosure.

Definition 4.37. *Let $expr_{\mathbf{f}}$ be a floating-point polynomial expression and $e^\sharp$ an abstract environment where $[\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp) = [p](\overline{\mathbf{x}})$ and $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$. Then,*

$$Enc(\pi^n_{e^\sharp(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}}))) = [p_{\min}, p_{\max}],$$

where

$$p_{\min} = \min_{I \in [0,n-1]^k} (Enc(\pi_{e^\sharp_{n,I}(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}})))) \text{ and}$$
$$p_{\max} = \max_{I \in [0,n-1]^k} (Enc(\pi_{e^\sharp_{n,I}(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}})))).$$

Note that by the monotone inclusion property, for each $I \in [0, n-1]^k$,

$$Enc(\pi^n_{e^\sharp_{n,I}(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}}))) \subseteq Enc(\pi^n_{e^\sharp(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}}))).$$

Thus, the upper boundary of the enclosure corresponds to the maximum of all the enclosure upper boundaries of all $\pi^n_{e^\sharp_{n,I}(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}}))$, and the lower boundary of the enclosure corresponds to the minimum of all the enclosure lower boundaries of all $\pi^n_{e^\sharp_{n,I}(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}}))$.

Moreover, by the monotone inclusion property,

$$\begin{aligned} Enc(\pi^{n+1}_{e^\sharp(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}})), e^\sharp(\overline{\mathbf{x}})) &\subseteq Enc(\pi^{n}_{e^\sharp(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}})), e^\sharp(\overline{\mathbf{x}})) \\ &\subseteq Enc(\pi^{1}_{e^\sharp(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}})), e^\sharp(\overline{\mathbf{x}})) \\ &= Enc(\pi_{e^\sharp(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}})), e^\sharp(\overline{\mathbf{x}})), \end{aligned}$$

and for n sufficiently large,

$$Opt([p](\overline{\mathbf{x}}), e^\sharp(\overline{\mathbf{x}})) = Enc(\pi^{n}_{e^\sharp(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}})), e^\sharp(\overline{\mathbf{x}}))$$

if and only if

$$Enc(\pi^{n}_{e^\sharp(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}})), e^\sharp(\overline{\mathbf{x}})) = Enc(\pi^{n+1}_{e^\sharp(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}})), e^\sharp(\overline{\mathbf{x}})).$$

Example 4.38. *In example 4.27, the estimate for the range enclosure for* $e^\sharp(x) = [-0.5, 1]$ *is*

$$Enc(\pi^{2}_{e^\sharp(x)}([p](x)), e^\sharp(x)) = [0.42, 2.7],$$

which is tighter than the enclosure obtained in example 4.12 for a segmentation degree $n = 1$*, namely*

$$Enc(\pi^{1}_{e^\sharp(x)}([p](x)), e^\sharp(x)) = [-0.67, 2.7].$$

Moreover, from the enclosure for $n = 2$*, we can note that* $[p](x) > 0$ *for all* $x \in [-0.5, 1]$*. This is important for instance when determining the existence of solutions of an equation such as* $[p](x) = [0, 0]$*.*

The time complexity of the recursive method is exponential to the number of variables, which determine the number of segments.

4.5 Minimal Method

When the purpose of using the recursive method is to abstract a floating-point expression into polyhedra — instead of finding zeroes in an interval valued polynomial — it may be convenient to reduce the abstraction into a more time and space efficient representation. This representation, though, must preserve the properties of interest of the initial abstraction, namely, convexity and range enclosure. In this case, we simply compute the poly-hull of the set delivered by the recursive method.

4.5.1 Definitions

The minimal method delivers the least polyhedron which includes all the polyhedra in the powerset obtained in the recursive method. This least polyhedron preserves the range enclosure property of the recursive method as well as the convexity property, while keeping a reduced representation. Thus, we have a precise and efficient way of abstracting floating-point polynomial expressions into polyhedra.

Definition 4.39. (Minimal Bernstein mapping.) *Let $expr_{\mathbf{f}}$ be a floating-point polynomial expression and $e^\sharp$ an abstract environment where $[\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp) = [p](\overline{\mathbf{x}})$ and $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$. For a segmentation degree $n \geq 1$ and $\mathbf{B} \in \mathbb{IR}^k$, the* minimal Bernstein mapping $\mathrm{m}\pi^n_{\mathbf{B}} : (\mathbb{R}^k \to \mathbb{IR}) \to \wp_{\mathrm{fn}}(\mathbb{R}^{k+1})$ *is defined as*

$$\mathrm{m}\pi^n_{e^\sharp}([p](\overline{\mathbf{x}})) \;=\; \bigcup_{I \in [0,n-1]^k} \pi_{e^\sharp_{n,I}}([p](\overline{\mathbf{x}})).$$

4.5.2 Polyhedra Abstraction

The following theorem shows the soundness of the minimal method.

Theorem 4.40. *$\forall e \in \gamma(e^\sharp)$, $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$, $e^\sharp(\overline{\mathbf{x}}) \in \mathbb{IR}^k$, where*

$$\uplus \left\{ \big(e(x_1), \ldots, e(x_k), c\big) \;\in\; \mathbb{F}^{k+1} \;\middle|\; \begin{array}{l} \forall c \;\in\; \mathbb{F} \;: \\ c \;\in\; [\![expr_{\mathbf{f}}]\!]_{fl}(e) \end{array} \right\} \;\subseteq\; \uplus\, \mathrm{m}\pi^n_{e^\sharp(\overline{\mathbf{x}})}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp))$$

Proof. By definition of the poly-hull operator and theorem 4.32. □

Example 4.41. *The minimal method, after applying the recursive method for a given segmentation degree n, computed the poly-hull of all the sets of control points. Following example 4.27 for the recursive method, let $P_1 = P_1^- \cup P_1^+$ $P_2 = P_2^- \cup P_2^+$. Then,*

$$\Big\{\big(x, c\big) \in \mathbb{R}^2 \;\mid\; \forall x \in [0,1] \;:\; \forall r(x) \in [p](x) \;.\; c \;=\; r(x)\Big\} \;\subseteq\; \uplus\, (P^- \;\cup\; P^+).$$

Figure 4.6 plots the convex hull (shadowed region).

4.5.3 Range Enclosures

The range enclosure is invariant under the poly-hull operator. Thus we have the following result for the range enclosure for the minimal Bernstein mapping.

Theorem 4.42. *Let $expr_{\mathbf{f}}$ be a floating-point polynomial expression and $e^\sharp$ an abstract environment where $[\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp) = [p](\overline{\mathbf{x}})$ and $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$. Then*

$$Enc(\mathrm{m}\pi^n_{e^\sharp(\overline{\mathbf{x}})}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp))) \;=\; Enc(\pi^n_{e^\sharp(\overline{\mathbf{x}})}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp))).$$

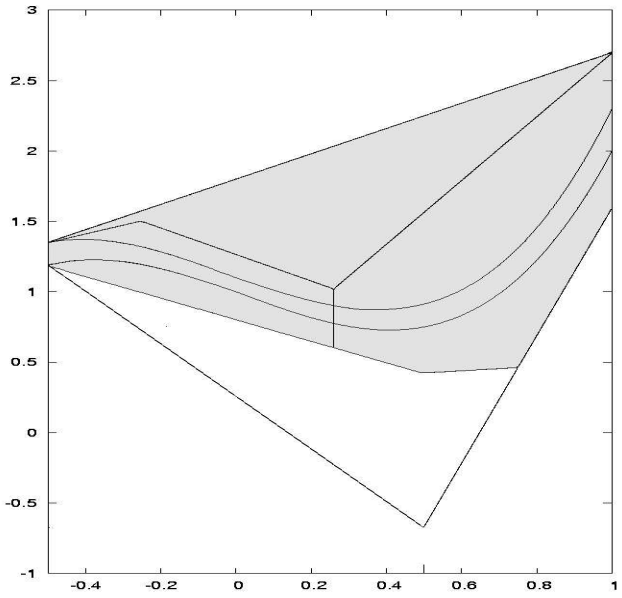


Figure 4.6: *Convex hull for* $[p](x) = [2, 2.1]x^3 + [-1, -0.9]x + [1, 1.1]$, $x \in [-0.5, 1] \cup [0, 1]$.

Note that

$$Opt([p](\overline{\mathbf{x}}), e^\sharp(\overline{\mathbf{x}})) \ \subseteq \ Enc(\mathrm{m}\pi^n_{e^\sharp(\overline{\mathbf{x}})}([p](\overline{\mathbf{x}})),\ e^\sharp(\overline{\mathbf{x}})).$$

Example 4.43. *Note in example 4.41 that the range enclosure is the same as the range enclosure obtained from the recursive method. In fact, the poly-hull operator computes a polyhedron which includes the same set of control points as in the recursive method.*

4.6 Discussion and Related Work

Bernstein Expansion. The Russian mathematician Sergei Natanovich Bernstein (1880-1968) worked on the theory of best approximation of functions. In 1911, Sergei Bernstein presented what are now called Bernstein polynomials that provide a constructive proof of Weierstrass Approximation Theorem [1] (1885) [Ber12]. Bernstein expansion of real polynomials has been studied for instance in [CS66, Riv70, Rok77, Rok79, Rok82, RR84, Fis90, RR03]. Applications of Bernstein expansion include the study of robustness stability [MMTG92, ZG98], solutions of polynomial inequalities [GG99, SG00] and computer aided geometric design [Far93].

Polyhedra Abstraction for Floating-Point Expressions. The development of the methods presented here has been originally motivated by the problems inherent in the

[1] The Weierstrass Approximation Theorem [Wei85] states that for a given continuous function $f : \mathbf{U} \to \mathbb{R}$ and a positive real $\epsilon > 0$, there exists a polynomial $p : \mathbb{R} \to \mathbb{R}$ where $| f(x) - p(x) | < \epsilon$, for all $x \in \mathbf{U}$.

intervalisation method [Min04b], which produces a coarse linear approximation to non-linear expressions. Intervalisation requires further adaptation to a polyhedral relational domain, which can turn to be a complex task for the polyhedra domain. However, as shown in [MSV+02], Bernstein expansion delivers more precise results than interval-based methods. Contrarily to intervalisation, our approach is independent of nonlinearity but the computational cost grows exponentially with the number of variables.

Of Efficiency of the Methods. Time complexity of the standard method (and its derived methods) grows exponentially to the number of variables in an interval valued polynomial. However, Delgado and Peña [DP03] presented a linear time complexity algorithm for computing Bernstein basis, that overcomes this problem. An improvement in efficiency can be achieved by reusing already calculated Bernstein coefficients to compute new ones. This idea is formalised and developed in the sweep procedure [Gar86, Gar93], which is based on the Casteljau algorithm for interpolation and is widely used in computer aided graphics design (CAGD) [Far93]. This procedure computes the Bernstein coefficients for subsets of boxes obtained by segmenting the original box. However, in order to produce polyhedral abstractions useful for analyses, an adapted sweep procedure should compute coefficients for every box, The standard method for floating-point expressions can be easily adapted to deal with integer expressions by adopting a conservative rounding of Bernstein coefficients in $\mathbb{R}$; namely, rounding up a real value c to the larger positive integer if $c > 0$, and to the smaller negative integer if $c < 0$.

Chapter 5

Polyhedra for Floating-Point Polynomial Quotients

In this chapter we present a method for abstracting floating-point polynomial quotients in which a division by zero may occur. For this, we introduce a branch and prune algorithm which excludes those abstract environments for which a division by zero in a polynomial quotient may occur. Zeroes for quotients are checked using a test based on Miranda's theorem which takes advantage of the range enclosure property of the Bernstein mappings defined in chapter 4. The abstraction delivers a set of polyhedra, which is important for the development of a backward analysis for the inference of possible divisions by zero.

5.1 Introduction

When abstracting floating-point expressions that include a division operator, one must check whether the divisor subexpression may evaluate to 0 for some values that instantiate the variables. If the subexpression evaluates to 0, then the division operation produces a floating-point division by zero exception, represented either by NaN or by $\pm\infty$. This exception may cause in turn a floating-point run-time error when querying a program. The purpose of this chapter is to present a polyhedral abstraction that includes the sets of values that can be assigned to variables in a floating-point expression which do not cause division by zero.

Existing approaches to abstracting floating-point expressions with division operators into a relational domain includes those developed in [Min04b, Min04a]. These relational

abstractions rely on the definition of the interval division operator, which is not defined when the divisor interval includes 0. In that case, the abstraction may fail (and therefore the analysis cannot proceed). A reason for this is the coarse linearisation method used (*intervalisation*), which may deliver an overestimated range enclosure that includes 0 whereas the optimal range enclosure does not.

By identifying parts of the variables domain, namely parts of an abstract environment, for which it can be proved that no division by zero can occur, the abstraction does not fail. This means also that the abstraction should deliver a powerset of a polyhedral domain, which in general is more precise but also more costly. In addition, the use of more precise methods for calculating range enclosures, such as those presented in chapter 4, can lead to the inclusion of larger parts of an abstract environment for which it can be proved that no division by zero can occur.

The central problem in this chapter is thus the identification of the parts of an abstract environment that cannot lead to a division by zero. Our approach is the introduction of a variation of a branch and prune method for identifying such parts of abstract environments. In a floating-point polynomial quotient, the divisor is tested against the existence of zeroes. For this, a test based on Miranda's existence theorem which uses the range enclosure property of Bernstein mappings, is performed. Those parts of an abstract environment that satisfy the test for the divisor, namely that includes a zero, are either discarded (pruned) or refined (branched). The refinement consists in segmenting an abstract environment. Those parts that do not satisfy the test, namely that do not include zeroes, are used for abstracting the expression following the abstract semantics of definition 3.16 and the powerset of the polyhedra domain of section 4.4. The termination of the method is guaranteed by ensuring a finite number of refinements.

Our contribution in this chapter is the definition of a test for zeroes existence in interval valued polynomials, based on Miranda's existence theorem and the range enclosing property of the Bernstein mappings. Moreover we introduce a variation of the branch and prune method for identifying segments of an abstract environment in which no zeroes exists. Based on this, we propose an abstraction of floating-point polynomial quotients into the polyhedra powerset.

5.2 Tests for Zero Existence in Interval Valued Polynomials

In order to prove the existence of zeroes of a function defined over a box, several existence theorems are available: namely for instance Kantorovich, Miranda's or Borsuk [AFHM04, FL05]. But in order to decide whether there may exist zeroes in an abstract environment, we can define tests based on existence theorems including Moore test [Moo72] and the test based on Miranda's theorem [MK80]. These tests require an estimate of the range enclosure of the function to be analysed. Such estimates are usually computed using interval

arithmetic, and includes the naïve method (the optimal range is replaced by the interval arithmetics evaluations of the function over the facets), the centred method [MK80] (based on calculating a *slope* matrix), and the facet-centred method [FL05, pp. 1350].

A weaknesses of these tests is the coarse estimate for range enclosures, as shown for instance in [FL05], where the use of the more general zero existence theorem by Borsuk [AFHM04] is required. However, we can compute tight estimates using the range enclosing property of Bernstein mappings. This idea was used for instance in [SG00] to solve systems of polynomials and interval valued polynomials respectively. We use the range enclosure property of Bernstein expansion to obtain an estimate of the range enclosure of an interval valued polynomial defined over a box.

Thus, by combining Miranda's existence theorem and the range enclosure property of Bernstein mappings, we define a test for an interval valued polynomial defined over an abstract environment which succeeds only if the abstract environment includes a zero for the given interval valued polynomial. In the following section we formalise the concept of zero of an interval valued polynomial.

5.2.1 Definitions

The concept of floating-point polynomial quotient is central in this chapter, and is formalised as follows.

Definition 5.1. (Floating-point polynomial quotient.) *Let $expr_{\mathbf{f}}$ be a floating-point expression and $e^\sharp$ be an abstract environment where $[\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp) = [p](\overline{\mathbf{x}}) \boxslash^\sharp [q](\overline{\mathbf{x}})$. We say that $expr_{\mathbf{f}}$ is a floating-point polynomial quotient if $\mathsf{vars}([q](\overline{\mathbf{x}})) \neq \emptyset$.*

For a divisor $[q](\overline{\mathbf{x}}) = [a^-, a^+]$ where $a^-, a^+ \in \mathbb{R}$, it is possible to multiply $[p](\overline{\mathbf{x}})$ by the inverse of such interval, thus eliminating the division operator.

In order to introduce the definition of zeroes of an interval valued polynomial, we need first to formalise the notion of zero of a polynomial, as follows.

Definition 5.2. (Zero of a polynomial.) *Let $\overline{\mathbf{z}} \in \mathbb{R}^k$. We say that $\overline{\mathbf{z}}$ is a* zero *of a k-variate polynomial $r(\overline{\mathbf{x}})$ if $r(\overline{\mathbf{z}}) = 0$.*

It is important to remark that we do not consider zeroes in $\mathbb{C}$.

The standard definition of a zero of an interval valued polynomial can be found for instance in [PCT95]; namely, for an interval valued polynomial $[p](\overline{\mathbf{x}})$ defined for $\overline{\mathbf{x}} \in \mathbb{R}^k$, we say that $\overline{\mathbf{z}} \in \mathbb{R}^k$ is a zero of $[p](\overline{\mathbf{x}})$ if there exists $r(\overline{\mathbf{x}}) \in [p](\overline{\mathbf{x}})$ where $r(\overline{\mathbf{z}}) = 0$. For the purpose of defining and computing abstractions of quotients of interval valued polynomials into polyhedra, we provide an alternative definition.

Definition 5.3. (Zero of an interval valued polynomial.) *Let $[p](\overline{\mathbf{x}})$, $\overline{\mathbf{x}} \in \mathbb{R}^k$, be an interval valued polynomial. We say that $\overline{\mathbf{z}} \in \mathbb{R}^k$ is a* zero *of $[p](\overline{\mathbf{x}})$ if $[0, 0] \subseteq [p](\overline{\mathbf{z}})$.*

5.2.2 Miranda's Existence Theorem

This section recalls Miranda's existence theorem, which generalises the intermediate value theorem to a higher number of dimensions, and is important in the study of existence of solutions for systems of nonlinear equations. For a given $\mathbf{B} \in \mathbb{IR}^k$, let $\mathbf{f} : \mathbf{B} \subseteq \mathbb{R}^k \to \mathbb{R}^k$ be a (nonlinear) continuous mapping where $\forall \overline{\mathbf{x}} \in \mathbf{B}$

$$\overline{\mathbf{x}} \;\mapsto\; (f_1(\overline{\mathbf{x}}), \ldots, f_k(\overline{\mathbf{x}})),$$

Miranda's existence theorem provides a general result for proving the existence of solutions for a system of nonlinear equations such as

$$\begin{cases} f_1(\overline{\mathbf{x}}) \;=\; 0 \\ \ldots \\ f_k(\overline{\mathbf{x}}) \;=\; 0. \end{cases}$$

For $\mathbf{B} \in \mathbb{IR}^k$, we say that $\overline{\mathbf{x}}^* \in \mathbf{B}$ is a solution of the system above if $\mathbf{f}(\overline{\mathbf{x}}^*) = \mathbf{0}$. In the following theorem, the term *facet* refers to a $k-1$ dimensional face of a k dimensional box $\mathbf{B} \in \mathbb{IR}^k$.

Theorem 5.4. (Miranda's Existence Theorem [Mir41].) *Let $\mathbf{B} \in \mathbb{IR}^k$ be a box denoted by $\mathbf{B} = [a_1^-, a_1^+] \times \ldots \times [a_k^-, a_k^+]$, and for some $i \in [1, k]$, let $\mathbf{B}_i^- = \{\overline{\mathbf{x}} \in \mathbf{B} \mid x_i = a_i^-\}$ and $\mathbf{B}_i^+ = \{\overline{\mathbf{x}} \in \mathbf{B} \mid x_i = a_i^+\}$ denote the pair of opposite parallel facets of $\mathbf{B}$ perpendicular to the i-th coordinate direction. Let $\mathbf{f} : \mathbf{B} \subseteq \mathbb{R}^k \to \mathbb{R}^k$ be a continuous mapping where $\overline{\mathbf{x}} \mapsto (f_1(\overline{\mathbf{x}}), \ldots, f_k(\overline{\mathbf{x}}))$. If*

$$f_i(\overline{\mathbf{x}}) \;\geq\; 0 \text{ for } \overline{\mathbf{x}} \in \mathbf{B}_i^+, \text{ and } f_i(\overline{\mathbf{x}}) \;\leq\; 0 \text{ for } \overline{\mathbf{x}} \in \mathbf{B}_i^-,$$

where $i \in [1, k]$, then $\mathbf{B}$ contains a zero $\overline{\mathbf{z}}$ of $\mathbf{f}$.

Miranda's theorem can be formulated in a more general form if the facets and function components are related by a permutation, namely, a bijective mapping $\tau : \{1, \ldots, k\} \to \{1, \ldots, k\}$ where

$$f_{\tau(i)}(\overline{\mathbf{x}}) \;\geq\; 0 \text{ for } \overline{\mathbf{x}} \in \mathbf{B}_i^+, \text{ and } f_{\tau(i)}(\overline{\mathbf{x}}) \;\leq\; 0 \text{ for } \overline{\mathbf{x}} \in \mathbf{B}_i^-.$$

Moreover, note that Miranda's theorem can be used to study the existence of zeroes of a function $f : \mathbb{R}^k \to \mathbb{R}$ in $\mathbf{B} \in \mathbb{IR}^k$, by studying the existence of zeroes of a mapping $\mathbf{g} : \mathbf{B} \subseteq \mathbb{R}^k \to \mathbb{R}^k$ where $\forall \overline{\mathbf{x}} \in \mathbf{B}$

$$\overline{\mathbf{x}} \mapsto (f(\overline{\mathbf{x}}), 0, \ldots, 0).$$

In general, this system of (possibly nonlinear) equations may have infinitely many so-

lutions. The following extension of Miranda's theorem to the case of interval valued polynomials is useful for showing the existence of solutions for

$$[0,0] \;\subseteq\; [p](\overline{\mathbf{x}});$$

namely, to show whether $[p](\overline{\mathbf{x}})$ has zeroes.

