POLYHEDRAL DOMAINS FOR ABSTRACT INTERPRETATION IN LOGIC PROGRAMMING

A THESIS SUBMITTED TO
THE UNIVERSITY OF KENT AT CANTERBURY
IN THE SUBJECT OF COMPUTER SCIENCE
FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY.

By
Patricia Mary Benoy

January 2002

Contents

List of Figures

List of Tables

Dedication

This thesis is dedicated to my father, Ralph Wheeler, whose patience instilled in me a love of mathematics. Had he still been alive, he would have been intrigued by this work.

Abstract

The motivations for program analyses are many. The impetus for the static analysis of programs initially ranged from the need for more efficient code to program transforms for parallel systems, but with the passage of time now also encompasses: providing the programmer with debugging information, program verification and program testing.

Abstract interpretation is a methodology for static analysis that was formalised at the beginning of the last decade and is employed in both declarative and imperative programming paradigms. Abstract interpretation is a rigorous approximation technique that allows an analysis to focus on a particular property and infer useful information about the run time behaviour of the program being analysed. In order to implement an abstract interpretation it is necessary to chose an abstract domain that expresses the property of interest with sufficient precision to be useful.

The focus of this thesis is on the use of two polyhedral domains as a means of capturing these properties of interest in logic programs. This thesis demonstrates how polyhedra can be employed to capture analyses that relate to dependency information with respect to some measure of size. Polyhedral analyses require a technique known as widening to ensure analysis convergence. Computational issues associated with widening prompted an investigation into the way linear spaces interact and how it is possible for a polyhedron to have different representations as a set of implicitly conjoined linear inequalities. Some useful computational insights are the outcome of this investigation.

The observation that certain linear inequalities might represent dependency information prompted a novel approach to capturing dependency analysis with the subclass of polyhedra that can be represented by those linear inequalities. This could be advantageous for analyses in environments with constraint support as no set-up costs for the analysis would be incurred as the existing environment would be capable of supporting the abstract domain. Central to this thesis is the isomorphism that reveals the precise extent to which these polyhedra can represent dependency information.

An argument-size analysis using polyhedra and a groundness analysis using certain polyhedral cones have been implemented up to the prototype stage.

Acknowledgements

First and foremost I must thank my supervisor Dr Andy King for his support throughout this time. Secondly, my grateful thanks to go to the Computing Laboratory here at Kent for the E.B.Spratt Bursary that made it possible for me to embark on a PhD.

The argument size analysis presented in Chapter 5 is based on work co-authored with my supervisor that I presented at the 6th International Workshop on Logic-Based program Synthesis and Transformation in Sweden.

As a researching student the phrase *virtual-ivory-tower-experience* comes to mind. Therefore, my thanks must go to authors, in particular R. Tyrell Rockafellar, who most emphatically "rocks", and who has opened the door to the fascinating subject of convex analysis, which I have hardly scraped the surface of in my investigations here. Also to Patrick and Radhia Cousot whose papers on abstract interpretation provided the impetus for this journey. I would also like to thank Tony Daniels, Jacob Howe and Jon Martin for their helpful comments on preliminary chapters and Jan Smaus for his comments on the first attempt at writing up the isomorphism.

On a more personal level my close family, Lawrence, Ivor, Rosalie and Becci have been great. They have put up with burnt offerings disguised as meals with commendable stoicism, ever since the time that twenty five fish fingers went the way of all flesh in my first year undergraduate days as I hit on the algorithm for the Fibonacci sequence. They have been there to listen when things have gone well and when when they haven't, in spite of the fact that none of them speak *computer science*!

I am so very fortunate in knowing and counting as friends, Maria Stergiou and John Crawford here at Kent, Jon Martin and last but not least Tony, husband of my daughter Olwynne, whose support and encouragement have made the last three years bearable.

Finally, Olwynne, my first born, who has supported me through the wonderful ups and awful downs since I became a student in 1989 with a faith in my ability that has been unshakable even in the face of my own despair. Her presence in my life has been a constant joy and her ability to reduce me to laughter when faced by perceived intellectual disaster has not only made this journey easier, it has helped me to learn far more about myself, the world and the meaning of life than I would have done otherwise. Thankyou Olwynne :-)

Chapter 1

Introduction

"Discovery consists of seeing what
everyone else has seen and thinking
what no-one else has thought."
- Albert Szent-Gyorgyi, *The Scientist Speculates*

The computing industry is going through the same metamorphosis that has overtaken the car industry in recent years: the gap between the emphasis on speed and safety as selling-points is narrowing. In computing, the underlying cause of this change lies in that fact that both the complexity of software and the impact it has on our lives as a controlling entity is increasing rapidly. Software is in control in areas as diverse as a real-time safety critical system and a credit-rating facility. A software system is considered to be safe if it can be formally verified that its behaviour is contained within certain measurable parameters. The parameters may range from the safety-critical kind associated with nuclear power stations and aircraft to the more mundane, but none-the-less important criteria that operate over our bank accounts.

Fast imperative languages like C and C++ are the preferred choice, although the rise of the popularity of Java, initially a useful tool for creating applets on web pages, suggests that a change is taking place - Java is presented as being a "safer" language than its C counterparts. Logic based languages are still perceived as too slow and of limited use, despite the fact that logic programming languages and constraint handling systems like Chip V[1] are used in the "real" world. No particular programming language is a universal panacea, but there are certain classes of problems that benefit from expression in a declarative language and a language based on logic has the potential to allow the verification that is going to be required of software systems in the future.

Logic programming was introduced by Robert Kowalski and Alain Colmeraur in the 1970's. The "eureka" step occurred with the realisation that a procedural interpretation could be applied to first order predicate logic. Logic, with a sound basis in mathematics would seem to be an ideal medium in which to specify programs, especially if verification is an issue. The classical logic that is the basis of logic programming languages has been pruned to achieve better computational efficiency. With the passage of time interpreters have given way to compilers and machine architectures have been designed that go a long way towards efficient implementation, notably the work of Taylor [Tay91] and van

[1] cosytec.com

Roy [vR90] for Prolog.

Of the several means and devices that have been explored to improve both declarative and imperative language performance, the automatic analysis of programs is of primary interest here, and in particular the methodology known as abstract interpretation. The initial impetus for abstract interpretation in logic programming was analyses that would allow compiling techniques aimed at more efficient space utilisation and/or program execution time. Increasingly though, the scope of application for such analyses in both declarative and imperative paradigms has been extended to encompass program verification, the provision of debugging information for programmers, and more recently for abstract testing, a generalisation of model-checking [CC00]. In effect, analyses can contribute not only to more efficient, but also to more reliable software.

1.1 Abstract Interpretation in the Analysis of Logic Programs

Patrick and Radhia Cousot formalised the lattice theoretic approach in the late seventies. They developed the theory of abstract interpretation as a means to elicit static analyses that could infer the dynamic properties of programs. Abstract interpretation is applied to program source code in order to carry out a static analysis that can infer an approximation of the meaning of the program's run-time behaviour. In this context approximation means ignoring irrelevant program details and focusing on a particular "property of interest" that can be exploited by a compiler. One source of variant program behaviour is the input data, so, typically, the property of interest is a property associated with data.

The abstract domain that represents the property of interest is the lynch pin of this methodology. A simple example [CC92a] is the abstract domain that approximates that of the positive and negative integers, due to the Ancient Greeks, known as the "rule of signs". Given that the required information is whether a particular variable at a particular program point is positive or otherwise, an abstract interpretation of a program might use the abstract values $+1$ and -1 to describe positive and negative integers respectively. Then, by re-interpreting operations like addition or multiplication according to the rule of signs, certain properties of a program may be deduced. For example, when a particular loop is entered it may be deduced that a certain program variable is positive. Abstract interpretation has been applied to logic programs to deduce a wide range of useful information, including types, modes and aliasing.

The choice of abstract domains for abstract interpretation and a survey of some of the most noteworthy examples in the literature are explored further in Chapter 3.

1.2 Research Objective

This thesis is broadly concerned with the static analysis of logic programs and in particular with abstract interpretation as a methodology. The overall research objective has been the investigation of the possibilities of using abstract domains of a polyhedral character. This idea has been pursued as positive results would mean that logic programming environments that include constraint solvers, like CLP(R) could conduct program analyses using the solving machinery that is already available without having to set up the machinery for an abstract domain, for example like that required for reduced order binary decision trees. The substance of this thesis is in three parts:

1. An investigation into widenings for polyhedral domains qualifies precisely how and when different representations of a polyhedron are possible. As a consequence of this understanding an optimisation technique for polyhedra under certain conditions is noted.

2. Central to the thesis is the presentation of an isomorphism between a subclass of certain polyhedra that are known as polyhedral cones, and a subclass of Boolean functions known as the definite functions. The subclass of polyhedral cones are known as abstract cones and the process of, and motivation for, the abstraction are expounded in Chapter 7. This association highlights the fact that this subclass is a finite abstraction of all polyhedral cones. Employing a finite domain in abstract interpretation has the advantage that termination of the analysis itself is not an issue.

3. Two implementation studies at the prototype level, using polyhedral abstract domains are presented. An argument size analysis is presented that involves a program transform and a static analysis of the run time properties of the transformed program. As the abstract domain is not finite, a simple widening technique is devised that forces the analysis to converge. Any property that can be characterised by definite Boolean functions can be characterised by the abstract cones, so the finite polyhedral domain is employed as an abstract domain for the purpose of groundness analysis for logic programs.

1.3 Thesis Prospectus

The remainder of this thesis is structured as follows:

- Chapters 2, 3 and 4 constitute a reference for the terminology and notation that is required to negotiate this thesis.
  - Chapter 2 comprises some lattice theory terminology and an introduction to logic programming.
  - Chapter 3 introduces abstract interpretation as an analysis methodology along with spatial approximation techniques and some pertinent examples from the literature. This chapter also introduces the notion of widening, a technique for accelerating analysis convergence, or in the case when the analysis domain is infinite, ensuring analysis termination.
  - Chapter 4 introduces the abstract domains of polyhedra and Boolean functions.
- Chapters 5, 6, 7 and 8 constitute the body of this thesis.
  - Chapter 5 presents an analysis that infers useful inter argument relationships with respect to size. This type of information is useful in termination analysis. The abstract domain employed is that of linear inequalities over rational numbers. The sets of inequalities that constitute the analysis output can be viewed spatially as polyhedra in $\mathbb{R}^n$. A technique known as widening is

employed to ensure analysis termination. The analysis is illustrated over a sample of structurally diverse Prolog programs.

  - Chapter 6 is an interesting discussion associated with a well-known widening for polyhedral domains. This widening can be viewed algebraically or spatially. This discussion concentrates on the spatial view of the widening providing insight into how it works and the circumstances when it is most useful. Further, insight is provided into how representations of polyhedra as sets of inequalities can uniquely qualify polyhedra. In the light of this information an alternative widening is suggested, and the association between this widening and the one introduced in the previous chapter is qualified.
  - Chapter 7 introduces abstract convex cones and the isomorphism with definite Boolean functions showing exactly how these two apparently diverse domains are connected.
  - Chapter 8 presents two prototypes that employ the finite domain of abstract convex cones to capture a groundness analysis. The analysis outcomes are commensurate with those for other analyses using domains such as Positive Boolean functions.

- Chapter 9 summarises and suggests further areas for investigation.

Throughout this thesis, a small square - ▪, denotes the end of both Definition and Example environments, and a slightly larger square - ■, denotes the end of Theorem, Proposition, Lemma or Corollary environments.

Chapter 2

Lattice Theory and Logic Programming

Familiarity with lattice and order theory, and with logic programming is assumed but some basic definitions are provided for completeness.
Throughout, the following notational conventions will be adhered to:

- ϕ denotes the empty set,
- in complex formulae the square brackets, $'['$ and $']'$ serve only to clarify the limit of the scope of universal and existential quantifiers.

2.1 Lattice and Order Theory

The sources for the definitions regarding lattice and order theory are taken from [Bir48] and [DP90].

Definition 2.1.1 A *partially ordered set* or *poset* is a set L in which a binary relation $\sqsubseteq$ is defined, which satisfies, for all l_1, l_2, $l_3 \in L$,

1. $l_1 \sqsubseteq l_1$. (Reflexive)
2. if $l_1 \sqsubseteq l_2$ and $l_2 \sqsubseteq l_1$ then $l_1 = l_2$. (Antisymmetric)
3. if $l_1 \sqsubseteq l_2$ and $l_2 \sqsubseteq l_3$ then $l_1 \sqsubseteq l_3$. (Transitive)

A poset L, is denoted $\langle L, \sqsubseteq \rangle$. ▪

Definition 2.1.2 Let $L\langle \sqsubseteq \rangle$, $L'\langle \sqsubseteq' \rangle$ be posets and $\psi : L \to L'$, then ψ is an *(strict) order embedding* if $\forall l_1, l_2 \in L \,.\, l_1 \sqsubseteq l_2 \leftrightarrow \psi(l_1) \sqsubseteq' \psi(l_2)$, $(\forall l_1, l_2 \in L \,.\, l_1 \sqsubset l_2 \leftrightarrow \psi(l_1) \sqsubset' \psi(l_2))$. ▪

Definition 2.1.3 Let $\langle L, \sqsubseteq \rangle$ be a poset and l_1, $l_2 \in L$ then if $l_1 \not\sqsubseteq l_2$ and $l_2 \not\sqsubseteq l_1$, then l_1 and l_2 are said to be *incomparable* and this is denoted $l_1 \not\parallel l_2$. ▪

Definition 2.1.4 Let L be a poset and $L' \subseteq L$, then $a \in L'$ is the *least* element of L' iff $\forall x \in L' \,.\, a \sqsubseteq x$. Similarly, $a \in L'$ is the *greatest* element of L' iff $\forall x \in L' \,.\, x \sqsubseteq a$. ▪

Definition 2.1.5 Let L be a poset and $L' \subseteq L$, then $a \in L$ is an *upper bound* of L' iff $\forall x \in L' \,.\, x \sqsubseteq a$; and a is the *least upper bound* of L' iff $a \sqsubseteq y$ for all upper bounds y of L'. ▪

Definition 2.1.6 Let L be a poset and $L' \subseteq L$, then similarly, $a \in L$ is a *lower bound* of L' iff $\forall x \in L' \,.\, a \sqsubseteq x$; and a is the *greatest lower bound* of L' iff $y \sqsubseteq a$ for all lower bounds y of L'. ▪

Definition 2.1.7 A *lattice*, L, is a poset such that for any two elements, $l_1, l_2 \in L$ there is a greatest lower bound, $l_1 \sqcap l_2$ and a least upper bound, $l_1 \sqcup l_2$. A lattice, L, is denoted by $\langle L, \sqsubseteq, \sqcup, \sqcap \rangle$. ▪

Definition 2.1.8 A *complete lattice*, L, is such that: $\forall L' \subseteq L$ there is a least upper bound or join, denoted: $\sqcup L'$ and a greatest lower bound, or meet, denoted: $\sqcap L'$. A complete lattice, L, is denoted $\langle L, \sqsubseteq, \sqcup, \sqcap, \top, \bot \rangle$, where $\top$ denotes the top or greatest element of L, and $\bot$ denotes the bottom or least element of L. ▪

Note that the definition of a complete lattice means that every finite lattice is complete, but every complete lattice is not finite.

Definition 2.1.9 Let L be a lattice and $\phi \neq L' \subseteq L$. Then L' is a *sub-lattice* of L iff $\forall a, b \in L' \,.\, a \sqcup b \in L' \;\wedge\; a \sqcap b \in L'$. ▪

Hereafter, in accordance with common practice, the least upper bound is referred to as the *join* and the greatest lower bound as the *meet.*

Definition 2.1.10 Let L be a poset, then L is a *chain* iff for any two elements, l_1, $l_2 \in L$ either $l_1 \sqsubseteq l_2$ or $l_2 \sqsubseteq l_1$. ▪

Definition 2.1.11 Let L be a poset and $\phi \neq L' \subseteq L$ is a chain. Then L is *chain-complete* iff $\forall\, L' \,.\, \sqcup\, L'$ exists. ▪

2.2 Logic Programming

The logic programming definitions and descriptions are drawn from Lloyd's seminal work [Llo93], and [Hog90]. Further details can be found in either of these texts. Simple example programs are included to illustrate the logic programming terminology. The analyses that are part of this thesis have been devised for logic programs, and in particular for definite programs consisting of definite clauses. The following definitions clarify these concepts.

2.2.1 A First Order Theory

Definition 2.2.1 A first order theory comprises:

- an alphabet,
- a first order language,
- a set of axioms,

- a set of inference rules.

∎

Definition 2.2.2 An *alphabet* comprises seven classes of symbol:

1. variables,
2. constants,
3. function symbols,
4. predicate symbols,
5. connectives - $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$,
6. quantifiers - $\{\exists, \forall\}$,
7. punctuation symbols - {(,), “,”}.

The first four classes vary from one alphabet to another, the second and third classes may be empty, however, the remaining three classes are the same for all alphabets. ∎

Definition 2.2.3 A *term* is defined inductively:

- a variable is a term,
- a constant is a term,
- if f is an n-ary function symbol and $t_1, \ldots, t_n$, are terms, then $f(t_1, \ldots, t_n)$ is a term.

∎

Definition 2.2.4 A *well-formed formula* is defined inductively:

- if p is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is a well-formed formula, (alternatively an *atomic formula* or simply an *atom*)
- if W_1, W_2 are well-formed formulae then so are their compositions, $\neg W_1$, $W_1 \wedge W_2$, $W_1 \vee W_2$, $W_1 \rightarrow W_2$, $W_1 \leftrightarrow W_2$,
- if W_1 is a well-formed formula and x is a variable then, $\forall x \,.\, W_1$ and $\exists x \,.\, W_1$ are also well-formed formulae. ∎

Definition 2.2.5 A *first order language* of a theory is the set of well formed formulae constructed from the symbols of the alphabet. ∎

Definition 2.2.6 The *scope* of $\forall x$ (respectively $\exists x$) in $\forall x \,.\, F$ (respectively $\exists x \,.\, F$) is F. A *bound occurrence* of a variable in a formula is an occurrence immediately following a quantifier or an occurrence within the scope of a quantifier, which has the same variable immediately after the quantifier. Any other occurrence of a variable is *free*. ∎

Definition 2.2.7 A *closed formula* is a formula with no free occurrences of any variable.
∎

The axioms of a first order theory are a designated subset of closed formulae in the language of the theory.

Definition 2.2.8 A *literal* is an atom or the negation of an atom. A *positive literal* is an atom, a *negative literal* is the negation of an atom. ▪

Definition 2.2.9 A *clause* is a closed formula of the form:

$$\forall x_1 \ldots \forall x_n \,.\, (L_1 \vee \ldots \vee L_m)$$

where each L_i is a literal and $x_1, \ldots, x_n$ are all the variables occurring in $L_1 \vee \ldots \vee L_m$. ▪

A well-founded formula is said to be in clausal form if it is a conjunction of clauses. Clauses in logic programs are assumed to be universally quantified for all variables so a clause:

$$\forall x_1 \ldots \forall x_n \,.\, (A_1 \vee \ldots \vee A_k \vee \neg B_1 \vee \ldots \neg B_m)$$

where $A_1, \ldots, A_k$ and $B_1, \ldots, B_m$ are atoms and $x_1, \ldots, x_n$ are variables occurring in these atoms, which is equivalent to:

$$\forall x_1 \ldots \forall x_n \,.\, (A_1 \vee \ldots \vee A_k) \leftarrow (B_1 \wedge \ldots \wedge B_m)$$

is denoted:

$$(A_1 \vee \ldots \vee A_k) \leftarrow (B_1 \wedge \ldots \wedge B_m)$$

Although the following definitions apply to clauses in a program, since the context is clear, the word *program* is omitted.

Definition 2.2.10 A *definite clause* is a clause of the form:

$$A \leftarrow B_1, \ldots, B_m$$

where A and $B_1, \ldots, B_m$ are atoms, which contains precisely one atom in its consequent. A is called the *head* and $B_1, \ldots, B_m$ the *body*. Each B_i is known as a *body atom*. ▪

Definition 2.2.11 A *definite program* is a finite set of definite clauses. ▪

Definition 2.2.12 A *definite goal* is a clause of the form:

$$\leftarrow B_1, \ldots, B_m$$

that is, a clause with an empty consequent. Each B_i is a *subgoal*. ▪

Definition 2.2.13 A *ground* term (*ground* atom) is a term (atom) that does not contain any variables. ▪

Definition 2.2.14 Let L be a first order language, then the *Herbrand universe* for L is the set of all ground terms formed from the constants and function symbols in L. ▪

Definition 2.2.15 Let L be a first order language, then the *Herbrand base* for L is the set of all ground atoms formed from the predicate symbols in L, with ground terms in the *Herbrand universe* as arguments. ▪

Definition 2.2.16 Let L be a first order language, then a *pre-interpretation* of L consists of:

- a set $D \neq \phi$, the domain of the pre-interpretation,
- for each constant in L, the assignment of an element in D,
- for each n-ary function symbol in L, the assignment of mapping from D^n to D.

■

Definition 2.2.17 Let L be a first order language, then the *Herbrand pre-interpretation*, of L consists of:

- the domain of the pre-interpretation, the Herbrand universe H,
- constants in L are assigned themselves in H,
- if f is an n-ary function symbol in L, then the mapping from H^n to H, defined by $(t_1, \ldots, t_n) \rightarrow f(t_1, \ldots, t_n)$ is assigned to f.

■

Definition 2.2.18 Let L be a first order language, then an *Herbrand interpretation*, I, of L based on the Herbrand pre-interpretation of L is the assignment of a mapping from H^n into $\{$*true*, *false*$\}$, for each n-ary predicate symbol in L. ■

For Herbrand interpretations the assignment to constant and function symbols is fixed so that an Herbrand interpretation, I, can be identified with a subset of the Herbrand base, that is, those atoms in the Herbrand base that are mapped to *true*.

Definition 2.2.19 An interpretation for which a formula expresses a true statement is a *model* of the formula. ■

Definition 2.2.20 Let L be a first order language and well-founded formulae $W_1, W_2 \in L$ then the relation *logical implication*, denoted $\models$, is defined: $W_1 \models W_2$ iff every model of W_1 is a model of W_2. ■

Definition 2.2.21 Let L be a first order language and well-founded formulae $W_1, W_2 \in L$ then the relation *derivability* denoted $\vdash$ is defined: $W_1 \vdash W_2$ iff W_2 is derivable using the inference rules of L. ■

Definition 2.2.22 Let L be a first order language and P a set of clauses, then an *Herbrand model* for P is an *Herbrand interpretation* for L which is a model for P. ■

Definition 2.2.23 Let a set of definite clauses, P, be a program. The *ground instantiation* of P, denoted G_P, is the set of all possible ground instances of all of the clauses in P over the Herbrand universe of P. ■

Definition 2.2.24 A *truth-assignment* to an atom is the assignment to the atom of either the value *true*, or the value *false*. ■

The set of all Herbrand interpretations for a definite program P, forms a complete lattice, ordered by set inclusion [KvE76]. Therefore, there is both a minimal (least element /bottom) and a maximal (greatest element/top) model for P. The maximal model for P is the Herbrand base, B_P, as this ensures that every atom that is a component of a clause in G_P is *true*, and hence, every clause in G_P is *true*. However, B_P may contain many atoms not present in any of the clauses in G_P, so their truth assignment is irrelevant. An Herbrand interpretation, I, is a truth-assignment to each atom in B_P. Applying this assignment to the atoms in G_P determines the truth values of the clauses in G_P, and hence, a truth-assignment to G_P itself. If an interpretation determines that G_P is *true*, then it is a model for G_P, and hence, for P.

Example 2.2.1 Consider the program, P :

```
related(X, florence) <- related(X, olwynne).
related(sophie, olwynne).
```

The Herbrand base, B_P :

```
related(florence, florence).
related(olwynne, florence).
related(sophie, florence).

related(florence, olwynne).
related(olwynne, olwynne).
related(sophie, olwynne).

related(florence, sophie).
related(olwynne, sophie).
related(sophie, sophie).
```

and the ground instantiation of P, denoted G_P :

```
clause 1:   related(florence, florence) <- related(florence, olwynne).
clause 2:   related(olwynne, florence) <- related(olwynne, olwynne).
clause 3:   related(sophie, florence) <- related(sophie, olwynne).
clause 4:   related(sophie, olwynne).
```

■

To find the least model it is necessary to find the smallest set of atoms whose individual assignment to *true* ensures that G_P, is *true*. It is convenient to consider the 4 clauses in reverse order as the truth of clause 4 affects that of clause 3.

- `clause 4` is *true*, only when `related(sophie, olwynne)` is *true*.
- since `related(sophie, olwynne)` is *true*, `clause 3` is *true* only when `related( sophie, florence)` is *true*,
- however, neither `related(sophie, olwynne)` nor `related(sophie, florence)` is in the antecedent or the consequent of `clause 2`, so `clause 2` can be *true* when both atoms, `related(olwynne, florence)` and `related(olwynne, olwynne)`, are *false*,

- similarly, `clause 1` can be *true* when both atoms, `related(florence, florence)` and `related(florence, olwynne)`, are *false*.