Theorem 5.5. (Miranda's Existence Theorem for Interval Valued Polynomials.) *Let* $\mathbf{B} \in \mathbb{IR}^k$ *be a box denoted by* $\mathbf{B} = [a_1^-, a_1^+] \times \ldots \times [a_k^-, a_k^+]$, *and for some* $i \in [1,k]$, *let* $\mathbf{B}_i^- = \{\overline{\mathbf{x}} \in \mathbf{B} \mid x_i = a_i^-\}$ *and* $\mathbf{B}_i^+ = \{\overline{\mathbf{x}} \in \mathbf{B} \mid x_i = a_i^+\}$ *denote the pair of opposite parallel facets of* $\mathbf{B}$ *perpendicular to the* i*-th coordinate direction. Let* $[\mathbf{f}] : \mathbf{B} \subseteq \mathbb{IR}^k \to \mathbb{IR}^k$ *be a continuous mapping where* $\overline{\mathbf{x}} \mapsto ([f_1](\overline{\mathbf{x}}), \ldots, [f_k](\overline{\mathbf{x}}))$. *If* $\exists f_i(\overline{\mathbf{x}}) \in [f_i](\overline{\mathbf{x}})$ *where*

$$f_i(\overline{\mathbf{x}}) \;\geq\; 0 \text{ for } \overline{\mathbf{x}} \in \mathbf{B}_i^+, \text{ and } f_i(\overline{\mathbf{x}}) \;\leq\; 0 \text{ for } \overline{\mathbf{x}} \in \mathbf{B}_i^-, \tag{5.1}$$

$i \in [1,k]$, *then* $\mathbf{B}$ *contains a zero* $\overline{\mathbf{z}}$ *of* $[\mathbf{f}]$.

5.2.3 Bernstein Tests

Miranda's theorem for interval valued polynomials can be applied as follows. Let $[p](\overline{\mathbf{x}})$ be an interval valued polynomial defined over $\mathbf{B} \in \mathbb{IR}^k$. Moreover, let

$$\mathit{Opt}([p](\overline{\mathbf{x}}), \mathbf{B}) \;=\; [p_{\min}, p_{\max}].$$

The verification of 5.1 for $[p](\overline{\mathbf{x}})$ is equivalent to checking whether

$$p_{\min} \;\leq\; 0 \;\leq\; p_{\max},$$

namely, whether $[0,0] \subseteq [p_{\min}, p_{\max}]$. Interval arithmetic-based methods for checking the condition above such as the naïve, centred or facet-centred provide coarse approximations to the range enclosure which may be inconclusive. We use the range enclosure property of Bernstein mappings to approximate the optimal range enclosure of $[p](\overline{\mathbf{x}})$ for $\overline{\mathbf{x}} \in \mathbf{B}$ by a (nonoptimal) range enclosure, for a segmentation degree $n \geq 1$. Moreover, recall that the optimal range enclosure of an interval valued polynomial can be obtained by the recursive Bernstein mapping for a sufficiently large segmentation degree n.

It must be noted that the signs of the range enclosure boundaries are preserved under affine transformation, since the range enclosure is invariant under affine transformation. Moreover, recall that our interest lies in zeroes in $\mathbb{R}$, not in $\mathbb{C}$. The following test is useful to develop a method for isolating the segmentations of an abstract environment that does not contain zeroes for a floating-point polynomial expression.

Definition 5.6. (Bernstein test.) *Let* $\mathbb{B} = \{0,1\}$. *Moreover, let* $\mathit{expr}_{\mathbf{f}}$ *be a floating-point polynomial expression defined for* $e^\sharp \in \mathit{AEnv}_{\mathbf{f}}$, *where* $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(\mathit{expr}_{\mathbf{f}})$. *The*

Bernstein test *for* $expr_{\mathbf{f}}$ *over* $e^\sharp(\overline{\mathbf{x}}) \in \mathbb{IR}^k$*, for a segmentation degree* $n \geq 1$*, is a mapping* $Bt^n_{e^\sharp(\overline{\mathbf{x}})} : \mathbb{IR} \to \mathbb{B}$ *where*

$$Bt^n_{e^\sharp(\overline{\mathbf{x}})}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp)) \;=\; \begin{cases} 1 & \text{if } [0,0] \not\subseteq Enc(\pi^n_{e^\sharp(\overline{\mathbf{x}})}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp))) \\ 0 & \text{otherwise.} \end{cases}$$

Example 5.7. *In example 4.12, the range enclosure for* $e^\sharp(x) = [-0.5, 1]$ *and* $n = 1$ *is* $[-0.67, 2.7]$*, which includes the zero interval* $[0,0]$ *and therefore the test evaluates to* 0*. However, as shown in example 4.27, the range enclosure for* $e^\sharp(x) = [-0.5, 1]$ *and* $n = 2$ *is* $[0.42, 2.7]$ *which does not include the zero interval* $[0,0]$ *and therefore the test evaluates to* 1*.*

The range enclosure processed in the test above is not necessarily optimal. Clearly, testing whether $[0,0]$ is not included in this possibly nonoptimal range enclosure ensures that $[0,0]$ is not included in the optimal range enclosure. When the Bernstein test delivers 1 for a given floating-point polynomial expression $expr_{\mathbf{f}}$ and abstract environment$e^\sharp$, then there exists no zeroes in $expr_{\mathbf{f}}$ for $e^\sharp$. That is to say, no change of sign occurs when evaluating the function in any point of any two opposed facets. Assume that for $p_{\min} \cdot p_{\max} > 0$, $p_{\min} > 0$ and $p_{\max} > 0$, where $p_{\max} \geq p_{\min}$. Since $p_{\min} > 0$, and moreover $[p](\overline{\mathbf{x}}) \subseteq [p_{\min}, p_{\max}]$, we have that $[0,0] \not\subseteq [p_{\min}, p_{\max}]$. Furthermore, $[0,0] \not\subseteq [p](\overline{\mathbf{x}})$ and therefore $\nexists\, \overline{\mathbf{z}} \in e^\sharp(\overline{\mathbf{x}})$ where $[0,0] \subseteq [p](\overline{\mathbf{z}})$. A similar reasoning applies to the case in which $p_{\min} < 0$ and $p_{\max} < 0$. Thus, for a floating-point expression $expr_{\mathbf{f}}$ defined over $e^\sharp$ where

$$Enc(\pi^n_{e^\sharp(\overline{\mathbf{x}})}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp))) \;=\; [p_{\min},\; p_{\max}],$$

for some $n \geq 0$, if $p_{\min} \cdot p_{\max} > 0$ then for each $[p](\overline{\mathbf{x}}) \subseteq [\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp)$, $\nexists \overline{\mathbf{z}} \in e^\sharp(\overline{\mathbf{x}})$ where $[0,0] \subseteq [p](\overline{\mathbf{z}})$. When the range enclosure is not optimal, the test can be invoked with a larger segmentation degree. This however, does not guarantee an optimal enclosure and thus the existence of zeroes cannot be assured.

When the Bernstein test delivers 0, zeroes in $[\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp)$ may exist. This is the case in which we can use Miranda's existence theorem to prove the existence of a zero, or else, the case where the optimal range enclosure does not include $[0,0]$.

5.3 Branch and Prune Method for Abstract Environments

In this section we present a method to approximate the set of segmentations which do not include any zero for a given interval valued polynomial; namely, for $[p](\overline{\mathbf{x}})$ and $\forall e \in \gamma(e^\sharp)$, we must find the sets of all $\overline{\mathbf{z}} \in \mathbb{R}^k$ where

$$[0,0] \;\not\subseteq\; [\![[p](\overline{\mathbf{z}})]\!]_{re}(e). \tag{5.2}$$

We can approximate the sets of all $\overline{\mathbf{z}} \in \mathbb{R}^k$ by

$$\left\{ e^\sharp(\overline{\mathbf{x}}) \in \mathbb{IR}^k \;\middle|\; \begin{array}{l} \forall \overline{\mathbf{z}} \in \mathbb{R}^k : \forall e^\sharp \in AEnv_{\mathbf{f}} : \\ \overline{\mathbf{z}} \in e^\sharp(\overline{\mathbf{x}}) \wedge [0,0] \not\subseteq [\![p]\!](\overline{\mathbf{z}})]\!]_{re}(e) \end{array} \right\}.$$

We define a mapping that builds a set of abstract environments for which 5.2 holds. This mapping explores this set of abstract environments and segments those environments which do not satisfy the Bernstein test a finite number of times. Those environments which satisfy the Bernstein test are proposed as an approximation to the solution for 5.2. Space and time complexity as well as soundness and completeness of this mapping are discussed. Moreover, we discuss possible merging of abstract environments to reduce the cardinality of the solution set.

5.3.1 Definitions

Recursive Segmentations. Recall by definition 4.22 that for a segmentation degree $n \geq 1$, $\overline{\mathbf{x}} \in \mathbb{R}^k$ and $e^\sharp \in AEnv_{\mathbf{f}}$, where $e^\sharp(\overline{\mathbf{x}}) \in \mathbb{IR}^k$,

$$e^\sharp_{n,I}(\overline{\mathbf{x}}) \;=\; \varphi_{n,I}(e^\sharp(\overline{\mathbf{x}})),$$

$I \in [0, n-1]^k$. We generalise the definition to sequences of successive segmentations. For instance, for a sequence of two successive segmentations,

$$e^\sharp_{n,I,J}(\overline{\mathbf{x}}) \;=\; \varphi_{n,J}(\varphi_{n,I}(e^\sharp(\overline{\mathbf{x}}))),$$

where $I, J \in [0, n-1]^k$. The general case is as follows.

Definition 5.8. (Multiple segmentation.) *Let $e^\sharp \in AEnv_{\mathbf{f}}$ where for $\overline{\mathbf{x}} \in \mathbf{B}$, $\mathbf{B} \in \mathbb{IR}^k$, $e^\sharp(\overline{\mathbf{x}}) \in \mathbb{IR}^k$. Moreover, let $n \geq 1$ be a segmentation degree. Then a* multiple segmentation *$e^\sharp_{n,I_d,I_{d-1},\ldots,I_0}$ of degree $d \in \mathbb{N}$ is defined by*

$$e^\sharp_{n,I_d,I_{d-1},\ldots,I_0}(\overline{\mathbf{x}}) \;=\; \varphi_{n,I_d}(\varphi_{n,I_{d-1}}(\ldots(\varphi_{n,I_0}(e^\sharp(\overline{\mathbf{x}}))))), \tag{5.3}$$

where $I_d, I_{d-1}, \ldots, I_0 \in [0, n-1]^k$.

Definition 5.9. (Direct segmentation.) *Let $e^\sharp \in AEnv_{\mathbf{f}}$ where for $\overline{\mathbf{x}} \in \mathbf{B}$, $\mathbf{B} \in \mathbb{IR}^k$, $e^\sharp(\overline{\mathbf{x}}) \in \mathbb{IR}^k$, and $d \in \mathbb{N}$. Moreover, let $n \geq 1$ be a segmentation degree. Then we say that $e^\sharp_{n,I_d,\ldots,I_{d-j+1},I_{d-j}}$, $1 \leq j \leq d-1$, is a* direct segmentation *of $e^\sharp_{n,I_d,\ldots,I_{d-j+1}}$, where $I_d, \ldots, I_{j+1}, I_j \in [0, n-1]^k$.*

Expression 5.3 can be denoted in a compact form by

$$\varphi_{n,I_d,I_{d-1},I_0}(e^\sharp(\overline{\mathbf{x}})), \;=\; \varphi_{n,I_d}(\varphi_{n,I_{d-1}}(\ldots(\varphi_{n,I_0}(e^\sharp(\overline{\mathbf{x}}))))).$$

Tree for segmentations. These segmentations can be represented by a finite connected, acyclic, directed graph, namely a tree, defined as follows. A *tree for segmentations* is a finite, labelled, directed tree denoted by $T = (V, E)$, where V is the set of vertices (or *nodes*) and E the set of edges. Each node is denoted by a box obtained from a segmentation for a variable. Thus, for $e^\sharp \in AEnv_{\mathbf{f}}$, the *root* node corresponds to $e^\sharp(\overline{\mathbf{x}})$, and the rest of nodes by a subbox of the root node, obtained from a multiple segmentation. Each edge connects two nodes where the destination node is a direct segmentation of the origin node. The destination node is called *successor node* and the origin node, *parent node.*

Definition 5.10. (Tree depth.) *We say that $d \in \mathbb{N}$ is the tree depth of a tree T if the maximum degree for every multiple segmentation is d.*

Definition 5.11. (Set of successor nodes.) *Let $\overline{\mathbf{x}} \in \mathbf{B}$, $\mathbf{B} \in \mathbb{IR}^k$, and for $e^\sharp \in AEnv_{\mathbf{f}}$, let $e^\sharp(\overline{\mathbf{x}}) \in \mathbb{IR}^k$ be a root node. Moreover, let $n \geq 1$ be a segmentation degree. The set of successors of a node $e^\sharp_{n,I_d,\ldots,I_{d-j}}(\overline{\mathbf{x}})$, $1 \leq j \leq d-1$, is delivered by the following operator:*

$$Succ_d(e^\sharp_{n,I_d,\ldots,I_{d-j}}(\overline{\mathbf{x}})) \;=\; \left\{ e^\sharp_{n,I_d,\ldots,I_{d-j-1}}(\overline{\mathbf{x}}) \in AEnv_{\mathbf{f}} \;\middle|\; \begin{array}{l} \forall I_i \in [0,n-1]^k,\ 1 \leq j \leq d-1,\ i \in [d-j-1,d] \\ e^\sharp_{n,I_d,\ldots,I_{d-j-1}}(\overline{\mathbf{x}}) \;=\; \varphi_{n,I_d,\ldots,I_{d-j-1}}(e^\sharp_{n,I_d,\ldots,I_{d-j}}(\overline{\mathbf{x}})) \end{array} \right\}.$$

For a segmentation degree $n \geq 1$, each node in a tree can have up to n^k child nodes.

Definition 5.12. (Terminal node.) *A* terminal node *is a node where the set of its successors is empty.*

For notational purposes, when constructing a segmentation tree of depth d, the root node is assigned depth d, the successor nodes of the root node are assigned depth $d-1$, and so forth. There are two types of terminal nodes: those with $d = 0$, and those with $d > 0$. The existence of each type in a tree is determined by each particular search performed.

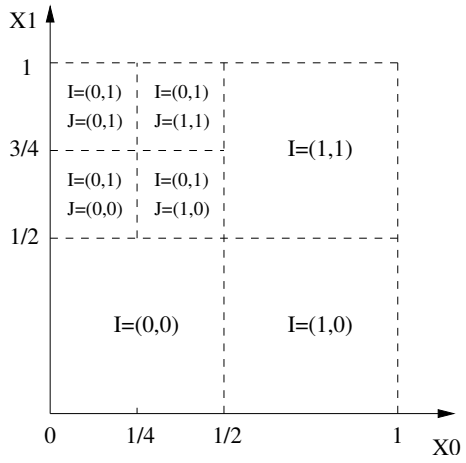
Definition 5.13. (Deepest terminal node.) *Let $e^\sharp \in AEnv_{\mathbf{f}}$ and $\overline{\mathbf{x}} \in \mathbf{B}$, $\mathbf{B} \in \mathbb{IR}^k$, where $e^\sharp(\overline{\mathbf{x}}) \in \mathbb{IR}^k$. For a tree depth $d \in \mathbb{N}$ and segmentation degree $n \geq 1$, a* deepest terminal node *in a tree is denoted are denoted by*

$$e^\sharp_{n,I_d,\ldots,I_0}(\overline{\mathbf{x}}) \;=\; \varphi_{n,I_d,\ldots,I_1}(e^\sharp(\overline{\mathbf{x}})),$$

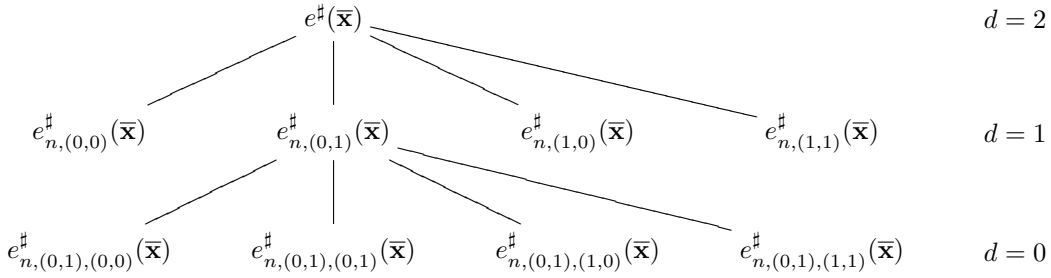
where $I_d, \ldots, I_0 \in [0, n-1]^k$.

The following example illustrates these definitions.

Example 5.14. *In example 4.24, for a segmentation degree $n = 2$, $I \in [0, n-1]^k$, where*



(a) Segmentations for $\mathbf{U}^2$ and $n = 2$.



(b) A segmentation tree for $n = 2$ and $d = 2$.

Figure 5.1: *Segmentation of* $\mathbf{U}^2$ *and its corresponding segmentation tree.*

$I = (0,1)$, and $e^\sharp(x_0, x_1) = \mathbf{U}^2$, let

$$\begin{aligned} e^\sharp_{2,(0,1)}(\overline{\mathbf{x}}) &= \varphi_{2,(0,1)}(e^\sharp(x_0, x_1)) \\ &= e^\sharp_{2,0}(x_0) \times e^\sharp_{2,1}(x_1) \\ &= [0, \frac{1}{2}] \times [\frac{1}{2}, 1]. \end{aligned}$$

For $J \in [0,1]^2$ where $J = (0,0)$,

$$\begin{aligned} e^\sharp_{2,(0,1),(0,0)}(\overline{\mathbf{x}}) &= \varphi_{2,(0,0)}(\varphi_{2,(0,1)}(\overline{\mathbf{x}})) \\ &= \varphi_{2,(0,0)}(e^\sharp_{2,(0,1)}(\overline{\mathbf{x}})) \\ &= \varphi_{2,(0,0)}([0, \frac{1}{2}] \times [\frac{1}{2}, 1]) \\ &= [0, \frac{1}{4}] \times [\frac{1}{2}, \frac{3}{4}]. \end{aligned}$$

Figure 5.1 depicts both the segmentations on the unit box $\mathbf{U}^2$ and the mapping of such set of segmentations onto a tree. The set of terminal nodes includes all the nodes occurring in the tree except $e^\sharp(\overline{\mathbf{x}})$ and $e^\sharp_{n,(0,1)}(\overline{\mathbf{x}})$; namely,

$$Succ_d(e^\sharp(\overline{\mathbf{x}})) \ \cup \ Succ_d(e^\sharp_{n,(0,1)}(\overline{\mathbf{x}})) \ \setminus \ \{e^\sharp(\overline{\mathbf{x}}),\ e^\sharp_{n,(0,1)}(\overline{\mathbf{x}})\}$$

The deepest terminal nodes are included in the set $Succ_d(e^\sharp_{n,(0,1)}(\overline{\mathbf{x}}))$.

5.3.2 Constructing and Traversing the Search Space

In this section we present a tree search mapping that builds a search space of abstract environments. This mapping performs a breadth-first search and prunes those nodes of the tree which do not hold the Bernstein test. The search is performed for all nodes up to a preset depth $d \in \mathbb{N}$. In order to ease the presentation of this operator, consider first the following worked example which illustrates the tree construction and search.

Example 5.15. *This example illustrates the search process through a tree of segmentations, for a given interval valued polynomial which has zeroes in the interval in which it is defined. This search identifies the segmentations which do not include zeroes for the interval valued polynomial. Thus, consider the interval valued polynomial*

$$[p](x) \ = \ [1,1]x^3 + [-1,-1]x + [0.25, 0.26],$$

defined for $x \in \mathbf{U}$. Figure 5.2 plots $[p](x)$, where it can be noted that $\mathbf{U}$ includes two zeroes. Let $e^\sharp \in AEnv_{\mathbf{f}}$ where $e^\sharp(x) = \mathbf{U}$. Moreover, let $n = 2$ be a segmentation degree

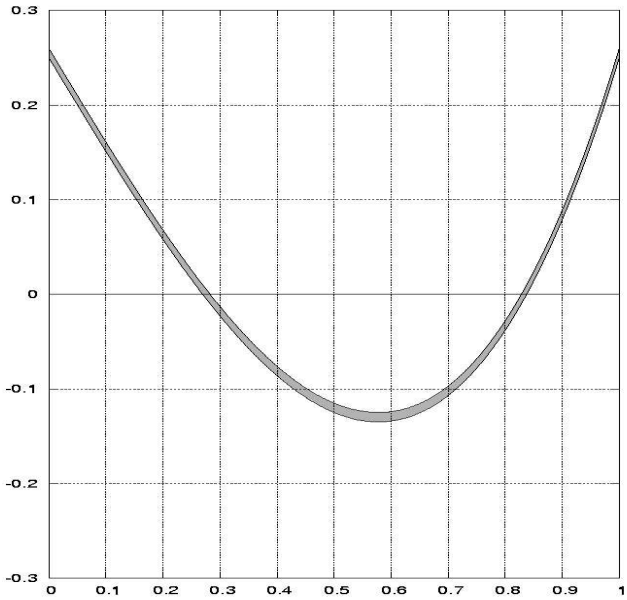


Figure 5.2: *Plot of* $[p](x) = [1,1]x^3 + [-1,-1]x + [+0.25,+0.26]$ *for* $x \in [0,1]$.