Hence, {`related(sophie, olwynne)`, `related(sophie, florence)`} is the smallest set of atoms whose assignment to *true* ensures that G_P is *true*, and hence, is the minimal model for both G_P and P. The minimal model for a program, P, is the smallest set of ground atoms whose assignment to *true* guarantees that every clause in G_P is *true*, that is, the intersection of all the models for P. The meaning of a program, P, is defined by its minimal model which constitutes the theorems that can be proven by the axioms of P. The minimal model of P is said to be the declarative meaning of P.

2.2.2 Fixpoint Semantics and the T_P Operator

Let P be a definite program, then 2^{B_P}, the set of all Herbrand interpretations of P, is a complete lattice under the partial order of set inclusion, $\subseteq$. The top element is B_P and the bottom element is ϕ. The join is set union and the meet set intersection.
There is an incremental method for constructing the minimal model. The first step is to chose an interpretation $I_1 \in B_P$. If there is a clause in G_P such that the interpretation I_1 assigns *true* to the atoms in the body of that clause, then an *immediate consequence* is that the head atom of that clause, the consequent, is also *true*. Let I_2 be all the atoms that are made *true*, then the outcome of applying this method to I_2 will be the third iterate I_3. This method can be applied successively and will generate only models of P. If the initial I is the empty set it has been shown that this sequence of iterates will stabilise and yield the minimal model.

The repeated application of this process can be formalised in the definition of the operator known as the T_P operator, but before it is introduced some further definitions are required.

Definition 2.2.25 Let L and M be posets. A map $\vartheta : L \to M$ is *monotonic* if $a \sqsubseteq b \in L$ implies $\vartheta(a) \sqsubseteq \vartheta(b)$. A monotonic map may also be called *order-preserving.* ▪

Definition 2.2.26 Let L' be a non-empty subset of a poset L. Further, let F be a finite subset of L'. Then L' is a *directed* set if $\forall F \subseteq L' \,.\, \exists a \in L'$ such that a is an upper bound of F. ▪

Definition 2.2.27 Let L and M be posets, and L' a directed set in L. A map $\vartheta : L \to M$ is *continuous* if, $\forall L' \in L \,.\, \vartheta(\sqcup L') = \sqcup \vartheta(L')$. ▪

Note that continuity implies monotonicity.

Definition 2.2.28 Let L be a complete lattice and $T : L \to L$ be a mapping. Then $a \in L$ is the *least fixpoint* of T if a is a fixpoint, that is $T(a) = a$, and for all fixpoints b of T, $a \sqsubseteq b$; and the set $\{b \in L \mid a \sqsubseteq b\}$ is the set of *post fixpoints* of T.
The *greatest fixpoint* is defined similarly. ▪

Let L be a complete lattice, $\langle L, \sqsubseteq, \sqcup, \sqcap, \top, \bot \rangle$, then by Tarski's fixpoint theorem [Tar55], a monotone mapping $T : L \to L$ has both a least fixpoint, *lfpT*, and a greatest fixpoint *gfpT*. Further, $lfpT = \sqcap\{x \mid T(x) = x\}$ and $gfpT = \sqcup\{x \mid T(x) = x\}$.

Definition 2.2.29 Let P be a definite and continuous program and 2^{B_P} the set of all Herbrand interpretations. Then the monotonic operator, $T_P : 2^{B_P} \rightarrow 2^{B_P}$ is defined: $T_P(I) = \{A \in B_P \mid A \leftarrow A_1, \ldots, A_n$ is a ground instance of a clause $\in P$ and $\{A_1, \ldots, A_n\} \subseteq I\}$ ▪

Hence, in the case of logic programs the semantics can be defined as the least fixpoint of the immediate consequences operator. By application of Tarski's fixpoint theorem [Tar55] it can be shown that the least fixpoint of the immediate consequences operator is coincident with the minimal model.
The following example illustrates the construction of the minimal model for a program P starting with the empty set and applying the T_P function until the iteration sequence stabilises. In the example program there are no function symbols so the Herbrand universe is finite.

Example 2.2.2 Let P be a program and H_P be the Herbrand universe of P :

$$\begin{array}{rcl} P & = & \{\texttt{p(X) <- q(X).} \\ & & \ \texttt{q(a).}\} \\ H_P & = & \{\texttt{a, b}\} \end{array}$$

Then the Herbrand base of P, namely B_P, and the ground instantiation of P, namely G_P are:

$$\begin{array}{rcl} B_P & = & \{\texttt{p(a), q(a),} \\ & & \texttt{p(b), q(b)}\} \\ & & \\ G_P & = & \{\texttt{p(a) <- q(a).} \\ & & \texttt{p(b) <- q(b).} \\ & & \texttt{q(a).}\} \end{array}$$

The minimal model of P, the fixpoint of T_P is derived:

$$\begin{array}{rcl} I_0 & = & \phi \\ I_1 = T_P(I_0) & = & \{\texttt{q(a)}\} \\ I_2 = T_P(I_1) & = & \{\texttt{q(a), p(a)}\} \\ I_3 = T_P(I_2) & = & \{\texttt{q(a), p(a)}\} \end{array}$$

Hence, the minimal model, that is, the meaning of P is $\{\texttt{q(a)}, \texttt{p(a)}\}$. ▪

Definition 2.2.30 For a program P, the set of all ground atoms, $\{q \mid P \cup \neg q \vdash \mathit{false}\}$ is the *success set* of P. ▪

Definition 2.2.31 A *substitution*, θ, is a finite set of the form $\{v_1/t_1, \ldots, v_n/t_n\}$ where each v_i is a variable, each t_i is a term distinct from v_i and the variables $v_1, \ldots, v_n$ are distinct. An element, v_i/t_i of a substitution is called a *binding* for v_i. ▪

Definition 2.2.32 A substitution θ, is called a *ground* or *grounding substitution* if all of the t_i are ground terms. ▪

The success set of P is coincident with the minimal model of P [KvE76].

Chapter 3

Abstract Interpretation & Spatial Approximation

The motivations for static program analysis in logic programming are program verification, to provide debugging information for the programmer or to infer information that can be used to improve program performance, in terms of space utilisation or execution speed. Abstract interpretation is a methodology for static analysis that focuses on a description, the abstraction, of a useful program property. When the description domain, the abstract domain, is numeric, representations can become unmanageably large, hence the application of spatial approximation techniques. Section 3.1 serves to introduce abstract interpretation and the technique known as widening, and 3.1.1 provides a brief overview of some of the applications of abstract interpretation, particularly in logic programming. Section 3.2 describes some spatial approximation techniques and includes some applications of abstract interpretation that employ these techniques.

3.1 Abstract Interpretation

Patrick and Radhia Cousot have formalised the methodology known as Abstract Interpretation and a detailed review of this is found in [CC92a, CC92b]. Abstract interpretation is a methodology that formalises the approximation of the run-time behaviour of a program with respect to a particular property of interest. In this context approximation means throwing away all but pertinent information and devising a means of characterising program behaviour in terms of the property of interest. The formal characterisation of program behaviour, the standard semantics, is the basis on which an abstract interpretation is built. Central to an abstract interpretation is the choice of abstract domain. Analyses should be both as efficient and as precise as possible so this choice is tempered by the capacity of the abstract domain to express the program descriptions succinctly and further, the computational complexity of operations on those descriptions. Hence, this choice dictates the efficacy of the analyses.

The relationship between an abstract semantics and a concrete semantics is qualified formally in the most precise way possible, for example, the relationship may be strong enough to be qualified as a Galois connection (see Definition 3.1.3). The formal association between concrete and abstract values allows the specification of sound abstract operations over the abstract domain. The property of interest, or abstraction, and its relationship with the concrete semantics induces the abstract or non-standard semantics, an approximation of the collecting semantics. Since the abstract semantics is an approximation of the collecting semantics it can be described by a fixpoint equation and its solution is the information that the analysis infers (this follows only when the collecting semantics is already a transition system).

Hence, consideration must be given to the operations required to compute the fixpoint, or at least some useful approximation of the fixpoint. Monotonicity is a requirement of fixpoint equations therefore the structure of abstract domains ranges from pre-orders to complete lattices. When analyses are defined over abstract domains with infinite or very long chains, they may require a technique that either forces or accelerates convergence. The notions of widening and narrowing were introduced to manage such situations and are reviewed in [CC92c].

Widening and Narrowing A widening is an operator on two successive iterates in a fixpoint calculation over increasing chains that approximates the fixpoint with a post fixpoint. The loss in precision is compensated for by operational tractability, as in practice post fixpoints can be easier to compute than fixpoints. The following definitions are due to [CC92c].

Definition 3.1.1 A *widening* denoted $\bigtriangledown$, on the preorder $\langle L, \sqsubseteq \rangle$ is defined by $\bigtriangledown : L \times L \to L$ such that $\forall x, y \in L \, . \, x \sqsubseteq x \bigtriangledown y$ and $\forall x, y \in L \, . \, y \sqsubseteq x \bigtriangledown y$ and for all increasing chains $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$, the increasing chain defined by $y_0 = x_0, \ldots, y_{i+1} = y_i \bigtriangledown x_{i+1}, \ldots$ is not strictly increasing, that is, $y_{l+1} \sqsubseteq y_l$ for some l. ▪

Narrowing, denoted $\triangle$, is a similar technique that is applied to decreasing chains.

Definition 3.1.2 A *narrowing* denoted $\triangle$, on the preorder $\langle L, \sqsubseteq \rangle$ is defined by $\triangle : L \times L \to L$ such that $\forall x, y \in L \, . \, (y \sqsubseteq x) \to (y \sqsubseteq (x \triangle y) \sqsubseteq x)$ and for all decreasing chains $x_0 \sqsupseteq x_1 \sqsupseteq \ldots$, the decreasing chain defined by $y_0 = x_0, \ldots, y_{i+1} = y_i \triangle x_{i+1}, \ldots$ is not strictly decreasing, that is, $y_{l+1} \sqsupseteq y_l$ for some l. ▪

Widening and narrowing can be applied independently or in conjunction with one another. Widening is discussed more fully in context in Chapter 5 and both techniques are described in detail in [CC92c]. In the Example 3.1.1, that follows, both techniques are employed. The narrowing effectively comes in to play to tighten up the approximation of the post fixpoint induced by the widening. In the example and throughout this text $\mathcal{P}$ denotes the power set.

Example 3.1.1 This example is taken from [CC92a] but is due to earlier texts by the same authors. The purpose of the abstract interpretation in this example is to derive an analysis that returns the possible range of integer values as a pair of bounds, for a variable with a value that controls the execution of a while loop. Let P denote the concrete domain and A the abstract domain. Consider the abstraction $\alpha \in P(\subseteq) \to A(\sqsubseteq)$ where $P = \mathcal{P}(\mathbb{Z})$, $A = \{[l, u] \mid l \in \mathbb{Z} \cup \{-\infty\} \wedge u \in \mathbb{Z} \cup \{+\infty\} \wedge l \leq u\} \cup \{\phi\}$, $min\mathbb{Z} = -\infty$, $max\mathbb{Z} = +\infty$ such that $\alpha(\phi) = \phi$ and $\alpha(X) = [minX, maxX]$, where $minX, maxX$, respectively, denote the minimum and maximum values of X. Hence, $\forall I \in A \, . \, \phi \sqsubseteq I$ and $[a, b] \sqsubseteq [c, d]$ iff $c \leq a \wedge b \leq d$. Since A has strictly increasing (infinite) chains, a widening is introduced such that $\forall I \in A \, . \, \phi \bigtriangledown I = I \bigtriangledown \phi = I$ and $[a, b] \bigtriangledown [c, d] = [\text{ if } c < a \text{ then } -\infty \text{ else } a, \text{ if } d > b \text{ then } +\infty \text{ else } b]$. The strictly decreasing chains of A are all finite, but can be very long, so a narrowing is defined, such that $\forall I \in A \, . \, \phi \triangle I = I \triangle \phi = \phi$ and $[a, b] \triangle [c, d] = [\text{if } a = -\infty \text{ then } c \text{ else } a, \text{ if } b = +\infty \text{ then } d \text{ else } b]$. The analysis of the following Prolog program:

```
program:- init(X,1), while(X).
init(X,X).
while(X):- inf(X,100,X), write(X),nl,
           Y is X + 2,
           while(Y).
while(X).
```

consists of solving the equation: $X = \big([1,1] \sqcup (X +^A [2,2])\big) \sqcap [-\infty, 99]$ where $\phi +^A I = I +^A \phi = \phi$ and $[a,b] +^A [c,d] = [a +^{A'} c, b +^{A'} d]$ with $-\infty +^{A'} x = x +^{A'} -\infty = -\infty$ and $+\infty +^{A'} x = x +^{A'} +\infty = +\infty$. The ascending abstract iteration sequence with widening is then:

$$\begin{aligned}
&X_0 = \phi\\
&X_1 = X_0 \bigtriangledown (([1,1] \sqcup (X_0 +^A [2,2])) \sqcap [-\infty, 99] = \phi \bigtriangledown [1,1] = [1,1]\\
&X_2 = X_1 \bigtriangledown (([1,1] \sqcup (X_1 +^A [2,2])) \sqcap [-\infty, 99] = [1,1] \bigtriangledown [1,3] = [1,+\infty]\\
&X_3 = X_2 \bigtriangledown (([1,1] \sqcup (X_2 +^A [2,2])) \sqcap [-\infty, 99] = [1,+\infty] \bigtriangledown [1,+99] = [1,+\infty]
\end{aligned}$$

and the descending abstract iteration sequence with narrowing which initialises the first iterate of the sequence, X_0, to the last iterate of the widening sequence, X_3 is:

$$\begin{aligned}
&X_0 = X_3 = [1,+\infty]\\
&X_1 = X_0 \bigtriangleup (([1,1] \sqcup (X_0 +^A [2,2])) \sqcap [-\infty, 99] = [1,+\infty] \bigtriangleup [1,99] = [1,99]\\
&X_2 = X_1 \bigtriangleup (([1,1] \sqcup (X_1 +^A [2,2])) \sqcap [-\infty, 99] = [1,99] \bigtriangleup [1,99] = [1,99]
\end{aligned}$$

The use of both the widening and narrowing means that the analysis is independent of the number of iterations in the while loop. ▪

It is interesting to note that convergence occurs within three iterations for the widening. In the first iteration the base case conditions are established. The second iterate is effectively the interval between the greatest lower bound and the greatest upper bound and this stabilises in the third iterate. The narrowing commences with this interval and tightens the upper bound, in this case to the least upper bound in the first iterate and this sequence converges in the second iterate.

Summary - the Design of an Abstract Interpretation The design of an abstract interpretation, can be seen as a sequence of steps:

1. identify the property of interest
2. choose a concrete collecting semantics that can capture this property - this can be chosen from the standard semantics for a programming language or an instrumented semantics
3. choose an abstract domain that will capture the required property as precisely as possible, balancing computational and representational constraints
4. establish as precise a connection as possible, between the concrete and abstract domains
5. identify the required abstract operations and affirm their soundness

Bearing in mind that the success of an analysis is measured not only in terms of what it delivers but the speed with which it is delivered, it is the third step, the choice of the abstract domain that is the most crucial.

3.1.1 Abstract Interpretation in Logic Programming

Approximating the Concrete Fixpoint Semantics with the Abstract Semantics The choice of concrete semantics is dictated by the property of interest as the semantics must be able to express this property in order for it to be abstracted. Ideally the collecting semantics are the most precise of the standard semantics expressed as a formalised transition system that can capture the property of interest. In logic programming the collecting semantics is typically the fixpoint semantics as characterised by the immediate consequences operator, denoted T_P, (see Definintion 2.2.29). The T_P operator defines the concrete semantics of a logic program as its least fixpoint, $\mathit{lfp}T_P = \bigcup_{n\geq 0} T_P^n(\phi)$ where $\forall n > 0 \,.\, T_P^n(\phi) \subseteq T_P^{n+1}(\phi)$. When the association between the concrete domain and the abstract domian is a Galois connection the properties of this strong association allow the inducing of the abstract (fixpoint) semantics.

Definition 3.1.3 Let $L\langle\sqsubseteq\rangle$, $L'\langle\sqsubseteq'\rangle$ be posets. Further, let $\alpha : L \to L'$ and $\gamma : L' \to L$. A *Galois connection* is a pair of mappings α, γ such that

$$\forall l \in L \;\; \forall l' \in L' \,.\, \alpha(l) \sqsubseteq' l \leftrightarrow l \sqsubseteq \gamma(l')$$

■

Hence, whenever α maps l to l' then $\gamma(\alpha(l))$ is an upper approximation of l.

This means that the analysis can be couched in terms of the solution to a fixpoint equation that is an abstraction of the collecting semantics and hence, the goal of an abstract interpretation over a logic program is $\alpha(\mathit{lfp}T_P)$. One approach is to compute an abstract image of the concrete iteration sequence, that is, an abstraction of $\mathit{lfp}T_P = \bigcup_{n\geq 0} T_P^n(\phi)$. The approach taken in this thesis is to compute the abstract fixpoint over a program that is an abstraction, $P^{\mathcal{A}}$, of the concrete program, P. In an ideal situation, $\alpha(\mathit{lfp}T_P) = \mathit{lfp}T_{P^{\mathcal{A}}}$, but in practice it may be difficult or impossible to compute the least fixpoint of the abstract iteration sequence and so there is recourse to a postfixpoint which, as an upper approximation of a fixpoint, is also correct (by Tarski's fixpoint theorem [Tar55]). In particular this is the case where the abstract domain is infinite as in the argument size analysis presented in Chapter 5.

An Overview In this thesis abstract interpretation as a vehicle for the analysis of logic programs is considered at the source code level as this is primarily where it has proven to be most useful. The two analyses introduced in this thesis are concerned with reasoning about data flow and are at the variable, rather than predicate level. An argument size analysis is presented in Chapter 5 and a groundness analysis in Chapter 8. The motivation for these types of analysis and a closer look at some of the more significant examples in the literature is presented in the respective introductions to these chapters. A brief overview of abstract interpretation in the wider context follows and then the next Section, 3.2, considers spatial approximation techniques.

The efficacy of program analysis is measured primarily in terms of the precision of the analysis and its cost in terms of space and time. If precision is adequate and the cost is cheap enough then the analysis may be incorporated into a compiler and its efficiency measured at a higher level in terms of reductions in code size, compile time and run time for a program. The interested reader is referred to Getzinger's paper [Get94], which despite its age, not only stands as a survey of the various types of analysis, but is also an account of the systematic incorporation of several of these analyses into van Roy's Aquarius compiler [vR90] in order to quantify their relative efficiency. Overall, as one would expect, complex domains are computationally more expensive than simpler ones, and the corresponding increase in program efficiency with more precise analyses is bought with a longer compile time and increased code size. The outstanding observation from Getzinger's work is that a substantial gain at reasonable cost is possible with relatively simple domains. Getzinger's evidence suggests that the primary driver for the choice of abstract domain should be the information that can definitely be used rather than the best precision affordable.

It may be, that for a certain class of program in a particular circumstances, a complex analysis could produce dividends, however, for general purposes abstract interpretation with the simpler domains is the most likely choice for incorporation into any system, commercial or otherwise. The CIAO program development system [HBC$^+$99] is such an example, the result of many ideas and years of collaborative work in the field, is a programming environment that offers the programmer/developer a number of applications of abstract interpretation, including program optimisation, a debugging facility, program validation and an assertion language. Some of the techniques used in the CIAO system have been incorporated into CHIP V, the constraint system used by Cosytec in Paris. It should also be noted that in the wider context there are two commercial companies marketing program verification tools [Cou00]; Polyspace Technologies base their product on the notion of abstract testing [CC00] for C and Ada, whilst Absint Angewandte Informatik GMBH offers, for example, abstract interpretation as a route to high quality C code generation.

3.2 Spatial Approximation Techniques

The driving force behind spatial approximation techniques is to render something difficult to model, simple. Such techniques are also driven by a need for a simple representation of the approximation, especially in the case where there are many variables involved and consequently the dimension is high.

In the following *resumé* of some of the most significant approaches, the figures depict some of the different approximations of the line $y = x$, (the left hand figure), and a triangle (the right hand figure) in the positive orthant of $\mathbb{R}^2$.

Most Specific Generalisation (*msg*) Approximations Argument size analysis can be specified by defining an appropriate pre-interpretation and by using *msg* approximations [GBS95]. *msg* approximations are formed by taking most specific generalisations of sets of atoms. With the aid of query-answer transforms, the analysis of [GBS95] can infer, for example, that the length of the first argument of naive reverse, `rev/2,` is the same as the length of the second argument. Although useful relationships can be inferred the pre-interpretation domain of the natural numbers expressed in terms of 0 and the successor predicate does not allow the expression of inter-argument size relationships of the kind that can be modelled by a non-strict linear inequality.

Affine Approximations The intuitive picture of an affine space is that of an infinite linear space like a line or a plane. In general an affine set $A \subseteq \mathbb{R}^n$, is such that $(1-\lambda)\bar{x} + \lambda\bar{y} \in A$ for every $\bar{x} \in A$, $\bar{y} \in A$, $\lambda \in \mathbb{R}$. This definition includes the empty set, a single point and $\mathbb{R}^n$ itself. In the context of imperative programs, the affine approach has been used to deduce invariant linear relations between the nested indices of DO loops in FORTRAN programs for the purpose of compiler optimisation [Kar76]. In [VD92, GDL92], invariants are characterised by affine spaces which can be represented and manipulated using matrices. In [VD92] abstract interpretation is applied to the task of inferring invariant argument size relations, a principle aid to proving program termination. Relationships over the natural numbers are of the form $\{\langle x_1, \ldots, x_n\rangle \mid c_0 + c_1x_1 + \ldots + c_nx_n = 0\}$ where $c_i \in \mathbb{Z}$. However, affine approximations can be very imprecise as demonstrated by the approximation of the triangle as the whole of $\mathbb{R}^2$, in Figure 1.

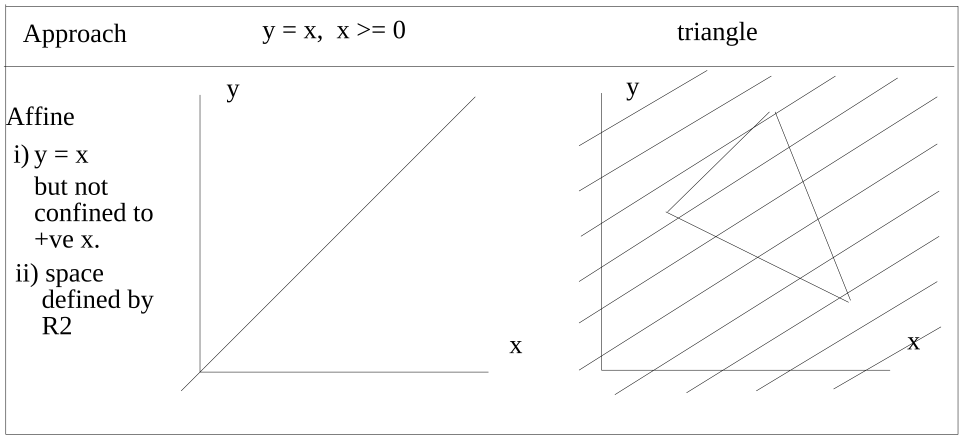


Figure 1: Affine Approximations

Interval Approximations Interval approximations offer more accuracy than the previous approach, but as can be seen from Figure 2, information is confined to lower and upper bounds that are scalars and inter-argument relationships are lost. In the context of termination analysis, as many of the spaces that are approximated are unbound, this method would lose significant relationships that are characterised by spatial boundaries. China [Bag94, Bag96, BGL92] is an analyser for CLP($\mathcal{R}$) and CLP($\mathcal{FD}$) that approximates conjunctions of constraints with bounding boxes. Bounding boxes are rectangular regions with sides parallel to the axes that are derived from sets of constraints using an algorithm that gradually restricts the bounds in order to represent each variable within an interval. Interval widening is required for termination analysis and constraints are solved by propagation around a constraint network [Bag94, Bag96]. An interval analysis for CLP($\mathcal{R}$), not dissimilar to that used in China, is described in [JBE94]. The purpose of the global analysis is program specialisation by adding derived redundant constraints to CLP($\mathcal{R}$) programs. Interval abstraction is employed with an implementation, in C, of GAIA (Generic Abstract Interpretation Algorithm). Narrowing is proposed in order to recover some of the precision lost in widening interval approximations, but not implemented. Results indicate a significant improvement in execution times on the tested

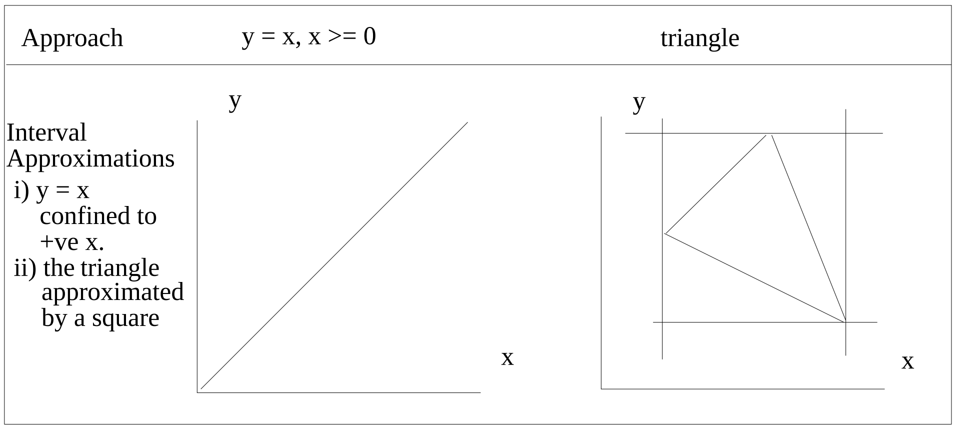

Approach
y = x, x >= 0
triangle
Interval Approximations
i) y = x confined to +ve x.
ii) the triangle approximated by a square
y
x
y
x

Figure 2: Interval Approximations

programs.

Simple Section Approximations Simple sections were developed as a means of detecting task parallelism, for example, parallelism between different loop nests. The idea is to model data accesses in a given program region within the simple sections. Although developed for imperative languages, simple sections [BK89] might be used to characterise simple inter-argument relationships. Simple sections represent spaces that are bound by hyperplanes of the form $x = c$, $x + y = c$ or $x - y = c$ where linear relationships are considered over the variables, x and y, and a constant, c. Intuitively, they are similar to interval approximations but with a further level of accuracy given by bounds that are at 45 degree angles to the horizontal and vertical bounds, as shown in Figure 3.

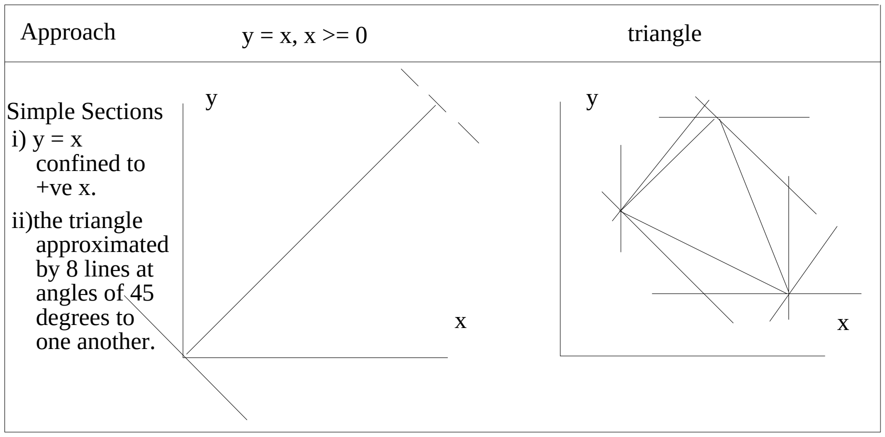

Approach
y = x, x >= 0
triangle
Simple Sections
i) y = x confined to +ve x.
ii)the triangle approximated by 8 lines at angles of 45 degrees to one another.
y
x
y
x

Figure 3: Simple Sections

Although simple sections are less complex to manipulate and do not require CLP($\mathcal{R}$)

support, they cannot afford the precision of polyhedral representations. Further, representations may become intractable for predicates of large arity.

Polyhedral Approximations The greater accuracy that is possible with polyhedral approximations makes this approach attractive and in consequence its suitability has been more widely investigated than other techniques. The polyhedral approach is by far the most expressive and for this reason was chosen as the most suitable, see Figure 4.

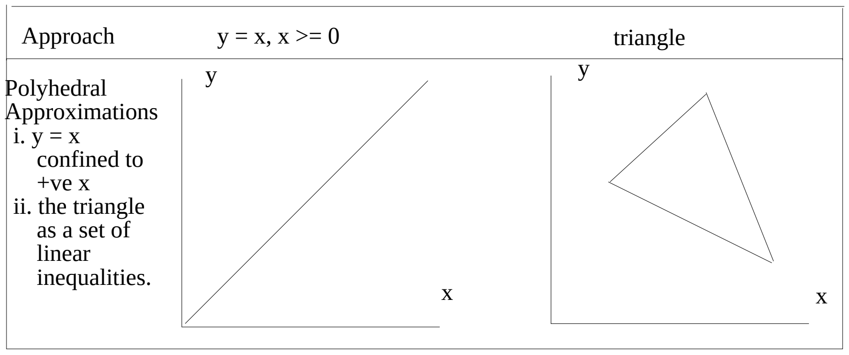

Approach
y = x, x >= 0
triangle
Polyhedral Approximations
i. y = x confined to +ve x
ii. the triangle as a set of linear inequalities.
y
x
y
x

Figure 4: Polyhedral Approximations

- In [MG92] a Prolog III program is presented for checking the invariants of the CLP($\mathcal{Q}$) program to partially mechanise the derivation of inter-argument relationships. The proof method is neither a decision procedure (it is not complete) nor is it automatic since it requires the user to postulate an inter-argument relation to be proven.

- In [van91], an analysis for inferring linear inequalities is proposed with a suite of matrix transformations for mechanising the derivation of the inequalities. However, as the analysis may not converge in finitely many steps a heuristic for finding fixpoints is proposed, otherwise user intervention is required. It is noted that useful fixpoints are not easy to find.

- The argument size analysis for logic programs of [Soh94] builds on the matrix transforms of [van91] but employs a widening to ensure termination. The widening, however, returns the affine hull of the convex hull and therefore loses precision. In order to avoid the widening and its loss of precision, an unfolding transform is proposed for the class of linearly recursive programs. A frame representation is used to check for a fixpoint and this representation is used in the convex union calculation.

- A widening for polyhedra is reported in [Hal79, HPR94] that refines the widening first proposed in [CH78]. Originally used for overflow and array bounds checking, these widenings essentially remove inequalities from the ith polyhedron that are not satisfied by the $(i + 1)$th polyhedron. So that less information is lost the widening reformulates the representation of the first polyhedron in order to maximise the number of common constraints. For example, to calculate $P_1 \bigtriangledown P_2$ where $P_1 = \{\langle x,y\rangle \in \mathbb{R}^2 \,|\, x = 0, y = 0\}$ and $P_2 = \{\langle x,y\rangle \in \mathbb{R}^2 \,|\, 0 \leq y \leq x \leq 1\}$, P_1 is re-expressed as $P_1 = \{\langle x,y\rangle \in \mathbb{R}^2 \,|\, 0 \leq y \leq x \leq 0\}$, yielding a result $P_1 \bigtriangledown P_2 = \{\langle x,y\rangle \in \mathbb{R}^2, |\, 0 \leq y \leq x\}$ rather than $\{\langle x,y\rangle \in \mathbb{R}^2 \,|\, 0 \leq x, 0 \leq y\}$. An algorithm for reformulating P_1 is detailed in [Hal79].

- In [Han95] Handjieva suggests the use of polyhedra to approximate numerical constraints in a program. The idea is that by approximating the constraints using a *backward semantics*, a form of the traditional T_P semantics, a program transform is achieved that has the same semantics as the original, but is computationally more efficient.

- A recent use of polyhedra is that of Howe and King in [HK99] and [HK00]. They propose an abstract interpretation, based on a Galois connection, of finite domain constraint logic programs. One tactic that finite domain constraint solvers employ is that of propagating linear constraints that restrict variable domains. This effectively reduces the search space and expedites problem solution. The analysis they propose allows the addition of supplementary constraints at compile time, as a program specialisation that preserves the semantics of the original program. These constraints may restrict variable domains further than is normally possible with the propagation tactics employed by the solver. Some of the analysed programs exhibited a significant execution time improvement and no analysed program had a slower execution time.

- Besson et al [BJT99] introduce a polyhedral approximation of the set of reachable states in a reactive system implemented in the synchronous language SIGNAL. The purpose of the analysis is program verification rather than a compiler optimisations. A reactive system monitoring the condition of some object may be set up to raise an alarm under certain conditions and possibly set in motion stabilising or recovery procedures. Verification in this context amounts to infer the prescribed conditions for the alarm state and confirming any subsequent states as stable or recovered. They define an operational semantics for SIGNAL and analyse an abstraction of the source code. Faced with the usual problems associated with abstract domains that contain infinite chains, two widening techniques are introduced. One of the widening techniques is based on a heuristic for the choice of an extra extreme ray. The second is based on the notion that certain increasing bounds can be replaced by their normed linear combination, as shown in Figure 5, where the increasing sequence of inverted triangles bound by the line $y = 2$ can be approximated by the infinite band $1 \leq y \leq 2$ rather than just the half space $y \leq 2$. The analysis is proven correct and the proposed widenings do ensure convergence in circumstances when the widening of [CH78] would lose information.

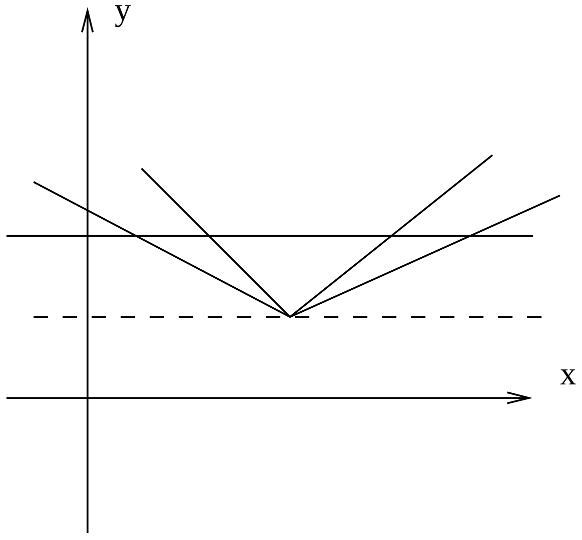


Figure 5: Infinite band $1 \leq y \leq 2$ vs. the half space $y \leq 2$

Chapter 4

Abstract Domains

The purpose of this chapter is to provide points of reference for the terminology relating to the abstract domains in this thesis. Both polyhedra and Boolean functions are considered over a totally ordered, finite set of variables, X, where $n = |X|$. It is sufficient to consider a finite set of variables because at each step an abstract interpreter considers only the variables in a particular clause, and of necessity these will be finite in number. The definitions presented here are taken predominantly from R. Tyrell Rockafellar's Convex Analysis [Roc70], with a few from S. Lay's Convex Sets and their Applications [Lay82].

4.1 Polyhedra and Cones

Throughout the text, the following conventions are adhered to, unless otherwise stated. In accordance with convention, $\mathbb{R}^n$ denotes n-dimensional space over real numbers and $\bar{x}$ is an n-tuple, $\langle x_1, \ldots, x_n \rangle$, denoting a point (column vector) in $\mathbb{R}^n$. Sets of points in $\mathbb{R}^n$ are usually denoted by the letter S, but in order that they be easily distinguishable from other sets, a polyhedron is denoted by a letter P. Similarly, a polyhedral cone is denoted by a letter C. Since polyhedra (cones or otherwise) are viewed as sets of points, $P \in \mathbb{R}^n$ is represented by $\{\bar{x} \in \mathbb{R}^n \mid c_1 \ldots c_m\}$, where the c_is are linear inequalities. However, for brevity, the representation of P may be written $\{c_1, \ldots, c_m\}$, where, in accordance with convention, the implicit conjunction of the constraints represented by the linear inequalities is denoted by a comma rather than $\wedge$. Also for brevity, a pair of opposing linear inequalities may be represented as an equality, for example, $\{x = y\}$ instead of $\{x \leq y, y \leq x\}$; and, in diagrams, if the polyhedron is a point it will be labelled as an n-tuple rather than a set of inequalities. It should be noted that set operations on polyhedra are operations on sets of points not operations on the sets of inequalities used to represent them. Should it be necessary to consider operations on the inequalities in the representation, it will be made clear in the text or by additional notation that will be defined in advance.
The following definitions serve to define the context for the polyhedral domains.

Definition 4.1.1 Let $\bar{x}$, $\bar{x}' \in \mathbb{R}^n$ then the *inner product* of two vectors, $\bar{x}$ and $\bar{x}'$ is denoted: $\langle \bar{x}, \bar{x}' \rangle = \Sigma_{i=1}^{n} x_i {x_i}'$. ▪

Definition 4.1.2 *Positive scalar multiplication* of a set of points, $S \subseteq \mathbb{R}^n$, $\lambda > 0$ is defined: $\lambda S = \{\lambda \bar{x} \mid \bar{x} \in S\}$. ▪

Definition 4.1.3 Let $S \subseteq \mathbb{R}^n$, then S is *affine* iff $\forall \bar{x}, \bar{y} \in S \,.\, \forall \lambda \in \mathbb{R} \,.\, [(1-\lambda)\bar{x} + \lambda\bar{y} \in S]$. ∎

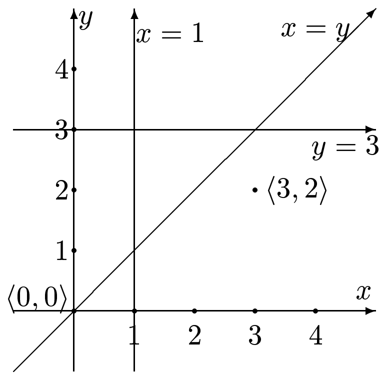
Definition 4.1.4 The *subspaces* of $\mathbb{R}^n$ are the affine sets which contain the origin. ∎

Definition 4.1.5 Let $S \subseteq \mathbb{R}^n$, then the *affine hull* of S, denoted $\text{aff}\, S$ is the smallest affine set containing S. ∎

Definition 4.1.6 A *hyperplane* is an affine set of dimension $(n-1)$ in $\mathbb{R}^n$. ∎

Example 4.1.1 (Examples of affine sets) Consider some of the affine spaces of $\mathbb{R}^2$, defined over $X = \{x, y\}$.

affine set		*subspace of* $\mathbb{R}^2$	*hyperplane of* $\mathbb{R}^2$
$\{\langle x, y\rangle \mid x = 3, y = 2\}$	point	no	no
$\{\langle x, y\rangle \mid x = 0, y = 0\}$	point	yes	no
$\{\langle x, y\rangle \mid x = 1\}$	line	no	yes
$\{\langle x, y\rangle \mid y = 3\}$	line	no	yes
$\{\langle x, y\rangle \mid x = y\}$	line	yes	yes



Affine sets in $\mathbb{R}^2$

∎

Definition 4.1.7 A sum: $\lambda_1\bar{x}_1 + \ldots + \lambda_k\bar{x}_k$ is a *convex combination* of $\bar{x}_1 \ldots \bar{x}_k$ iff $\forall \lambda_i \,.\, 0 \leq \lambda_i \wedge \Sigma_{i=1}^{k} \lambda_i = 1$. ∎

Definition 4.1.8 Let $S \subseteq \mathbb{R}^n$, then S is a *convex set* iff it contains all the convex combinations of its elements. ∎

Intuitively, in any set of points, if all the points in a line joining any two points in the set are within the set itself, then the set is convex.

Definition 4.1.9 Let $S \subseteq \mathbb{R}^n$, then the *convex hull* of S, denoted $conv(S)$, is the intersection of all the convex sets that contain S, that is, $conv(S) = \cap\{S' \in \mathbb{R}^n \mid S \subseteq S', S' \text{ is a convex set}\}$ ∎

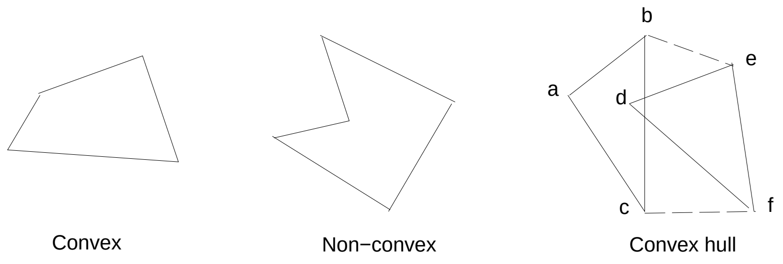


Figure 6: Convexity

Example 4.1.2 Figure 6 illustrates convexity and the convex hull, `a,b,e,f,c` of two convex sets, the triangles `a,b,c` and `d,e,f`. ▪

Given any arbitrary collection of convex sets in $\mathbb{R}^n$, there is a unique largest convex set included in all of those sets, their intersection, and a unique smallest set containing all of those sets, their convex hull. In other words, the collection of all convex sets in $\mathbb{R}^n$ is a complete lattice under the natural partial ordering corresponding to inclusion.

Definition 4.1.10 The *open ball*, with centre $\bar{x}$ and radius $\delta \in \mathbb{R}$ is denoted $B(\bar{x}, \delta) = \{\bar{y} \in \mathbb{R}^n \mid d(\bar{x}, \bar{y}) < \delta\}$, where $\delta > 0$ and $d(\bar{x}, \bar{y}) = \langle \bar{x} - \bar{y}, \bar{x} - \bar{y} \rangle^{1/2}$ returns the distance between two points in $\mathbb{R}^n$. ▪

Definition 4.1.11 Let $S \subseteq \mathbb{R}^n$ and $\bar{x} \in S$, then $\bar{x}$ is an *interior* point of S iff $\exists \delta > 0$ such that $B(\bar{x}, \delta) \subset S$. The set S is *open* iff for all $\bar{x} \in S$, $\bar{x}$ is an interior point of S ▪

Definition 4.1.12 A set S is *closed* iff its complement, $\sim S = \mathbb{R}^n / S$ is open. The *closure* of a set S is the intersection of all of the closed sets containing S and is denoted $cl(S)$. ▪

Definition 4.1.13 Let $\bar{x} \in \mathbb{R}^n, \bar{\mu} \in \mathbb{R}^n$ such that $\bar{\mu} \neq \bar{0}$ and $\eta \in \mathbb{R}$, then a *closed half-space* in $\mathbb{R}^n$ is a set of points:

$$\{\bar{x} \mid \langle \bar{x}, \bar{\mu} \rangle \leq \eta\}$$

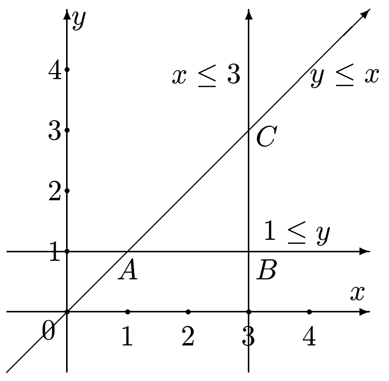
▪

Definition 4.1.14 A *polyhedron* is a set of points in $\mathbb{R}^n$ which can be expressed as the intersection of a finite collection of closed half-spaces, that is, as the solution set of some finite system of linear inequalities of the form

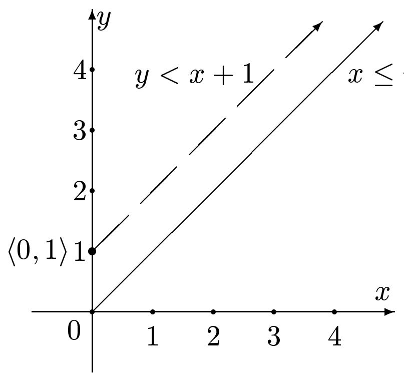
$$\{\bar{x} \mid \langle \bar{x}, \bar{\mu}_i \rangle \leq \eta_i\} \qquad i \in 1, \ldots, m$$

▪

Thus a closed half-space is a set of points that satisfy a non-strict linear inequality and a polyhedron is a set of points that satisfy the conjunction of a set of non-strict linear inequalities. For example, $P = \{1 \leq y,\ y \leq x, x \leq 3\}$ represents a triangle in $\mathbb{R}^2$, as in Figure 7.1. The preceding definition means that polyhedra are both convex and closed and $Poly^n$ denotes the set of polyhedra in $\mathbb{R}^n$.
Since the convex hull of two or more polyhedra may not be closed, the join operator is defined as the closure of the convex hull. The convex hull of the union of a bounded space



1. Triangle ABC in $\mathbb{R}^2$



2. Convex hull of $\{x = 0, y = 1\}$ and $\{0 \leq x, x \leq y, y \leq x\}$ in $\mathbb{R}^2$ is $\{0 \leq y, x \leq y, \underline{y < x+1}\} \cup \{\langle 0,1 \rangle\}$

Figure 7: Triangle and Convex Hull example in $\mathbb{R}^2$

and an unbounded space will not be closed, for example, in Figure 7.2, the "bound" $y < x + 1$ is not closed.

Proposition 4.1.1 $\langle Poly^n, \subseteq, \overline{\cup}, \cap \rangle$ is a lattice under the natural partial ordering of set inclusion. The top element is $\mathbb{R}^n$, the bottom element is the empty set, the closure of the convex hull is the join, denoted $\overline{\cup}$, and the meet is intersection.

Proof It is sufficient to show that the closure of the convex hull is the join for polyhedra. Let $P_i \in Poly^n$. Since all P_i are convex, by definition, $conv(P_1 \cup \ldots \cup P_n)$ is the smallest convex set containing all of the P_is. There are two cases to consider:

- If $conv(P_1 \cup \ldots \cup P_n)$ is closed, then $conv(P_1 \cup \ldots \cup P_n) \in Poly^n$.
- If $conv(P_1 \cup \ldots \cup P_n)$ is not closed then, by the definition of closure and [Roc70] [Theorem 19.6], the smallest closed polyhedral convex set containing $conv(P_1 \cup \ldots \cup P_n)$ is $\mathrm{cl}\,(conv(P_1 \cup \ldots \cup P_n))$. Since $\mathrm{cl}\,(conv(P_1 \cup \ldots \cup P_n))$ is polyhedral $\mathrm{cl}\,(conv(P_1 \cup \ldots \cup P_n)) \in Poly^n$.

By the definition of closure, if a convex set S is closed then $S = \mathrm{cl}\, S$, therefore in general the smallest polyhedron containing an arbitrary collection $P_1, \ldots, P_n$ of polyhedra is $\mathrm{cl}\,(conv(P_1 \cup \ldots \cup P_n))$. Hence the closure of the convex hull is the join for $Poly^n$. ■

Definition 4.1.15 A subset S of $\mathbb{R}^n$ is called a *cone* iff it is closed under positive scalar multiplication. ▪

Definition 4.1.16 A *polyhedral cone* is a set of points in $\mathbb{R}^n$ that can be expressed as the intersection of a finite system of closed half-spaces

$$\{\bar{x} \mid \langle \bar{x}, \bar{\mu}_i \rangle \leq \bar{0}\} \qquad i \in 1, \ldots, m.$$

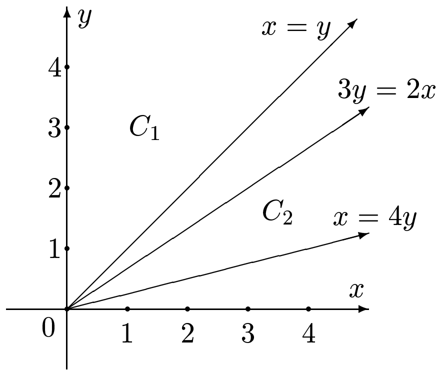
▪

This means that a cone is polyhedral iff it can be expressed as the intersection of a finite set of closed half spaces whose boundary hyperplanes pass through the origin.

Example 4.1.3 (Examples of Polyhedral Cones) In the table that follows the representation of each cone is a set of implicitly conjoined linear inequalities:

polyhedral cones in $\mathbb{R}^2$	*representation*
non-negative orthant	$\{0 \leq x, 0 \leq y\}$
origin	$\{0 \leq x,\ x \leq 0,\ 0 \leq y,\ y \leq 0\}$
C_1	$\{0 \leq x, x \leq y\}$
C_2	$\{0 \leq y, 3y \leq 2x,\ x \leq 4y\}$

Note that both C_1 and C_2 are subsets of the non-negative orthant in $\mathbb{R}^2$.



Polyhedral cones C_1 and C_2 in $\mathbb{R}^2$

■

The preceding definition means that polyhedral cones are cones that are closed, convex and include the origin. The set of all polyhedral cones defined over a set of n variables X, is denoted $Cone^n$, and therefore, $Cone^n \subset Poly^n$. The following definition is required for Proposition 4.1.2 that demonstrates that $Cone^n$ is a lattice.

Definition 4.1.17 Let S_1, S_2 be convex sets in $\mathbb{R}^n$, then their *sum* is defined:

$$S_1 + S_2 = \{\bar{x_1} + \bar{x_2} \mid \bar{x_1} \in S_1, \bar{x_2} \in S_2\}$$

■

Proposition 4.1.2 $\langle Cone^n, \subseteq, \overline{\cup}, \cap \rangle$ is a lattice.

Proof Since polyhedral cones, like all polyhedra, are ordered by set inclusion it is sufficient to demonstrate that (i) the intersection of an arbitrary collection of polyhedral cones is a polyhedral cone, and (ii) the convex hull of an arbitrary collection of polyhedral cones is a polyhedral cone.