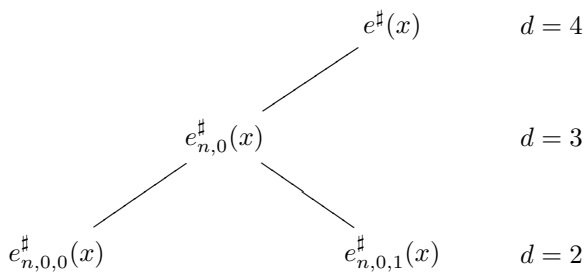
and $d = 4$ *the tree depth. Note that*

$$Enc([p](x), e^\sharp(x)) \;=\; [-5/12, 312/1200]$$

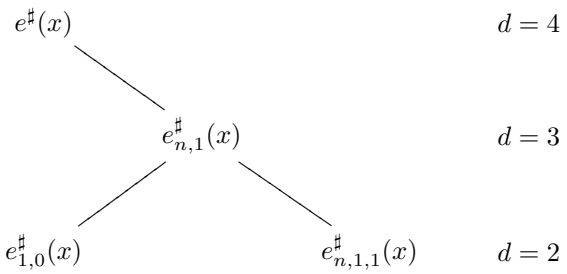
and thus the Bernstein test fails. Then we must process the successors of $e^\sharp(x)$*; namely,* $e^\sharp_{n,0}$ *and* $e^\sharp_{n,1}$*, for which*

$$\begin{aligned} Enc([p](x), e^\sharp_{n,0}) &= [-3/24, 312/1200] \text{ and} \\ Enc([p](x), e^\sharp_{n,1}) &= [-3/24, 312/1200], \end{aligned}$$

The Bernstein test fails and we must process the successors for $d = 2$*. This is illustrated in the following segmentation tree:*



and



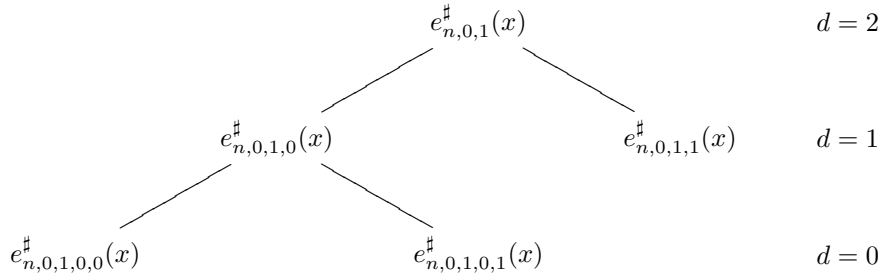
where

$e^\sharp(x) = [0,1]$
$e^\sharp_{n,0}(x) = [0,1/2] \quad e^\sharp_{n,1}(x) = [1/2,1]$
$e^\sharp_{n,0,0}(x) = [0,1/4] \quad e^\sharp_{n,0,1}(x) = [1/4,1/2] \quad e^\sharp_{n,1,0}(x) = [1/2,3/4] \quad e^\sharp_{n,1,1}(x) = [3/4,1]$

Nodes $e^\sharp_{n,0,1}(x)$ *and* $e^\sharp_{n,1,1}(x)$ *fail the Bernstein test since*

$$Enc([p](x), e^\sharp_{n,0,1}(x)) = [-2/3, 305/500] \text{ and}$$
$$Enc([p](x), e^\sharp_{n,1,1}(x)) = [-1/2, 3820/12000].$$

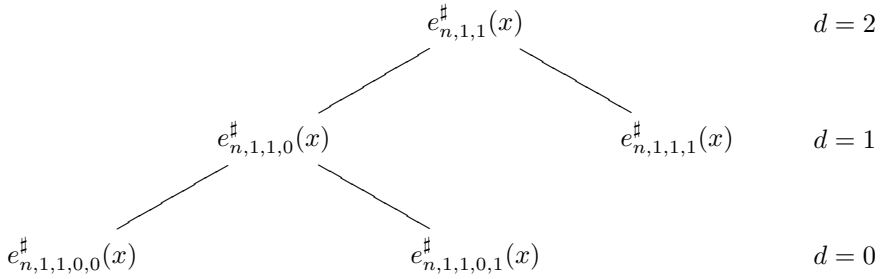
As the tree depth has not been reached, we must calculate the successors of these nodes; namely,



where

$$e^\sharp_{n,0,1,1}(x) = [3/8, 1/2]$$
$$e^\sharp_{n,0,1,0,0}(x) = [1/4, 5/16] \quad e^\sharp_{n,0,1,0,1}(x) = [5/16, 3/8]$$

and



where

$$\begin{array}{l} e^\sharp_{n,1,1,1}(x) \;=\; [7/8, 1] \\ e^\sharp_{n,1,1,0,0}(x) \;=\; [3/4, 13/16] \quad e^\sharp_{n,1,1,0,1}(x) \;=\; [13/16, 7/8]. \end{array}$$

In level $d = 1$*, we have that*

$$\begin{array}{rcl} Enc([p](x), e^\sharp_{n,0,1,0}(x)) & = & [-37/512, 164/6400] \text{ and} \\ Enc([p](x), e^\sharp_{n,1,1,0}(x)) & = & [-7/96, 2284/38400]. \end{array}$$

These nodes do not hold the Bernstein test. Thus we compute the Bernstein test to their successors. We note that nodes $e^\sharp_{n,0,1,0,0}(x)$ *and* $e^\sharp_{n,1,1,0,1}(x)$ *do not hold the Bernstein test. Therefore such nodes can be excluded from the set of intervals that do not contain zeroes. Thus, the set of abstract environments for which there exists no zeroes for* $[p](x)$ *in* $x \in [0, 1]$ *for up to tree depth* $d = 3$ *is*

$$NR^{n,3}(\{e^\sharp\}, [p](x)) \;=\; \left\{\; e^\sharp_{n,0,0},\; e^\sharp_{n,1,0}, e^\sharp_{n,0,1,1},\; e^\sharp_{n,1,1,1}, e^\sharp_{n,0,1,0,1},\; e^\sharp_{n,1,1,0,0} \;\right\},$$

where $NR^{n,3}$ *denotes a tree search operator defined below.*

We present now a formalisation of the operator

Definition 5.16. (Tree search operator.) *The mapping that delivers the set of abstract environments for which no zero exists for a given floating-point polynomial expression, for tree depth* $d \in \mathbb{N}$*, where* $NR^{n,d} : \wp_{\text{fn}}(AEnv_{\mathbf{f}}) \times (\mathbb{R}^k \to \mathbb{IR}) \to \wp_{\text{fn}}(AEnv_{\mathbf{f}})$*,* $n, d \in \mathbb{N}$*,* $n \geq 1$*, is defined for* $m \geq 1$ *as*

$$\begin{array}{rcl} NR^{n,0}(E, [q](\overline{\mathbf{x}})) & = & \{e^\sharp \in AEnv_{\mathbf{f}} \;\mid\; \forall e^\sharp \in E \;:\; Bt^m_{e^\sharp(\overline{\mathbf{x}})}([q](\overline{\mathbf{x}})) = 1\} \\ NR^{n,d}(E, [q](\overline{\mathbf{x}})) & = & g_1(E, [q](\overline{\mathbf{x}})) \;\cup\; NR^{n,d-1}(g_2(E, [q](\overline{\mathbf{x}})),\; [q](\overline{\mathbf{x}})), \end{array}$$

where

$$g_1(E,[q](\overline{\mathbf{x}})) = \left\{ e^\sharp_{n,I_d,\ldots,I_{d-j}} \in AEnv_{\mathbf{f}} \;\middle|\; \begin{array}{c} \forall e^\sharp_{n,I_d,\ldots,I_{d-j}} \in E, \\ I_{d-j},\ldots,I_d \in [0,n-1]^k,\ 1 \le j \le d-1, \\ s.t.\ Bt^m_{e^\sharp_{n,I_d,\ldots,I_{d-j}}(\overline{\mathbf{x}})}([q](\overline{\mathbf{x}})) = 1 \end{array} \right\}$$

$$g_2(E,[q](\overline{\mathbf{x}})) = \left\{ e^\sharp_{n,I_d,\ldots,I_{d-j-1}} \in AEnv_{\mathbf{f}} \;\middle|\; \begin{array}{c} \forall e^\sharp_{n,I_d,\ldots,I_{d-j}} \in E,\ 1 \le j \le d-1, \\ I_{d-j-1},\ldots,I_d \in [0,n-1]^k, \\ s.t.\ Bt^m_{e^\sharp_{n,I_d,\ldots,I_{d-j}}(\overline{\mathbf{x}})}([q](\overline{\mathbf{x}})) = 0, \\ e^\sharp_{n,I_d,\ldots,I_{d-j-1}}(\overline{\mathbf{x}}) \in Succ_d(e^\sharp_{n,I_d,\ldots,I_{d-j}}(\overline{\mathbf{x}})) \end{array} \right\}.$$

This mapping is initially queried with a singleton set consisting of an abstract environment, namely $\{e^\sharp\}$. This abstract environment is then segmented into n segmentations. A pruning is performed on those segmentations for which the Bernstein test does not hold. Then each remaining segmentation is further segmented. This process of segmenting and pruning is repeated up to d times. As a result, we have a (possibly empty) set of terminal nodes (segmentations) up to tree depth d which hold the Bernstein test.

The pruning of tree nodes reduces the number of nodes evaluated in the tree. It is not necessary to further segment a segmentation for which it has been proved that no zero is included in it. This saves processing time and does not affect the final result of the search.

The $NR^{n,d}$ mapping for $n \ge 1$ and $d \in \mathbb{N}$ is sound since it is guaranteed that no segmentation delivered by the mapping includes any zero. A finite number of recursions are computed, since in the recursive definition, $d \in \mathbb{N}$ is decreased monotonically and $\langle \mathbb{N}, \le \rangle$ is a well-ordered set, thus reaching the nonrecursive case of the definition of the tree search mapping in a finite number of recursive calls.

Theorem 5.17.

$$\bigcup_{e^\sharp_\ell \in NR^{n,d}(\{e^\sharp\},[p](\overline{\mathbf{x}}))} e^\sharp_\ell(\overline{\mathbf{x}}) \;\subseteq\; e^\sharp(\overline{\mathbf{x}}).$$

Proof. By proposition 4.29 the set union of the successors of a node are included in the parent node, and by corollary 4.30, each node is included in its parent node. Since the root node has no parent, by the mentioned proposition and corollary, every node is included in the root node. □

5.3.3 Optimisation of the Search Space

In this section we propose first an optimisation to reduce the number nodes in the search space that hold the Bernstein test. This optimisation does not lose information obtained from the initial solution tree. Second, we propose ways of improve the accuracy of the solution tree.

Tree compactification. A box that has been segmented into a set of segmentations, can be reconstructed back either by the set union mapping of the segmentations or by the poly-hull mapping of the segmentations; namely

$$\uplus(Succ_d(e^\sharp_{n,I_d,\ldots,I_{d-j}}(\overline{\mathbf{x}}))) \;=\; \cup(Succ_d(e^\sharp_{n,I_d,\ldots,I_{d-j}}(\overline{\mathbf{x}}))).$$

This observation is useful to optimise the tree in the case where all the successors of a tree node hold the Bernstein test. Thus, the successors can be pruned. This optimisation is of special importance when the interval valued polynomial evaluates to an interval in which either of its boundaries are closed to zero. In that case, the Bernstein mappings are likely to deliver Bernstein coefficients for which the Bernstein test fails, up to a certain tree depth. Recall that when a node fails the test and the maximum tree depth is not reached, then that node is further expanded. For a sufficiently large tree depth, it can be proved that all the expanded nodes hold the Bernstein test. Thus, we can propagate the optimisation upward toward the first parent node that failed the test, and mark that node as *free* of zeroes.

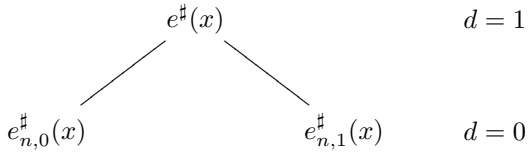
Example 5.18. *Consider the interval valued polynomial*

$$[p](x) \;=\; [-1,-1]x^3 \;+\; [1,1]x^2 \;+\; [1,1]x \;+\; [0.5,0.6]$$

defined for $x \in [-1,1]$. *Let* $e^\sharp \in AEnv_{\mathbf{f}}$ *where* $e^\sharp(x) = [-1,1]$. *Then*

$$Enc(\pi_{e^\sharp(x)}([p](x))) \;=\; [-7/6, 8/5],$$

and the Bernstein test fails. Let $n = 2$ *be a segmentation degree* $n = 2$ *and* $d = 1$ *a tree depth. Then we can construct a tree such as*



The enclosures for each successor node, namely

$$Enc(\pi_{e^\sharp_{n,0}(x)}([p](x))) \;=\; [1/6, 8/5] \text{ and}$$
$$Enc(\pi_{e^\sharp_{n,1}(x)}([p](x))) \;=\; [1/2, 8/5],$$

hold both the Bernstein test. Thus we can compactify the tree into its root node $e^\sharp(x)$. *However, it is necessary to preserve the tighter enclosure and set of Bernstein coefficients*

obtained from the successors of the root, to ensure a sound (and also more precise) polyhedra abstraction.

Note that in order to avoid unnecessary segmentations that do not include any zero, it is convenient to select a small segmentation degree n. This is in contrast with the purpose of finding tight range enclosures for an interval valued polynomial, in which case a larger n allows for tighter enclosures.

It is important to note that the notion of compactification is similar to the idea underlying the minimal method. However, the main difference here with respect to that method is that we compactify subsets of segmentations obtained from an initial box, whereas in the minimal method we compactify the whole set of segmentations obtained from an initial box. Moreover, segmenting the whole set of segmentations is unsound when the least one segmentation includes possible zeroes.

Moreover, note that when each child in the successor set fail the Bernstein test in the lowest tree level, then the parent node must be excluded from the solution set.

5.4 Polyhedra Abstraction

In this section we propose abstractions for floating-point polynomial quotients into the polyhedra domain. Since the division operator is not defined when the divisor is zero, we must examine the existence of zeroes in the divisor polynomial. First of all note that the zeroes of a floating-point polynomial expression are included in the set of floating-point values for which the concrete semantics of the polynomial expression evaluate to 0. A divisor floating-point polynomial is approximated by an interval valued polynomial. If the concrete evaluation of an interval valued polynomial for a floating-point value, evaluates to an interval which includes the interval $[0, 0]$, then the floating-point value is a zero in the floating-point polynomial.

In the following definition we present an operator for the abstraction of floating-point polynomial quotients into polyhedra. We use the operator $NR^{n,d}$ to identify those segmentations that do not include a zero for a given floating-point expression. Then it defines an interval valued polynomial from which a polyhedron is constructed. The presented operator delivers a set of polyhedra which does not contain zeroes for the polynomial quotients defined over a box.

Definition 5.19. (Polynomial quotient polyhedra abstraction.) *Let the map* $\Upsilon^{n,d}_{e^\sharp} : AEnv_{\mathbf{f}} \times (\mathbb{R}^k \to \mathbb{IR}) \to \wp(\mathbb{CP}_{k+1})$ *can be defined for different Bernstein mappings. From definition 3.16, let* $p_{\mathbf{f}} \oslash_{\mathbf{f},\mathbf{r}} q_{\mathbf{f}}$ *be a floating-point polynomial quotient, where* $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(p_{\mathbf{f}} \oslash_{\mathbf{f},\mathbf{r}} q_{\mathbf{f}})$, $[\![p_{\mathbf{f}} \oslash_{\mathbf{f},\mathbf{r}} q_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp) = [r](\overline{\mathbf{x}})$, *and where*

$$\begin{aligned}[r](\overline{\mathbf{x}}) \;=\; & ([p](\overline{\mathbf{x}}) \;\boxslash^\sharp\; Opt([q](\overline{\mathbf{x}}), e^\sharp(\overline{\mathbf{x}}))) \;\boxplus^\sharp \\ & \varepsilon_{\mathbf{f}}([p](\overline{\mathbf{x}}) \;\boxslash^\sharp\; Opt([q](\overline{\mathbf{x}}), e^\sharp(\overline{\mathbf{x}}))) \;\boxplus^\sharp\; [-mf_{\mathbf{f}}, mf_{\mathbf{f}}].\end{aligned}$$

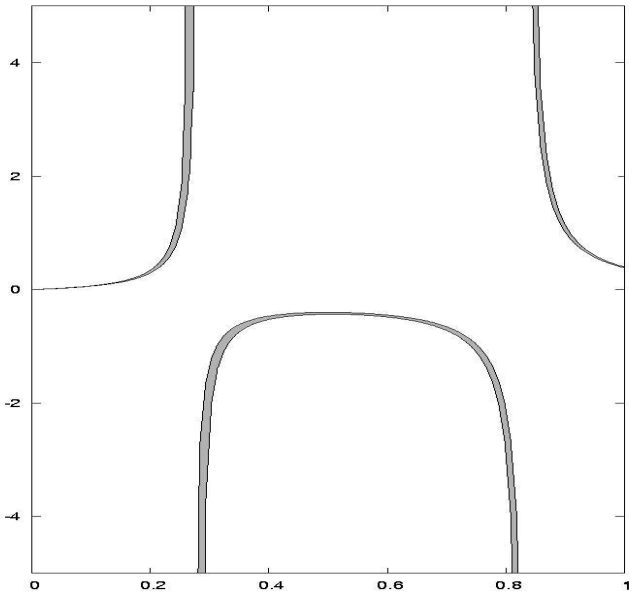


Figure 5.3: *Plot of* $[r](x) = [0.1, 0.1]x \boxslash^\sharp ([1,1]x^3 + [-1,-1]x + [-0.25,-0.24])$ *for* $x \in [-1,1]$.

Then the polyhedra abstraction operator $\Upsilon^{n,d}$ *for* $[r](\overline{\mathbf{x}})$ *is defined as*

$$\Upsilon^{n,d}_{e^\sharp}([r](\overline{\mathbf{x}})) \;=\; \left\{ P \;\in\; \mathbb{CP}_{k+1} \;\middle|\; \begin{array}{l} \forall\; s^\sharp \in NR^{n,d}(\{e^\sharp\}, [q](\overline{\mathbf{x}})) \;:\; \exists\; P \in \mathbb{CP}_{k+1} \;.\; \exists\; [a^-, a^+] \in \mathbb{IR} \;. \\ Enc(\pi_{s^\sharp}[q](\overline{\mathbf{x}}), s^\sharp(\overline{\mathbf{x}})) \;=\; [a^-, a^+] \;\wedge \\ [r'](\overline{\mathbf{x}}) \;=\; \big([p](\overline{\mathbf{x}}) \;\boxslash^\sharp\; [a^-, a^+]\big) \;\boxplus^\sharp\; \varepsilon_{\mathbf{f}}\big([p](\overline{\mathbf{x}}) \;\boxslash^\sharp\; [a^-, a^+]\big) \\ \quad \boxplus^\sharp\; [-mf_{\mathbf{f}}, mf_{\mathbf{f}}] \;\wedge \\ P \;=\; \uplus\, \pi_{s^\sharp}([r](\overline{\mathbf{x}})) \end{array} \right\}.$$

The abstraction delivered by Υ is sound, as shown in chapter 4 for the Bernstein mapping. The following example illustrates the abstraction of polynomial quotients into polyhedra.

Example 5.20. *Let*

$$[r](x) \;=\; \frac{[0.1, 0.1]x}{[1,1]x^3 + [-1,-1]x + [-0.25,-0.24]},$$

for $x \in \mathbf{U}$*, as plotted in figure 5.3, and where the divisor expression corresponds to* $[p](x)$ *of example 5.15. We calculate the range enclosure of* $[p](x)$ *for each abstract environment that has been proved not to include zeroes, that is to say, for each abstract environment*

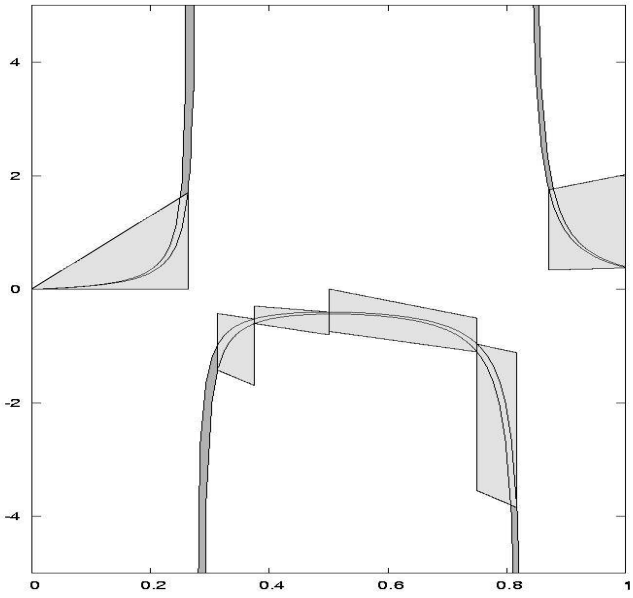


Figure 5.4: *Plot of set of polyhedra for the abstraction of* $[r](x)$ *for* $x \in [0, 1]$.

included in $NR^{n,3}(\{e^\sharp\}, [p](x))$*; namely,*

$$\begin{aligned}
Enc([p](x), e^\sharp_{n,0,0}) &= [3/192, 312/1200] \\
Enc([p](x), e^\sharp_{n,1,0}) &= [-7/48, -1308/19200] \\
Enc([p](x), e^\sharp_{n,0,1,1}) &= [-3/241, -9564/153600] \\
Enc([p](x), e^\sharp_{n,1,1,1}) &= [19/384, 312/1200] \\
Enc([p](x), e^\sharp_{n,0,1,0,1}) &= [-111/1536, -27012/1228800] \\
Enc([p](x), e^\sharp_{n,1,1,0,0}) &= [-7/96, -3264/153600].
\end{aligned}$$

Thus, we can build a convex hull from each enclosure, as shown in definition 5.19. Figure 5.4 plots $[r](x)$ *and a set of polyhedra which approximates parts of* $[r](x)$*. A deeper search in the tree would lead to a larger number of polyhedra around the asymptotes of* $[r](x)$*. This means the resulting abstraction would be more precise. Moreover, it is important to remark the purpose of this example is to illustrate the process: when abstracting a floating-point polynomial quotient as part of an analysis process, higher tree depth is required in order not to loose information of boxes near a zero.*

The recursive and minimal Bernstein mappings can also be used to abstract polynomial quotients, though at a high cost. Moreover, the gain in precision does not payoff the high cost especially when a segmentation due to the existence of (possible) zeroes has been computed. That is to say, a segmentation has already been processed, which guarantees tighter enclosures in the abstraction.