- (i) [Roc70][Theorem 2.5] states that the intersection of an arbitrary collection of convex cones is a convex cone, and the intersection of an arbitrary collection of polyhedra is a polyhedron. Therefore the intersection of an arbitrary collection of polyhedral cones is both a cone and polyhedral, that is, a polyhedral cone.

- (ii) Let C_1, $C_2 \in \mathit{Cone}^n$. [Roc70][Theorem 3.8] states that $\mathit{conv}(C_1 \cup C_2) = C_1 + C_2$ and $C_1 + C_2$ is a convex cone. Further, [Roc70][Corollary 19.3.2] states that the sum of two polyhedral convex sets is polyhedral. Hence, $\mathit{conv}(C_1 \cup C_2) = C_1 + C_2$ is a polyhedral cone. Since addition is both associative and commutative, the convex hull of an arbitrary collection of polyhedral cones is a polyhedral cone. ■

Note that the convex hull of two polyhedral cones is always polyhedral and therefore always closed, so, it is with a slight abuse of notation that $\overline{\cup}$ denotes the join operator for Cone^n.
The convex cones that are considered in this work are confined to the non-negative orthants of $\mathbb{R}^n$. These cones are, by definition, unbounded and it is useful to be precise about the direction in which they are unbounded. The following definitions are required.

Definition 4.1.18 Let $A \subseteq \mathbb{R}^n$ be an affine set and $\bar{\eta} \in \mathbb{R}^n$, then the *translate* of A by $\bar{\eta}$ is defined by the set: $\{\bar{x} + \bar{\eta} \mid \bar{x} \in A\}$. An affine set A is *parallel* to an affine set A' iff they are translates of one another. ■

The notion of a translate can now be extended from affine sets to half-lines.

Definition 4.1.19 Let $\bar{y} = \langle y_1, \ldots, y_n \rangle$, $\bar{y}' = \langle y'_1, \ldots, y'_n \rangle$ be points in $\mathbb{R}^n$ and $l = \{\lambda \bar{y} \mid \lambda \in \mathbb{R} \wedge \lambda \geq 0\}$, $l' = \{\lambda \bar{y'} \mid \lambda \in \mathbb{R} \wedge \lambda \geq 0\}$ be half-lines in $\mathbb{R}^n$. l and l' are *translates* of one another iff $\operatorname{aff} l$ is parallel to $\operatorname{aff} l'$. ■

Consider an unbounded convex set C and a point $\bar{x} \in C$. Intuition dictates that C must contain some entire half-line starting at x, otherwise C cannot be unbounded. Each closed half-line in $\mathbb{R}^n$, then, should have a direction and two half-lines have the same direction if and only if they are translates of one another. Hence, a direction of $\mathbb{R}^n$ is an equivalence class of the collection of all closed half-lines of $\mathbb{R}^n$ that are translates of one another. This allows a formal definition of the directions of an unbounded convex set. An unbounded convex set is said to recede in certain directions that are qualified by the following Definition 4.1.20.

Definition 4.1.20 Let S be a non-empty convex set in $\mathbb{R}^n$, then S *recedes in the direction* of $\bar{y}$ where $\bar{y} \neq \bar{0}$, iff $\bar{x} + \lambda \bar{y} \in S$ for every $\lambda \geq 0$ and $\bar{x} \in S$. ■

Definition 4.1.21 Let S, S' be non-empty convex sets in $\mathbb{R}^n$, such that S recedes in the direction of $\bar{y}$. S' recedes in the *opposite direction* to the direction that S recedes in iff $\forall \bar{x} \in S' . \forall \lambda \geq 0 . [\bar{x} + \lambda(-\bar{y}) \in S']$. ■

Informally the recession cone of a convex set can be viewed as the set of directions that the convex set recedes in, augmented with the zero vector. The formal definition follows and this completes the context definitions for the polyhedral domains.

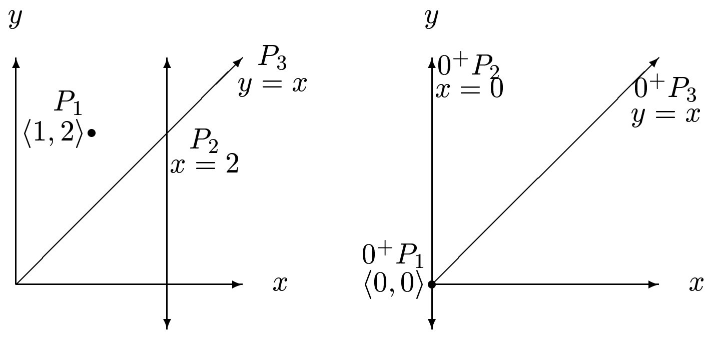
Theorem 4.1.1 [Roc70][Theorem 8.1] Let S be a non-empty convex set in $\mathbb{R}^n$ and $\bar{y} \in \mathbb{R}^n$. The recession cone of S, denoted 0^+S, is a convex cone containing the origin, defined:

$$0^+S = \{\bar{y} \mid S + \bar{y} \subseteq S\}$$

■

Example 4.1.4 The following examples of polyhedra and their recession cones are in $\mathbb{R}^2$.

Polyhedron	Recession Cone
$P_1 = \{x = 1, y = 2\}$	$0^+P_1 = \{x = 0, y = 0\}$
$P_2 = \{x = 2\}$	$0^+P_2 = \{x = 0\}$
$P_3 = \{x = y,\ x \geq 0,\ y \geq 0\}$	$0^+P_3 = \{x = y,\ x \geq 0,\ y \geq 0\}$



■

4.2 Def_X a Class of Boolean Function

Throughout this thesis Boolean functions are represented by propositional formulae. The definitions in this section establish the distinguishing characteristics of Boolean functions in Def_X.

Definition 4.2.1 Let $Bool = \{false, true\}$ and $Bool^n$ denote the set of n-tuples of values in $Bool$; then a Boolean function is a mapping from $Bool^n$ to $Bool$. Let the set of Boolean functions over a set of variables, X, be $Bool_X$. Let $f \in Bool_X$ and $|X| = n$, then $f : Bool^n \to Bool$, where the ith element of the Boolean n-tuple is associated with the i-th element of the set, X. ■

This means that a Boolean function defined over n variables maps 2^n n-tuples of Boolean values to a single Boolean value. An example follows.

Example 4.2.1 Let $X = \{x, y\}$, the Boolean function represented by the formula $x \wedge y$, is described by the following set of mappings:

$$\begin{array}{l}\{\langle true, true\rangle \mapsto true,\\ \langle true, false\rangle \mapsto false,\\ \langle false, true\rangle \mapsto false,\\ \langle false, false\rangle \mapsto false\}\end{array}$$ ■

An n-tuple that is mapped by a function to $true$ is a model for that function. By defining a mapping, $truevar$, from n-tuples of Boolean values to the powerset of X, the mapping from a function to its set of models in terms of the variables in its representation is clarified.

Definition 4.2.2 *truevar*: $Bool^n \rightarrow \mathcal{P}(X)$
Let $\langle b_1, \ldots, b_n \rangle \in Bool^n$, and $X = \{x_1, \ldots, x_n\}$, then:

$$truevar(\langle b_1, \ldots, b_n \rangle) = \{x_i \in X \mid b_i = true\}$$
■

Definition 4.2.3 $model_X\colon Bool_X \rightarrow \mathcal{P}(\mathcal{P}(X))$ is defined:

$$model_X(f) = \{truevar(\bar{b}) \mid f(\bar{b}) = true\}$$
■

An example follows.

Example 4.2.2 Let $X = \{x, y\}$, the function represented by the formula $x \vee y$, returns the following mappings:

$$\begin{array}{l} \{\langle true, true\rangle \mapsto true \\ \langle true, false\rangle \mapsto true, \\ \langle false, true\rangle \mapsto true, \\ \langle false, false\rangle \mapsto false\} \end{array}$$

$$\begin{array}{rl} \text{then} & truevar(\langle true, true\rangle) = \{x, y\} \\ & truevar(\langle true, false\rangle) = \{x\} \\ & truevar(\langle false, true\rangle) = \{y\} \\ \\ \text{hence} & model_X(x \vee y) = \{\{x, y\}, \{x\}, \{y\}\} \end{array}$$
■

Since Boolean functions are distinguished by their sets of models, there is a natural partial ordering corresponding to set inclusion.

Definition 4.2.4 Let f_1, $f_2 \in Bool_X$, then f_1 *entails* f_2, denoted $f_1 \models f_2$, iff $model_X(f_1) \subseteq model_X(f_2)$. ■

Definition 4.2.5 A function $f \in Bool_X$ is *positive* iff $X \in model_X(f)$. The set of positive Boolean functions over X is denoted Pos_X. ■

Definition 4.2.6 A Boolean function f is *definite* iff $f \in Pos_X$ and $\forall M, M' \in model_X(f) \,.\, [(M \cap M') \in model_X(f)]$. The set of functions in Pos_X that are definite is denoted Def_X. ■

By Definition 4.2.6, a function in Def_X has a set of models that are closed under intersection, however, a Boolean function may have a set of models that is closed under intersection but not be in Pos_X, as in Example 4.2.3.

Example 4.2.3 Let $X = \{x, y\}$ and $f = x \wedge \neg y$. Therefore, $model_X(f) = \{\{x\}\}$ which is closed under intersection, but $X \notin model_X(f)$ and therefore $f \notin Pos_X$, and consequently $f \notin Def_X$. ■

Definition 4.2.7 Let $\uparrow S = \{M' \mid M \in S \wedge M \subseteq M' \subseteq X\}$. A function $f \in Bool_X$ is said to be *monotonic* iff $\uparrow model_X(f) = model_X(f)$. The set of monotonic Boolean functions over X is denoted Mon_X. ■

Both $\langle Def_X, \models, \dot{\vee}, \wedge\rangle$ and $\langle Pos_X, \models, \vee, \wedge, \rangle$ are complete lattices, where the join in Def_X is denoted by $\dot{\vee}$, and $f_1 \dot{\vee} f_2 = \wedge\{f \in Def_X \mid (f_1 \models f) \wedge (f_2 \models f)\}$ [AMSS98]. The meet in both cases is classical conjunction, however, whilst the join in Pos_X is classical disjunction this is not the case in Def_X, as shown in Example 4.2.4.

Example 4.2.4 Let $X = \{x, y\}$, then since $X \in \mathit{model}_X(x \leftarrow y) = \{\phi, \{x\}, \{x, y\}\}$, and $\{\phi, \{x\}, \{x, y\}\}$, is closed under intersection it follows that $(x \leftarrow y) \in \mathit{Def}_X$. However, $(x \vee y) \in \mathit{Pos}_X$ but $\{x\}, \{y\} \in \mathit{model}_X(x \vee y)$ and $\{x\} \cap \{y\} = \phi$ and $\phi \notin \mathit{model}_X(x \vee y)$ therefore $(x \vee y) \notin \mathit{Def}_X$. ▪

The join in Def_X is computed from formulae in Orthogonal Reduced Monotonic Body Form, as follows [AMSS98]:

Definition 4.2.8 Let $f \in \mathit{Def}_X$ and $f = \bigwedge_{i=1}^{n} x_i \leftarrow M_i$ where $M_i \in \mathit{Mon}_X \setminus \{x_i\}$, and $\forall S \subseteq X \,.\, f \wedge \bigwedge S \models x_i \leftrightarrow \bigwedge S \models M_i \vee x_i$, then f is in Orthogonal Reduced Monotonic Body Form. Further, let $f' = \bigwedge_{i=1}^{n} x_i \leftarrow M'_i$ also be in Def_X and in ORMBF. Then, $f \dot{\vee} f' = \bigwedge_{i=1}^{n} (x_i \leftarrow (M_i \wedge M'_i))$. ▪

The orthogonal form of RMBF ensures that transitive dependencies are explicit. An example that illustrates the computation of the join for Def_X and how it differs from classical disjunction, follows:

Example 4.2.5 Let $X = \{x, y, z\}$ and $f_i \in \mathit{Def}_X$, where $f_1 = x \wedge (y \leftrightarrow z)$ and $f_2 = y \wedge (x \leftrightarrow z)$ then in ORMBF, $f_1 = (x \leftarrow \mathit{true}) \wedge (y \leftarrow z) \wedge (z \leftarrow y)$ and $f_2 = (x \leftarrow z) \wedge (y \leftarrow \mathit{true}) \wedge (z \leftarrow x)$. Hence, $f_1 \dot{\vee} f_2 = x \leftarrow (\mathit{true} \wedge z) \wedge y \leftarrow (z \wedge \mathit{true}) \wedge z \leftarrow (x \wedge y) = (x \wedge y) \leftrightarrow z$. Note that $\mathit{model}_X((x \wedge y) \leftrightarrow z) = \{\phi, \{x\}, \{y\}, \{x, y, z\}\}$ whereas the set of models for the outcome of classical disjunction is $\mathit{model}_X(x \wedge (y \leftrightarrow z) \vee y \wedge (x \leftrightarrow z)) = \{\{y\}, \{x, y, z\}\}$. ▪

Definition 4.2.9 Let $f_1, f_2 \in \mathit{Bool}_X$, then f_1 is *logically equivalent* to f_2, denoted $f_1 \equiv f_2$, iff $f_1 \models f_2 \wedge f_2 \models f_1$. ▪

Chapter 5

Argument Size Analysis with CLP($\mathcal{R}$)

5.1 Introduction

Inter-argument size relationships have a variety of uses. Some examples follow: In IBM Prolog Horspool [Hor90] proposed the use of argument size relationships for improving the memory management of cdr-coded lists by representing them as arrays. In Reform Prolog [Mil90], in order to avoid overheads incurred by run-time unfolding, during compilation the `for` loop is used to implement recursion; in this context argument relationships can allow structurally recursive predicates to be recognised and the compiler can deduce the recursion bound by inspection of the input arguments. Argument relationships can also be employed in planning the evaluation of queries in deductive databases [Ull85] and in optimising database queries [KS93].

Further, argument size relations can play a significant role in termination analysis. The aim of termination analysis, is to answer the question: Is this predicate guaranteed to terminate for some class of queries? and if so to identify the class of queries. Termination analysis is useful in program verification, but also on other counts. For example, as a de-bugging aid, if a programmer believes that a program will always terminate and an analysis fails to infer just that, then the program (and perhaps the analyser) require scrutiny [SSS97]. Such an analysis is also useful to an optimising compiler that might transform a program, as it is essential that the transform is a program that terminates for the same class of queries as the original program. The part argument size analysis can play in deducing termination properties lies in the association of some notion of size with a predicate call. Then if it is known that terms are rigid (their size is constant over all possible substitutions) and if certain inter-argument size relations can be proven, termination is guaranteed. The underlying principle is that a notion of term size, formalised by Plümer [Plu95], is established and if the size of an input term is strictly decreasing from call to recursive call, then the derivation will necessarily be finite, as size is a strictly non-negative attribute. Van Gelder and Ullman first proposed the use of constraints amongst argument sizes to model the necessary conditions for termination in [UvG88]. For structurally recursive predicates like `append/3`, as the input argument of the recursive body call is a subterm of the input argument in the initial call queries will terminate. However, for predicates that have recursive behaviour, but whose input arguments to their recursive body calls are not strict subterms of those in the initial call, the situation is more complex. Argument size relationships at the auxiliary predicate level are required to infer the argument size relationships of the predicate at the uppermost level. For example, for `quicksort/2`, defined in the table of analyses on page 54, this means inferring that the recursive body call to `quicksort/2` is smaller than that of the initial call, by determining the argument size relationships for the auxiliary predicate `partition/4`. The work presented here, based on [BK96], is an argument size analysis able to infer useful relationships for predicates like `quicksort/2`. This work is an abstract interpretation of logic programs. To keep the implementation simple the analyses here are primarily concerned with lists which are ubiquitous in recursive logic programs, however, the techniques involved can be extended to other size metrics, for example terms [CT97] and level mappings [MK97].

An abstract program is devised and argument size information is captured in the form of sets of inequalities that can be viewed as polyhedra. The approach to termination analysis that is employed in this work is attractive on two counts. By employing a relaxation technique used in disjunctive constraint programming [DB93], the solver and projection machinery of CLP($\mathcal{R}$) and the clp(Q,R) libraries of SICStus are able to compute the convex hull as the solution to a system of linear inequalities. This approach is advantageous as it has not been necessary to manipulate matrices as in [Kar76, van91, VD92]; frames as in [CH78, Hal79]; or implement the Chernikova conversion mechanism as in [Wil93]. Further, results are comparable or better than those achieved by the more complex widening in [CC92c], without loss of precision.

Polyhedral approximations of the kind considered here are also employed in the analysis of constraint logic programs to approximate numeric constraints in [SG97]. The pre-interpretation domain is $\mathbb{R}^n$ where a subset of $\mathbb{R}^n$ is expressed in terms of both frame (vertices and rays) and a linear constraint system of both equalities and inequalities. This technique is employed further, [SG98] by using linear constraints to augment the regular approximation analysis of conventional logic programs (as opposed to CLP).

This chapter is structured as follows: Section 5.2 presents the analysis method with a worked example. Section 5.3 covers the theory behind the abstraction process, formalises the association between the abstract and concrete domains and introduces notation. Section 5.4 considers the impact of predicate structure on widening tactics. Sections 5.5 and 5.6 cover the computation of the convex hull and outline the implementation. Section 5.7 includes a table and summaries of some interesting analyses. The chapter concludes with a summary and indications for future work in Section 5.8.

5.2 Worked Example

In Prolog program text environments variables are represented by single upper case letters or strings that begin with uppercase letters, predicate names begin with lower case letters and the calligraphic letter $\mathcal{C}$ stands for a set of linear constraints. Consider an argument size analysis for the predicate append, here denoted `append/3`. Analysis is of an abstract program, denoted $\texttt{append}^{\mathcal{A}}\texttt{/3}$. The arguments of each predicate in the abstract program represent the sizes with respect to list length of the arguments of the corresponding predicate in the original program. The original program is hereafter referred to as the *concrete* program. Relationships that hold between arguments of $\texttt{append}^{\mathcal{A}}\texttt{/3}$ are interpreted as inter-argument size relationships for `append/3`, in the original program.

```
append([], S, S).                append^A(0, S, S).
append([R|Rs], S, [R|T]) :-      append^A(RRs, S, RT) :-
    append(Rs, S, T).                RRs = 1 + Rs,
                                     RT = 1 + T,
                                     append^A(Rs, S, T).
```

The analysis terminates with a solution to a fixpoint equation that characterises the inter-argument relationships. The i^{th} iterate is denoted by I_i and I_0 is the empty set of points. Each iteration in the fixpoint calculation takes an I_i as input and generates an I_{i+1} as output. To compute I_{i+1}, terms in each body atom of each clause are unified with terms in the same atoms in I_i. Since I_0 is empty, I_1 contains only those relationships explicit in the unit clause of `append`$^{\mathcal{A}}$`/3`, that is:

$$I_1 = \{\langle r,s,t\rangle \in \mathbb{R}^3 \mid r = 0, s = t\}$$

I_1 contains just a single set of inequalities, but the number of sets of inequalities for a predicate can grow at each iteration. Therefore, to keep the size of each iterate small and manageable, the sets of inequalities are approximated. Each set of inequalities characterises a set of points, in this example, a space in $\mathbb{R}^3$. Instead of maintaining several sets of inequalities for each predicate, the union of the spaces they represent is approximated by the closure of their convex hull. The closure of the convex hull under union of sets of points is the smallest closed convex set of points that contains the union. As the closure of the convex hull of an arbitrary number of polyhedral sets is also polyhedral, it can be represented by a single set of inequalities so that only one such set need be maintained for each predicate. Hence I_2 is computed as follows:

$$\begin{aligned}
I_2 &= \{\langle r,s,t\rangle \in \mathbb{R}^3 \mid r = 0, s = t\} \,\overline{\cup} \\
&\quad \{\langle r,s,t\rangle \in \mathbb{R}^3 \mid r = 1 + r', s = s', t = 1 + t', r' = 0, s' = t'\} \\
&= \{\langle r,s,t\rangle \in \mathbb{R}^3 \mid r = 0, s = t\} \,\overline{\cup} \\
&\quad \{\langle r,s,t\rangle \in \mathbb{R}^3 \mid r = 1, s = t - 1\} \\
&= \{\langle r,s,t\rangle \in \mathbb{R}^3 \mid 0 \le r, r \le 1, t = r + s\}
\end{aligned}$$

Although the closure of the convex hull is an approximation, useful relationships are preserved; for example, the inter-argument size relationship $t = r + s$, that is common to both clauses of the predicate.

Since the analysis domain is infinite the technique known as widening is employed to contain the growth of the set for each predicate and enforce convergence to those inequalities that are invariant in all iterations. In order to incorporate the widening process into the iteration sequence, the third iterate is named I_3', as follows:

$$\begin{aligned}
{I_3}' &= \{\langle r,s,t\rangle \in \mathbb{R}^3 \mid r = 0, s = t\} \,\overline{\cup} \\
&\quad \{\langle r,s,t\rangle \in \mathbb{R}^3 \mid r = 1 + r', s = s', t = 1 + t', 0 \le r', r' \le 1, t' = r' + s'\} \\
&= \{\langle r,s,t\rangle \in \mathbb{R}^3 \mid r = 0, s = t\} \,\overline{\cup} \\
&\quad \{\langle r,s,t\rangle \in \mathbb{R}^3 \mid 1 \le r, r \le 2, t = r + s\} \\
&= \{\langle r,s,t\rangle \in \mathbb{R}^3 \mid 0 \le r, r \le 2, t = r + s\}
\end{aligned}$$

Then the widening is introduced so that:

$$I_3 = I_2 \bigtriangledown {I_3}' = \{\langle r, s, t\rangle \in \mathbb{R}^3 \mid 0 \leq r, t = r + s\}$$

Only those inequalities present in both I_2 and ${I_3}'$, are present in I_3. The last two steps are repeated by naming the fourth iterate ${I_4}' = \{\langle r, s, t\rangle \in \mathbb{R}^3 \mid 0 \leq r, r \leq 3, t = r + s\}$, and computing $I_4 = I_3 \bigtriangledown {I_4}'$ demonstrating that:

$$I_4 = \{\langle r, s, t\rangle \in \mathbb{R}^3 \mid 0 \leq r, t = r + s\}$$

and the iteration sequence has converged.

To recap, for each predicate, each iteration generates a space described by the intersection of a set of inequalities and successive iterations return a space that both includes and extends the previous space. The widening defined here retains only those inequalities that are invariant from one iteration to the next. Spatially, these inequalities characterise boundaries common to each iterate. The inequalities that are discarded are those that represent the unconstrained growth in size of an argument that is a list. Once widening commences termination of the analysis will follow, since at each iteration the finite set of inequalities can only decrease or remain the same.

5.3 Semantics

5.3.1 Concrete Domain

Let *Vars* be a denumerable set of variables, Σ a set of function symbols and constant symbols and Π a set of predicate symbols. The set of finite trees over *Vars* and Σ is T_{Herb}, and the set of finite trees over just Σ is D_{Herb}. Hence, a program P is a finite set of clauses of the form: $p(\bar{t}) \leftarrow p_1(\bar{t_1}), \ldots, p_n(\bar{t_n})$ where p, $p_i \in \Pi$, $\bar{t}$, $\bar{t_i} \in T^n_{Herb}$. The interpretation base for P is $B_{Herb} = \{p(\bar{t}) \mid p \in \Pi \wedge \bar{t} \in D^n_{Herb}\}$. The variables in a syntactic object, o, are denoted $var(o)$. $\exists\{x_1, \ldots, x_n\} . h$ and $\overline{\exists} X\ h$ are abbreviations for $\exists x_1 \ldots \exists x_n . h$ and $\exists(var(h) \setminus X) . h$ where h is an Herbrand constraint.

5.3.2 Abstract Domain

Let *Lin* be the system of sets of linear constraints over $\mathbb{R}$ and the set of constraint predicates, $\{\leq, =\}$. *Lin* is ordered by entailment, denoted $\models$, and $\forall \mathcal{C}_i \in Lin . \mathcal{C}_1 = \mathcal{C}_2 \leftrightarrow \mathcal{C}_1 \models \mathcal{C}_2 \wedge \mathcal{C}_2 \models \mathcal{C}_1$. Since many different sets of linear equalities are equivalent in this way *Lin* is quotiented by equivalence to give $Lin/\!= (\models)$ which is a poset. *Lin* is closed under variable elimination and $\exists\{x_1, \ldots, x_n\} . \mathcal{C}$ and $\overline{\exists} X . \mathcal{C}$ are abbreviations for $\exists x_1 \ldots \exists x_n . \mathcal{C}$ and $\exists(var(\mathcal{C}) \setminus X) . \mathcal{C}$ where $\mathcal{C} \in Lin$. Since a polyhedron is defined as the set of points included in the intersection of a set of closed half-spaces and a closed half-space can be represented by a non-strict linear inequality, there is a total mapping from Lin^n to polyhedra in $\mathbb{R}^n$, namely, $Poly^n$. Hence, given a totally ordered finite set of variables $X = \{x_1, \ldots x_n\}$ when $[\mathcal{C}]_= \in Lin^n$, then $[\mathcal{C}]_= = \{\bar{x}' \in \mathbb{R}^n \mid (\bigwedge^n_{i=1} x_i = x'_i) \models \mathcal{C}\}$, and the meet and join for polyhedra, set intersection and the closure of the convex hull, respectively, also serve as the meet and join for Lin^n.
Let *Vars* be a denumerable set of variables, and Lin^n a set of linear constraints over a totally ordered set of variables $X \subseteq Vars . |X| = n$. Hence, P is a finite set of clauses of the form: $p(\bar{t}) \leftarrow \mathcal{C}, p_1(\bar{t_1}), \ldots, p_n(\bar{t_n})$ where p, $p_i \in \Pi$, $\mathcal{C} \in Lin^n$, $\bar{t}$, $\bar{t_i} \in T^n_{Herb}$.

The semantics of a program in the abstract domain can be expressed in s-style for constraint logic programs [BGLM91]. The interpretation base B_{Lin} for a program is the set of constrained atoms of the form $p(\bar{x}) \leftarrow \mathcal{C}$ quotiented by equivalence. Equivalence, $\sim$, is defined:

$$p(\bar{x}) \leftarrow \mathcal{C} \sim p(\bar{x}') \leftarrow \mathcal{C}' \;\leftrightarrow\; \overline{\exists} var(\bar{x}) \,.\, \mathcal{C} = \overline{\exists} var(\bar{x}) \,.\, [\mathcal{C}' \wedge (\bar{x} = \bar{x}')]$$

where $\mathcal{C}$, $\mathcal{C}'$ are sets of inequalities. The preorder on inequalities lifts to a preorder on interpretations, $\mathcal{P}(B_{Lin})/\approx (\sqsubseteq)$ where:

$$I \sqsubseteq I' \text{ iff } \forall [p(\bar{x}) \leftarrow \mathcal{C}]_\sim \in I \,.\, \exists [p(\bar{x}) \leftarrow \mathcal{C}']_\sim \in I' \,.\, \mathcal{C} \models \mathcal{C}'$$

The preorder also defines an equivalence relation: $I \approx I'$ iff $I \sqsubseteq I' \wedge I' \sqsubseteq I$ which, in turn, defines the poset $\mathcal{P}(B_{Lin})/\approx (\sqsubseteq)$ where $[I]_\approx \sqsubseteq [I']_\approx$ iff $I \sqsubseteq I'$. Note that, although an interpretation I is a set of constrained atoms of the form $p(\bar{x}) \leftarrow \mathcal{C}$, where $\mathcal{C}$ is a set of implicitly conjoined inequalities, the ordering is defined semantically with respect to the constraints over the arguments in each predicate. This means that for any set of constraints $\mathcal{C}$, $\mathcal{C}' \,.\, \mathcal{C} \models \mathcal{C}'$ it is also true that in terms of the polyhedra P, P' that $\mathcal{C}$ and $\mathcal{C}'$ represent, $P \subseteq P'$.

5.3.3 Program Abstraction

The formal mapping from terms to the natural numbers that returns the size of a term is known as a norm. Norms are qualified by type and a size metric that is applied to ground terms [Gia93]:

Definition 5.3.1 Let T be a term system, then a norm denoted $|.|_\zeta$, is a function: $|.|_\zeta : T \to \mathbb{N}$. ▪

In this case the measure that is required is that of list length and this is defined:

Definition 5.3.2 A norm denoted $|.|_{len}$, that measures the length of a list, is a partial mapping, $|.|_{len} : T \to \mathbb{N}$, defined:

$$\begin{array}{lcll} |t|_{len} & = & 0 & \text{if } t \text{ is a variable} \\ |t|_{len} & = & 0 & \text{if } t = [] \\ |t|_{len} & = & 1 + |Tail|_{len} & \text{if } t = [Head|Tail] \end{array}$$

▪

The list length norm is lifted to non-ground terms to enable the abstraction of programs and interpretations that necessarily reference non-ground terms. Hence, the co-domain is not simply the non-negative integers, it comprises expressions constructed from non-negative integers, variables which can have non-negative integer values and the operator '+'. Such a norm is effectively a symbolic norm.

Definition 5.3.3 Let *Vars* and *Vars'* each be a distinct denumerable set of variables. Let T be a term system described over *Vars*, and $\mathcal{N}$ is the set of finite expressions constructed from $\mathbb{N} \cup \{+\} \cup Vars'$. A norm denoted $|.|_{len}$, that measures the length of a list, is a partial mapping, $|.|_{len} : T \to \mathcal{N}$, defined:

$$\begin{array}{lcll} |t|_{len} & = & t & \text{if } t \text{ is a variable} \\ |t|_{len} & = & 0 & \text{if } t = [] \\ |t|_{len} & = & 1 + |\mathit{Tail}|_{len} & \text{if } t = [\mathit{Head}|\mathit{Tail}] \end{array}$$

■

In order to formally define program abstraction, it is necessary to ensure that constraints over like terms are mapped to constraints over like variables in the numerical domain. This is achieved using the mapping sol, which is based on the solved form algorithm of [LMM88]. Let X denote a finite set of variables. An equation is of the form $s = t$ where both s and t are terms and Eqn denotes a set of equations. Similarly, let Eqn_{Lin} denote the set of linear equations of the form $x = e$ where $e \in \mathcal{N}$.

Definition 5.3.4 The mapping $sol : \mathcal{P}(Eqn) \to \mathcal{P}(Eqn)$ is defined by: $sol(E) = \{E' \mid E \leadsto^* E'\}$ where $*$ is transitive closure and the relation, $Eqn \leadsto Eqn$ is the least binary relation defined by:

- $\{x = t\} \cup E \leadsto \{x = s\} \cup E \qquad \text{if } t = s \wedge s \notin X$
- $\{x = t\} \cup E \leadsto \{x = y\} \cup E \qquad \text{if } t = y \wedge y \in X$

where $X \subset Vars$. ■

The mapping $\mathfrak{a}_{len}$ is defined to allow the deduction of constraints based on the list-length norm. This mapping is fundamental to program abstraction and although Lin is defined over $\mathbb{R}$ as $|.|_{len}$ is defined over $\mathbb{N}$, the abstraction implicitly includes non-negative constraints on all variables; these constraints are necessary as the analysis is concerned with size which is non-negative.

Definition 5.3.5 Let $E \in \mathcal{P}(Eqn)$, then $\mathfrak{a}_{len} : \mathcal{P}(Eqn) \to \mathcal{P}(Eqn_{Lin})$ is defined by:

$$\mathfrak{a}_{len}(E) = \{x = |t|_{len}) \mid x = t \in sol(E)\}$$

■

Definition 5.3.6 The abstraction of a program P, to its abstract counterpart, $P^{\mathcal{A}}$, is defined:

$$P^{\mathcal{A}} = \left\{ p(\bar{x}_0) \leftarrow \mathfrak{a}_{len}(E),\, p_1(\bar{x_1}), \ldots p_n(\bar{x}_n) \;\middle|\; \begin{array}{l} w \in P, \\ w = p(\bar{t_0}) \leftarrow p_1(\bar{t_1}), \ldots, p_n(\bar{t_n}), \\ E = sol(\bar{x_i} = \bar{t_i}), \\ \cup_{i=0}^{n} \bar{x_i} \cap var(w) = \phi \end{array} \right\}$$

■

Example 5.3.1 Consider append/3:

```
w1  append([], Y, Y).
w2  append([X|Xs], Y, [X|Z]) :-
        append(Xs, Y, Z).
```

and the application of both *sol* and $\mathfrak{a}_{len}$ to each clause as prescribed by the abstraction:

w_1				
		$sol(\{x_{0i} = t_{0i}\})$	$\mathfrak{a}_{len}(sol(\{x_{0i} = t_{0i}\}))$	
$\{x_{01} = t_{01},$	$t_{01} = []\}$	$\leadsto \{x_{01} = []\}$	$\{x_{01} = \lvert[]\rvert_{len}\}$	$\leadsto \{x_{01} = 0\}$
$\{x_{02} = t_{02},$	$t_{02} = y\}$	$\leadsto \{x_{02} = y\}$	$\{x_{02} = \lvert y\rvert_{len}\}$	$\leadsto \{x_{02} = y\}$
$\{x_{03} = t_{03},$	$t_{03} = y\}$	$\leadsto \{x_{03} = y\}$	$\{x_{03} = \lvert y\rvert_{len}\}$	$\leadsto \{x_{03} = y\}$
w_2				
		$sol(\{x_{0i} = t_{0i}\})$	$\mathfrak{a}_{len}(sol(\{x_{0i} = t_{0i}\}))$	
$\{x_{01} = t_{01},$	$t_{01} = [x\vert xs]\}$	$\leadsto \{x_{01} = [x\vert xs]\}$	$\{x_{01} = \lvert[x\vert xs]\rvert_{len}\}$	$\leadsto \{x_{01} = 1 + \lvert xs\rvert_{len}\}$
				$\leadsto \{x_{01} = 1 + xs\}$
$\{x_{02} = t_{02},$	$t_{02} = y\}$	$\leadsto \{x_{02} = y\}$	$\{x_{02} = \lvert y\rvert_{len}\}$	$\leadsto \{x_{02} = y\}$
$\{x_{03} = t_{03},$	$t_{03} = [y\vert z]\}$	$\leadsto \{x_{03} = [y\vert z]\}$	$\{x_{03} = \lvert[y\vert z]\rvert_{len}\}$	$\leadsto \{x_{03} = 1 + \lvert z\rvert_{len}\}$
				$\leadsto \{x_{03} = 1 + z\}$
		$sol(\{x_{1i} = t_{1i}\})$	$\mathfrak{a}_{len}(sol(\{x_{1i} = t_{1i}\}))$	
$\{x_{11} = t_{11}$	$t_{11} = xs\}$	$\leadsto \{x_{11} = xs\}$	$\{x_{11} = \lvert xs\rvert_{len}\}$	$\leadsto \{x_{11} = xs\}$
$\{x_{12} = t_{12}$	$t_{12} = y\}$	$\leadsto \{x_{12} = y\}$	$\{x_{12} = \lvert y\rvert_{len}\}$	$\leadsto \{x_{12} = y\}$
$\{x_{13} = t_{13}$	$t_{13} = z\}$	$\leadsto \{x_{13} = z\}$	$\{x_{13} = \lvert z\rvert_{len}\}$	$\leadsto \{x_{13} = z\}$

Hence, an abstract version of `append`/3 is derived; the far right column is a more concise expression of the abstract program, included for clarity.

w_1 $\texttt{append}^{\mathcal{A}}(\texttt{X}_{01}, \texttt{X}_{02}, \texttt{X}_{03}).$ $\texttt{X}_{01} = \texttt{0},$ $\texttt{X}_{02} = \texttt{Y},$ $\texttt{X}_{03} = \texttt{Y}.$	$\leadsto$	w_1 $\texttt{append}^{\mathcal{A}}(\texttt{0}, \texttt{Y}, \texttt{Y}).$
w_2 $\texttt{append}^{\mathcal{A}}(\texttt{X}_{01}, \texttt{X}_{02}, \texttt{X}_{03}) :-$ $\texttt{X}_{01} = \texttt{1} + \texttt{Xs},$ $\texttt{X}_{02} = \texttt{Y},$ $\texttt{X}_{03} = \texttt{1} + \texttt{Z},$ $\texttt{X}_{11} = \texttt{Xs},$ $\texttt{X}_{12} = \texttt{Y},$ $\texttt{X}_{13} = \texttt{Z},$ $\texttt{append}^{\mathcal{A}}(\texttt{X}_{11}, \texttt{X}_{12}, \texttt{X}_{13}).$	$\leadsto$	w_2 $\texttt{append}^{\mathcal{A}}(\texttt{1} + \texttt{Xs}, \texttt{Y}, \texttt{1} + \texttt{Z}) :-$ $\texttt{append}^{\mathcal{A}}(\texttt{Xs}, \texttt{Y}, \texttt{Z}).$

5.3.4 Analysis Framework - a Galois Connection

The notion of abstraction is extended to interpretations to qualify the relationship between an abstract and a concrete interpretation:

Definition 5.3.7 Abstraction $\alpha : \mathcal{P}(B_{Herb}) \rightarrow \mathcal{P}(B_{Lin})/\approx$ is defined:

$$\alpha(J) = \{[p(\bar{x}) \leftarrow \mathfrak{a}(E)]_{\equiv} \mid p(\bar{t}) \in J \wedge E = sol(\bar{x} = \bar{t})\}$$

■

The definition of α allows a reciprocal definition of γ:

Definition 5.3.8 Concretisation $\gamma : \mathcal{P}(B_{Lin})/\!\approx\, \rightarrow \mathcal{P}(B_{Herb})$ is defined:

$$\gamma([I]_{\approx}) = \cup\{J \mid \alpha(J) \sqsubseteq I\}$$

■

Lemma 5.3.1 $\langle\alpha, \gamma\rangle$ is a Galois Connection.

Proof Since $\gamma([I]_{\approx}) = \cup\{J \mid \alpha(J) \sqsubseteq I\}$ it follows that:

$$\alpha(J) \sqsubseteq [I]_{\approx} \leftrightarrow J \subseteq \gamma([I]_{\approx})$$

Hence, $\langle\alpha, \gamma\rangle$ is a Galois Connection. ■

5.3.5 Concrete Semantics

The fixpoint semantics of P is defined as the least fixpoint of the immediate consequences operator T^g_P, that is, $\mathcal{F}[\![P]\!] = \mathit{lfp}(T^g_P)$.

Definition 5.3.9 $T^g_P : \mathcal{P}(B_{Herb}) \rightarrow \mathcal{P}(B_{Herb})$ is defined:

$$T^g_P(I) = \left\{ p(\bar{t}') \;\middle|\; \begin{array}{l} w \in P,\;\; w = p(\bar{t}) \leftarrow p_1(\bar{t}_1), \ldots, p_n(\bar{t}_n), \\ p_i(\bar{t}_i{}') \in I, \\ (\bar{t} = \bar{t}') \models \overline{\exists} var(\bar{t})\,.\,(\wedge_{i=1}^n(\bar{t}_i = \bar{t}_i{}')) \end{array} \right\}$$

■

5.3.6 Abstract Semantics

Let $P^{\mathcal{C}}$ denote a (constraint) logic program, then the immediate consequences operator, for $T_{P^{\mathcal{C}}}$, is defined:

Definition 5.3.10 $T_{P^{\mathcal{C}}} : \mathcal{P}(B_{Lin})/\approx\; \rightarrow \mathcal{P}(B_{Lin})/\approx$ is defined as $T_{P^{\mathcal{C}}}([I]_{\approx}) = [J]_{\approx}$ where:

$$J = \left\{ [p_i(\bar{x}) \leftarrow \mathcal{C}]_{\sim} \;\middle|\; \begin{array}{l} w \in P^{\mathcal{C}},\;\; w = p_0(\bar{x}) \leftarrow \mathcal{C}', p_1(\bar{x}_1), \ldots, p_n(\bar{x}_n), \\ [w_i]_{\sim} \in I,\;\; w_i = p_i(\bar{y}_i) \leftarrow \mathcal{C}_i, \\ \forall i\,.\, var(w) \cap var(w_i) = \phi, \\ \forall i \neq j\,.\, var(w_i) \cap var(w_j) = \phi, \\ \mathcal{C} = \overline{\exists} var(\bar{x})\,.\,[\mathcal{C}' \wedge \bigwedge_{i=1}^n (\bar{x}_i = \bar{y}_i \;\wedge\; \mathcal{C}_i)] \end{array} \right\}$$

■

In fact, $T_{P^{\mathcal{C}}}$ returns one or more constrained atoms for each predicate p_i. Each constrained atom is of the form $[p_i(\bar{x}) \leftarrow \mathcal{C}_\eta]_{\sim}$ where each $\mathcal{C}_\eta$ is a set of implicitly conjoined linear inequalities. Recall from the previous Section 5.2 that for each predicate, the join operator is applied over all $\mathcal{C}_\eta$ so that the output from each iteration is a single set of constraints for each predicate.
In the same way as the preorder on inequalities lifts to interpretations, so the join on inequalities is lifted to interpretations, $\bigsqcup : \mathcal{P}(B_{Lin})/\!\approx\rightarrow B_{Lin}/\!\approx$:

$$\bigsqcup_{\eta\in\mathcal{I}}\{[p_i(\bar{x}) \leftarrow \mathcal{C}_\eta]_{\sim}\} = [p_i(\bar{x}) \leftarrow \overline{\cup}_{\eta\in\mathcal{I}}\mathcal{C}_\eta]_{\sim}$$

Let $P_{\mathcal{A}}$ be an abstract program, then the immediate consequences operator for the abstract domain is defined:

Definition 5.3.11 $T_{P\mathcal{A}} : \mathcal{P}(B_{Lin})/\approx \to \mathcal{P}(B_{Lin})/\approx$ is defined as $T_{P\mathcal{A}}([I]_\approx) = [J]_\approx$ where:

$$J = \bigcup_{i=1}^{n} \left\{ \bigsqcup_{\eta \in \mathcal{I}} \{[p_i(\bar{x}) \leftarrow \mathcal{C}_\eta]_\sim\} \;\middle|\; \begin{array}{l} w \in P^{\mathcal{A}},\;\; w = p_0(\bar{x}) \leftarrow \mathcal{C}', p_1(\bar{x}_1), \ldots, p_n(\bar{x}_n), \\ [w_i]_\sim \in I,\;\; w_i = p_i(\bar{y}_i) \leftarrow \mathcal{C}_i, \\ \forall i \,.\, var(w) \cap var(w_i) = \phi, \\ \forall i \neq j \,.\, var(w_i) \cap var(w_j) = \phi, \\ \mathcal{C}_\eta = \overline{\exists} var(\bar{x}) \,.\, [\mathcal{C}' \wedge \bigwedge_{i=1}^{n} (\bar{x}_i = \bar{y}_i \;\wedge\; \mathcal{C}_i)] \end{array} \right\}$$

■

5.3.7 Widening

To recap, each iteration of the $T_{P\mathcal{A}}$ operator returns a set of linear inequalities for each predicate that defines a polyhedron in $\mathbb{R}^n$, and since $T_{P\mathcal{A}}$ is continuous, for each predicate the sequence of generated polyhedra will be increasing. When there are alternative clauses for a predicate the iteration sequence returns the closure of the convex hull of the alternative polyhedra. Approximating in this way helps to reduce the number of individual inequalities for each predicate in an interpretation. However, $Poly^n$ is infinite and is not chain complete[1], so the sequence of iterations may not converge. In order to ensure that the analysis converges, a technique known as widening is used to force convergence to a post fixpoint. The following proposition is adapted from [CC92c].

Proposition 5.3.1 Let $\langle L, \sqsubseteq\rangle$ be a poset, $F : L \to L$ be continuous, $\bot \in L$ be such that $\bot \sqsubseteq F(\bot)$ and $\bigtriangledown \in L \times L \to L$ be a widening, then the upward iteration sequence with widening, x_i where $i, k \in \mathbb{N}$, is defined:

$$\begin{array}{rll} x_0 = & \bot & \\ x_{i+1} = & x_i & \text{if } F(x_i) \sqsubseteq x_i \\ x_{i+1} = & F(x_i) & \text{else if } i \leq k \\ x_{i+1} = & x_i \bigtriangledown F(x_i) & \text{else if } i > k \end{array}$$

will converge and its limit $\mathcal{Z}$ is such that $lfp(F) \sqsubseteq \mathcal{Z}$ and $F(\mathcal{Z}) \sqsubseteq \mathcal{Z}$. ■

The tenet of a widening is that it trades precision for finiteness. The widening presented here, an adaption of the widening of [CH78] lifted to interpretations, retains those inequalities that are invariant from one iteration to the next. By definition, $T_{P\mathcal{A}}$ is continuous on the poset $\mathcal{P}(B_{Lin})/\approx$ $(\sqsubseteq)$ so that $[I_i]_\approx \sqsubseteq [I_{i+1}]_\approx$. Both $[I_i]_\approx$ and $[I_{i+1}]_\approx$ contain at most one set of inequalities for each predicate symbol, and the widening lifts naturally from the predicate level to that of interpretations. In each iteration the set of inequalities expressing variable constraints for **each predicate** describes a polyhedron. For each predicate the polyhedron described in $[I_i]_\approx$ will be a subset or equal to the polyhedron described in $[I_{i+1}]_\approx$. Consider then, $p(\bar{x}) \leftarrow \mathcal{C}_i \in [I_i]_\approx$ and $p(\bar{x}) \leftarrow \mathcal{C}_{i+1} \in [I_{i+1}]_\approx$. It follows that $\mathcal{C}_i \models \mathcal{C}_{i+1}$. Since $\mathcal{C}_i \models \mathcal{C}_{i+1} \leftrightarrow \forall \beta \in \mathcal{C}_{i+1} \,.\, [\mathcal{C}_i \models \beta]$,

[1]The increasing chain of polyhedra, $P_1, \ldots P_i, \ldots \in \mathbb{R}^2$ each circumscribed by a circle, S, does not converge to a unique polyhedron.

in order to identify inequalities that are invariant from one iteration to the next, it is necessary to retain only those inequalities in $\mathcal{C}_i$ that are entailed by $\mathcal{C}_{i+1}$.

Definition 5.3.12 $\bigtriangledown : \mathcal{P}(B_{Lin})/\!\approx \times \mathcal{P}(B_{Lin})/\!\approx \rightarrow \mathcal{P}(B_{Lin})/\!\approx$ is defined:

$$[I_i]_{\approx} \bigtriangledown [I_{i+1}]_{\approx} = [\{[p(\bar{x}) \leftarrow \mathcal{C}_i \bigtriangledown \mathcal{C}_{i+1}]_{\sim} \mid [p(\bar{x}) \rightarrow \mathcal{C}_i]_{\sim} \in I_i \wedge [p(\bar{x}) \rightarrow \mathcal{C}_{i+1}]_{\sim} \in I_{i+1}\}]_{\approx}$$

where $\mathcal{C}_i \bigtriangledown \mathcal{C}_{i+1} = \{\beta \in \mathcal{C}_i \mid \mathcal{C}_{i+1} \models \beta\}$ ▪

Since interpretations are ordered by entailment over the constraints on each atom it follows that $\mathcal{C}_i \models \mathcal{C}_{i+1}$. In terms of the polyhedra P_i, P_{i+1} that $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$ represent, this means that $P_i \subseteq P_{i+1}$. (Therefore, spatially, if a constraint $\beta \in \mathcal{C}_i$ is entailed by $\mathcal{C}_{i+1}$ then P_{i+1} is a subset or equal to the half-space represented by β, and the boundary of β acts as a bound on both P_i and P_{i+1}.) Both sets, $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$ are maintained as lists of inequalities in the ground representation. A fixpoint is reached as the widening does not allow the list representing $\mathcal{C}_i \bigtriangledown \mathcal{C}_{i+1}$ to grow by introducing any new inequalities[2]. Hence, the iteration sequence with widening where $i, k \in \mathbb{N}$ is as follows:

$$\begin{array}{rll} I_0 = & \phi & \\ I_{i+1} = & I_i & \text{if } T_{P_{\mathcal{A}}}(I_i) \sqsubseteq I_i \\ I_{i+1} = & T_{P_{\mathcal{A}}}(I_i) & \text{else if } i \leq k \\ I_{i+1} = & I_i \bigtriangledown T_{P_{\mathcal{A}}}(I_i) & \text{else if } i > k \end{array}$$

The widening is applied after a bounded number of iterations, k, and in order to select a bound that will maximise precision, the structure of the predicates in the program must be considered. In the following section three different types of structure are identified.

5.4 Widening Criteria

The following Sections 5.4.1, 5.4.2 and 5.4.3 show how different classes of predicate dictate different optimal choices for a bound on the number of iterations. In practice of course, the cost of classifying the predicate and identifying a strategy has to be weighed against the cost incurred by unnecessary iterations when an upper approximation of the optimal choice of bound is chosen.

5.4.1 Widening with Uniform Increments

Consider the append$^{\mathcal{A}}$/3 program of Section 5.2 listed below.

```
append^A([], S, S).             append^A(0, S, S).
append([R|Rs], S, [R|T]) :-     append^A(1 + Rs, S, 1 + T) :-
    append(Rs, S, T).               append^A(Rs, S, T).
```

Each iteration of the analysis generates an atom append$^{\mathcal{A}}(r, s, t) \leftarrow \mathcal{C}$ where the variables r and t are both incremented by 1 relative to the previous iterate. That is, the i^{th} iteration of the analysis, $[I_i]_{\approx}$, takes the form $I_i = \{[\text{append}^{\mathcal{A}}(x, y, z) \leftarrow \mathcal{C}_i]_{\sim}\}$ where

[2]If the solver guarantees the return of inequalities with the same meaning in the same syntactic form then a naive widening, the intersection of the inequalities in $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$ is sufficient.

each $\mathcal{C}_i$ represents a polyhedron $P_i \in \mathbb{R}^3$. For example, with reference to Section 5.2, the iterations I_1 and I_2 represent the polyhedra P_1 and P_2 respectively:

$$P_1 = \{\langle r, s, t\rangle \,|\, r = 0, s = t\}, \quad P_2 = \{\langle r, s, t\rangle \,|\, 0 \le r, r \le 1, t = r + s\}$$

Generally for $\texttt{append}^{\mathcal{A}}/3$, $P_{i+1} = P_1 \overline{\cup} \{\langle 1 + r,\, s,\, 1 + t\rangle \mid \langle r,\, s,\, t\rangle \in P_1\}$ so that P_{i+1} extends and includes that of P_i. Each P_{i+1} can be obtained from P_i in the same way since P_{i+1} differs from P_i by uniform increments in the first and third dimensions. Although inter-argument relationships $0 \le r$ and $t = r + s$ are entailed implicitly by P_1, they are not explicit until P_2 and the ensuing P_i. Widening can therefore be invoked on the third iterate by imposing a bound of $k = 2$, without loss of significant information. The invariant constraints are confirmed in the third iteration I_3, where $P_3 = \{\langle r, s, t\rangle \,|\, 0 \le r, t = r + s\}$. The increments are uniform in the first and third dimensions because the order of terms and subterms in the recursive call is preserved.