5.5 Discussion and Related Work

Interval-Based Branch and Prune Methods. Branch and prune methods based on interval arithmetic for solving systems of polynomials includes [vHMK97], which is implemented in the global optimisation system Numerica [vHMD97]. The method proposed in this chapter does not require a preprocessing for zero isolation (such as for instance in [PCT95]). On the contrary, it computes zeroes existence in a given bounding box and either refines or excludes those boxes that contain at least a zero. By assigning a sufficiently large recursion degree $d \in \mathbb{N}$ and reverting to the valuation of the Bernstein test, we would obtain those boxes for which the range enclosure could include $[0, 0]$. The use of segmentations in the intervalisation method may not be precise enough to determine the existence of zeroes in an interval valued polynomial defined over a box, since intervalisation provides a rough approximation to the range enclosure of functions.

Inclusion and Exclusion Methods. Two different approaches can be taken in order to obtain parts of an abstract environment that do not include any zero; namely zero inclusion and zero exclusion methods. In the first approach, after enclosing the set of zeroes in the abstract environment, we must construct sets of convex polyhedra for the parts of the abstract environment not included in such enclosure. This approach may become complex and tedious in order to further partition the parts of the abstract environment for the purpose of constructing convex polyhedra [BL03]. In the second approach we isolate parts of the abstract environment for which it can be proved no zero is included and where the construction of convex polyhedra is trivial. However, this method may produce a large number of abstract environment partitions that must be processed. A bisection method based on zero exclusion was proposed in [MS87]. The main difference with our approach is that we compute interval valued polynomials whereas Morgan processed more general functions.

Solving Quantified Constraints. The branch and prune method presented here can be used also to solve quantified constraints, namely, parametric constraints with quantified parameters. This can be achieved by reverting the definition of the Bernstein test [Gol06]. For instance, the parametric constraint $\alpha \cdot x^2 + \beta = 1$ on the variable x, where $\alpha \in [1, 2]$ and $\beta \in [1/2, 1]$, leads to the quantified constraint

$$\big(\exists \alpha \in [1, 2],\ \exists \beta \in [1/2, 1]\big)\big(\alpha \cdot x^2 + \beta = 1\big).$$

This constraint can be reformulated into an interval valued polynomial equation such as

$$[1, 2]x \ + \ [1/2, 1] \ = \ [1, 1]$$

Then, the approximation of the set of solutions for such quantified constraint can be performed by solving this polynomial equation by using the reversed Bernstein test.

Chapter 6

Forward Relational Analysis for Constraint Logic Programs

In this chapter we present an analysis for the verification of the nonexistence of floating-point overflow when querying a logic program. Central in this analysis is the use of the finite polyhedra powerset domain, which proves appropriate for the precise abstraction of nonlinear expressions. We present precise abstractions for nonlinear assignments and nonlinear tests, which allow for verifying the absence of errors where linear versions would inevitably report a potential hazard. This analysis does not halt if a potential overflow is detected. Instead, it always reaches the output variables of the analysed program. Reaching such variables is necessary in order to produce the needed information to run a backward analysis for the inference of input that do not cause overflow. We analyse different classes of overflow including the intermediate overflow.

6.1 Introduction

Program analysis for the detection of floating-point run-time errors such as overflow or division by zero is important for preventing abnormal *behaviours* that may lead to disastrous results. Existing work is usually concerned with imperative programs [Gou01, Min04b], and up to date, the analysis of errors due to floating-point in logic programs has been focused on the analysis for termination [SS02, SM04]. Logic programs represent a unifying language for denoting models for physical systems [JL87, Col90, HJM+91, Hon93, OV93,

JM94, Hon94, vE99] — unifying in the sense that problems of different nature can be described in the common language of logic programming. In modelling physical systems, floating-point arithmetic plays a central role as the preferred arithmetic for computation. In this respect, logic programs with floating-point computations are vulnerable to errors inherent of floating-point arithmetic such as overflow. In this chapter we address the problem of analysing logic programs with floating-point computations using abstract interpretation and the polyhedra domain. Our analysis focuses on the detection of floating-point overflow.

In particular, we use the finite powerset domain of polyhedra. It allows for delivering highly precise abstractions which are important for the analysis of nonlinear expressions, and especially when an expression is not defined for all the elements of the domain. This is the case for instance of an expression with possible divisions by zero. By defining the set of subsets of a domain for which an expression is defined, we can compute an analysis that will reach the output variables of the program. This is necessary in order to have available the required information for computing a backward analysis, from the output to the input variables of a program.

The logic programs we analyse include constraints on the input and output variables, and these constraints as well as the expressions involved, are abstracted into polyhedra. From these constraints we can define three classes of overflow; namely, the absolute overflow, in which the result of evaluating an expression has no representation in the underlying floating-point system; the relative overflow, in which the declared constraints on the output variables are violated; and the intermediate overflow, in which an absolute overflow occurs when evaluating a subexpression. The purpose of the analysis is to constrain the sets of values that can instantiate the output variables and that do not fall into any of these three classes of overflow.

Our contribution in this chapter is the presentation of abstractions for (possibly nonlinear) assignments and tests in the powerset of polyhedra. These abstractions are general in that linear and nonlinear expressions are considered. Moreover, we introduce a forward analysis for the detection of floating-point overflow and overflow due to division by zero which always reaches the output variables of a program. This is necessary in order to develop a backward analysis which infers preconditions on the input variables of a program to avoid possible divisions by zero.

6.2 Formal Language

In order to define an operational semantics for both forward and backward analyses, we formalise first the concrete language we analyse, and also its abstract version.

6.2.1 Logic Programs

Logic programs considered in this analysis include two classes of numerical atoms; namely, assignment and test atoms. Assignment atom is denoted by

$$v \text{ is } expr_{\mathbf{f}},$$

where for a substitution θ, where $\mathsf{vars}(expr_{\mathbf{f}}) \subseteq \mathsf{vars}(\theta)$, θ instantiates all variables in $\mathsf{vars}(expr_{\mathbf{f}})$. We call variable v the assignee and the floating-point expression $expr_{\mathbf{f}}$ the assigner. Test atom is denoted by

$$expr_{\mathbf{f}} \ \leq \ expr_{\mathbf{f}}.$$

Logical connectives such as **or** and **and** are inherent to logic programming. Thus we do not include explicitly such operators in our concrete semantics. We assume that each argument in a test has the same floating-point format. For simplicity, we do not consider casting between different formats. The corresponding build-in in a Prolog system such as SICStus is denoted by $=</2$. A clause with empty body is denoted by

$$p(\overline{\mathbf{x}}) \leftarrow$$

whereas a clause with nonempty body is denoted by

$$p(\overline{\mathbf{x}}) \leftarrow p_1(\overline{\mathbf{x}}_1), \ldots, p_n(\overline{\mathbf{x}}_n).$$

Moreover, a query is denoted by

$$\leftarrow p_1(\overline{\mathbf{x}}_1), \ldots, p_n(\overline{\mathbf{x}}_n).$$

An empty logic program is denoted by $\triangledown$. Note that an atomic predicate includes the $\leftarrow$ symbol to differentiate it from a polynomial. When a logic program is denoted in Prolog-like syntax, it is platform and floating-point system dependent. We refer to this notation as dependent notation. Otherwise, we assume left-to-right SLD resolution and no preset floating-point system. We refer to this notation as independent notation. Most Prolog systems, such as SICStus [oCS06] use the IIIE-754 standard double format to represent their floating-point system values. The theoretical framework presented in this chapter uses independent notation, whereas the examples use dependent notation. In what follows, we may refer to a logic program as *program*.

A normalisation process is required for the case of arguments that are grounded in the head atom of a clause. This process consists in declaring the grounded input arguments as tests and the grounded output arguments as assignments.

Example 6.1. *An atomic clause such as* $p(0,1)$ *is normalised as*

$$p(X,Y) \leftarrow X == 0,\ Y \text{ is } 1.$$

6.2.2 Syntactic Conditions for Correctness

Logic programs analysed here are moded: each argument position is assumed to be either an input or an output. Moreover, programs are required to be well-moded [AM94], where a correctness condition between the input arguments and the output arguments must hold. Though well-modedness is commonly used to study termination of logic programs [Plü90, EBC99, AL95], our purpose here is to ensure that the input and output arguments of predicates that compute floating-point expressions (which includes assignments and tests) do not cause a run-time instantiation error.

Let $p(m_1,\ldots,m_k)$ be a mode for predicate p/k, where each m_i, $i \in [1,k]$ is preset as an input or output argument, namely, $m_i \in In \cup Out$, where $In \cap Out = \emptyset$. Thus, by $p(\tilde{u},\tilde{v})$ we denote an atom where the sequence $\tilde{u}$ denotes the terms that occur in the input arguments, and the sequence $\tilde{v}$ denotes the terms that occur in the output arguments. We say that a definite clause

$$p_0(\tilde{t}_0,\tilde{s}_{n+1}) \leftarrow p_1(\tilde{s}_1,\tilde{t}_1),\ldots,p_n(\tilde{s}_n,\tilde{t}_n)$$

is *well-moded* if for each $i \in [1,n+1]$,

$$\mathsf{vars}(\tilde{s}_i) \subseteq \bigcup_{j=0}^{i-1} \mathsf{vars}(\tilde{t}_j).$$

A query

$$\leftarrow q_1(\tilde{s}_1,\tilde{t}_1),\ldots,q_n(\tilde{s}_n,\tilde{t}_n)$$

is called *well-moded* if and only if the clause

$$q \leftarrow q_1(\tilde{s}_1,\tilde{t}_1),\ldots,q_n(\tilde{s}_n,\tilde{t}_n)$$

is *well-moded*, where q is an atom of arity zero. Moreover, we say that a program is *well-moded* if each clause in it is well-moded. We assume left-to-right SLD resolution. Thus in the first atom in a well-moded query, the input arguments are ground. These ground argument positions are available as output $\tilde{t_0}$ for any input arguments in the body of the clause. For instance for input argument $\tilde{s_1}$, by well-modedness condition, $\mathsf{vars}(s_1) \subseteq \mathsf{vars}(t_0)$, but $\mathsf{vars}(t_0)$ are grounded so no instantiation error can occur in p_0 or p_1. In a definite clause or query in which all atoms have input and output arguments, assume the input arguments of an atom are grounded. By definition of well-modedness, this atom can instantiate the output arguments from the input arguments. Moreover,

these output arguments are input arguments to another atom (or atoms), thus ensuring that the subsequent output of these other atoms will be instantiated as well. Thus, well-modedness of a definite clause or query ensures the persistence of groundness throughout all the atoms. The following result formalises this persistence of groundness in well-moded programs (a proof of the following proposition can be found in [AP94]).

Proposition 6.2. *Let P be a well-moded logic program and let denote by $\tilde{A}$ the well-moded query $\leftarrow q_1(\tilde{s}_1, \tilde{t}_1), \ldots, q_n(\tilde{s}_n, \tilde{t}_n)$. Then for each answer substitution σ of $\tilde{A}$ in P, $\tilde{A}\sigma$ is ground.*

Recall that a substitution maps each variable in its domain to either another variable or a (possibly ground) term. By the proposition above, each variable in the answer substitution is mapped to a ground term. In the context of our analysis, a concrete environment corresponds to an answer substitution obtained from a well-moded query to a well-moded program. Thus, a concrete environment is defined for each output variable of a well-moded query. In what follows we assume programs are well-moded and queries to them are also well-moded, since well-modedness ensures that the necessary input to an atom being queried is available. This is important to avoid instantiation run-time errors for instance in assignment and test predicates.

6.3 Concrete Semantics

The concrete semantics of logic programs considered here, which includes floating-point arithmetic, is defined by means of a transition system of program states. A program state consists of a tuple of a predicate and a concrete environment. The transitive closure of a transition from the initial state to the empty program state (or final state) defines the concrete semantics.

6.3.1 Definitions

Central in defining a concrete semantics for logic programs with floating-point arithmetic are the assignment and test states. This section presents the necessary definitions to introduce such states in order to define the concrete semantics of a program.

Definition 6.3. (Program state.) *A* program state *is defined as a tuple of the form*

$$\langle g_1, \ldots, g_k;\ R \rangle,$$

where $g_1, \ldots, g_k \in \mathsf{Atoms}$ and $R \in \wp_{\mathrm{fn}}(Env_{\mathbf{f}})$ is a set of concrete environments defined for each variable in $g_1, \ldots, g_k$.

The set of states is denoted by States.

In the definition above, we use sets of environments in order to take into account the concrete semantics presented in definition 3.5, which is defined for sets of floating-point values.

Definition 6.4. (Query state.) *A* query state *is a program state of the form*

$$\langle g_1, \ldots, g_k;\ R_0 \rangle,$$

where $g_1, \ldots, g_k \in \mathsf{Atoms}$ *and* $R_0 \in \wp_{\mathrm{fn}}(Env_{\mathbf{f}})$ *is the initial set of concrete environments.*

Program states that involve numerical atoms are defined as follows.

Definition 6.5. (Assignment state.) *An* assignment state *is defined by*

$$\langle v \ \leftarrow \ expr_{\mathbf{f}},\ g_1, \ldots, g_k;\ R \rangle \ \in \ \mathsf{States}.$$

In an assignment atom, we assume that the variables occurring in the expression are input variables, whereas the assignee is an output variable. However, in a test atom every variable occurring in the expressions is assumed to be an input variable, and further, no output variables can occur. This restriction on output variables still ensures well-modedness, as no atom in a well-moded clause can obtain input from nonexisting output.

Definition 6.6. (Test state.) *A* test state *is defined by*

$$\langle expr_{1,\mathbf{f}} \ \leq \ expr_{2,\mathbf{f}},\ g_1, \ldots, g_k;\ R \rangle \ \in \ \mathsf{States}.$$

When a variable is instantiated in an assignment or in a unification with a query, we must extend the definition of the concrete environments in a program. Let $e \in Env_{\mathbf{f}}$, and for a set $V_{\mathbf{f}}$ of variables, let $v \in V_{\mathbf{f}}$ be a variable which has been instantiated to a value $c \in \mathbb{F}_{\mathbf{f}}$. Then, we say that e is extended to v for value c and denote it $e[v \mapsto c]$. This clarification is important in the following definition.

Definition 6.7. (Assignment state transition.) *The* transition *from an assignment state to another state for a program* P *is defined for* $R, R' \in \wp(Env_{\mathbf{f}})$ *as*

$$\langle v \leftarrow expr_{\mathbf{f}}, g_1, \ldots, g_k;\ R \rangle \rightarrow_P \langle g_1, \ldots, g_k;\ R' \rangle,$$

where $R' \ = \ \{\ e \in Env_{\mathbf{f}} \ \mid \ \forall e \in R \ : \ \exists c \in [\![expr_{\mathbf{f}}]\!]_{fl}(e)\ .\ e[v \mapsto c]\ \}.$

Recall that our analysis studies logic programs with numerical atoms. Thus, a transition from an assignment state must reach another state with a numerical atom. Recall that each variable in a substitution can be instantiated only once.

Definition 6.8. (Test state transition.) *The* transition *from a test state to another state for a program P is defined for $\leq$ and for $R, R' \in \wp(Env_{\mathbf{f}})$ as*

$$\langle\ expr_{1,\mathbf{f}}\ \leq\ expr_{2,\mathbf{f}},\ g_1, \ldots, g_k;\ R\ \rangle\ \rightarrow_P\ \langle\ g_1, \ldots, g_k;\ R'\ \rangle,$$

where $R' = \{e \in Env_{\mathbf{f}} \mid \forall e \in R : \exists v_1 \in [\![expr_{1,\mathbf{f}}]\!]_{fl}(e)\ .\ \exists v_2 \in [\![expr_{2,\mathbf{f}}]\!]_{fl}(e) : v_1 \leq v_2\}$.

The *concrete semantics domain* is defined by the lattice

$$\widehat{C}\ =\ \langle \wp(\mathsf{States}), \subseteq, \cup, \cap, \emptyset, \mathsf{States}\rangle,$$

ordered by set inclusion, and equipped with set intersection and set union, and in which the bottom element is the empty set and the top element is the set of all states.

We define the concrete semantics of a logic program by means of a transition system, which is defined as a set of relations between states of a program.

6.3.2 Operational Semantics

The following definition and results are standard in operational semantics. However, we propose a reformulation that suits the context of our analysis, in which concrete environments as well as assignment and test predicates are central.

First we need the definition of an operator that *removes* from the definition of a concrete environment those variables that do not occur in the head atom of a clause. Then we define a mapping

$$proj_{\overline{\mathbf{x}}} : \wp_{\mathrm{fn}}(Env_{\mathbf{f}}) \rightarrow \wp_{\mathrm{fn}}(Env_{\mathbf{f}})$$

for $e \in Env_{\mathbf{f}}$ and $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq V_{\mathbf{f}}$, by means of the polyhedra existential quantifier operator; namely, for $R, R' \subseteq Env_{\mathbf{f}}$, $proj_{\overline{\mathbf{x}}}(R) = R'$ where $\forall e \in R$, $e(\overline{\mathbf{x}}) \in \mathbb{IR}^n$, $\exists e' \in R'$ where $e'(\overline{\mathbf{x}}) = \exists_{\overline{\mathbf{x}}}(e(\overline{\mathbf{x}}))$.

We define an immediate consequence operator over the complete lattice of powerset of states. Since such operator is monotonic and defined over a complete lattice, the least fixpoint exists.

Definition 6.9. (Immediate consequence operator.) *The* immediate consequence operator $\mathcal{F}^{\widehat{C}}_P : \wp(\mathsf{States}) \rightarrow \wp(\mathsf{States})$ *is defined over the complete lattice of powerset of states as*

$$\mathcal{F}^{\widehat{C}}_P(I)\ =\ \left\{\ \langle\ p(\overline{\mathbf{x}});\ R'\ \rangle\ \middle|\ \begin{array}{l} \forall\ \langle p(\mathbf{x}); R\rangle,\ \langle p_1(\mathbf{x}_1); R_1\rangle, \ldots, \langle p_k(\mathbf{x}_k); R_k\rangle\ \in\ I\ : \\ p(\mathbf{x}) \leftarrow p_1(\overline{\mathbf{x}}_1), \ldots, p_k(\overline{\mathbf{x}}_k)\ \in\ P\ \wedge \\ R'\ =\ R\ \cap\ \cap_{i=1}^{k}\ proj_{\overline{\mathbf{x}}_i}(R_i) \end{array}\ \right\}.$$

Since $\mathcal{F}_P^{\widehat{C}}$ is monotonic and defined over a complete lattice, the least fixpoint semantics exists and is defined as follows.

Definition 6.10. (Fixpoint semantics of a program P over $\widehat{C}$.) *The fixpoint semantics of a program P over $\widehat{C}$, denoted $\mathcal{F}^{\widehat{C}}(P)$ is defined as*

$$\mathcal{F}^{\widehat{C}}(P) \;=\; \mathrm{lfp}(\mathcal{F}_P^{\widehat{C}}).$$

The operational semantics of a program P over the concrete semantics domain $\widehat{C}$ is defined by means of the transition mapping $\rightarrow_P$ and its closure. We denote the closure of $\rightarrow_P$ by $\rightarrow_P^\star$.

Definition 6.11. (Operational semantics.) *The operational semantics of a program P over $\widehat{C}$ is defined on atomic goals as*

$$\mathcal{O}^{\widehat{C}}(P) \;=\; \{\langle\, p(\overline{\mathbf{x}});\; R\,\rangle \;\mid\; \langle\, p(\overline{\mathbf{x}});\; \mathsf{States}\,\rangle \;\rightarrow_P^\star\; \langle\, \triangledown;\; R\,\rangle\}.$$

In the definition above, the state $\langle \triangledown;\ R\rangle$ denotes that the empty program has been reached and that an answer substitution for the query to the program is available. The answer substitution can be obtained from R by mapping each variable of $\overline{\mathbf{x}}$ onto the co-domain of each $e \in R$.

The operational semantics is related to the fixpoint semantics as

$$\mathcal{O}^{\widehat{C}}(P) \;=\; \mathcal{F}^{\widehat{C}}(P).$$

6.4 Abstract Semantics

The analysis is carried out using the powerset of polyhedra since the possible occurrence of a division by zero should not halt the process. Alternatively it could be performed initially using the polyhedra domain and reformulating the abstraction into the powerset when a possible division by zero is detected. This last option would be an improvement in efficiency with respect to time and use of space, but the possible loss of precision while using the polyhedral domain could also result in an inconclusive analysis. In fact, violation of assertions would deliver the empty set, as our analysis cannot halt until the output point of a program $\triangledown$ is reached.

6.4.1 Definitions

We use abstract states to define the abstract semantics of a logic program. An alternative approach consists in annotating the abstract version of a logic program with assertions in the abstract domain, as done for instance in [KL02, KL03].

Definition 6.12. (Abstract program state.) *Let* $A \in \mathsf{Atoms}$ *and* $\kappa \in \wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n})$ *be a success pattern for* A*. Then an* abstract program state *is defined to be any pair of the form* $\langle A, \kappa \rangle$*.*

The set of abstract states is denoted by $\mathsf{States}^\sharp$. Abstract program states will be used to express conditions in the variables of the atom that hold after querying the atom.

Definition 6.13. (Abstract semantics domain.) *The* abstract semantics domain *is defined by the complete lattice* $\widehat{A}$ *of powerset of abstract program states as*

$$\widehat{A} \;=\; \langle\; \wp_{\mathrm{fn}}(\mathsf{States}^\sharp), \subseteq, \cup, \cap, \emptyset, \mathsf{States}^\sharp \;\rangle$$

where $\subseteq$ *is the set inclusion relation,* $\cup$ *is the set union operator and* $\cap$ *is the set intersection operator.*

6.4.2 Assignments

In this section we define the abstract state transition for assignments. In a (concrete) program state $\langle p(\overline{\mathbf{x}}), g_1, \ldots, g_k;\ R\rangle$, each input variable in p is instantiated to a value within the interval $[-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}]$.