Definition 5.4.1 Let p/n be a recursive predicate where in the recursive clause the consequent is $p(\bar{t})$ and in the antecedent the recursive call is $p(\bar{t}')$, then p/n is *ordered structurally recursive* if $\exists t'_i \in \bar{t}' \,.\, \exists t_i \in \bar{t} \,.\, t'_i$ is a subterm of t_i. ▪

It is conjectured that for ordered structurally recursive predicates with a single recursive clause all of the invariant constraints can be found within three iterations.

5.4.2 Widening within a Hierarchy

Consider the $\texttt{quicksort}^{\mathcal{A}}$ program described in Section 5.7. The program consists of a hierarchy of several predicates, where the top level predicate $\texttt{quicksort}^{\mathcal{A}}$ has calls in the body to other predicates, the auxiliaries: $\texttt{partition}^{\mathcal{A}}$ and $\texttt{append}^{\mathcal{A}}$. Therefore, each I_i will consist of up to three constrained atoms of the form, $[p(\bar{x}) \leftarrow \mathcal{C}]_\sim$, but at most one for each predicate symbol, p. However, I_{i+1} can only include $[\texttt{quicksort}^{\mathcal{A}}(\bar{x}) \leftarrow \mathcal{C}]_\sim$, when I_i includes both $[\texttt{partition}^{\mathcal{A}}(\bar{x}) \leftarrow \mathcal{C}]_\sim$ and $[\texttt{append}^{\mathcal{A}}(\bar{x}) \leftarrow \mathcal{C}]_\sim$. $\texttt{partition}^{\mathcal{A}}$ and $\texttt{append}^{\mathcal{A}}$ are structurally recursive with uniform increment and therefore can be widened with $k = 2$.
In general, however, precision can be lost if a predicate is widened before the analysis of its auxiliaries has stabilised. One method of coping with this is to reference the Strongly Connected Components of the call graph of clause dependencies. By inspecting the clauses of a program, the call graph and its Strongly Connected Components can be computed. The SCCs of the $\texttt{quicksort}^{\mathcal{A}}$ program, for example, are the clause sets:

$$\begin{array}{ll}\{\{\texttt{append}^{\mathcal{A}}/3/1\}, & \{\texttt{append}^{\mathcal{A}}/3/2\}, \\ \{\texttt{partition}^{\mathcal{A}}/4/1\}, & \{\texttt{partition}^{\mathcal{A}}/4/2, \texttt{partition}^{\mathcal{A}}/4/3\}, \\ \{\texttt{quicksort}^{\mathcal{A}}/2/1\}, & \{\texttt{quicksort}^{\mathcal{A}}/2/2\}\}\end{array}$$

where $p/a/m$ abbreviates the m^{th} clause defining the predicate p of arity a.

By considering the SCCs in topological order with the deepest predicates first, the fixpoint may be computed in the usual bottom-up way, see Figure 8. In this case analysis begins with the base cases of the deepest predicates, progressing upwards to derive fixpoints for $\texttt{partition}^{\mathcal{A}}$ and $\texttt{append}^{\mathcal{A}}$, before moving on to $\texttt{quicksort}^{\mathcal{A}}$. A complete analysis for $\texttt{quicksort}^{\mathcal{A}}$ is given in the table.

Interpretation	*Step*
$I_1 = \{[\texttt{append}^{\mathcal{A}}(x,y,z) \leftarrow x = 0,\ y = z]_{\sim}\}$	base
$I_2 = \{[\texttt{append}^{\mathcal{A}}(x,y,z) \leftarrow 0 \leq x \leq 1,\ z = y + x]_{\sim}\}$	recurse
$I_3 = \{[\texttt{append}^{\mathcal{A}}(x,y,z) \leftarrow z = y + x,\ 0 \leq x]_{\sim}\}$	widen
$I_4 = \{[\texttt{append}^{\mathcal{A}}(x,y,z) \leftarrow z = y + x,\ 0 \leq x]_{\sim}\}$	stabilise
$I_5 = I_4 \cup \{[\texttt{partition}^{\mathcal{A}}(w,x,y,z) \leftarrow x = 0,\ y = 0,\ z = 0]_{\sim}\}$	base
$I_6 = I_4 \cup \{[\texttt{partition}^{\mathcal{A}}(w,x,y,z) \leftarrow x = z + y,\ 0 \leq y \leq 1,\ y \leq x]_{\sim}\}$	recurse
$I_7 = I_4 \cup \{[\texttt{partition}^{\mathcal{A}}(w,x,y,z) \leftarrow x = z + y,\ y \leq x,\ 0 \leq y]_{\sim}\}$	widen
$I_8 = I_4 \cup \{[\texttt{partition}^{\mathcal{A}}(w,x,y,z) \leftarrow x = z + y,\ y \leq x,\ 0 \leq y]_{\sim}\}$	stabilise
$I_9 = I_8 \cup \{[\texttt{quicksort}^{\mathcal{A}}(x,y) \leftarrow x = 0,\ y = 0]_{\sim}\}$	base
$I_{10} = I_8 \cup \{[\texttt{quicksort}^{\mathcal{A}}(x,y) \leftarrow y = x,\ 0 \leq x \leq 1]_{\sim}\}$	recurse
$I_{11} = I_8 \cup \{[\texttt{quicksort}^{\mathcal{A}}(x,y) \leftarrow y = x,\ 0 \leq x]_{\sim}\}$	widen
$I_{12} = I_8 \cup \{[\texttt{quicksort}^{\mathcal{A}}(x,y) \leftarrow y = x,\ 0 \leq x]_{\sim}\}$	stabilise

The topological ordering minimises the size of the interpretations as the auxiliary predicates are dealt with one by one rather than simultaneously and each SCC is widened separately. Further, since a predicate will not be referenced until its auxiliaries have been analysed, fewer computations will lead to failure.

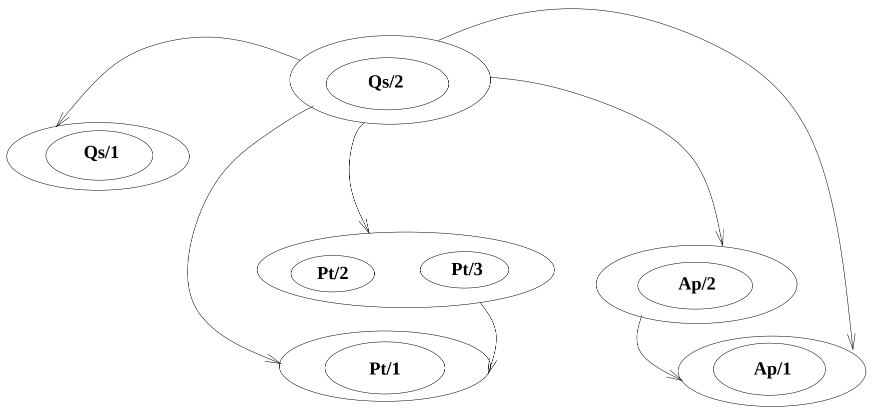


Figure 8: Dependencies between SCCs of the $\texttt{quicksort}^{\mathcal{A}}$ program

5.4.3 Widening with Non-uniform Increments

Under certain conditions all pertinent information may not be found in three iterations, even when the SCCs are considered in the topological order described in the previous Section 5.4.1. Two classes of predicate have been identified that require a more intelligent widening strategy. Consider the $\texttt{split}^{\mathcal{A}}\texttt{/3}$ predicate:

```
split([], [], []).              splitᴬ(0, 0, 0).
split([X|Xs], [X|Os], Es):-     splitᴬ(1+Xs, 1+Os, Es):-
  split(Xs, Es, Os).              splitᴬ(Xs, Es, Os).
```

Elements of the first list are placed alternately in the second and third lists. Associating a polyhedron P_i with each iteration I_i the definition of $\texttt{split}^{\mathcal{A}}\texttt{/3}$ is such that, given any P_i in the sequence, P_{i+1} and P_{i+2} are derived as follows:

$$P_{i+1} = P_i \,\bar{\cup}\, \{\langle 1+x, 1+z, y\rangle \mid \langle x,y,z\rangle \in P_i\},$$
$$P_{i+2} = P_i \,\bar{\cup}\, \{\langle 2+x, 1+y, 1+z\rangle \mid \langle x,y,z\rangle \in P_i\}$$

This means that $\texttt{split}^{\mathcal{A}}\texttt{/3}$ has a bi-modal incrementation behaviour where a uniform increment of 2, 1 and 1 in the first, second and third dimensions respectively, occurs in every second iteration. Consequently, the invariant condition is not explicit until the fourth iteration and cannot be confirmed until the fifth, so that a bound, $k = 4$, is required. An analysis for $\texttt{split}^{\mathcal{A}}$ is shown below where $I_i = \{[\texttt{split}^{\mathcal{A}}(x,y,z) \leftarrow \mathcal{C}_i]_{\sim}\}$. The invariant constraints are in bold type.

$\mathcal{C}_1$	$x = 0$	$y = 0$	$z = 0$			
$\mathcal{C}_2$	$x \leq 1$	$0 \leq x$	$z = 0$	$y = x$		
$\mathcal{C}_3$	$\mathbf{x = y + z}$	$\mathbf{0 \leq y}$	$\mathbf{x \leq 2y}$	$y \leq x$	$y \leq 1$	
$\mathcal{C}_4$	$\mathbf{x = y + z}$	$\mathbf{0 \leq y}$	$\mathbf{x \leq 2y}$	$\mathbf{2y \leq x + 1}$	$\mathbf{y \leq x}$	$x \leq y + 1$
$\mathcal{C}_5$	$\mathbf{x = y + z}$	$\mathbf{0 \leq y}$	$\mathbf{x \leq 2y}$	$\mathbf{2y \leq x + 1}$	$\mathbf{y \leq x}$	

In this case the recursive call for `split/3` does not preserve the ordering of terms and subterms in the initial call. The widening as described in [CC92c] would converge after three iterations having only deduced $x = y + z$.

To recap then, the ideal moment to apply the widening is when the iterations have captured all of the invariant relationships. However, as shown above, detecting this moment may present some difficulties and since widening prematurely can lead to a loss of information, this implementation permitted the user to vary k. The choice of which is the most propitious moment to widen will be tempered by the usual trade-off between cost and accuracy.

5.5 The Computation of the Convex Hull

The convex hull of an arbitrary collection of polyhedra is defined:

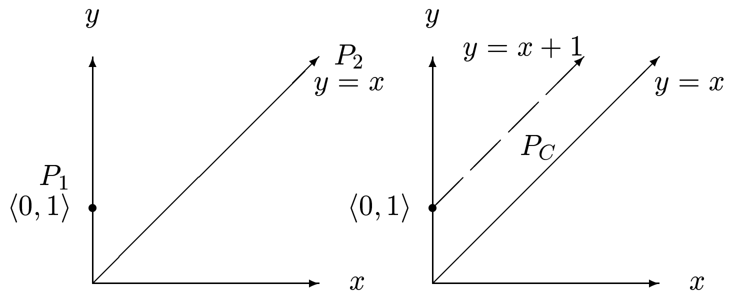
Theorem 5.5.1 Let $C_1, \ldots C_n$ be an arbitrary collection of non-empty convex sets in $\mathbb{R}^n$. Then the convex hull of the union of the collection, C_h, is defined:

$$C_h = \cup\{\Sigma_{i=1}^{n} \lambda_i C_i\}$$

where the union is taken over all finite convex combinations, that is for all combinations such that $0 \leq \lambda_i \leq 1$ and $\Sigma_{i=1}^{n} \lambda_i = 1$. [Roc70][Theorem 3.3] ∎

The convex hull of an arbitrary collection of polyhedra in $\mathbb{R}^n$ is not always a polyhedron, if there is an unbound polyhedron in the collection then the convex hull will not be closed, as demonstrated by the following example:

Example 5.5.1 Consider the convex hull of the polyhedra $P_1 = \langle 0, 1\rangle$ and $P_2 = \{0 \leq x,\, 0 \leq y,\, x = y\}$:



The convex hull, $P_C = conv(P_1 \cup P_2) = \{\langle 0,\ 1\rangle\} \cup \{0 \leq x,\ x \leq y,\ y < x+1\}$ which is not closed, despite the fact that both P_1 and P_2 are closed. Consider the point $P' = \langle 1,\ 2\rangle$ on the line $y = x + 1$. There is no linear combination of $P_1 = \{\langle 0,\ 1\rangle\}$ with any point in $P_2 = \{\langle 0,\ 0\rangle, \ldots, \langle 1,\ 1\rangle, \ldots, \langle 2,\ 2\rangle, \ldots\}$, that is the point $\langle 1,\ 2\rangle$. The smallest closed convex space containing the convex hull will be $\mathrm{cl}\,(P_C) = \{0 \leq x,\ x \leq y,\ y \leq x + 1\}$, and this is a polyhedron. The closure of P_C requires $y \leq x + 1$ rather than $y < x + 1$.

5.5.1 The Closure of the Convex Hull

To define the closure of the convex hull of a collection of polyhedra some account must be taken of the direction in which unbound polyhedra recede and this requires some terms of reference at the individual variable level. When a polyhedron is described over a set of variables $\bar{x}$, for each variable ,$x_i \in \bar{x}$, x_i has a lower bound x_i^{lb}, and an upper bound x_i^{ub}. For example, let $P = \{x \geq 0,\ y \leq x,\ x \leq 7\}$, then $x \in [0, 7]$ where $[0, 7]$ denotes the interval (in this case closed) between 0 and 7, so that $x^{lb} = 0$ and $x_{ub} = 7$. In this case y is constrained to the same interval and has the same lower and upper bounds as x. Recall the definition of a recession cone:

Theorem 5.5.2 Let P be a non-empty convex set in $\mathbb{R}^n$. The recession cone, 0^+C is a convex cone containing the origin, that is:

$$0^+C = \{y \mid C + y\ \subseteq\ C\}$$

[TR70] [Theorem 8.1] ∎

The recession cone of a convex set can be reformulated in terms of its unbound variables:

$$\begin{array}{lcll}
0^+P & = & \{r \mid P + r \subseteq P\} & \\
 & = & \{r \mid \forall x \in P\,[x + r \in P]\} & \\
 & = & \{\langle r_1, \ldots, r_n\rangle \mid \forall \langle x_1, \ldots, x_n\rangle \in P\,.\,\langle x_1 + r_1, \ldots, x_n + r_n\rangle \in P\} & \\
\text{where} & & r_i \in [-\infty, +\infty] & \text{if } x_i \in [-\infty, +\infty[\\
 & & r_i \in [-\infty, 0] & \text{if } x_i^{lb} = -\infty \\
 & & r_i \in [0, +\infty] & \text{if } x_i^{ub} = +\infty \\
 & & r_i = 0 & \text{otherwise}
\end{array}$$

Lemma 5.5.1 Let $P_1\ P_2$ be polyhedral convex sets in $\mathbb{R}^n$. Then given the reformulation of the recession cone of a polyhedron, $P_1 + 0^+P_2$ can be expressed in similar terms.

Proof There are two cases to consider:

- P_2 is bound.

  Since P_2 is bound, $0^+P_2 = \bar{0}$ [Roc70] Theorem 8.4. Therefore, $P_1 + 0^+P_2 = P_1$.

- P_2 is unbound.

$$\begin{array}{rcll} P_1 + 0^+P_2 & = & \{\langle s_i, \ldots, s_n\rangle \mid s_i = y_i + r_i,\ \langle y_1, \ldots, y_n\rangle \in P_1, & \\ & & \langle r_1, \ldots, r_n\rangle \in 0^+P_2\} & \\ \text{where} & & s_i \in [-\infty, +\infty] & \text{if } r_i \in [-\infty, +\infty] \\ & & s_i \in [-\infty, y_i^{ub}] & \text{if } r_i \in [-\infty, 0] \\ & & s_i \in [y_i^{lb}, +\infty] & \text{if } r_i \in [0, +\infty] \\ & & s_i \in [y_i^{lb}, y_i^{ub}] & \text{if } r_i = 0 \end{array}$$

  This allows a definition with reference to P_2 as follows:

$$\begin{array}{rcll} P_1 + 0^+P_2 & = & \{\langle s_i, \ldots, s_n\rangle \mid s_i = y_i + r_i, \langle y_1, \ldots, y_n\rangle \in P_1, & \\ & & \langle x_1, \ldots, x_n\rangle \in P_2,\ \langle r_1, \ldots, r_n\rangle \in 0^+P_2\} & \\ \text{where} & & s_i \in [-\infty, +\infty] & \text{if } x_i \in [-\infty, +\infty] \\ & & s_i \in [-\infty, y_i^{ub}] & \text{if } x_i^{lb} = -\infty \\ & & s_i \in [y_i^{lb}, +\infty] & \text{if } x_i = +\infty \\ & & s_i \in [y_i^{lb}, y_i^{ub}] & \text{otherwise} \end{array}$$

■

This computation forms an integral part of the definition of the closure of the convex hill as it allows the directions in which unbound polyhedra recede to be taken into account. $P_1 + 0^+P_2$ must include P_1 as the recession cone of P_2 includes the origin (all recession cones contain the origin by definition). A recession cone that is not the zero vector, recedes in certain directions, therefore, by definition of the sum operation on convex sets, $P_1 + 0^+P_2$ recedes in the directions of P_2. Intuitively, $P_1 + 0^+P_2$ is P_1 stretched or extended infinitely in the directions of P_2. Thus, the closure of the convex hull is the union of the linear combinations of points in P_1 and P_2 as the λ_i's range over the open interval $[0, 1]$ with P_1 receding in the directions of P_2 and P_2 receding in the directions of P_1. This effectively ensures the closure of the convex hull when either or both polyhedra are unbound.

Theorem 5.5.3 Let $P_1, \ldots P_n$ be an arbitrary collection of non-empty polyhedral convex sets in $\mathbb{R}^n$. Let $P = \mathrm{cl}\,(conv(P_1 \cup \ldots \cup P_n))$. Then P is also a polyhedral convex set and:

$$P = \cup\{\lambda_1 P_1 + \ldots + \lambda_n P_n\}.$$

where the union is taken over values for λ_i, where $0 \leq \lambda_i \leq 1$ and $\Sigma_{i=1}^{n} = 1$, but, 0^+P_i is substituted for $\lambda_i P_i$ when $\lambda_i = 0$, [Roc70][Theorem 19.6]. ■

The convex hull of two polyhedra is, by definition, the linear combination of all pairs of points one from each polyhedron. Recall that the closure of the convex hull is denoted $\overline{\cup}$, so where $P_1, P_2 \in Poly^n$, then $P_1 \overline{\cup} P_2$, is the union of:

1. the linear combination ranging over λ_i's where $0 < \lambda_i < 1$,
2. the sum of P_1 with the recession cone of P_2 and

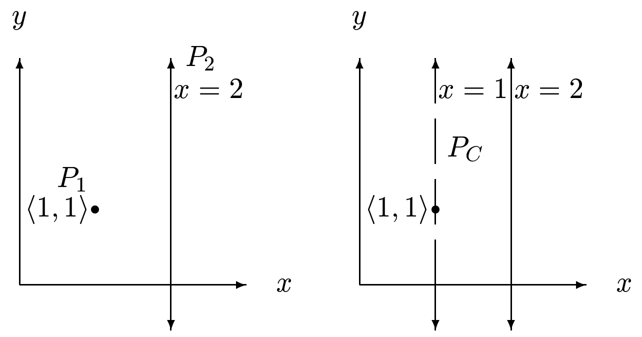
3. the sum of P_2 with the recession cone of P_1.

Consider $P_1 \overline{\cup} P_2 = P' \cup (0^+P_1 + P_2) \cup (P_1 + 0^+P_2)$, where

$$P' = \left\{ \bar{x} \;\middle|\; \begin{array}{lll} \bar{x} = \lambda_1\bar{y} + \lambda_2\bar{z} & \lambda_1 + \lambda_2 = 1 & \lambda_i > 0 \\ A_1\bar{y} \le B_1 & A_1\bar{z} \le B_1 & \end{array} \right\}$$

An example follows:

Example 5.5.2 Consider the point $P_1 = \{\langle 1,\, 1\rangle\}$ and the line $P_2 = \{x = 2\}$ in $\mathbb{R}^2$.



$$\begin{aligned} P_C = P_1 \overline{\cup} P_2 &= P' \cup (0^+P_1 + P_2) \cup (P_1 + 0^+P_2) \\ &= \{1 < x, x < 2\} \cup (\{\langle 0,\, 0\rangle\} + \{x = 2\}) \\ &\quad \cup(\{\langle 1,\, 1\rangle\} + \{x = 0\}) \\ &= \{1 < x, x < 2\} \cup \{x = 2\} \cup \{x = 1\} \\ &= \{1 \le x, x \le 2\} \end{aligned}$$

■

Note that in both cases the addition of the the recession cones of P_1 and P_2 has the effect of stretching, or extending, the convex hull so that it is closed.

5.5.2 The Matrix Method

Consider the matrix computation method proposed for the closure of the convex hull of $P_1 \cup P_2$, a linear relaxation of:

$$P_{12} = \left\{ \bar{x} \;\middle|\; \begin{array}{ll} \bar{x} = \lambda_1\bar{y} + \lambda_2\bar{z}, & \lambda_1 + \lambda_2 = 1,\, \underline{0 \le \lambda_i}, \\ A_1\bar{y} \le B_1, & A_2\bar{z} \le B_2, \end{array} \right\}$$

Matrix Method

The computation method can be reformulated into an equivalent expression which reflects the informal argument above:

$$P_{12} = \left\{ \bar{x} \;\middle|\; \begin{array}{ll} \forall x_i \in \bar{x} \,.\, \forall y_i \in \bar{y} \,.\, \forall z_i \in \bar{z} \,. & x_i \in [min(y_i^{lb}, z_i^{lb}), max(y_i^{ub}, z_i^{ub})] \\ \wedge\; x_i \models (\bar{x} = \lambda_1\bar{y} + \lambda_2\bar{z}, & \lambda_1 + \lambda_2 = 1,\, \underline{0 \le \lambda_i}, \\ A_1\bar{y} \le B_1, & A_2\bar{z} \le B_2) \end{array} \right\}$$

This means that the values of $x_i \in \bar{x}$ are considered separately and despite the fact that in principle, where c_1, c_2 are arbitrary constants and $i \neq j$, if i) $x_i = c_1$ when $\lambda_1 = 0$ and $\lambda_2 = 1$ and ii) $x_j = c_2$ when $\lambda_1 = 1$ and $\lambda_2 = 0$, then there is no $\bar{x}$ where x_i and x_j can take those values simultaneously, in this computation there is no constraint that can express this restriction. Thus, in the matrix method of computation the union of the two spaces is interpreted as a variable-wise union of intervals for each x_i in $\langle x_1, \ldots, x_n \rangle \in P_1$ with those intervals for each x_i in $\langle x_1, \ldots, x_n \rangle \in (P' \cup P_2)$, and vice versa for each x_i in $\langle x_1, \ldots, x_n \rangle \in P_2$. This does not take into account the fact that the bounds on $\bar{y}$ do not allow the points $\langle x_1, \ldots, x_n \rangle$ to recede into infinite space. This method effectively, computes $conv(P_1 \cup P_2)$ extended in the directions that both P_1 and P_2 recede in. The following Proposition demonstrates that the matrix method returns the closure of the convex hull in the general case:

Proposition 5.5.1 Let $P_i \in Poly^n$ and $P_1 = A_1\bar{y} \leq B_1$, $P_2 = A_2\bar{z} \leq B_2$ and

$$P_{12} = \left\{ \bar{x} \left| \begin{array}{ll} \bar{x} = \lambda_1\bar{y} + \lambda_2\bar{z} & \lambda_1 + \lambda_2 = 1,\ 0 \leq \lambda_i, \\ A_1\bar{y} \leq B_1, & A_2\bar{z} \leq B_2 \end{array} \right. \right\}$$

then $P_{12} = P_1 \bar{\cup} P_2$.

Proof By [Roc70][Theorem 19.6], $P_1 \bar{\cup} P_2 = \cup\{\lambda_1 P_1 + \lambda_2 P_2\}$, where $\lambda_i \in [0, \ldots, 1]$ and $\lambda_1 + \lambda_2 = 1$, but, 0^+P_i is substituted for $\lambda_i P_i$ when $\lambda_i = 0$. Hence, $P_1 \bar{\cup} P_2 = P' \cup (0^+P_1 + P_2) \cup (0^+P_2 + P_1)$, where

$$\begin{array}{rcl} P' & = & \{\bar{x} \mid \bar{x} = \lambda_1\bar{y} + \lambda_2\bar{z},\ A_1\bar{y} \leq B_1,\ A_1\bar{z} \leq B_1 \\ & & \lambda_1 + \lambda_2 = 1,\ \lambda_i > 0\} \end{array}$$

By definition of P_{12}, $P' \subseteq P_{12}$, $P_1 \subseteq P_{12}$, $P_2 \subseteq P_{12}$, hence, let $P_{12} = P' \cup P_1 \cup P_2 \cup P''$ where $P'' = P_{12} \setminus (P' \cup P_1 \cup P_2)$. Since the recession cone of a polyhedra always includes the zero vector, it follows that $P_1 \subseteq (P_1 + 0^+P_2)$ and $P_2 \subseteq (P_2 + 0^+P_1)$. Therefore $(P' \cup P_1 \cup P_2) \subseteq P_{12}$ and $(P' \cup P_1 \cup P_2) \subseteq P_1 \bar{\cup} P_2$. (1)

The proof is in two parts:

- $P_{12} \subseteq \text{cl}\,(conv(P_1 \cup P_2))$ The proof is by contradiction.

$$\begin{array}{rll} \text{suppose} & \exists\bar{x} \in P_{12} \,.\, \bar{x} \notin \text{cl}\,(conv(P_1 \cup P_2)) & \\ \text{therefore} & \exists\bar{x} \in P'' \,.\, \bar{x} \notin ((P_1 + 0^+P_2) & \\ & \cup(P_2 + 0^+P_2)) & \text{by (1)} \end{array}$$

Consider $P_1 + 0^+P_2$:

$$\begin{array}{rcll} P_1 + 0^+P_2 & = & \{\langle s_i, \ldots, s_n \rangle \mid s_i = y_i + r_i, \langle y_1, \ldots, y_n \rangle \in P_1, & \\ & & \langle z_1, \ldots, z_n \rangle \in P_2,\ \langle r_1, \ldots, r_n \rangle \in 0^+P_2\} & \\ \text{where} & & s_i \in [-\infty, +\infty] & \text{if } z_i \in [-\infty, +\infty] \\ & & s_i \in [-\infty, y_i^{ub}] & \text{if } z_i^{lb} = -\infty \\ & & s_i \in [y_i^{lb}, +\infty] & \text{if } z_i = +\infty \\ & & s_i \in [y_i^{lb}, y_i^{ub}] & \text{otherwise} \end{array}$$

But

$$P_{12} = \left\{ \bar{x} \;\middle|\; \begin{array}{ll} \forall x_i \in \bar{x} \,.\, \forall y_i \in \bar{y} \,.\, \forall z_i \in \bar{z} \,. & x_i \in [min(y_i^{lb}, z_i^{lb}), max(y_i^{ub}, z_i^{ub})] \\ \wedge\; x_i \models (\bar{x} = \lambda_1 \bar{y} + \lambda_2 \bar{z}, & \lambda_1 + \lambda_2 = 1,\; \underline{0 \le \lambda_i}, \\ A_1 \bar{y} \le B_1, & A_2 \bar{z} \le B_2) \end{array} \right\}$$

therefore, every $\bar{x} \in P''$ is either in $(P_1 + 0^+P_2)$ or in $(P_1 + 0^+P_2)$.

- $\mathrm{cl}\,(conv(P_1 \cup P_2)) \subseteq P_{12}$

  Similarly, the argument follows that every $\bar{x} \in (P_1 + 0^+P_2) \setminus \{P_1\}$ and every $\bar{x} \in (P_2 + 0^+P_1) \setminus \{P_2\}$ is also in P''.

Hence $P_{12} = P_1 \bar{\cup} P_2$. ■

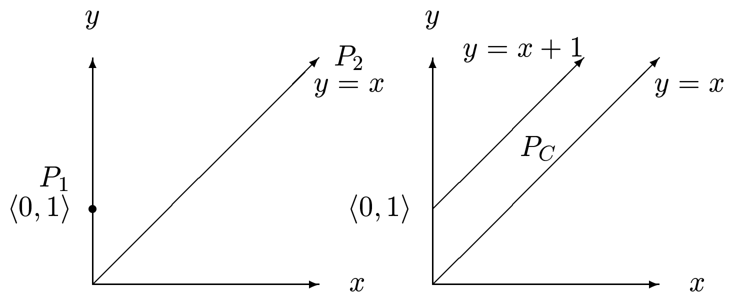
5.5.3 The Computation of the Closure of the Convex Hull

There are various approaches [CH78, HPR94, Wil93] to computing the convex hull of polyhedra. Some rely on the frame representation, some on the constraint representation as sets of linear inequalities and some on both. For the frame representation a polyhedron is represented as a system of generators, that is, two finite sets, V and R of vertices and rays, respectively:

$$P = \left\{ \Sigma_{v_i \in V} \lambda_i v_i + \Sigma_{r_j \in R} \mu_j r_j \,|\lambda_i \ge 0,\; \mu_j \ge 0,\; \Sigma_i \lambda_i = 1 \right\}$$

The convex hull P of two polyhedra P_1 and P_2, respectively represented by $\langle V_1, R_1 \rangle$ and $\langle V_2, R_2 \rangle$, is then given by $\langle V, R \rangle$ where $V = V_1 \cup V_2$ and $R = R_1 \cup R_2$ [Ker94].

Example 5.5.3 Consider again, the point P_1 and the half line P_2, referred to at the beginning of this section and illustrated in the diagram that follows.



The convex hull of P_1 and P_2 is the space P_C. Both the constraint and frame representations of P_1, P_2 and P_C are given below.

$$P_1 = \left\{ \langle x, y\rangle \in \mathbb{R}^2 \;\middle|\; \begin{array}{l} x = 0, \\ y = 1 \end{array} \right\} \qquad V_1 = \left\{ \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \qquad R_1 = \phi$$

$$P_2 = \left\{ \langle x, y\rangle \in \mathbb{R}^2 \;\middle|\; \begin{array}{l} x = y, \\ 0 \leq x, \end{array} \right\} \qquad V_2 = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\} \qquad R_2 = \left\{ \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$$

$$P_C = \left\{ \langle x, y\rangle \in \mathbb{R}^2 \;\middle|\; \begin{array}{l} 0 \leq x, \\ x \leq y, \\ y \leq x + 1 \end{array} \right\} \qquad V_C = \left\{ \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\} \qquad R_C = \left\{ \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$$

■

In [Ker94], for example, both representations are used with the convex hull being computed from the frame and ray representation and intersection from the representation as sets of inequalities. An improved version of the Chernikova algorithm is used to compute one representation from another.

CLP($\mathcal{R}$) provides the projection and solver machinery for manipulating sets of inequalities but a naive approach to calculating the convex hull may flounder, as non-linear constraints that cannot be reduced to linear constraints will be indefinitely postponed. Consider two arbitrary polyhedra, P_1 and P_2, represented in standard form:

$$P_1 = \{\bar{x} \in \mathbb{R}^n \mid A_1\bar{x} \leq \bar{B}_1\}, \quad P_2 = \{\bar{x} \in \mathbb{R}^n \mid A_2\bar{x} \leq \bar{B}_2\}$$

The convex hull of $P_1 \cup P_2$, P_C, is then defined by:

$$P_C = \left\{ \bar{x} \in \mathbb{R}^n \;\middle|\; \begin{array}{lll} \bar{x} = \sigma_1\bar{x}_1 + \sigma_2\bar{x}_2, & \sigma_1 + \sigma_2 = 1, & 0 \leq \sigma_i \\ A_1\bar{x}_1 \leq \bar{B}_1, & A_2\bar{x}_2 \leq \bar{B}_2 & \end{array} \right\}$$

However, the equation $\sigma_1\bar{x}_1 + \sigma_2\bar{x}_2 = 1$, is non-linear and in a constraint language that delays non-linear constraints the worst case can result in an infinite loop [HJM$^+$92]. A relaxation technique employed by [DB93], reformulates this system of equations and inequalities allowing the computation of the convex hull without recourse to the frame and ray representation. It is asserted that this relaxation is equivalent to the convex hull, but this is clearly not the case as the solution to a system of linear inequalities will be another system of linear inequalities which will be closed. For this reason this computation method cannot be equivalent to the convex hull, although, as has been seen it is equvalent to the closure of the convex hull. The technique amounts to reformulating the equations above by putting: $\bar{y}_1 = \sigma_1\bar{x}_1$ and $\bar{y}_2 = \sigma_2\bar{x}_2$ so that:

$$\bar{x} = \bar{y}_1 + \bar{y}_2, \quad A_1\bar{y}_1 \leq \sigma_1\bar{B}_1, \quad A_2\bar{y}_2 \leq \sigma_2\bar{B}_2$$

and P_C is defined by:

$$P_C = \left\{ \bar{x} \in \mathbb{R}^n \;\middle|\; \begin{array}{lll} \bar{x} = \bar{y}_1 + \bar{y}_2, & \sigma_1 + \sigma_2 = 1, & 0 \leq \sigma_i, \\ A_1\bar{y}_1 \leq \sigma_1\bar{B}_1, & A_2\bar{y}_2 \leq \sigma_2\bar{B}_2, & \end{array} \right\}$$

Example 5.5.4 To illustrate this method, consider example 5.5.3. Substituting for the matrices $A_1$and A_2, and the vectors B_1 and B_2, the above system of equations, where

$\bar{x} = \langle x, y \rangle$ and $\bar{y}_i = \langle x_i, y_i \rangle$, is as follows:

$$P_C = \left\{ \bar{x} \in \mathbb{R}^2 \;\middle|\; \begin{array}{l} x = x_1 + x_2,\, y = y_1 + y_2, \\ \sigma_1 + \sigma_2 = 1,\, 0 \leq \sigma_1,\, 0 \leq \sigma_2, \\ x_1 \leq 0,\, -x_1 \leq 0,\, y_1 \leq \sigma_1,\, -y_1 \leq -\sigma_1, \\ x_2 - y_2 \leq 0,\, -x_2 + y_2 \leq 0,\, -x_2 \leq 0 \end{array} \right\}$$

■

Hence, $P_C = \{0 \leq x, x \leq y, y \leq x + 1\}$ is derived through projection onto $\bar{x}$. The solution to the intersection of a set of non-strict inequalities is the intersection of a set of non-strict inequalities. Therefore, after projection onto $\bar{x}$, this computation will also return the intersection of a set of non-strict inequalities, which, by definition is polyhedral.

Proposition 5.5.1 confirms that the matrix method returns the closure of the convex hull, and so all that remains now, is to confirm that the linear relaxation technique employed above is simply a computational technique and not an upper approximation of the closure of the convex hull. Since the relaxed system of equations and inequalities is algebraically equivalent to the original system and no variable in the original system is removed from the relaxed system, the two systems are equivalent. Hence the relaxation technique returns the closure of the convex hull. The significant difference between the two methods lies in the fact that linear constraint solvers can solve the system of inequalities expressed as a relaxation, but may flounder with the original formulation.

It is interesting, though, to consider what happens when the original system is composed of homogeneous equations and inequalities, that is, $B_i = \bar{0}$. In this case the λ_i's effectively disappear from the calculation altogether which is then reduced to addition over each pair of points $\bar{y} \in P_1$ and $\bar{z} \in P_2$. However, spatially, systems of homogeneous equations and (non-strict) inequalities represent polyhedral cones. Since the convex hull of an arbitrary collection of polyhedral cones is always polyhedral (and therefore closed and convex), and the computation reduces to addition, the relaxation technique is confirmed.

5.6 A Bottom-up Interpreter

The bottom-up analyser is a simplified version of the meta-interpreter listed in [CL95]. The main difference between the two analysers is that interpretations are ground. This simplifies the meta-interpreter as atoms can be looked up in the interpretation without inducing aliasing and therefore variance (goal renaming) is not an issue.

For a given clause, the interpreter unifies each (non-ground) body atom in the definition of the clause with a matching (ground) atom in the interpretation, and then projects onto the head. Built-ins are solved directly. The task of unifying a body atom with an atom in the interpretation reduces to decoding the ground representation of constraints. For example, if the body atom is `p(X, Y, Z)` and the interpretation contains `atom(p, 3, [less(plus(var(1), var(2)), 0)])` then the call to decode from the ground representation:

```
decode([lesseq(plus(minus(var(1)), var(2)), 0)], 3, [X, Y, Z])
```

imposes the constraint `Y` $\leq$ `X` on the constraint store. Constraints on head variables are projected using meta-programming built-ins to obtain an output in a ground form.

Polyhedral abstractions from each of the clauses of a predicate are then combined incrementally as a convex hull. The convex hull is implemented as a binary operation so for a predicate of, say, three clauses returning the polyhedra P_1, P_2, P_3, the convex hull is calculated: $(P_1 \bar{\cup} P_2) \bar{\cup} P_3$. There is no loss of precision since the operation is both associative and commutative. Recall that the convex hull is described over a vector of variables $\bar{x}$ and $\bar{x} = \bar{y}_1 + \bar{y}_2$, the sets of equations over each y_i are generated by a single recursive pass over the ground representation of constraints over each $\bar{x}_i$. The equations and inequalities are posted to the constraint store and the their solution projected onto $\bar{x}$ is the convex hull. By employing a builtin meta-programming facility the constraints that define the convex hull are encoded into a ground representation.

5.6.1 Implementation Issues

The computation of the convex hull requires projection and the widening requires an entailment test. Both of these operations raise some implementation issues.

- **Projection** In CLP($\mathcal{R}$) constraints on head variables are projected using the meta-programming built-in `dump/3` [HJM+92], and the projection is encoded and output in a ground form. For example, with constraint store: $\{$`X` $=$ `4` $+$ `Y`, `Z` $<$ `X`$\}$, the call:

  ```
  dump([X, Y], [var(1), var(2)], Cons)
  ```

  instantiates `Cons` to `[var(2) = var(1) - 4]` which corresponds to a ground representation of the projection onto `X` and `Y`. `dump/3` was the only meta-programming facility in CLP($\mathcal{R}$) employed in the implementation.

  The `call_residue/2` built-in in SICStus has a similar role to `dump/3` in CLP($\mathcal{R}$). Apart from some syntactic differences, the main difference between the solvers CLP($\mathcal{R}$) and SICStus 3 lies in their respective projection facilities. In SICStus, projection is rather more complex as inequalities and equalities have to be dealt with separately, and consequently, the analyser implemented in SICStus 3 had to take this into account. `call_residue/2` accesses the residual constraints, that is, those constraints that are not satisfiable at the time of calling. For example, the call:

  ```
  call_residue({X = Y + Z, X = P}, Cons)
  ```

  instantiates `Cons` to `[[Z]-Z = P-Y]`, denoting a single residual constraint, `Z = P-Y`, on the variable `Z`. Note, however, that `Cons` excludes the equality relationship between `X` and `P`. `call_residue/2` requires an input argument, a goal, and the residue that is returned constitutes those subgoals unsatisfied after the call to goal. Therefore, the program is dynamically modified with the call:

  ```
  assert(dummy(Target_Vars))
  ```

  where `Target_Vars` are the variables that are the projection target. A `dummy/1` goal is then passed to `call_residue/2` to retrieve the constraints. A final post-processing phase assembles the residual constraints with the equality constraints.

- **Entailment** The widening operation identifies those constraints that are invariant from one iteration to the next. At the implementation level this involves an entailment test which is based on the premise that for a linear constraint c to be entailed by a set of constraints $\mathcal{C}$, it is sufficient to show that $\mathcal{C} \wedge \neg c$ has no solution [Col90].
- **Arithmetic in CLP($\mathcal{R}$) and SICStus 3** In both CLP($\mathcal{R}$) and SICStus 3 clp(R) the coefficients of variables within the inequalities are represented as real numbers so, for example, the inequality $2x \leq 3y$ may be stored in one iteration as $x - 1.5y \leq 0$, and as $x - 1.4999999y \leq 0$ in another. Although the widening ensures analysis termination, roundoff errors can lead to the loss of invariant relationships. Roundoff errors in CLP($\mathcal{R}$) were not a problem, since a small ϵ is used as slack in numerical comparisons. However, roundoff problems encountered in the initial port to SICStus 3 clp(R) (in the `Trd/3` problem (see Section 5.7)) hastened the transfer to clp(Q).

5.7 Example Analyses

There follows some analyses of programs which have traditionally proved difficult to analyse.

`leq/2` The `leq(X, Y)` predicate, adapted from [van91, pp. 55], holds if $x \leq y$. The base case constraint establishes the non-negativity of size, which is perpetuated in the recursive case as the size of `succ`(y) is one greater than that of y and the size of x is constant. The `leq(X, Y)` predicate is an example for which the analysis of [van91] does not terminate. In [CC92c] a similar predicate is analysed in a finite number of iterations through widening; the widening is a refinement of those proposed in [CH78, Hor90] and is precise enough to infer $x \leq y$.

`trd/2` The affine approach [GDL92, Kar76, VD92] cannot deduce any information for the `trd(X, Y)` predicate of [VD92]. In contrast, the polyhedral approach with widening can infer useful results. The non-negativity of x and y follows from $\frac{2}{3}x \leq y$ and $y \leq \frac{3}{2}x$. The norm in this example and those that follow is list length.

`perm/2` The `perm(L,P)` predicate enumerates all the permutations `P` of the list `L`. To deduce termination it is necessary to infer that `L` in the recursive call is smaller than `[H|T]`. This can only be inferred by deducing an inter-argument relation for `del/3`, namely, that the third argument of `del/3` is smaller than the second argument. It is precisely this relation that the analysis infers.

`quicksort/2` Like `perm/2`, `quicksort/2` is not structurally recursive and therefore it is necessary to infer that for `partition/4`, the arguments `L` and `G` in the recursive calls are strictly smaller than `[X|Xs]`.

`split/3` Although `split/3` is structurally recursive it is unusual in that the recursive call switches the last two arguments around. It can be deduced that the sum of the sizes of the second and third arguments is equal to size of the first. Further, the second argument will either be the same size as the third or at most 1 element larger than the third.

In the table that follows the first column displays the original program, the second its abstraction with respect to argument size and the third the set of inter-argument relations derived by analysis.

Program	*Abstract Program*	*Inter-argument Relationships*

leq(X, X).
leq(X, succ(Y)):-
   leq(X, Y).
	leq$^{\mathcal{A}}$(X, X):-
	   0 $\leq$ X.
	leq$^{\mathcal{A}}$(X, 1+Y):-
	   0 $\leq$ X,
	   0 $\leq$ Y,
	   leq$^{\mathcal{A}}$(X, Y).
		leq$^{\mathcal{A}}$(x, y) :-
		   $\{x \leq y,$
		   $0 \leq x\}$

trd([], []). trd([_,_|X], [_,_,_|Y]):- trd(X, Y). trd([_,_,_|X], [_,_|Y]):- trd(X, Y).	trd$^{\mathcal{A}}$(0, 0). trd$^{\mathcal{A}}$(2+X, 3+Y):- trd$^{\mathcal{A}}$(X, Y). trd$^{\mathcal{A}}$(3+X, 2+Y:- trd$^{\mathcal{A}}$(X, Y).	trd$^{\mathcal{A}}$(x, y) :- $\{\frac{2}{3}x \leq y,$ $y \leq \frac{3}{2}x\}$
perm([], []). perm([H|T], [A|P]):- del(A, [H|T], L), perm(L, P). del(X, [X|Y], Y). del(U, [Y|Z], [Y|W]):- del(X, Z, W).	perm$^{\mathcal{A}}$(0, 0). perm$^{\mathcal{A}}$(1+T, 1+P):- del$^{\mathcal{A}}$(_, 1+T, L), perm$^{\mathcal{A}}$(L, P). del$^{\mathcal{A}}$(_, 1+Y, Y). del$^{\mathcal{A}}$(_, 1+Z, 1+W):- del$^{\mathcal{A}}$(_, Z, W).	perm$^{\mathcal{A}}$(x, y) :- $\{y = x,$ $0 \leq x\}$ del$^{\mathcal{A}}$(x, y, z) :- $\{z = y - 1\}$

quicksort([], []).
quicksort([X|Xs], S):-
   part(X, Xs, L, G),
   quicksort(L, SL),
   quicksort(G, SG),
   append(SL, [X|SG], S).
 	
part(_, [], [], []).
part(X, [Y|Ys], L, [Y|G]):-
   X =< Y,
   part(X, Ys, L, G).
part(X, [Y|Ys], [Y|L], G):-
   Y =< X,
   part(X, Ys, L, G).
 	
append([], Y, Y).
append([X|Xs], Y, [X|Zs]):-
   append(Xs, Y, Zs).
	quicksort$^{\mathcal{A}}$(0, 0).
	quicksort$^{\mathcal{A}}$(1+Xs, S):-
	   part$^{\mathcal{A}}$(_, Xs, L, G),
	   quicksort$^{\mathcal{A}}$(L, SL),
	   quicksort$^{\mathcal{A}}$(G, SG),
	   append$^{\mathcal{A}}$(SL, 1+SG, S).
	
	part$^{\mathcal{A}}$(_, 0, 0, 0).
	part$^{\mathcal{A}}$(X, 1+Ys, L, 1+G):-
	   part$^{\mathcal{A}}$(X, Ys, L, G).
	part$^{\mathcal{A}}$(X, 1+Ys, 1+L, G):-
	   part$^{\mathcal{A}}$(X, Ys, L, G).
	
	append$^{\mathcal{A}}$(0, Y, Y).
	append$^{\mathcal{A}}$(1+Xs, Y, 1+Zs):-
	   append$^{\mathcal{A}}$(Xs, Y, Zs).
		quicksort$^{\mathcal{A}}$(x, y) :-
		   $\{x = y,$
		   $0 \leq x\}$
		
		part$^{\mathcal{A}}$(w, x, y, z) :-
		   $\{x = y + z,$
		   $y \leq x,$
		   $0 \leq x\}$
		
		append$^{\mathcal{A}}$(x, y, z) :-
		   $\{z = x + y,$
		   $0 \leq x\}$

split([], [], []):- split([X|Xs], [X|Os], Es):- split(Xs, Es, Os).	split$^{\mathcal{A}}$(0, 0, 0):- split$^{\mathcal{A}}$(1+Xs, 1+Os, Es):- split$^{\mathcal{A}}$(Xs, Es, Os).	split$^{\mathcal{A}}$(x, y, z) :- $\{x = y + z,$ $y \leq x,$ $y - \frac{1}{2}x \leq \frac{1}{2},$ $\frac{1}{2}x \leq y,$ $0 \leq y\}$

5.8 Summary

The work described in this chapter is an implementation study in argument size analysis based on polyhedral approximations. The approach to polyhedral approximation presented here has much to commend it as it combines accuracy with a relatively simple implementation. The relative simplicity of the implementation was allowed: i) by the use of a computation method for the convex hull that requires only a constraint representation and ii) by employing languages with constraint support, namely CLP($\mathcal{R}$) and SICStus 3. A relaxation technique employed in disjunctive constraint programming [DB93] allows the expression of the join computation, the closure of the convex hull, as the solution to a system of linear inequalities. Recall that there are occasions when the convex hull of two polyhedra may not be closed (and therefore is not polyhedral). It is demonstrated that the computation method employed here is an approximation of the convex hull that is the closure of the convex hull. The closure of the convex hull is the most precise approximation of disjunctions of convex spaces that is polyhedral, and it is this polyhedral property that is important as this property entails representation as the solution to a system of linear inequalities that represent closed half-spaces. Therefore by exploiting the computational machinery of CLP($\mathcal{R}$) and the clp(Q,R) libraries of SICStus, it has not been necessary to manipulate matrices or frames, or implement the Chernikova conversion mechanism to switch between the representations.
The advantage of the method with a single representation over the method with two representations can only really be assessed if the various time complexities of the operations in question are taken into account. If both methods use the same technique for solving linear equations and inequalities, then the cost of intersection is the same for both methods. This leaves the convex hull operation in the single representation case to be weighed against the combined cost of: i) either a) maintaining both representations and updating at least one representation using the conversion mechanism each time an operation is carried out or b) carrying one representation and converting when necessary to the other, and ii) the cost of the convex hull operation over frames and rays. The convex hull operation over frames and rays, as simple set unions, will be relatively fast. However, the cost of maintaining both representations at each computational step or converting between the two representations when necessary, will depend on both the time complexity of the conversion mechanism and on how often the conversion operation takes place. The frequency with which the convex hull is required will also have some bearing on this comparison. In designing an analyser, Howe and King in [HK01] note that in the analysis of 8 large programs, including the Aquarius compiler itself, the frequency of the join operation counts for less than 10% of the operations required by the analyser. Howe and King's analyser is occupied with groundness dependency analysis but the overall technique, albeit streamlined, of abstract interpretation over abstract programs is the same as the analysis here. Suppose then that the convex hull is relatively infrequently required, then the impact on the overall time cost could be less significant than might be expected by a straightforward comparison between the convex hull over systems of linear inequalities and over frames and rays. Hence, providing the time cost is accepatable, the method with a single representation may well have the advantage over the method with two representations.