Definition 6.14. (Abstract assignment state.) *An* abstract assignment state *is defined by*

$$\langle\; v \leftarrow expr_{\mathbf{f}},\ g_1, \ldots, g_k;\ \kappa \;\rangle \;\in\; \mathsf{States}^\sharp.$$

In the following definition, $\cap^\star$ denotes the polyhedra powerset meet operator, as defined in section 2.8.

Definition 6.15. (Abstract assignment state transition.) *The* transition *from an abstract assignment state to another abstract state for a program* P *is defined as*

$$\langle\; v \;\leftarrow\; expr_{\mathbf{f}}, g_1, \ldots, g_k;\ \kappa \;\rangle \Rightarrow_P \langle\; g_1, \ldots, g_k;\ \kappa \cap^\star \kappa' \;\rangle$$

for $\kappa, \kappa' \in \wp_{\mathrm{fn}}(\mathbb{CP}_n)$, $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$ *where*

$$E = \{e^\sharp \in AEnv_{\mathbf{f}} \mid \forall Q \in \kappa \;:\; e^\sharp(x_i) = \exists_{\mathsf{vars}(x)\setminus\{x_i\}} Q,\ i \in [1,n]\}$$

and

$$\kappa' \;=\; \{\; Q \in \mathbb{CP}_{n+1} \;\mid\; \forall e^\sharp \in E \;:\; Q = \uplus\, \pi([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp)) \;\wedge\; Q \subseteq [-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}]^{n+1} \;\}.$$

where the $n+1$ *dimension in the polyhedra corresponds to variable* v*.*

Recall that our analysis studies logic programs with numerical atoms. Thus the transition from an abstract assignment state reaches another state with a numerical atom.

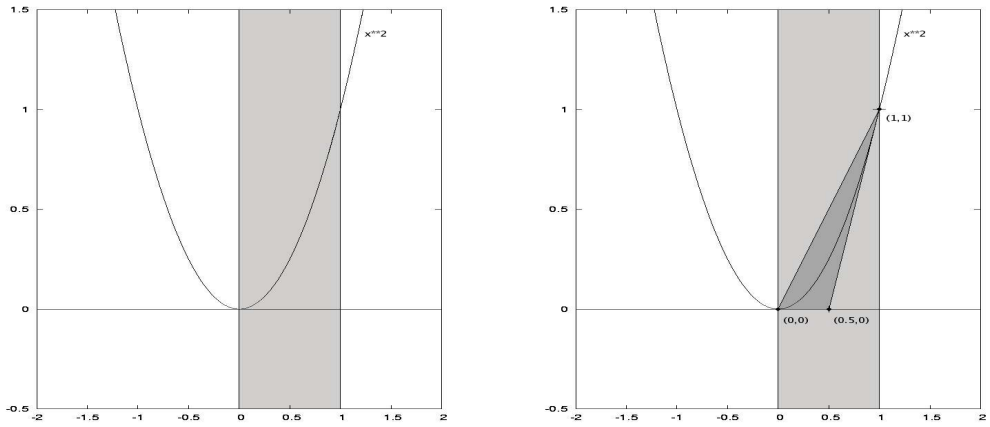


Figure 6.1: *Plot of input assertion and initial output assertion for* $y \leftarrow [1,1]x^2$ *(left); and upgrade of output assertion (darker gray) (right).*

We can be achieved higher precision in the computation of κ' by considering the recursive method. For some segmentation degree $m \geq 1$,

$$\kappa' \;=\; \{\; Q \in \mathbb{CP}_{n+1} \;\mid\; \forall e^\sharp \in E \;:\; Q \in \Upsilon^m_{e^\sharp}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp)) \;\wedge\; Q \subseteq [-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}]^{n+1} \;\}.$$

Moreover, we can extended the definition above for the case of expressions with possible divisions by zero. Thus, for a segmentation degree $m \geq 1$ and tree depth d,

$$\kappa' \;=\; \{\; Q \in \mathbb{CP}_{n+1} \;\mid\; \forall e^\sharp \in E \;:\; Q \in \Upsilon^{m,d}_{e^\sharp}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp)) \;\wedge\; Q \subseteq [-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}]^{n+1} \;\}.$$

Moreover, note that for $m = 1$ and $d = 0$,

$$\begin{aligned} \uplus\, \pi_{e^\sharp(\overline{\mathbf{x}})}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp)) \;&=\; \Upsilon^m_{e^\sharp}([p](\overline{\mathbf{x}})) \\ &=\; \Upsilon^{m,d}_{e^\sharp}([p](\overline{\mathbf{x}})). \end{aligned}$$

The following example illustrates the definition of an abstract assignment state for a floating-point expression that does not include polynomial quotients.

Example 6.16. *Let* $y \leftarrow expr_{\mathbf{f}}$ *be an assignment atom where for some* $e^\sharp \in AEnv_{\mathbf{f}}$*,*

$$[\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp) \;=\; [1,1]x^2.$$

Moreover, for $x \in \mathsf{vars}(expr_{\mathbf{f}})$ *let* $e^\sharp(x) = [0,1]$*. Then, by applying the Bernstein mapping on* $[1,1]x^2$ *we get the set of control points*

$$\pi_{e^\sharp(x)}([1,1]x^2) \;=\; \{\; (0,0),\; (0.5,0),\; (1,1) \;\}$$

which corresponds to the set of linear constraints

$$\begin{cases} -x + y \ \leq \ 0, \\ -y + 2 \cdot x \ \leq \ 0, \\ -y \ \leq \ 0. \end{cases}$$

The intersection of the input assertion with these linear constraints are the linear constraints themselves. Figure 6.1 plots the input, initial output, and upgraded output assertions.

6.4.3 Tests

In the case of test atoms, we assume that each variable is an input variable and that it is bounded by an interval included in $[-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}]$. Let $Q \in \mathbb{CP}_n$ be denoted by the set

$$\{(c_1, \ldots, c_n) \ \in \ \mathbb{R}^n \ \mid \ (c_1, \ldots, c_n) \ \in \ Q\}.$$

Moreover, let $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(Q)$, $\overline{\mathbf{x}} = (x_1, \ldots, x_n)$, and let $e^\sharp \in AEnv_{\mathbf{f}}$ be defined by

$$e^\sharp(x_i) \ = \ \exists_{\mathsf{vars}(\overline{\mathbf{x}}) \setminus \{x_i\}} \ Q, \quad i \in [1, n-1].$$

Then

$$Enc(Q, e^\sharp(\overline{\mathbf{x}})) \ = \ [\min(c_n), \ \max(c_n)]. \tag{6.1}$$

The definition that follows delivers those abstract environments for which the values that instantiate the variables make the test hold. This definition considers both linear and nonlinear tests.

Definition 6.17. (Abstract test state transition.) *The* transition *from an abstract test state to another abstract state for a program P is defined as*

$$\langle \ expr_{1,\mathbf{f}} \ \leq \ expr_{2,\mathbf{f}}, g_1, \ldots, g_k; \ \kappa \ \rangle \ \Rightarrow_P \ \langle \ g_1, \ldots, g_k; \ \kappa' \ \rangle,$$

where for $\kappa, \kappa' \in \wp_{\mathrm{fn}}(\mathbb{CP}_m)$, and $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{1,\mathbf{f}}) \cup \mathsf{vars}(expr_{2,\mathbf{f}})$, and $\overline{\mathbf{x}} = (x_1, \ldots, x_m)$,

$$\kappa' \ = \ \left\{ s^\sharp(\overline{\mathbf{x}}) \in \mathbb{CP}_m \ \middle| \ \begin{array}{l} \exists e^\sharp \in AEnv_{\mathbf{f}} \ . \ \forall Q \in \mathbb{CP}_m \ : \\ Q \in \kappa \ \wedge \ e^\sharp(x_i) = \exists_{\mathsf{vars}(\overline{\mathbf{x}}) \setminus \{x_i\}} Q, \ i \in [1, m] \ : \\ \forall Q' \ \in \ \Upsilon^{n,d}_{e^\sharp}([\![expr_{2,\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp) \ \boxminus^\sharp \ [\![expr_{1,\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp)) \ : \\ \forall s^\sharp \in AEnv_{\mathbf{f}} \ : \ s^\sharp(\overline{\mathbf{x}}) \subseteq e^\sharp(\overline{\mathbf{x}}) \ \wedge \\ Enc(Q', s^\sharp(\overline{\mathbf{x}})) = [a^-, a^+] \ \wedge \ a^- \geq 0 \end{array} \right\}.$$

6.4.4 Operational Semantics

In this section we compute the success patterns for logic programs with floating-point computations.

Definition 6.18. (Abstract fixpoint semantics operator.) *The* abstract fixpoint semantics operator $\mathcal{F}_P^{\wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n})} : \wp(\mathsf{States}^\sharp) \to \wp(\mathsf{States}^\sharp)$ *is defined over the complete lattice of powerset of abstract states as*

$$\mathcal{F}_P^{\wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n})}(A) \;=\; \left\{ \langle p(\overline{\mathbf{x}});\ \kappa' \rangle \in \wp_{\mathrm{fn}}(\mathsf{States}^\sharp) \;\middle|\; \begin{array}{l} \forall \langle p(\overline{\mathbf{x}}), \kappa \rangle \in A \;: \\ \forall \langle p_1(\overline{\mathbf{x}}_1);\ \kappa_1 \rangle, \ldots, \langle p_k(\overline{\mathbf{x}}_k);\ \kappa_k \rangle \in A \;: \\ p(\overline{\mathbf{x}}) \leftarrow p_1(\overline{\mathbf{x}}_1), \ldots, p_k(\overline{\mathbf{x}}_k) \in P \wedge \\ \kappa' \;=\; \kappa \;\cap^\star\; \cap^\star_{i \in [1,k]}\; \exists_{\overline{\mathbf{x}}_i}(\kappa_i) \end{array} \right\}.$$

Definition 6.19. (Abstract fixpoint semantics of a program P over $\wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n})$.) *The* abstract fixpoint semantics *of a program P over $\wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n})$, denoted $\mathcal{F}^{\wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n})}(P)$ is defined as*

$$\mathcal{F}^{\wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n})}(P) \;=\; \mathrm{lfp}(\mathcal{F}_P^{\wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n})}).$$

The convergence to a fixpoint can be accelerated with the use of widening operators for the polyhedra domain [CC77, CH78, Hal79] and widening operators for the finite powerset of the polyhedra domain [BHRZ03a].

Definition 6.20. (Abstract operational semantics.) *The* abstract operational semantics *of a program P is defined by the transitive closure of the abstract states transition on atomic goals as*

$$\mathcal{O}_P^{\wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n})} \;=\; \{\langle p(\overline{\mathbf{x}});\ \kappa \rangle \in \mathsf{States}^\sharp \;\mid\; \langle p(\overline{\mathbf{x}});\ \mathbb{R}^n \rangle \;\Rightarrow_P^\star\; \langle \nabla;\ \kappa \rangle\}$$

The abstract operational semantics soundly approximates the concrete operational semantics since each state transition in the concrete is soundly approximated by an abstract state transition. This soundness can be assured since the computation of assertions in each state is defined by means of the sound abstraction operators presented in chapters 4 and 5. Thus for the concrete and the abstract operational semantics for a logic program we have the following soundness result.

Theorem 6.21.

$$\forall \langle p(\overline{\mathbf{x}}); R \rangle \in \mathcal{O}_P^{\widehat{C}} \;:\; \exists \langle p(\overline{\mathbf{x}}); \kappa \rangle \in \mathcal{F}_P^{\wp_{\mathrm{fn}}(\mathbb{CP}_n)} \,.\, \forall e \in R \;:\; \{e(\overline{\mathbf{x}})\} \subseteq^\star \kappa.$$

Theorem 6.22.

$$\mathcal{O}^{\wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n})}(P) \;=\; \mathcal{F}^{\wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n})}(P).$$

6.5 Detecting Floating-Point Overflow

In chapter 3, we defined floating-point overflow for floating-point arithmetic as the result of an operation which has no representation in the underlying floating-point system; namely, for a floating-point system of format $\mathbf{f}$, those values not included in the interval $[-Mf_{\mathbf{f}}, -Mf_{\mathbf{f}}]$ overflow, and are denoted by Ω. In this section we generalise the meaning of the term *overflow*. Informally, we say that the result of evaluating a floating-point expression overflows if it is larger or smaller than preset values.

6.5.1 Absolute and Relative Overflow

For n variables (in both input and output positions) in the query state atoms, we define a bounding box $\mathbf{B}_{\mathsf{abs}} \in \mathbb{IR}^n$, where

$$\mathbf{B}_{\mathsf{abs}} \;=\; [-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}] \times \ldots \;(k \; times) \;\ldots \times [-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}].$$

We call $\mathbf{B}_{\mathsf{abs}}$ the *absolute overflow bounding box*. Moreover, we can set constraints on the variables in the query state atoms, which bound the possible values each variable can take. For this case we define a bounding box $\mathbf{B}_{\mathsf{rel}} \in \mathbb{IR}^n$, where

$$\mathbf{B}_{\mathsf{rel}} \;=\; [r_1^-, r_1^+] \times \ldots \;(k \; times) \;\ldots \times [r_n^-, r_n^+].$$

We call $\mathbf{B}_{\mathsf{abs}}$ the *relative overflow bounding box*. Note that $[r_i^-, r_i^+] \subseteq [-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}]$, for $i \in [1, n]$. A formal definition of overflow for our analysis is presented below.

Definition 6.23. (Absolute overflow.) *We say that a floating-point expression $expr_{\mathbf{f}}$, where $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$, overflows for an abstract environment $e^\sharp \in AEnv_{\mathbf{f}}$ if*

$$Enc([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp), e^\sharp(\overline{\mathbf{x}})) \;\not\subseteq\; [-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}].$$

In our analysis, absolute overflow refers to the overflow inherent in a floating-point system of format $\mathbf{f}$, in which the result of evaluating a floating-point expression is too large or too small to be represented in format $\mathbf{f}$. We consider another class of overflow, which is relative to a given interval. The boundaries of the relative overflow interval are representable in the floating-point system in use. A formal definition of relative overflow is as follows.

Definition 6.24. (Relative overflow.) *We say that a floating-point expression $expr_{\mathbf{f}}$, where $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$, overflows for an abstract environment $e^\sharp \in AEnv_{\mathbf{f}}$ relative to an interval $[r^-, r^+] \in \mathbb{IR}$ where $[r^-, r^+] \subseteq [-Mf_{\mathbf{f}}, Mf_{\mathbf{f}}]$ if*

$$Enc([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp), e^\sharp(\overline{\mathbf{x}})) \;\not\subseteq\; [r^-, r^+].$$

The relative overflow is useful to analyse programs with constraints on the variables. Both classes of overflow can be extended to every variable in an expression; namely, for an expression $expr_{\mathbf{f}}$ where $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$, $\overline{\mathbf{x}} \in \mathbb{R}^k$, we specify the absolute overflow of $expr_{\mathbf{f}}$ by $\mathbf{B}_{\mathbf{f},\mathsf{abs}} \in \mathbb{IR}^k$ and the relative overflow of $expr_{\mathbf{f}}$ by $\mathbf{B}_{\mathbf{f},\mathsf{abs}} \in \mathbb{IR}^k$.

Example 6.25. *In example 6.16, we can constrain variable y by $y \in \mathbf{B}_{\mathbf{f},\mathsf{rel}} = [0, 1/2]$. Note that*

$$Enc([\![expr_{\mathbf{f}}]\!]^{\sharp}_{fl}(e^{\sharp}), e^{\sharp}(x)) \ \subseteq \ [0, 1],$$

and thus $[0, 1/2] \not\subseteq [0, 1]$, which indicates a relative overflow for y.

Note that the analysis for relative overflow is different to the analysis for precision, which investigates the difference between the expected result and the actual result (for works on floating-point arithmetic precision analysis see for instance [Gou01] and [GMP01]). The analysis for relative overflow verifies that the possible values taken by the output variables of a program are consistent with a set of constraints of such variables.

Moreover, note that in the analysis presented in this chapter, we use the empty set to denote that any input to a given floating-point expression overflows (absolute overflow in our context) or divides by zero (which is a cause of absolute overflow). However, both in the case of absolute and relative overflow, parts of the polyhedra may be within the representable or allowed limits. In such a case, we are interested in retrieving such *nonoverflowing* parts. Our approach is simple: for the case of absolute overflow we intersect each polyhedron in the powerset, defined over $\mathbb{R}^k$, with $\mathbf{B}_{\mathbf{f},\mathsf{abs}} \in \mathbb{IR}^k$ and for the case of relative overflow, we intersect each polyhedron in the powerset with $\mathbf{B}_{\mathbf{f},\mathsf{rel}} \in \mathbb{IR}^k$. Whenever the intersection does not include the element of the powerset, then such element is discarded as it may cause an overflow. The case of absolute overflow was formalised in definitions 6.15 and 6.17 for abstract assignment and test state transitions, respectively. For the case of relative overflow, such definitions can be easily reformulated by replacing $\mathbf{B}_{\mathbf{f},\mathsf{abs}}$ by $\mathbf{B}_{\mathbf{f},\mathsf{rel}}$.

The output of the forward analysis is used to perform a backward analysis. The backward analysis assumes that there may be possible overflow (such as those caused by division by zero), but it must deliver conditions of the input which do not produce any floating-point run-time error. The worst case scenario is that all input to a program can cause a floating-point run-time error, in which case both the forward and the backward analyses deliver an empty set. Consider example 5.20 in which it is abstracted a quotient of interval valued polynomials into a finite set of polyhedra. In that case, none of the polyhedra includes zeroes and thus no overflow occurs; namely, the forward analysis computes only those polyhedra that do not include zeroes. Thus, when analysing a clause with more than one assignment or test, the powerset meet operator $\cap^{\star}$ of any two polyhedron that do not include any point that overflows, delivers a set of polyhedra that also do not include any point that overflows. Moreover, this also holds for the poly-hull operator.

These polyhedra are then used by the backward analysis to infer conditions in the input variables. None of these polyhedra would be considered by the forward analysis if they included some zero. In that case, the output of the forward analysis would be an empty set, which would lead the backward analysis to deliver an empty set as well.

6.5.2 Intermediate Overflow

Subexpressions of a floating-point expression may also overflow. Thus, we refer to intermediate overflow as the overflow in a subexpression. In order to detect intermediate overflow, an expression is broken down into subexpressions, which undergo a separate analysis each.

To illustrate this, consider the following program for the computation of the amplitude of energy density; namely,

```
density(X,Y, Result) :-
        Result is sqrt(X * X + Y * Y).
```

Assume double precision in the floating-point system. Note that in the assignment atom, for `X = 1e300`, an intermediate overflow may occur, since

$$10^{300} \cdot 10^{300} \gg Mf_{\mathbf{f}} = 10^{308}.$$

The analysis computes the set of polyhedra for which `X*X`, `Y*Y`, and `X*X+Y*Y` do not overflow. For this purpose, it is defined three new assignments and a forward analysis on each is performed, in order to detect possible overflow. Section 8.3 explains in detail how the analysis processes sets of polyhedra for expressions with the square root function using the concept of logarithmic segmentation.

6.6 Discussion and Related Work

Analyses of Logic Programs with Floating-Point Arithmetic. A related line of research that involves logic programming and run-time errors due to floating-point arithmetic is presented in [SS02, SM04]. These works deal with the problem of termination in logic programs with floating-point arithmetic. It is shown that for certain settings in the floating-point rounding method, a query to a program may not terminate. This work is closely related also to the issue of fixpoint convergence of programs, which is central in static analysis. Intuitively, a program that does not converge may not terminate.

Invariants. The study of nonlinear invariants for programs with floating-point computations is important for the analysis of reactive systems. For this purpose, Sankaranarayanan [San05] provides techniques for finding (possibly) nonlinear invariants in imperative programs with floating-point computations. This work is founded in abstract

interpretation, and uses weakly relational domains, which avoid exponential space complexity. However, no specific treatment of rounding errors is considered. In [BRCZ05], techniques for inferring polynomial invariants are presented. These techniques are useful for the verification of numerical properties of programs that cannot otherwise be verified with linear invariants. However, no specific treatment of floating-point arithmetic is considered in this work either. In our analysis, we work with linear invariants for the analysis of programs of nonlinear expressions, although at the high space cost of using the finite powerset of polyhedra. Thus, we trade analysis cost for precision, as well as an analysis general enough to cope with a wide class of nonlinear expressions.

Chapter 7

Backward Relational Analysis for Constraint Logic Programs

In this chapter we present a backward analysis for the inference of preconditions in the input variables of a logic program to avoid floating-point run-time overflow. We instantiate the backward analysis presented in [KL02] and further extended in [KL03] to the powerset of polyhedra. However, we show that the powerset of polyhedra does not require a definition of a logical model as it does in [KL03]. Moreover, we synthesize a universal quantifier operator from which it can be defined a Galois connection that ensures the correctness of the analysis.

7.1 Introduction

Backward analysis propagates constraints from the output of a program toward the input in order to infer which conditions must hold in the input to obtain a *safe* output. That is to say, for a given postcondition, backward analysis infers the preconditions that ensures the postcondition holds. Backward analysis for logic programs has been developed, for instance, for safe calls inference [KL02, KL03], for type inference [LK02], for set sharing inference [LK04, LL05], for suspension analysis [GK03], and for termination inference [GC01].

In this chapter we propose a backward analysis for the inference of preconditions in the input to a program from which it can be ensured that no floating-point overflow can

occur whenever the inferred preconditions are satisfied; that is to say, for the inference of nonoverflow. This backward analysis is useful, for instance,

- for prototype development; namely, for ensuring that the implementation of a model for a physical system, the coding of the model parameters, and input to the prototype program do not *conflict* with the underlying floating-point system;
- for code reusability; namely, for reusing mathematical models for physical system, where the composition of models is mathematically sound but the composition of implemented components may not be;
- for *defensive programming* development; namely, for protecting critical parts in the code that have been left unprotected against potentially hazardous input, and where the inference of *safe* preconditions in large or complex code can be a tedious or even impractical task if delegated to the programmer.

In the design of a backward analysis, two approaches can be followed: the design by abstract interpretation on a backward collecting semantics (from *scratch*); or by inverting an already designed forward analysis. The second approach in general reduces the design effort as the correctness of the backward analysis can be determined from the correctness of the forward analysis [HL94, KL03]. In the backward analysis presented here, we use the outcome of the forward analysis presented in chapter 6. We follow *King's & Lu* backward framework [KL02, KL03], which requires the abstract domain to include a pseudo-complement operator that guarantees the existence of the maximal preconditions that does not violate a given postcondition. Our analysis is designed for the powerset of the polyhedra domain. Moreover, following [KL03], we define existential and universal quantifier operators for the powerset of the polyhedra domain from which we can guarantee the correctness of the analysis.

In particular the backward analysis presented here can be instantiated to infer conditions for either absolute or relative overflow. This is achieved by parameterising the initial element in the lattice from which the backward inference starts; that is to say, whether the initial element corresponds to $\mathbf{B}_{\text{abs}}$ or $\mathbf{B}_{\text{rel}}$, as defined in chapter 6.

Our contribution in this chapter is the introduction of a backward analysis for the inference of numerical properties in logic programs that compute floating-point arithmetic operations. The analysis infers preconditions to avoid floating-point overflow, such as for instance that caused by division by zero. Central in this analysis is the use of the powerset of polyhedra, for which a logical model is defined, which makes it appropriate for use in *King's & Lu* backward framework. Moreover, we present an application for the inference of preconditions for safe motion of robots.

7.2 Pseudo-Complementation in Polyhedra Domains

The backward analysis framework presented by *King's & Lu* relies on the existence of a pseudo-complementation operator in the abstract domains. Using this operator we can obtain the maximal preconditions that must hold any success call to an atom. First we introduce the basic concepts needed to develop *King's & Lu* backward framework. Then we study two domains of interest: namely the polyhedra domain and the powerset of polyhedra domain. We show that both domains are equipped with a pseudo-complement operator and thus are appropriate for backward analysis.

7.2.1 Definitions

For a given lattice $\langle D, \leq, \vee, \wedge \rangle$, the pseudo-complement operator, if it exists in $\widehat{D}$, is defined by the join and meet operators of $\widehat{D}$ [GPR96, Ran02]; namely, for $d_1, d_2 \in D$, the pseudo-complement of d_1 relative to d_2, denoted by $wr(d_1, d_2)$, is a mapping $wr : D \to D$ defined as

$$wr(d_1, d_2) \;=\; \vee \{d \in D \mid d \wedge d_1 \leq d_2\}.$$

We say that a domain $\widehat{D}$ is completely meet-distributive if $\forall a, b_1, \ldots, b_n \in D$, $n \in \mathbb{N}$,

$$a \;\wedge\; (\vee_{i=1}^{n}\, b_i) \;=\; \vee_{i=1}^{n}\, (a \;\wedge\; b_i).$$

A domain that is completely meet-distributive implies that it is a complete Heyting algebra [Bir67]. This in turn implies that the domain is equipped with a pseudo-complement operator [GS98]. Moreover, condensing domains [MS93, GRS05] are equipped with a pseudo-complement operator. In addition, the powerset of any domain is equipped with a pseudo-complement operator.

7.2.2 Pseudo-Complement Operators

We define the pseudo-complement of polyhedra relative to another polyhedra through the concept of maximal weakest polyhedra, as follows.

Definition 7.1. (Maximal weakest polyhedra.) *For $P, Q \in \wp_{\mathrm{fn}}(\mathbb{CP}_n)$, the* maximal weakest polyhedra *of P relative to Q, denoted $wr^{\star}(P, Q)$, is defined by the polyhedra $C \in \wp_{\mathrm{fn}}(\mathbb{CP}_n)$ where*

$$C \;=\; \cup \left\{ D \in \mathbb{CP}_n \;\mid\; P \;\cap^{\star}\; \{D\} \;\subseteq^{\star}\; Q \right\}.$$

In [KL03], the pseudo-complement operator is defined through the negation of constraints in Lin_X. Our approach is different. We define a pseudo-complement powerset operator for the powerset of bounded convex polyhedra, which in fact is a meet powerset

operator. First, recall that the polyhedra powerset meet operator is defined as a mapping $\cap^\star : \wp(\mathit{Hull}) \times \wp(\mathit{Hull}) \to \wp(\mathit{Hull})$ where

$$\{s_1, \ldots, s_m\} \ \cap^\star \ \{t_1, \ldots, t_n\} \ = \ \Omega^{\subseteq}_{\mathbb{CP}_n}\big(\{s_1 \cap t_1, \ldots, s_1 \cap t_n, \ldots, s_m \cap t_1, \ldots, s_m \cap t_m\big)\}.$$

That is to say, the finite polyhedra powerset meet operator is defined as a finite set of maximal weakest polyhedra.

Definition 7.2. (Pseudo-complement operator for the powerset of polyhedra.) *The* pseudo-complement operator for the powerset of polyhedra *is defined as*

$$wr^\star(\{s_1, \ldots, s_m\}, \{t_1 \ldots, t_n\}) \ = \ \{s_1, \ldots, s_m\} \ \cap^\star \ \{t_1 \ldots, t_n\}.$$

7.3 Backward Fixpoint Semantics

In this section we present the backward fixpoint semantics for the inference of preconditions on queries to clauses in a logic program with floating-point computations. First we define the concept of call patterns with is central in specifying preconditions. Then we define quantifier operators, which are central in *King's and Lu* backward framework. From these concepts we can then present a parametrised operator for the inference of preconditions. This operator is parametrised on the initial element in the lattice of powerset of polyhedra from which the inference starts. Results on correctness of the analysis follows.

7.3.1 Definitions

Here we formalise the concept of precondition under the term of *call pattern*. First, recall that an abstract program state that includes a postcondition $f \in \wp_{\mathrm{fn}}(\mathbb{CP}_n)$ is denoted by

$$\langle\ p(\overline{\mathbf{x}});\ f\ \rangle_{\mathrm{b}} \ \in \ \mathsf{States}^\sharp.$$

The concept of success pattern has been introduced chapter 6. Here we formalise this concept as follows.

Definition 7.3. (Success pattern.) *We say that* $f \in \wp_{\mathrm{fn}}(\mathbb{CP}_n)$ *describes a* success pattern *for an atom* $p(\overline{\mathbf{x}})$ *if*

$$\langle p(\overline{\mathbf{x}});\ \mathbb{R}^n\rangle \ \Rightarrow^\star_P \ \langle \triangledown;\ f\rangle.$$

where $\{\mathbb{R}^n\}$ *is the top element of* $\wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n})$.

Definition 7.4. (Call pattern.) *We say that* $d \in \wp_{\mathrm{fn}}(\mathbb{CP}_n)$ *describes a* call pattern *for an atom* $p(\overline{\mathbf{x}})$ *if*

$$\langle p_i(\overline{\mathbf{x}}_i);\ d\rangle \ \not\Rightarrow^\star_P \ \langle \triangledown;\ \emptyset\rangle$$

where $\{\mathbb{R}^n\}$ *is the top element of* $\wp_{\mathrm{fn}}(\widehat{\mathbb{CP}_n})$.

In our analysis, the empty set $\emptyset$ is considered as a call pattern where there exists no values for the input variables of a query that can succeed. We denote an abstract program state for an atom $p(\overline{\mathbf{x}})$ which includes a precondition by

$$\langle\ p(\overline{\mathbf{x}});\ d\ \rangle_{\mathrm{b}}\ \in\ \mathsf{States}^{\sharp}.$$

7.3.2 Quantification Operators

Central in *King's and Lu* backward framework is the definition of a universal quantifier operator, by which it can be guaranteed the correctness of the analysis. Thus, we must engineer a universal quantifier operator for the powerset of polyhedra, which is monotonic and additive. If such properties hold then we can define a Galois connection as in [KL03] that ensures the correctness of the analysis. Note that the universal quantifier operator for powerset of polyhedra can be defined by means of the existential quantifier operator for the same domain. The existential operator for the powerset can be defined in turn as an extension of the existential operator for the polyhedra domain, by applying the latter operator to each element in the powerset.

In [KL03, example 6], a definition of such operators is provided for the polyhedral domain Lin_X [LS97]. The generalisation of the operators for the full polyhedra domain is straightforward. First, we need the concept of free variable, defined as follows. For a linear constraint of the form $\sum_{v\in V} a_v v \geq 0$, we denote by

$$fr(\sum_{v\in V} a_v v \geq 0)$$

the set of free variables in the constraint; namely, the set of variables whose coefficients are different to zero:

$$fr(\sum_{v\in V} a_v v \geq 0)\ =\ \{v \in V\ \mid\ \forall a_v \in \mathbb{R}\ :\ a_v \neq 0\},$$

Let ξ_P denote a set of linear constraints of the form $\sum_{v\in V} a_v v \geq 0$ that define an element $P \in \mathbb{CP}_n$. Then,

$$fr(\xi_P)\ =\ \cup_{\ell\in\xi_P}\ fr(\ell),$$

where $\cup$ is the set union operator. Then the existential quantifier operator is defined as a mapping $\exists_v : \mathbb{CP}_m \to \mathbb{CP}_n$ where

$$\exists_v(P)\ =\ \{e \in \xi_P\ \mid\ \forall e \in \xi_P\ :\ v \notin fr(e)\}$$

That is to say, those constraints that define P and that do not include a variable v are removed from the set ξ_P of constraints. Moreover, the universal quantifier operator is

defined as a mapping $\forall_v : \mathbb{CP}_n \to \mathbb{CP}_n$ where

$$\forall_v(P) \;=\; \begin{cases} \xi_P & \text{if } v \notin fr(\xi_P) \\ \emptyset & \text{otherwise.} \end{cases}$$

That is to say, the operator delivers P if every constraint in ξ_P does not include variable v; otherwise, the operator delivers the bottom element of the lattice, namely, the empty set.

Moreover, we can extend the definition of these operators to the case of the powerset of polyhedra; namely, the existential quantifier powerset operator is a mapping $\exists_v^\star : \wp_{\text{fn}}(\mathbb{CP}_n) \to \wp_{\text{fn}}(\mathbb{CP}_n)$ defined for a set of variables V, where $v \in V$, as

$$\exists_v^\star(Q) \;=\; \{\exists_v(P) \in \mathbb{CP}_n \;\mid\; \forall P \in \mathbb{CP}_n \;:\; P \in Q\}$$

The universal quantifier powerset operator is a mapping $\forall_v^\star : \wp_{\text{fn}}(\mathbb{CP}_n) \to \wp_{\text{fn}}(\mathbb{CP}_n)$ defined for a set of variables V, where $v \in V$, as

$$\forall_v^\star(Q) \;=\; \{\forall_v(P) \in \mathbb{CP}_n \;\mid\; \forall P \in \mathbb{CP}_n \;:\; P \in Q\}.$$

Both the existential and universal quantifier operators are monotonic and both project a variable away from a set of constraints. The main difference between them, as noted in [KL03], is the direction of approximation; namely, for $P \in \mathbb{CP}_n$ defined for a set V of variables,

$$\forall_v(P) \;\subseteq\; P \;\subseteq\; \exists_v(P).$$

Moreover, for $Q \in \wp_{\text{fn}}(\mathbb{CP}_n)$,

$$\forall_v(Q) \;\subseteq^\star\; Q \;\subseteq^\star\; \exists_v(Q).$$

This observation is important to show the correctness of the backward analysis.

7.3.3 Backward Operators

In order to define a backward fixpoint semantics for logic programs that compute floating-point arithmetic, we must first present two particular backward operators. One of these operators delivers call patterns for assignment atoms; the other delivers call patterns for test atoms. Informally, a backward operator for assignments infers which are the possible values that can instantiate the variables in an assigner expression, so that the assignee variable is instantiated to those values inferred by a forward semantics operator. A backward operator for tests infers which are the possible values that can instantiate the variables in the expressions for which the test holds.

Definition 7.5. (Backward semantics for assignments.) *The backward semantics*

for an assignment $v \leftarrow expr_{\mathbf{f}}$ *where* $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{\mathbf{f}})$, $\overline{\mathbf{x}} = (x_1, \ldots, x_m)$, *is defined as*

$$\mathcal{D}^{\wp_{\mathrm{fn}}(\mathbb{CP}_m)}_{v \text{ is } expr_{\mathbf{f}}} = \left\{ \langle v \leftarrow expr_{\mathbf{f}};\ \mathbf{B_f} \rangle_{\mathrm{b}} \in \mathsf{States}^\sharp \;\middle|\; \begin{array}{l} \exists e^\sharp \in AEnv_{\mathbf{f}} \,.\, e^\sharp(\overline{\mathbf{x}}) \subseteq \mathbf{B_f} \,. \\ \forall Q \in \mathbb{CP}_{m+1} \ :\ Q \in \Upsilon^{n,d}_{e^\sharp}([\![expr_{\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp)) \wedge \\ \mathbf{B_f} = \exists_v Q \end{array} \right\},$$

where the $m+1$ *dimension in the polyhedra corresponds to variable* v *and*

$$\mathbf{B_f} \in \{\mathbf{B}_{\mathbf{f},\mathsf{abs}}, \mathbf{B}_{\mathbf{f},\mathsf{rel}}\}.$$

In the definition that follows, it is important to recall expression 6.1 in page 96.

Definition 7.6. (Backward semantics for inequality tests.) *The backward semantics for a test* $expr_{1,\mathbf{f}} \leq expr_{2,\mathbf{f}}$ *where* $\mathsf{vars}(\overline{\mathbf{x}}) \subseteq \mathsf{vars}(expr_{1,\mathbf{f}}) \cup \mathsf{vars}(expr_{2,\mathbf{f}})$ *and* $\overline{\mathbf{x}} = (x_1, \ldots, x_m)$ *is defined by*

$$\mathcal{D}^{\wp_{\mathrm{fn}}(\mathbb{CP}_m)}_{expr_{1,\mathbf{f}} \leq expr_{2,\mathbf{f}}} = \left\{ s^\sharp(\overline{\mathbf{x}}) \in \mathbb{CP}_m \;\middle|\; \begin{array}{l} \exists e^\sharp \in AEnv_{\mathbf{f}} \,.\, \forall Q \in \mathbb{CP}_m \ :\ Q \in \mathbf{B_f} \wedge \\ e^\sharp(x_i) = \exists_{\mathsf{vars}(\overline{\mathbf{x}}) \setminus \{x_i\}} Q,\ i \in [1,m] \ : \\ \forall Q' \in \Upsilon^{n,d}_{e^\sharp}([\![expr_{2,\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp) \boxminus^\sharp [\![expr_{1,\mathbf{f}}]\!]^\sharp_{fl}(e^\sharp)) \ : \\ \forall s^\sharp \in AEnv_{\mathbf{f}} \ :\ s^\sharp(\overline{\mathbf{x}}) \subseteq e^\sharp(\overline{\mathbf{x}}) \wedge \\ Enc(Q', s^\sharp(\overline{\mathbf{x}})) = [a^-, a^+] \ \wedge\ a^- \geq 0 \end{array} \right\}.$$

where

$$\mathbf{B_f} \in \{\mathbf{B}_{\mathbf{f},\mathsf{abs}}, \mathbf{B}_{\mathbf{f},\mathsf{rel}}\}.$$

We define a backward fixpoint semantics operator that uses the existential and universal quantifier operators, which are monotonic, reductive and extensive. This ensures the correctness of the analysis, as shown in [KL03]. Moreover, this operator is parameterised with the initial element in the lattice from which the inference starts; namely, for instance $\mathbf{B}_{\mathbf{f},\mathrm{abs}} \in \mathbb{R}^k$ or $\mathbf{B}_{\mathbf{f},\mathrm{rel}} \in \mathbb{R}^k$ for absolute or relative overflow, respectively.

Definition 7.7. (Parametric backward fixpoint semantics operator.) *Let* P *be a constraint logic program over* $\mathbb{CP}_n$. *Then operator* $\mathcal{B}^{\mathbb{CP}_n}_P Int \to Int$ *is defined by*

$$\mathcal{B}^{\wp_{\mathrm{fn}}(\mathbb{CP}_n)}_{P,\mathbf{B_f}}(\mathcal{D}^{\wp_{\mathrm{fn}}(\mathbb{CP}_n)}_P) = \left\{ \langle p(\overline{\mathbf{x}});\ e \rangle_{\mathrm{b}} \in \mathsf{States}^\sharp \;\middle|\; \begin{array}{l} \forall \{\ \langle p_i(\overline{\mathbf{x}}_i);\ f_i \rangle_{\mathrm{f}}\}^m_{i=1} \subseteq \mathcal{F}^{\wp_{\mathrm{fn}}(\mathbb{CP}_n)}_P \,. \\ \forall \{\ \langle p_i(\overline{\mathbf{x}}_i);\ d_i \rangle_{\mathrm{b}}\}^m_{i=1} \subseteq \mathcal{D}^{\wp_{\mathrm{fn}}(\mathbb{CP}_n)}_P \,. \\ \exists p(\overline{\mathbf{x}}) \leftarrow c,\ p_1(\overline{\mathbf{x}}_1), \ldots, p_m(\overline{\mathbf{x}}_m) \in P \ : \\ e_{m+1} = \{\mathbf{B_f}\} \ \wedge\ e_i = d_i \cap^\star (f_i \cap^\star e_{i+1}) \wedge \\ e \subseteq \forall^\star_{\mathsf{vars}(\overline{\mathbf{x}})}(e_0) \ \wedge\ e_0 = wr^\star(c, e_1) \end{array} \right\},$$

where $\mathcal{F}_P^{\wp_{\mathrm{fn}}(\mathbb{CP}_n)}$ *is the set of success patterns of the program and where*

$$\mathbf{B_f} \;=\; \{\mathbf{B_{f},}_{abs},\; \mathbf{B_{f},}_{rel}\}.$$

In the definition above, $\mathcal{D}_P^{\wp_{\mathrm{fn}}(\mathbb{CP}_n)}$ denotes the set of *initial* call patterns, which are defined for assignments and tests, and for every other class of predicate. Such a set is defined by polyhedra intersection with either $\mathbf{B_{f},}_{\mathrm{abs}}$ or $\mathbf{B_{f},}_{\mathrm{rel}}$, following initial constraints required in the variables of each head atom. Moreover, d_i represents the conditions that must hold $p_i(\overline{\mathbf{x}}_i)$ to avoid overflow errors when it is queried; moreover, $e_i \subseteq^\star d_i$, which ensures that the sub-goal $p_{i+1}(\overline{\mathbf{x}}_{i+1}), \ldots, p_n(\overline{\mathbf{x}}_n)$ does not cause an overflow. If the conditions represented by e_i hold, then no overflow occur when querying $p_i(\overline{\mathbf{x}}_i), \ldots, p_n(\overline{\mathbf{x}}_n)$.

An important result to show the correctness of a backward framework for logic programs verification is the relationship between quantifier operators found in [KL03]. King and Lu found that a forward and backward verification mechanism has the same verification strength, since it can be constructed a Galois connection using the abstract domain and the quantifier operators. Moreover, King and Lu noted that such operators are monotonic, reductive ($\exists_x$) and extensive ($\forall_x$), and thus it is possible to define a Galois connection. To see this note first that

$$\exists_x^\star \;\circ\; \forall_x^\star(P) \;\subseteq^\star\; P \;\subseteq^\star\; \forall_x^\star \;\circ\; \exists_x^\star(P)$$

That is to say, $\forall_x^\star \circ \exists_x^\star$ is extensive and $\exists_x^\star \circ \forall_x^\star$ is reductive. Moreover, $\exists_x$ and $\forall_x$ are monotonic. Thus,

$$\langle \mathbb{CP}_n, \forall_x, \mathbb{CP}_n, \exists_x \rangle$$

is a Galois connection. Hence by theorem 5 in [KL03],

$$\langle \wp_{\mathrm{fn}}(\mathbb{CP}_n), \forall_x^\star, \wp_{\mathrm{fn}}(\mathbb{CP}_n), \exists_x^\star \rangle$$

is a Galois connection. That is to say, both semantics have the same verification power.

7.4 Discussion and Related Work

Our analysis is particular in that it focuses on inferring numerical preconditions in queries to a program. If such preconditions are satisfied, then floating-point overflows are avoided. In order to infer numerical properties, we use the finite powerset of polyhedra. Our backward analysis could be easily adapted to use octagons, by defining the corresponding meet operator for powerset. This could lead to more efficient backward analyses than the analysis presented here for polyhedra. A limitation of our approach is that in general we cannot calculate invariants for nonlinear expressions. Chapter 6 discusses related work on computation of invariants of nonlinear expressions.

Chapter 8

Applications and Experimental Results

8.1 Introduction

In this chapter we apply the analysis techniques developed for the analysis of logic programs with nonlinear floating-point expressions. We present an application for designing robot trajectories and an application for designing satellite orbits.

The analysis is implemented in C++, which allows for encapsulation and scalability, as well as efficiency. Using C++ allows moreover for using multiprecision arithmetic libraries such as GMP [1]. The GMP library provides a series of functions defined over the rationals, the implementation of which is sound. This is important for implementing the Bernstein mappings. Moreover, the analysis uses the Parma Polyhedra Library (PPL) [BRZH02,BHRZ03b] for computing operations related to the polyhedra and polyhedra powerset domains. This library in turn uses the GMP library. Although most existing libraries for the manipulation of convex polyhedra use rational coefficients (namely for instance Polka [Jea04, HPR97]), two exceptions can be found: the CDD library [Fuk01], which uses floating-point arithmetic and is not except of rounding errors, and the analyser in [Min04b], which employs interval floating-point arithmetic soundly to analyse floating-point arithmetic itself. The implementation of the Bernstein mappings and subsequent forward and backward analyses follows very closely the definitions presented in chapters 3, 4 and 5.

The C++ code was compiler with the GNU gcc 3.2.2 and the experiments were con-

[1] GMP is a GNU Multiple Precision Library. The home page can be found at http://www.swox.com/gmp

ducted in a ThinkPad X20 (600Mhz, Pentium III, 20Gb, 192MB of memory).

8.2 Robot Trajectory Verification

8.2.1 Introduction

The verification of a trajectory of a parallel robot before using the machine for instance for manufacturing operations, can be of critical importance. The verification of a trajectory commonly consists in checking whether a set of *validity criteria* holds for the given machine specifications. This includes for example the verification of whether any possible trajectory lies within the reachable workspace of a robot or whether the physical limitations of the machine such as maximum leg length are satisfied. The verification of the robot geometry has been addressed in [GPV95, CWMM04]. An algorithm for checking the validity of a trajectory has been proposed in [Mer94]. Extensions of this latter work using interval analysis have been proposed in [Mer01, Mer04], where complex trajectories with possible nondeterministic floating-point rounding errors are considered. However, the floating-point rounding errors are assumed to be bounded by some pre-established ranges.

In our approach, we first implement the mathematical model for the robot motion trajectory through a logic program prototype. This implementation is in fact a direct *transcription* of the model and *captures* the control flow as well as possible problems that can be found in the machine-dependent floating-point system used. Moreover, we can specify valid ranges for variables. This machine dependency is important when scaling the prototype to a larger nonprototypical implementation.

8.2.2 Prototyping in Logic Programming

The following program is a prototype for verifying that the leg's length of a robot is bounded by a maximum length over a period of time. The problem at hand is to verify whether the leg's length will be bounded by for any allowed period of time. The analysis treats this case as a relative overflow verification problem.

```
jointTrajectory(Time) :-
    Time >= 0,
    Time1 is Time + 1/100,
    Time1 <= 4,
    Length is 2 * Time - Time * Time + sin(Time) / 3,
    Length <= 1.25,
    jointTrajectory(Time1).

jointTrajectory(Time) :-
    write('Error: maximum length surpassed.'),
    !, fail.
```

The functor `sin/1` in the program above corresponds to the trigonometric function sinus and is defined for radians. Moreover, this program is well-moded since each atom

has available all the input before being queried.

8.2.3 Adaptive Method

In practice, in order to attain higher precision of abstraction in some problems, we can redefine the NR mapping (see definition 5.16) by parameterising the test, instead to assuming the Bernstein test Bt. We can define tests to comply with specific requirements of the analysed program. In the prototype above, we define a leg length test which delivers 1 if the leg's length is less than or equal to 1.25 and delivers 0 otherwise.

$$LLt(t) \;=\; \begin{cases} 1 & \text{if } [\![2 \cdot t - t^2]\!]^{\sharp}_{fl}(e^{\sharp}) \boxplus^{\sharp} [-1/3 - mf_{\mathbf{f}}, 1/3 + mf_{\mathbf{f}}] \;\leq\; [1.25, 1.25] \\ 0 & \text{otherwise.} \end{cases}$$

Given the set of abstract environments delivered by the redefined NR operator, we can apply then a Bernstein mapping to each of these abstract environments to obtain polyhedra. In the following section we present experimental results that show the performance of this adaptive method by means of time of computation and precision attained.

8.2.4 Experimental Results

In the table of figure 8.1, it can be noted first that for depth 1, we have the standard segmentation presented in chapter 4. Moreover, for depths larger than 1, the adaptive method presents several performances in terms of number of solution segments and precision loss. The method achieves the best compromise between time of computation, number of solution segments delivered and precision loss for values of n between 8 and 16. This may be surprising since the bisection method, namely for $n = 2$, does not perform better. As argued in [Vu05], bisection methods are suitable for cases with isolated solutions. In this case, however, we have a continuum of solutions included in the intervals $[0, 0.711]$ and $[0.711, 1.289]$. These two intervals are obtained by backward analysis and correspond in fact to the set of values for the input variable (time) for which no relative overflow can occur.

Moreover, note that here the maximum number of possible branching in the adaptive method is predetermined by n and the depth (d). Thus, the maximum number of branching is n^d. However, this number is not reached since $[0.711, 1.289] \subset [0, 4]$, that is, there exists subintervals in $[0, 4]$ that do not cause relative overflow and that are larger than the width of the shortest interval in the maximum depth.

n	depth	time	segments	precision loss
2	1	$0.01s$	1	$9.16667 \cdot 10^{-1}$
	2	$0.05s$	1	$9.16667 \cdot 10^{-1}$
	3	$0.08s$	3	$1.66667 \cdot 10^{-1}$
	4	$0.12s$	3	$1.66667 \cdot 10^{-1}$
	5	$0.14s$	5	$5.72917 \cdot 10^{-2}$
4	1	$0.03s$	2	$9.16667 \cdot 10^{-1}$
	2	$0.07s$	6	$1.66667 \cdot 10^{-1}$
	3	$0.10s$	12	$1.43229 \cdot 10^{-2}$
	4	$0.14s$	14	$4.80143 \cdot 10^{-3}$
	5	$0.18s$	18	$2.23796 \cdot 10^{-4}$
8	1	$0.04s$	6	$1.66667 \cdot 10^{-1}$
	2	$0.08s$	12	$1.43229 \cdot 10^{-2}$
	3	$0.12s$	18	$2.23796 \cdot 10^{-4}$
	4	$0.15s$	18	$2.23796 \cdot 10^{-4}$
	5	$0.20s$	24	$1.2214 \cdot 10^{-5}$
16	1	$0.04s$	12	$1.66667 \cdot 10^{-1}$
	2	$0.10s$	38	$4.80143 \cdot 10^{-3}$
	3	$0.14s$	54	$2.23796 \cdot 10^{-4}$
	4	$0.19s$	66	$1.2214 \cdot 10^{-5}$
	5	$0.24s$	76	$1.20146 \cdot 10^{-6}$
32	1	$0.09s$	26	$5.72917 \cdot 10^{-2}$
	2	$0.21s$	70	$2.23796 \cdot 10^{-4}$
	3	$0.28s$	76	$1.2214 \cdot 10^{-5}$
	4	$0.38s$	86	$1.20146 \cdot 10^{-6}$
	5	$0.47s$	120	$3.14214 \cdot 10^{-8}$
64	1	$0.18s$	54	$1.43229 \cdot 10^{-2}$
	2	$0.37s$	102	$2.23796 \cdot 10^{-4}$
	3	$0.55s$	152	$3.40391 \cdot 10^{-6}$
	4	$0.75s$	200	$1.00247 \cdot 10^{-7}$
	5	$0.93s$	292	$1.31022 \cdot 10^{-9}$
128	1	$0.36s$	108	$1.43229 \cdot 10^{-2}$
	2	$0.73s$	302	$8.27114 \cdot 10^{-5}$
	3	$1.10s$	452	$1.00247 \cdot 10^{-7}$
	4	$1.51s$	474	$5.61182 \cdot 10^{-9}$
	5	$1.92s$	640	$3.31866 \cdot 10^{-11}$

Figure 8.1: *Adaptive method for robot leg length (n, segmentation degree; tree depth; analysis time in seconds; number of segments that hold nonoverflow criteria; precision loss in the abstraction).*

8.3 Analytical Orbital Mechanics

8.3.1 Introduction

In this experiment, we show the utility of the techniques presented in this work for the detection and inference of absolute and intermediate overflows. We present a small prototype for computing parameters for the calculation of orbits of satellites, where the values in consideration may have several orders of magnitude. Moreover, we consider nonlinear multivariate expressions.

8.3.2 Prototyping in Logic Programming

The small prototype below computes parameters related to the orbit of satellite requires information on the position and velocity of an orbiting object in the plane.

```
orbit(Rx, Ry, Vx, Vy, Gmp, R, EE) :-
    massDistance(Rx, Ry, R),
    totalEnergy(Vx, Vy, Gmp, R, EE).

massDistance(Rx, Ry, R) :-
    R is sqrt(Rx * Rx + Ry * Ry).

totalEnergy(Vx, Vy, Gmp, R, EE) :-
    Gmp > 0,
    EE is (Vx * Vx + Vy * Vy)/2 - Gmp/R.
```

The function sqrt/1 is included in the mathematics module of SICStus and computes the square root of its argument. This program is in fact a simplified model which is not intended to include all the possible parameters to fully specify an orbit.

8.3.3 Logarithmic Segmentation

When considering large domains such as for instance in double precision the range $[0, Mf_{\mathbf{f}}] \approx [0, 10^{308}]$, the segmentation method presented in chapter 4 may be impractical; namely, the number of segmentations required to provide a reasonably significant solution may be of the order of 10^{308} which is clearly computationally unfeasible. In this section we introduce a logarithmic segmentation, namely, a segmentation that bisects the domain into two subdomains according to the order of magnitude. For instance, the logarithmic segmentation of $[0, 10^{308}]$ would be the two intervals $[0, 10^154]$ and $[10^{154}, 10^{308}]$.

More formally, given an interval $[a, b]$, let $\ell_a = \log_{10} a$ and $\ell_b = \log_{10} b$. Then, for $\ell_b - \ell_a > 2$, the logarithmic segmentation of $[a, b]$ delivers the intervals

$$[a,\ 10^{\lceil \frac{\ell_a + \ell_b}{2} \rceil}] \qquad \text{and} \qquad [10^{\lceil \frac{\ell_a + \ell_b}{2} \rceil}, b].$$

For $\ell_b - \ell_a \leq 2$, then the segmentation in chapter 4 can be applied.

The logarithmic segmentation is useful for floating-point expressions that may grow exponentially, such as for instance in the expressions in the above prototype.

8.3.4 Experimental Results

We assume the underlying floating-point system corresponds to double precision. In the case of the mass distance, the analysis uses first logarithmic segmentation and then standard segmentation (with segmentation degree $n = 2$). The analysis shows in 0.11 seconds that no intermediate overflow can occur in the output of the mass distance predicate if

$$Rx \in [0,\ 6.25 \cdot 10^{153}] \qquad \text{and} \qquad Ry \in [0,\ 6.25 \cdot 10^{153}].$$

These constraints can be tightened further at higher time and space cost. Under the conditions above on the input variables, no intermediate overflow can thus occur, and therefore no absolute overflow can occur either. The analysis moreover, uses the square function to show that the floating-point square root does not overflow if invoked for values different to NaN or $\pm\infty$. Thus, assuming no intermediate overflow occurs, the output variable R is bounded by

$$R \in [0, 10^{154}].$$

In the case of the total energy, the analysis infers in a similar way than for the mass distance case, that no intermediate overflow can occur if the input variables are bounded by

$$Vx \in [0,\ 6.25 \cdot 10^{153}] \qquad \text{and} \qquad Vy \in [0, 6.25 \cdot 10^{153}].$$

Note that for $R = 0$, division by zero occurs. Thus, in order to avoid division by zero, the variable R is bounded by $R \in [mf_{\mathbf{f}}, 10^{154}]$. Moreover, $Gmp \in [mf_{\mathbf{f}}, Mf_{\mathbf{f}}]$. However, an intermediate overflow is detected in the division of Gmp by R. Thus, R is further bounded to $[1, 10^{154}]$, which ensures no overflow can occur. The analysis concludes that

$$EE \in [-Mf_{\mathbf{f}},\ 3.90625 \cdot 10^{307}].$$

That is, if the input variables hold the inferred nonoverflow conditions, then no overflow can occur in the output variables.

This analysis can be a useful tool to improve the design of programs by detecting and inferring conditions for overflow in floating-point expressions.

Chapter 9

Conclusion

This thesis has presented techniques for the abstraction of floating-point expressions into the polyhedra domain and the finite powerset of polyhedra domain. Moreover, a forward and a backward analyses for detecting and inferring conditions of overflow and division by zero have been introduced. The main aim of this thesis has been oriented around the gain in precision in the abstractions and subsequent analyses, in order to detect and infer several classes of overflow that may occur in software applications with nonlinear floating-point expressions, which are common for instance in complex (physical) systems. In this way, the main contributions of this thesis are summarised as follows:

- The introduction of a nonlinear abstract semantics based on interval valued polynomials that has increased precision in comparison with other abstractions for floating-point arithmetic. This semantics preserves nonlinear information which proves important to avoid linearisation and thus, loss of precision. Our abstraction holds in the abstract interpretation framework of [Min04a]. This abstract semantics will prove useful for abstracting floating-point expressions into the polyhedra domain.

- The adaptation of a series of methods based on Bernstein polynomials to abstract the nonlinear abstract semantics introduced in chapter 3, into polyhedra, and also to approximate the range enclosure of such semantics. Such methods vary in cost of computation as well as in the degree of precision attained. Methods that achieve a compromise between time and space efficiency, as well as precision are also proposed.

- The definition of a test for root existence in interval valued polynomials, based on Miranda's root existence theorem and the range enclosing property of the Bernstein mappings, and the introduction of a variation of the branch and prune method for

identifying segments of an abstract environment in which no roots exists. Based on this, we propose an abstraction of floating-point polynomial quotients into the polyhedra powerset.

- The introduction of general abstractions for assignments and tests in the powerset of polyhedra. These abstractions are general in that linear and nonlinear expressions are considered. Moreover, we introduce a forward analysis for the detection of floating-point overflow and overflow due to division by zero which always reaches the output program points. This is necessary in order to develop a backward analysis that infers preconditions on the input variables of a program to avoid possible divisions by zero.

- The introduction of a backward analysis for the inference of numerical properties in logic programs that perform floating-point arithmetic operations. The analysis is specialised in the inference of preconditions to avoid floating-point overflow, such as for instance that caused by division by zero. Central in this analysis is the use of the powerset of polyhedra.

Bibliography

[ABB+03] Y. Aït Ameur, G. Bel, F. Boniol, S. Pairault, and V. Wiels. Robustness analysis of avionics embedded systems. In *LCTES'03*, pages 123–132. ACM Press, 2003.

[AFHM04] G. Alefeld, A. Frommer, G. Heindl, and J. Mayer. On the existence theorems of Kantorovich, Miranda and Borsuk. *Electronic Transactions on Numerical Analysis*, 17:102–111, 2004.

[Age96] European Space Agency. Ariane 501 inquiry board report. Paris, July 19, 1996. Full report available at http://ravel.esrin.esa.it/docs/esa-x-1819eng.pdf.

[AL95] K.R. Apt and I. Luitjes. Verification of logic programs with delay declarations. In *4th International Conference on Algebraic Methodology and Software Technology (AMAST '95), (invited lecture)*, volume 936 of *Lecture Notes in Computer Science*, pages 66–90. Springer-Verlag, 1995.

[AM94] K. R. Apt and E. Marchiori. Reasoning about Prolog programs: from modes through types to assertions. *Formal Aspects of Computing*, 6(6A):743–765, 1994.

[AP94] K. R. Apt and A. Pellegrini. On the occur-check-free PROLOG programs. *ACM Transactions on Programming Languages and Systems*, 16(3):687–726, May 1994.

[Apt90] K. R. Apt. Introduction to logic programming. *Handbook of Theoretical Computer Science*, B: Formal Models and Semantics:495–574, 1990.

[Bag98] R. Bagnara. A hierarchy of constraint systems for data-flow analysis of constraint logic-based languages. *Science of Computer Programming*, 30(1–2):119–155, 1998.

[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 *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI'03)*, pages 196–207, San Diego, California, USA, June 7–14 2003. ACM Press.

[Ber12] S. Bernstein. Démonstration du théorème de Weierstrass fondeé sur le calcul des probabilités. *Harkov Soobs. Matem ob-va*, (13):1–2, 1912.

[Ber52] S. Bernstein. *Collected Works*, volume 1. USSR Academy of Sciences, 1952.

[Ber54] S. Bernstein. *Collected Works*, volume 2. USSR Academy of Sciences, 1954.

[BHRZ03a] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. Submitted for publication to Science of Computer Programming, 2003.

[BHRZ03b] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. *The Parma Polyhedra Library. User's Manual (version 0.5)*. Department of Mathematics, University of Parma, Italy, April 28, 2003. Documentation and software available at http://www.cs.unipr.it/ppl/.

[BHZ04] R. Bagnara, P. M. Hill, and E. Zaffanella. Widening operators for powerset domains. In G. Levi and B. Steffen, editors, *Proceedings of the 5th International Conference on Verification, Model Checking and Abstract Interpretation*, volume 2937 of *LNCS*, pages 135–148. Springer, January 2004.

[Bir67] G. Birkhoff. *Lattice theory*, volume XXV of *Colloquium Publications*. American Mathematical Society, Providence, Rhode Island, 3rd edition, 1967.

[BK89] V. Balasundaram and K. Kennedy. A technique for summarizing data access and its use in parallelism enhancing transformations. In B. Knobe, editor, *Proceedings of the ACM SIGPLAN'89 Conference on Programming Language Design and Implementation (PLDI)*, volume 24 of *ACM SIGPLAN Notices*, pages 41–53, Portland, Oregon, USA, 1989. ACM Press.

[BL03] F. G. Boese and W. J. Luther. Accurate enclosure of the zero set of multivariate polynomials. *BIT Numerical Mathematics*, 43:245–261, 2003.

[BRCZ05] R. Bagnara, E. Rodríguez-Carbonell, and E. Zaffanella. Generation of basic semi-algebraic invariants using convex polyhedra. In C. Hankin, editor, *12th International Static Analysis Symposium, SAS 2005*, Lecture Notes in Computer Science, Imperial College London, UK, 7-9 September 2005. Springer-Verlag.

[BRZH02] R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In M. V. Hermenegildo

and G. Puebla, editors, *Static Analysis: Proceedings of the 9th International Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 213–229, Madrid, Spain, 2002. Springer-Verlag, Berlin.

[BZGH01] R. Bagnara, E. Zaffanella, R. Gori, and P. M. Hill. Boolean functions for finite-tree dependencies. In R. Nieuwenhuis and A. Voronkov, editors, *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2001)*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 575–589, Havana, Cuba, 2001. Springer-Verlag, Berlin.

[CC76] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *ISOP'76*, pages 106–130, Dunod, Paris, France, 1976.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, NY.

[CC82] P. Cousot and R. Cousot. Induction principles for proving invariance properties of programs. In D. Néel, editor, *Tools & Notions for Program Construction: an Advanced Course*, pages 75–119. Cambridge University Press, Cambridge, UK, August 1982.

[CC92a] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992. (The editor of Journal of Logic Programming has mistakenly published the unreadable galley proof. For a correct version of this paper, see http://www.di.ens.fr/cousot.).

[CC92b] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.

[CC04] R. Clarisó and J. Cortadella. The octahedron abstract domain. In R. Giacobazzi, editor, *Static Analysis, 11th International Symposium, SAS 2004*, volume 3148 of *Lecture Notes in Computer Science*, pages 312–327, Verona, Italy, August 2004. Springer.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY.

[Che68] N. V. Chernikova. Algorithm for discovering the set of all solutions of a linear programming problem. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 8(6):282–293, 1968.

[Col90] A. Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7):69–90, 1990.

[Cou05] P. Cousot. The verification grand challenge and abstract interpretation. In *Verified Software: Theories, Tools, Experiments (VSTTE)*, ETH Zürich, Switzerland, October 10-13, 2005.

[CS66] G. T. Cargo and O. Shisha. The Bernstein form of a polynomial. *Journal of Research of the National Bureau of Standards - Section B. Mathematical Sciences*, 70B(1):79–81, 1966.

[CT04] P. Clauss and I. Tchoupaeva. A symbolic approach to Bernstein expansion for program analysis and optimization. In E. Duesterwald, editor, *Compiler Construction, 13th International Conference, CC 2004, held as part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004*, volume 2985 of *Lecture Notes in Computer Science*, pages 120–133, Barcelona, Spain, March 29 - April 2, 2004. Springer.

[CV88] J. M. Chesneaux and J. Vignes. Sur la robustesse de la méthode CESTAC. *Comptes Rendus de l'Académie des Sciences*, 1(307):855–860, 1988.

[CV92] J. M. Chesneaux and J. Vignes. Les fondaments de l'arithmétique stochastique. *Comptes Rendus de l'Académie des Sciences*, 1(315):1435–1440, 1992.

[CWMM04] D. Chablat, Ph. Wenger, F. Majou, and J-P. Merlet. An interval analysis based study for the design and the comparison of three-degrees-of-freedom parallel kinematic machines. *The International Journal of Robotics Research*, 23(6):615–624, 2004.

[Dem84] J. W. Demmel. Underflow and the reliability of numerical software. *SIAM Journal in Scientific and Statistical Computing*, 5(4):887–919, 1984.

[DGB92] S. Debray, D. Gudeman, and P. Bigot. Detection and optimisation of suspension-free logic programs. *The Journal of Logic Programming*, 29(1–3):171–194, 1992.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, United Kingdom, 1990.

[DP03] J. Delgado and J. M. Peña. A linear complexity algorithm for the Bernstein basis. In *International Conference on Geometric Modeling and Graphics (GMAG'03)*, pages 162–167. IEEE, July 2003.

[EBC99] S. Etalle, A. Bossi, and N. Coco. Termination of well-moded programs. *Journal of Logic Programming*, 38(2):243–257, 1999.

[Eda97] A. Edalat. Domains for computation in mathematics, physics and exact real arithmetic. *Bulletin of Symbolic Logic*, 3(4):401–452, 1997.

[EP97] A. Edalat and P. J. Potts. Exact real computer arithmetic. Technical Report DOC 97/9, Department of Computing, Imperial College, London, 1997.

[ES99] A. Edalat and P. Snderhauf. A domain-theoretic approach to real number computation. *Theoretical Computer Science*, 210(1):73–98, 1999.

[Far93] G. Farin. *Curves and surfaces for computer aided geometric design. A practical guide*. Computer Science and Scientific Computing. Academic Press, San Diego, CA 92101, US, 3rd edition, 1993.

[Fer04] J. Feret. Static analysis of digital filters. In D. A. Schmidt, editor, *Programming Languages and Systems, 13th European Symposium on Programming, ESOP 2004, held as part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004*, volume 2986 of *Lecture Notes in Computer Science*, pages 33–48, Barcelona, Spain, March 29 - April 2, 2004. Springer.

[FHW00] M. Falaschi, P. Hicks, and W. Winsborough. Demand transformation analysis for concurrent constraint programs. *The Journal of Logic Programming*, 42(3):185–215, March 2000.

[Fis90] H. C. Fischer. Range computations and applications. Contributions to Computer Arithmetic and Self-Validating Numerical Methods, pages 197–211, Amsterdam, 1990.

[FL05] A. Frommer and B. Lang. Existence tests for solutions of nonlinear equations using Borsuk's theorem. *SIAM Journal on Numerical Analysis*, 43(3):1348–1361, 2005.

[Fuk01] K. Fukuda. *cddlib reference manual, cddlib Version 092a*. McGill University, Montreal, Canada, 2001. Manual and software available at http://www.cs.mcgill.ca/ fukuda/software/cdd_home/cdd.html.

[Gar86] J. Garloff. Convergent bounds for the range of multivariate polynomials. *Interval Mathematics 1985*, 212:37–56, 1986.

[Gar93] J. Garloff. The Bernstein algorithm. *Interval Computations*, 2:154–168, 1993. (Interval Computations is now known as Reliable Computing).

[GC01] S. Genaim and M. Codish. Inferring termination conditions for logic programs using backwards analysis. In *International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 681–690. Springer-Verlag, 2001.

[GG99] J. Garloff and B. Graf. Solving strict polynomial inequalities by Bernstein expansion. In N. Munro, editor, *Symbolic Methods in Control System Analysis and Design*, pages 339–352, London, 1999. IEEE.

[GK03] S. Genaim and A. King. Goal-independent suspension analysis for logic programs with dynamic scheduling. In P. Degano, editor, *European Symposium on Programming*, volume 2618 of *Lecture Notes in Computer Science*, pages 84–98. Springer-Verlag, April 2003.

[GL70] D. I. Good and R. L. London. Computer interval arithmetic: Definition and proof of correct implementation. *Journal of the Association for Computing Machinery*, 17(4):603–612, October 1970.

[GMP] E. Goubault, M. Martel, and S. Putot. Asserting the precision of floating-point computations: a simple abstract interpreter. In D. Le Métayer, editor, *Programming Languages and Systems, 11th European Symposium on Programming, ESOP 2002, held as part of the Joint European Conference on Theory and Practice of Software ETAPS 2002, Grenoble, France, April 8-12, 2002*, volume 2305 of *Lecture Notes in Computer Science*, pages 133–150. Springer.

[GMP01] E. Goubault, M. Martel, and S. Putot. Concrete and abstract semantics of floating-point operations. Technical report, CEA – Recherche Technologique, LIST–DTSI–SLA, CEA F91191 Gif-Sur-Yvette Cedex, France, August 2001.

[GMP06] E. Goubault, M. Martel, and S. Putot. Some future challenges in the validation of control systems. In *Embedded Real Time Software, ERTS'06*, Toulouse, France, January 25-27, 2006.

[Gol91] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, March 1991.

[Gol06] A. Goldsztejn. A branch and prune algorithm for the approximation of non-linear AE-solution sets. In *21st ACM Symposium on Applied Computing*

Track Reliable Computations and their Applications 2006, Dijon, France, 2006.

[Gou01] E. Goubault. Static analyses of the precision of floating-point operations. In Patrick Cousot, editor, *Static Analysis, 8th International Symposium, SAS 2001*, volume 2126 of *Lecture Notes in Computer Science*, pages 234–259, Paris, France, July 16-18, 2001. Springer.

[GPR96] R. Giacobazzi, C. Palamidessi, and F. Ranzato. Weak relative pseudo-complements of closure operators. *Algebra Universalis*, 36(3):405–412, 1996. Also available as Technical Report LIX/95/04, 1995 LIX, Ecole Polytechnique, 91128 Palaiseau Cedex, France.

[GPV95] C. Gosselin, L. Perreault, and C. Vaillancourt. Simulation and computer-aided kinematic design of three-degree-of-freedom spherical parallel manipulators. *Journal on Robotic Systems*, 12(12):857–869, 1995.

[GRS05] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract domains condensing. *ACM Transactions on Computational Logic (ACM-TOCL)*, 6(1):33–60, 2005.

[GS98] R. Giacobazzi and F. Scozzari. A logical model for relational abstract domains. *ACM Transactions on Programming Languages and Systems*, 20(5):1067–1109, 1998.

[Hal79] N. Halbwachs. *Détermination Automatique de Relations Linéaires Vérifiées par les Variables d'un Programme*. PhD thesis, Université scientifique et médicale de Grenoble, Grenoble, France, March 1979. Theèse de 3ème cycle d'informatique.

[Han75] E. R. Hansen. A generalized interval arithmetic. In K. Nickel, editor, *Proceedings of the International Symposium in Interval Mathematics*, volume 29 of *Lecture Notes in Computer Science*, pages 7–18, Karlsruhe, West Germany, May 20-24, 1975. Springer-Verlag.

[Hig02] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, US, second edition, 2002. xxx+680 pages.

[HJM+91] N. Heintze, J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. *The CLP(R) Programmer's Manual Version 1.1*. November 1991.

[HJvE01] T. J. Hickey, Q. Ju, and M. H. van Emden. Interval arithmetic: From principles to implementation. *Journal of the ACM*, 48(5):1038–1068, 2001.

[HKL04] J. M. Howe, A. King, and L. Lu. Analysing logic programs by reasoning backwards. In M. Bruynooghe and K-K. Lau, editors, *Program Development in Computational Logic*, volume 3049 of *Lecture Notes in Computer Science*, pages 152–188. Springer-Verlag, May 2004.

[HL94] J. Hughes and J. Launchbury. Reversing abstract interpretations. *Science of Computer Programming*, 22(3):307–326, 1994.

[Hon93] H. Hong. RISC-CLP(Real): Constraint logic programming over real numbers. In F. Benhamou and A. Colmerauer, editors, *Constraint Logic Programming: Selected Research*. MIT Press, 1993.

[Hon94] H. Hong. RISC-CLP(CF) - constraint logic programming over complex functions. In F. Pfenning, editor, *Logic Programming and Automated Reasoning, 5th International Conference, LPAR'94*, volume 822 of *Lecture Notes in Computer Science*, pages 99–113, Kiev, Ukraine, July 16-22, 1994. Springer.

[HPR97] N. Halbwachs, Y. E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.

[HS95] H. Hong and V. Stahl. Bernstein form is inclusion monotone. *Computing*, 55(1):43–53, 1995.

[HS02] P. M. Hill and F. Spoto. Generalising *Def* and *Pos* to type analysis. *Journal of Logic and Computation*, 12(3):497–542, June 2002.

[HW89] C. Hall and D. Wise. Generating function versions with rational strictness patterns. *Science of Computer Programming*, 12:39–74, 1989.

[Jea04] B. Jeannet. *Convex Polyhedra Library, release 2.0.2*. Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), Campus Universitaire de Beaulieu, F-35042 Rennes Cedex, France, February 27th, 2004. Documentation for the *New Polka* library available at http://www.irisa.fr/prive/Bertrand.Jeannet/newpolka.html.

[JL87] J. Jaffar and J-L. Lassez. Constraint logic programming. In 14*th ACM Symposium on Principles of Programming Languages*, pages 111–119, Munich, Germany, 1987. ACM.

[JM94] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–582, 1994.

[Kah95] W. Kahan. The improbability of probabilistic error analyses for numerical computations (invited speaker). In K. Kirchgssner, O. Mahrenholtz, and

R. Mennicken, editors, *Proceedings of the Third International Congress on Industrial and Applied Mathematics (ICIAM 95)*, volume 87, Hamburg, Germany, July 3-7, 1995. Mathematical Research. Transparencies available at http://www.cs.berkeley.edu/wkahan/.

[KJ93] W. Krandick and J. R Johnson. Efficient multiprecision floating-point multiplication with optimal directional rounding. In E. E. Swartzlander, M. J. Irwin, and J. Julien, editors, *11th IEEE Symposium on Computer Arithmetic*, pages 228–233, Windsor, Canada, June 1993. IEEE Computer Society Press, Los Alamitos, CA, US.

[KL02] A. King and L. Lu. A backward analysis for constraint logic programs. *Theory and Practice of Logic Programming*, 2(4-5):517–547, July 2002. (Theory and Practice of Logic Programming was formally known as The Journal of Logic Programming; see http://www.cwi.nl/projects/alp/Welcome/appeal.html).

[KL03] A. King and L. Lu. Forward versus backward verification of logic programs. In Catuscia Palamidessi, editor, *International Conference on Logic Programming*, volume 2916 of *Lecture Notes in Computer Science*, pages 315–330. Springer-Verlag, December 2003.

[KM85] P. Kornerup and D. W. Matula. Finite precision lexicographic continued fraction number system. In *7th IEEE Symposium on Computer Arithmetic*, Urbana, USA, 1985. IEEE Computer Society Press, Los Alamitos, CA. Reprinted by E. E. Swartzlander, *Computer Arithmetic*, Vol. 2, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[LeV92] H. LeVerge. A note on Chernikova's algorithm. Technical Report RR. 635, IRISA, France, February 1992.

[LK02] L. Lu and A. King. Backward Type Inference Generalises Type Checking. In M. Hermenegildo and G. Puebla, editors, *Ninth International Static Analysis Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 85–101. Springer-Verlag, September 2002. Also see http://www.springer.de/comp/lncs/index.html.

[LK04] L. Lu and A. King. Backward pair sharing analysis. In Y. Kameyama and P. Stuckey, editors, *Seventh International Symposium on Functional and Logic Programming*, volume 2998 of *Lecture Notes in Computer Science*, pages 132–146. Springer-Verlag, April 2004.

[LL05] X. Li and L. Lu. Inverting abstract unification for set-sharing. In *PPDP'05*, pages 129–138, Lisbon, Portugal, July 11-13, 2005. ACM.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming, Second Edition*. Springer-Verlag, second edition, 1987.

[LS97] N. Lindenstrauss and Y. Sagiv. Automatic termination analysis of logic programs. In *Proceedings of the Eleventh International Conference on Logic Programming*, pages 63–77. MIT Press, 1997.

[Mar] M. Martel. Propagation of roundoff errors in finite precision computations: a semantics approach. In D. Le Métayer, editor, *Programming Languages and Systems, 11th European Symposium on Programming, ESOP 2002, held as part of the Joint European Conference on Theory and Practice of Software ETAPS 2002, Grenoble, France, April 8-12, 2002*, volume 2305 of *Lecture Notes in Computer Science*, pages 133–150. Springer.

[Mar02] M. Martel. Static analysis of the numerical stability of loops. In M. Hermenegildo and G. Puebla, editors, *Static Analysis, 9th International Symposium, SAS 2002*, volume 2477 of *Lecture Notes in Computer Science*, pages 133–150, Madrid, Spain, September 17-20, 2002. Springer.

[Mer94] J-P. Merlet. Trajectory verification in the workspace for parallel manipulators. *The International Journal of Robotics Research*, 13(4):326–333, 1994.

[Mer01] J-P. Merlet. A generic trajectory verifier for the motion planning of parallel robots. *Journal of Mechanical Design, Transactions of the ASME (American Society of Mechanical Engineers)*, 123(4):510–515, December 2001.

[Mer04] J-P. Merlet. Solving the forward kinematics of Gough-type parallel manipulator with interval analysis. *International Journal of Robotics Research*, 23(3):221–236, 2004.

[Min00] A. Miné. Représentation d'ensembles de contraintes de somme ou de différence de deux variables et application à l'analyse automatique de programmes. Master's thesis, Laboratoire d'Informatique de l'ENS, 18 Septembre 2000.

[Min01] A. Miné. The octagon abstract domain. In *AST 2001 in WCRE 2001*, IEEE, pages 310–319. IEEE CS Press, October 2001.

[Min04a] A. Miné. *Weakly Relational Numerical Abstract Domains*. PhD thesis, École Normale Supérieure, Département d'Informatique, Université Paris IX Dauphine, Paris, France, December 6, 2004 2004. xiv+308 pages. Original title: *Domaines numériques abstraits faiblement relationnels*.

[Min04b] A. Miné. Relational abstract domains for the detection of floating-point run-time errors. In D. A. Schmidt, editor, *Programming Languages and*

Systems, 13th European Symposium on Programming, ESOP 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, volume 2986 of *Lecture Notes in Computer Science*, pages 3–17, Barcelona, Spain, March 29 - April 2, 2004. Springer.

[Min06a] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 2006.

[Min06b] A. Miné. Symbolic methods to enhance the precision of numerical abstract domains. In *VMCAI'06*, 2006.

[Mir41] C. Miranda. Un'osservazione su un teorema di Brouwer. *Bolletino Unione Mathematica Italiana*, 2(3):5–7, 1941.

[MK80] R. E. Moore and J. B. Kioustelidis. A simple test for accuracy of approximate solutions to nonlinear (or linear) systems. *SIAM Journal on Numerical Analysis*, 17:521–529, 1980.

[MMTG92] S. Malan, M. Milanese, M. Taragna, and J. Garloff. b^3 Algorithm for robust performances analysis in presence of mixed parametric and dynamic perturbations. In *31st Conference on Decision and Control*, pages 128–133, Tucson, Arizona, 1992.

[Moo66] R. E. Moore. *Interval Analysis*. Cliffs. Prentice-Hall, 1966.

[Moo72] R. E. Moore. A test for existence of solutions to nonlinear systems. *SIAM Journal on Numerical Analysis*, 14:611–615, 1972.

[Moo79] R. E. Moore. Methods and applications of interval analysis. Philadelphia, PA, USA, 1979. Society for Industrial and Applied Mathematics.

[MP94] V. Maslov and W. Pugh. Simplifying polynomial constraints over integers to make dependence analysis more precise. In B. Buchberger and J. Volkert, editors, *CONPAR'94 - VAPP VI, Third Joint International Conference on Parallel and Vector Processing*, volume 854 of *Lecture Notes in Computer Science*, Linz, Austria, September 6-8, 1994. Springer.

[MRTT53] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. The double description method. In H. W. Kuhn and A. W. Tucker, editors, *Annals of Mathematic Studies*, number 28 in Contribution to the Theory of Games – Volume II. Princeton University Press, 1953.

[MS87] A. P. Morgan and V. Shapiro. Box-bisection for solving second-degree systems and the problem of clustering. *ACM Transactions on Mathematical Software*, 13(2):152–167, 1987.

[MS93] K. Marriott and H. Søndergaard. Precise and efficient groundness analysis for logic programs. *ACM Letters on Programming Languages and Systems*, 2(1-4):181–196, March–December 1993.

[MSV+02] R. Martin, H. Shou, I. Voiculescu, A. Bowyer, and G. Wang. Comparison of interval methods for plotting algebraic curves. *Computer Aided Geometric Design*, 19(7):553–587, 2002.

[MV99] B. D. McCullough and H. D. J. Vinod. The numerical reliability of econometric software. *Journal of Economic Literature*, XXXVII(2):633–665, June 1999.

[oCS06] Swedish Institute of Computer Science. *SICStus Prolog User's Manual*. Intelligent Systems Laboratory, Swedish Institute of Computer Science, Kista, Sweden, March 2006. Available at http://www.sics.se/.

[oEE85] Institute of Electrical and Electronics Engineers. IEEE Standard 754 for Binary Floating Point Arithmetic. *IEEE*, 1985. Reprinted in SIGPLAN Notices, 22(2):9-25, 1987.

[Off92] United States General Accounting Office. Patriot missile software problem. (GAO/IMTEC-92-26), 1992.

[OV93] W. J. Older and A. Velino. Constraint arithmetic on real intervals. In F. Benhamou and A. Colmerauer, editors, *Constraint Logic Programming: Selected Research*, pages 175–196. MIT Press, 1993.

[PCT95] M. S. Petković, C. Carstensen, and M. Trajković. Weierstrass formula and zero-finding methods. *Numerische Mathematik*, 69:353–372, 1995.

[Plü90] L. Plümer. *Termination proofs for logic programs*, volume 446 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1990.

[Ran02] F. Ranzato. Pseudocomplements of closure operators on posets. *Discrete Mathematics*, 248(1-3):143–155, 2002.

[Riv70] T. J. Rivlin. Bounds of a polynomial. *J. Res. Nat. Bur. Standards Sect. B*, Vol. 74B:47–54, 1970.

[Rok77] J. Rokne. Bounds for an interval polynomial. *Computing*, Vol. 18:225–249, 1977.

[Rok79] J. Rokne. A note on the Bernstein algorithm for bounds for an interval polynomial. *Computing*, Vol. 21:159–170, 1979.

[Rok82] J. Rokne. Optimal computation of the Bernstein algorithm for the bound of an interval polynomial. *Computing*, Vol. 28:239–246, 1982.

[RR84] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions.* Series in Mathematics and its Applications. Ellis Horwood, Chichester, England, 1984.

[RR03] H. Ratschek and J. Rokne. *Geometric Computations with Interval and New Robust Methods: Applications in Computer Graphics, GIS and Computational Geometry.* Series in Mathematics and its Applications. Albion/Horwood, Chichester, England, December 2003.

[RT73] J. E. Robertson and K. S. Trivedi. The status of investigations into computer hardware design based on the use of continued fractions. *IEEE Transactions on Computers*, C-22:555–560, 1973.

[San05] S. Sankaranarayanan. *Mathematical analysis of programs.* PhD thesis, Department of Computer Science, Stanford University, US, 2005.

[Sco89] M. Scott. Fast rounding in multiprecision floating-slash arithmetic. *IEEE Transactions on Computers*, 38:1049–1052, 1989.

[Sei83] R. B. Seidensticker. Continued fractions for high-speed and high-accuracy computer arithmetic. In *6th IEEE Symposium on Computer Arithmetic*, Aarhus, Denmark, 1983. IEEE Computer Society Press, Los Alamitos, CA.

[SG00] A. P. Smith and J. Garloff. Solution of systems of polynomial equations by using Bernstein expansion. Technical Report 129, Konstanzer Schriften in Mathematik und Informatik, Universität Konstanz, Fach D 188, 78457 Konstanz, Germany, September 2000.

[Sim05] A. Simon. Relational analysis of floating-point arithmetic. In *NSAD Joint Workshop with VMCAI*, January 21, 2005.

[Ske92] R. Skeel. Roundoff error and the Patriot missile. *SIAM News*, (25), July 11, 1992.

[SKH02] A. Simon, A. King, and J. M. Howe. Two variables per linear inequality as an abstract domain. In M. Leuschel, editor, *Proceedings of Logic Based Program Development and Transformation*, volume 2664 of *Lecture Notes in Computer Science*, pages 71–89. Springer-Verlag, 2002.

[SM04] A. Serebrenik and F. Mesnard. On termination of binary CLP programs. In S. Muñoz-Hernández and J. M. Gómez-Pérez, editors, *14th workshop on*

logic programming environments, WLPE' 04, in 20th International Conference on Logic Programming, ICLP'04, 1 quai Duguay Trouin, 35400 Saint-Malo, France, September 6-10, 2004.

[SS02] A. Serebrenik and D. De Schreye. On termination of logic programs with floating-point computations. In M. V. Hermenegildo and G. Puebla, editors, *9th International Static Analysis Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 151–164. Springer-Verlag, 2002.

[Sta95] V. Stahl. *Interval methods for bounding the range of polynomials and solving systems of nonlinear equations*. PhD thesis, University of Linz, Austria, 1995.

[VACS94] M. Vinícius, A. Andrade, J. L. D. Comba, and J. Stolfi. Affine arithmetic. In *INTERVAL94, International Conference on Interval and Computer- Algebraic Methods in Science and Engineering*, 1994.

[vE99] M.H. van Emden. The logic programming paradigm in numerical computation. In K. R. Apt, V. W. Marek, M. Truszczynski, and D. S. Warren, editors, *The Logic Programming Paradigm*, pages 257–276. Springer-Verlag, 1999.

[vEA76] M. H. van Emden and Kowalski R. A. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23:733–742, 1976.

[vHMD97] P. van Hentenryck, L. Michel, and Y. Deville. *Numerica. A Modeling Language for Global Optimization*. The MIT Press, Cambridge, Massachusetts and London, England, 1997.

[vHMK97] P. van Hentenryck, D. McAllester, and D. Kapur. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis*, 34(2):797–827, 1997.

[Vig96] J. Vignes. A survey of the CESTAC method. In J-C. Bajard, editor, *Real Numbers and Computer Conference*, 1996.

[Vu05] X-H. Vu. *Rigorous solution techniques for numerical constraint satisfaction problems*. PhD thesis, Faculté Informatique et Communications, Institut d'Informatique Fondamentale, École Polytechnique Fédérale de Lausanne, Laussane, Switzerland, March 2005.

[Vui90] J. E. Vuillemin. Exact real computer arithmetic with continued fractions. *IEEE Transactions on Computers*, 39(8), 1990.

[Wei85] K. Weierstrass. Uber die analytische darstellbarkeit sogenannter willkurlicher functionen einer reellen veranderlichen. *Sitzungsberichte der Koniglich*

Preussischen Akademie der Wissenshcaften zu Berlin, pages 633–639,789–805, 1885.

[WH98] P. Wadler and R. J. M. Hughes. Projections for strictness analysis. In *Functional Programming and Computer Architecture*, volume 274 of *Lecture Notes in Computer Science*, pages 385–407. Springer-Verlag, 1998.

[Wil80] J. H. Wilkinson. Turing's work at the National Physical Laboratory and the construction of Pilot ACE, DEUCE and ACE. In N. Metropolis, J. Howlett, and G-C. Rota, editors, *A History of Computing in the Twentieth Century: A Collection of Essays*, pages 101–114, New York, US, 1980. Academic Press.

[Zet91] M. Zettler. Subdivision and degree elevation for Bernstein polynomials, 1991. diploma thesis, Fachhochschule Konstanz, Fachbereich Informatik, Germany.

[ZG98] M. Zettler and J. Garloff. Robustness analysis of polynomials with polynomial parameter dependency using Bernstein expansion. *IEEE Transactions on Automatic Control*, 43:425–431, 1998.