Irrespective of representation, convergence of the analysis iterates with an infinite abstract domain is an issue. Convergence of the analysis iterates is enforced by the technique known as widening. By observation it is apparent that the precision of the suggested widening can be improved by delaying its invocation until the propitious moment. Currently a heuristic is employed to gauge that moment. Forecasting that moment may or may not be cost effective as it must be weighed against the cost of superfluous iterations incurred in some circumstances by the use of the heuristic. Hence, this is an area for future investigation. The widening here is an adaptation of the more complex widening of [CC92c] and it returns comparable or better results than those of [CC92c].

Further, this polyhedral work can be applied in other areas. For example, in [KSB97] the polyhedral technique was applied to the automation of time complexity analysis for parallel environments; if program tasks can be assigned a measure of time complexity that can be compared to some threshold for spawning, then the programmer is released from the error-prone task of specifying process assignment and scheduling. Possible future refinements of the techniques expounded here include: in terms of precision in some contexts widening can be improved by a single narrowing step and SCCs can be used to improve the iteration strategy. In particular polyhedral widenings require investigation on several counts. Not only is the feasibility of deducing the propitious moment to widen of interest but observation of the way in which polyhedral widenings work prompts questions about how these techniques work when the polyhedra are of a dimension that is less than the n-space in which they are embedded. Hence, the investigation that follows in the next chapter.

Chapter 6

Polyhedral Widenings

In the previous chapter a relatively simple widening was devised, an adaptation of a complex widening suggested by Halbwachs and Cousot [CC92c]. Some interesting questions arise concerning the original widening of Halbwachs and Cousot. Since several systems of inequalities may have the same solution, several systems of inequalities can represent the same space, so will this widening always return the same space irrespective of the representation? Are there sequences of increasing polyhedra for which this widening will not work, and if not, why? Questions like this prompted the investigation that follows. Further, Halbwachs asserts and proves that the widening is representation independent, but it is not a proof that appeals to the intuitive understanding of what is happening spatially. This chapter also offers a proof that satisfies spatial understanding as a complement to Halbwachs semantic proof, but also draws attention to the precondition that must be satisfied if the widening is to be representation independent. In order to attain these ends a close look is taken at just how a polyhedron is defined uniquely by the intersection of a set of closed half-spaces. This amounts to identifying which subset of points in each half-space actively contributes to the delineation and substance of the polyhedron. This allows the qualification of both i) how different representations of a polyhedron relate to one another and ii) when different representations are possible. In the process of this investigation, it also becomes apparent just how the widening of the previous chapter is related to Halbwachs original widening, and why it works after a bounded number of iterations.

With reference to binary operations, for example, $a \varrho b$, where ϱ is a binary operator, then a and b are referred to as the operands. It will be demonstrated that a set of constraints defines the bounds of a polyhedron and both widenings return the common bounds of their operands, also a polyhedron. The spatial insights afford an understanding of the widening process that not only identifies when it is most useful but also allows a new interpretation of the widening that amounts to a simpler computation.

Throughout the discussion that follows only minimal representations of polyhedra are considered, that is, representations such that the removal of any inequality in the representation set would change the shape of the polyhedron. The notion of a *minimal* representation, is defined formally in the Section 6.1 that describes the terms of reference for the discussion and proofs that follow. Unless otherwise stated, the source of the definitions in this section is [Roc70].

6.1 Preliminary Definitions

The definitions that follow provide the terminology for the discussion and proofs that constitute this chapter. The definitions are interspersed with explanatory text and diagrams.

Definition 6.1.1 The *dimension* of a convex set $S \in \mathbb{R}^n$ is the dimension of $\operatorname{aff} S$ and is denoted $\dim S$. ∎

Note that if the dimension, say m, of some subset of $\mathbb{R}^n$ is pertinent then the descriptor is prefixed with m. For example, an m-dimensional polyhedron may be referred to as an m-polyhedron.
The concept of interior (defined within Definition 4.1.11 of *open*) can be absorbed into that of relative interior allowing for the expression of the relative interior of an m-polyhedron in $\mathbb{R}^n$, where $m < n$, which has a natural interior relative to its affine hull.

Definition 6.1.2 The unit ball is defined, $B = \{\bar{x} \mid d(\bar{x}, \bar{0}) \leq 1\}$, where d is a function that returns the distance between two points, in this case, $\bar{x} \in \mathbb{R}^n$ and $\bar{0}$, the origin. ∎

Definition 6.1.3 The *relative interior* of a convex set, $S \in \mathbb{R}^n$, denoted $\operatorname{ri} S$, is defined as the interior which results when S is regarded as a subset of its affine hull. Hence, where $\epsilon \in \mathbb{R}$, $\operatorname{ri} S = \{\bar{x} \in \operatorname{aff}\ S \mid \exists \epsilon > 0, (\bar{x} + \epsilon B) \cap (\operatorname{aff} S) \subset S\}$. ∎

This means that the relative interior of a set S is all of those points in the affine hull of S, also in S but such that every adjacent point to each of those points is also in S. This is in effect, defining every point in S within its affine hull that is not in its boundary, without actually referring to the boundary.

Definition 6.1.4 The *relative boundary* of a convex set $S \in \mathbb{R}^n$, is the set difference: $\operatorname{rb} S = (\operatorname{cl} S) \setminus (\operatorname{ri} S)$. ∎

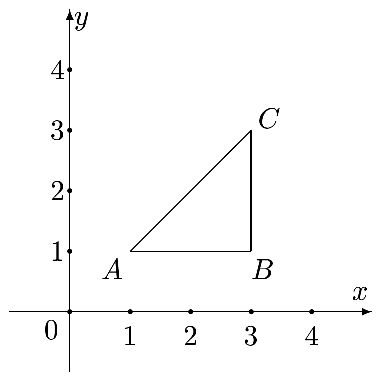
Recall that when the convex set is a polyhedron, then the closure of the polyhedron is equal to the polyhedron.

Definition 6.1.5 An m-polyhedron in $\mathbb{R}^n$, where $m \leq n$, is said to be *embedded* in each of the affine sets, including $\mathbb{R}^n$, that contains it. ∎

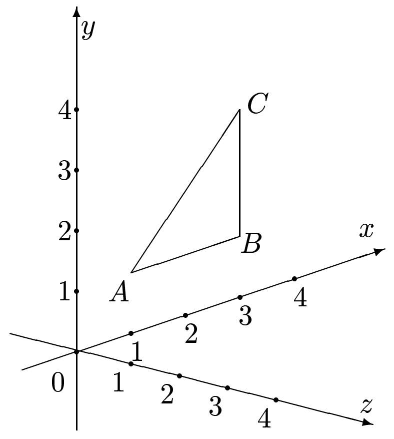
Definition 6.1.6 The set of all closed half-spaces in $\mathbb{R}^n$ is denoted Half^n. ∎

Since a closed half-space has only one possible representation as a linear inequality, no notational distinction is made between them but the meaning will be clear from the context. However, in the discussion that follows it is necessary to distinguish between the sets of inequalities that represent polyhedra and the sets of points represented by their implicit intersection. Given the definition of Half^n, when $P \in \mathcal{P}(\mathit{Half}^n)$ it follows by Definition 4.1.14 that the intersection of the elements of P is a polyhedron. In the remainder of this chapter the intersection of the elements of P, the set of points, is denoted explicitly by $\cap P$ to distinguish it from P, the set of linear inequalities.

Definition 6.1.7 Let $\cap P \in \mathit{Poly}^n$, then $\beta \in \mathit{Half}^n$ *supports* $\cap P$ iff $\cap P \subseteq \beta$ and $\cap P \cap \operatorname{rb} \beta \neq \phi$. ∎



1. Triangle ABC in $\mathbb{R}^2$



2. Triangle ABC in $\mathbb{R}^3$

Figure 9: Triangle ABC in $\mathbb{R}^2$ and $\mathbb{R}^3$.

ABC	representation	relative interior	relative boundary	affine hull
in $\mathbb{R}^2$	$\cap T_1$ where $T_1 = \{1 \leq y,\ y \leq x,\ x \leq 3\}$	ri $\cap T_1 = \cap\{1 < y,\ y < x,\ x < 3\}$	rb $\cap T_1 = AB \cup BC \cup CA$	aff $\cap T_1 = \mathbb{R}^2$
in $\mathbb{R}^3$	$\cap T_2$ where $T_2 = \{1 \leq y,\ y \leq x,\ x \leq 3,\ 0 \leq z,\ z \leq 0\}$	ri $\cap T_2 = \cap\{1 < y,\ y < x,\ x < 3\}$ embedded in $z = 0$	rb $\cap T_2 = AB \cup BC \cup CA$ embedded in $z = 0$	aff $\cap T_2 = z = 0$ in $\mathbb{R}^3$

Table 1: Triangle ABC in $\mathbb{R}^2$ and $\mathbb{R}^3$.

If $\beta \in \mathit{Half}^n$ supports $\cap P \in \mathit{Poly}^n$ then β is a said to be a closed supporting half-space to $\cap P$. Figure 9 and Table 1, illustrate the difference between the notions of an n-polyhedron in $\mathbb{R}^n$ as opposed to an m-polyhedron in $\mathbb{R}^n$, where $m < n$. In the table, AB for example, denotes the set of points in the line joining the points A and B.

Recall (Definition 4.1.6) that an affine set of dimension $(n-1)$ in $\mathbb{R}^n$ is called a *hyperplane*. Intuitively a hyperplane of $\mathbb{R}^n$ has two sides, like a line in $\mathbb{R}^2$ or a plane in $\mathbb{R}^3$. In general then, if α_1 and α_2 are n-dimensional closed half-spaces in $\mathbb{R}^n$ and $\alpha_1 \cup \alpha_2 = \mathbb{R}^n$, then $h = \alpha_1 \cap \alpha_2 = \operatorname{rb} \alpha_1 = \operatorname{rb} \alpha_2$ is a hyperplane of $\mathbb{R}^n$.

Definition 6.1.8 A *supporting hyperplane*, h to $\cap P \in \mathit{Poly}^n$ is such that $h = rb(\beta)$, where β is a closed supporting half-space to $\cap P$. If $\cap P \subseteq h$, in which case both closed half-spaces with relative boundary, h, are supporting half-spaces then h is an *improper* supporting hyperplane. If $\cap P \not\subseteq h$, then h is a *proper* supporting hyperplane. [Bro83] ∎

Definition 6.1.9 A pair of closed half-spaces α_1, α_2 is *opposing* iff $rb(\alpha_1) = rb(\alpha_2)$ and $\alpha_1 \neq \alpha_2$. ∎

representation of ABC in $\mathbb{R}^3$	proper supporting hyperplanes	relative boundary	improper supporting hyperplane	supporting half-spaces	opposing supporting half-spaces
$\cap T_2$ where $T_2 =$ $\{1 \leq y, y \leq x,$ $x \leq 3,$ $0 \leq z, z \leq 0\}$	$x = y$ $x = 3$ $y = 1$	$AC \cup$ $BC \cup$ AB	$z = 0$	T_2	$0 \leq z$ $z \leq 0$

Table 2: Triangle ABC in $\mathbb{R}^3$

This means that if a pair of opposing closed half-spaces α_1, α_2 both support $\cap P \in Poly^n$, then $h = \text{rb}\,\alpha_1 = \text{rb}\,\alpha_2$ is an improper supporting hyperplane to $\cap P$, as $\cap P \subseteq h$. Pairs of opposing closed half-spaces (and the opposing linear inequalities that represent them) are in general denoted α_1 and α_2.
Figure 10 and Table 2 illustrate the association between the proper supporting hyperplanes of $\cap P \in Poly^n$ with the relative boundary of $\cap P$. The triangle ABC is again embedded in $\mathbb{R}^3$, and, for example, the intersection of the hyperplane $x = y$ with ABC is the line AC, a subset of the relative boundary of ABC.

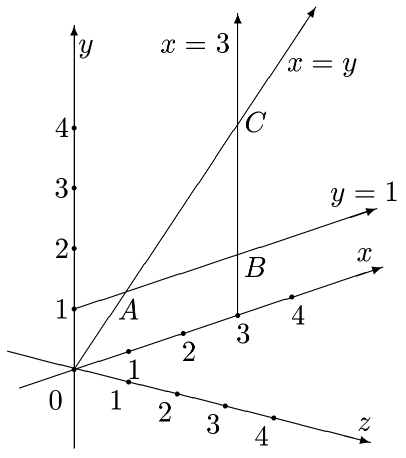


Figure 10: Triangle ABC in $\mathbb{R}^3$

The following definitions clarify the different ways in which supporting hyperplanes intersect with polyhedra that they support.

Definition 6.1.10 A subset f, of a polyhedron $\cap P \in Poly^n$, is called a *proper face* of $\cap P$ if there exists a proper supporting hyperplane h to $\cap P$ such that $f = h \cap \cap P$. The *improper faces* of $\cap P$ are $f = \phi$ and $f = \cap P$. [Lay82] ▪

Let $\cap P \in Poly^n$, then from the above definition if $\dim \cap P = n$ the set of proper faces F of $\cap P$ will comprise k-faces where $0 \leq k \leq n - 1$. (The 0-faces will be the extreme points of $\cap P$.) However, if $\dim \cap P = m$ and $0 \leq m \leq n-1$, the set of faces F of $\cap P$ will comprise k-faces where $0 \leq k \leq m-1$.

Definition 6.1.11 A *facet*, f^c of an m-polyhedron, $\cap P$, is a proper face of $\cap P$ where $\dim f^c = k$ and $0 \leq k = m-1$. [Bro83] ▪

However, there are proper supporting hyperplanes that can support a polyhedron at a face that is not a facet. In Figure 11 of the triangle ABC embedded in $\mathbb{R}^3$, the hyperplane $x = 1$ supports ABC at the point $\langle 1, 1\rangle$ and although this point is not a facet of ABC, it is a face of ABC. In general therefore, if a proper supporting hyperplane of an m-polyhedron, $\cap P \in Poly^n$, does not support $\cap P$ at a facet of $\cap P$, then it supports $\cap P$ at a k-face where $0 \leq k \leq m-2$. This is illustrated in Figure 11 and Table 3 that follow:

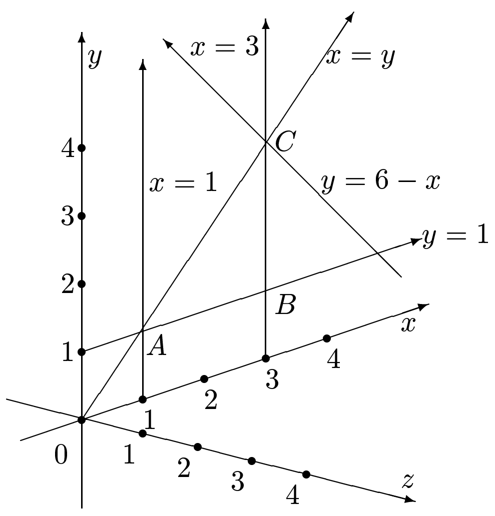


Figure 11: Triangle ABC in $\mathbb{R}^3$ and some of its supporting hyperplanes.

representation of ABC in $\mathbb{R}^3$ (dim)	proper supporting hyperplanes	proper faces (dim)	facets (dim)	improper supporting hyperplane	improper faces (dim)
$\cap T_2$ where $T_2 =$ $\{1 \leq y, y \leq x,$ $x \leq 3,$ $0 \leq z, z \leq 0\}$ (2)	$x = y$ $x = 3$ $y = 1$ $x = 1$ $y = 6 - x$ ...	AC (1) BC (1) AB (1) $A = \langle 1,1\rangle$ (0) $C = \langle 3,1\rangle$ (0) ...	AC (1) BC (1) AB (1)	$z = 0$	ϕ (na) ABC (2)

Table 3: Triangle ABC in $\mathbb{R}^3$ and some of its supporting hyperplanes.

It is interesting to note that a face of a polyhedron that is not a facet is also a face of at least two of the facets of the polyhedron and the union of the proper and improper faces of a polyhedron form a complete lattice ordered by set inclusion.

Definition 6.1.12 The *tangent hyperplanes* to $\cap P$ are the unique supporting hyperplanes of an n-polyhedron in $\mathbb{R}^n$. ■

This means that the tangent hyperplanes are the affine hulls of the facets of $\cap P$. This definition concurs with the notion that the set of facets of a polyhedron is unique.
The relative boundary of a half-space is a hyperplane so that it is said that a half-space is bound by a hyperplane, or that a half-space has a boundary hyperplane. It follows then that the intersection of the set of supporting half-spaces that are each bound by a tangent hyperplane to a polyhedron uniquely defines that polyhedron, as in the following theorem:

Theorem 6.1.1 An n-dimensional closed convex set $C \in \mathbb{R}^n$ is the intersection of the closed half-spaces tangent to it. [Roc70][Theorem 18.8]. ■

For example, Table 4 illustrates these concepts with reference to the earlier Figure 9 of triangle ABC in $\mathbb{R}^2$.

representation of ABC in $\mathbb{R}^2$ dimension 1	tangent hyperplanes dimension 2	facets	improper supporting hyperplane	improper faces
$\cap T_1$ where $T_1 =$				
$\{1 \leq y,$	$y = 1$	AB	none	$\cap T_1$
$y \leq x,$	$x = y$	AC		
$x \leq 3\}$	$x = 3$	BC		

Table 4: Tangent hyperplanes and facets of ABC in $\mathbb{R}^2$

Since each tangent hyperplane to an n-polyhedron in $\mathbb{R}^n$ is unique, it follows that the set of supporting half-spaces tangent to $\cap P$ is also unique. In the previous Definition 6.1.12 the tangent hyperplane is considered relative to the affine hull of an n-polyhedron, $\cap P$ in $\mathbb{R}^n$, that is, relative to $\mathbb{R}^n$. Now, suppose that $\cap P \in Poly^n$ such that $\dim \cap P < n$. Following a similar line of reasoning to that used by Rockafellar, to extend the notion of the interior of n-polyhedra in $\mathbb{R}^n$ to the relative interior of m-polyhedra, $m \leq n$ in $\mathbb{R}^n$, the notion of a tangent hyperplane can be absorbed into that of a relative tangent hyperplane; the definition follows immediately.

Definition 6.1.13 Let $\cap P$ be an m-polyhedron in $\mathbb{R}^n$, $m \leq n$, then the affine hull of a facet of $\cap P$ is a *relative tangent hyperplane* of $\cap P$. ■

Note that the dimension of the relative tangent hyperplanes to $\cap P$, will be $m - 1$, that is, relative to the affine hull of $\cap P$, not to $\mathbb{R}^n$ in which $\cap P$ is embedded. This means that if the dimension of $\cap P$ is less than n the relative tangent hyperplanes are the affine hulls of the facets of $\cap P$. Table 5 illustrates the concept of a relative tangent hyperplane with reference to the earlier Figure 11 of the triangle ABC in $\mathbb{R}^3$.
For the following definitions let $F^c(\cap P)$ denote the facets of $\cap P \in Poly^n$ and $F(\cap P)$ denote the faces of $\cap P$. Note that $F^c(\cap P) \subseteq F(\cap P)$, but $F(\cap P) \nsubseteq F^c(\cap P)$.

representation of ABC in $\mathbb{R}^3$ dimension 2	relative tangent hyperplanes dimension 2	facets	improper supporting hyperplane	improper faces
$\cap T_2$ where $T_2 =$				
$\{1 \leq y,$	$y = 1$	AB		
$y \leq x,$	$x = y$	AC		
$x \leq 3,$	$x = 3$	BC		
$0 \leq z, z \leq 0\}$			$z = 0$	ABC

Table 5: Relative tangent hyperplanes and facets of ABC in $\mathbb{R}^3$

Definition 6.1.14 Let $\beta \in \mathit{Half}^n$ and $\cap P \in \mathit{Poly}^n$ then β is a *constraint* on $\cap P \leftrightarrow \cap P \subseteq \mathrm{rb}\,\beta \vee (\cap P \subseteq \beta \wedge \mathrm{rb}\,\beta \cap \cap P \in F^c(\cap P))$ and β is said to *constrain* $\cap P$. ▪

This means that if β constrains $\cap P$ then β supports $\cap P$. On the other hand, if β supports $\cap P$ then β may or may not constrain $\cap P$.

Hence, a set of constraints defines a unique polyhedron in $\mathbb{R}^n$, not because the set of constraints that define the polyhedron are necessarily unique, but because, by definition, a constraint contributes to the unique set of attributes that define the polyhedron, namely the affine hull that contains $\cap P$ and the relative tangent hyperplanes that contain the facets of $\cap P$.

Definition 6.1.15 $P \in \mathcal{P}(\mathit{Half}^n)$ is *minimal* iff $\forall \beta \in P \,.\, [\cap P \neq \cap(P \setminus \{\beta\})]$. ▪

This definition means that P is minimal only when every element of P constrains $\cap P$ and further, that no two elements contain the same facet within their relative boundaries. However, each element that contributes to the delineation of the affine hull of $\cap P$ will of necessity contain $\cap P$ within its relative boundary. For example:

Example 6.1.1 Consider the polyhedron $\cap P \in \mathbb{R}^2$, where $P = \{y \leq x, 1 \leq y, x \leq 2\}$. The closed half-space $1 \leq x$ supports $\cap P$ at the point $\langle 1, 1\rangle \in \cap P$ as $\langle 1, 1\rangle \in \mathrm{rb}\,(1 \leq x)$. Let $P' = P \cup \{1 \leq x\}$. Although $\cap P = \cap P'$ and all of the elements in each set support $\cap P$, P is a minimal representation and P' is not, as the removal of $1 \leq x$ from P' would not change the shape of $\cap P'$. ▪

Definition 6.1.7 of *supports* can be rephrased as shown by the following Lemma.

Lemma 6.1.1 Let $\beta \in \mathit{Half}^n$ and $\cap P \in \mathit{Poly}^n$, then the following are equivalent:

1. β *supports* $\cap P \leftrightarrow \cap P \subseteq \beta \wedge \cap P \cap \mathrm{rb}\,\beta \neq \phi$
2. β *supports* $\cap P \leftrightarrow \cap P \subseteq \mathrm{rb}\,\beta \vee (\cap P \subseteq \beta \wedge \mathrm{rb}\,\beta \cap \cap P \in F(\cap P))$

Proof Note that by definition, $\mathrm{rb}\,\beta$ is a supporting hyperplane of $\cap P$.
Assume that 1. holds, then there are two cases to consider:

- $\cap P \cap \mathrm{rb}\,\beta = \cap P$

  Since $\cap P \cap \mathrm{rb}\,\beta = \cap P$, and $\cap P$ is an improper face of $\cap P$, it follows that in this case $\mathrm{rb}\,\beta$ is an improper supporting hyperplane of $\cap P$, by Definition 6.1.10.

- $\cap P \cap \operatorname{rb} \beta \neq \cap P$

  Let $s = \cap P \cap \operatorname{rb} \beta$, since $\operatorname{rb} \beta$ is a supporting hyperplane of $\cap P$ and s is a non-empty subset of $\cap P$, it follows that s is a face of $\cap P$, and $\operatorname{rb} \beta$ is a proper supporting hyperplane of $\cap P$ at s.

Now assume that 2) holds, there are two cases to consider:

- $\cap P \subseteq \operatorname{rb} \beta$

  Since $\cap P \subseteq \operatorname{rb} \beta$, it follows that $\cap P \subseteq \beta \ \wedge \ \cap P \cap \operatorname{rb} \beta \neq \phi$.

- $\cap P \subseteq \beta \ \wedge \ \operatorname{rb} \beta \cap \cap P \in F(\cap P)$

  Since $\operatorname{rb} \beta \cap \cap P \in F(\cap P)$, then $\operatorname{rb} \beta \cap \cap P \neq \phi$, by definition of F. Hence, $\cap P \subseteq \beta \ \wedge \ \cap P \cap \operatorname{rb} \beta \neq \phi$.

■

6.2 Alternative Representations and Rotations

When $m = n$, the tangent hyperplanes are the relative boundaries of the closed supporting half-spaces of $\cap P$ and as the tangent hyperplanes are unique, so are the closed supporting half spaces of $\cap P$. On the other hand, if $m < n$ the relative tangent hyperplanes of $\cap P$ are affine subsets of the relative boundaries of the closed supporting half-spaces of $\cap P$. It is in this case that there are alternative representations of the same polyhedron. With a minimal representation in terms of shape, Definition 6.1.15, it is now possible to consider precisely what conditions allow alternative representations. Further, since the widening is defined in terms of exchanging elements of alternative representations it is important to understand when this can happen.

Section 6.2.1 describes the terms and introduces the notation that supports this discussion. Section 6.2.2 takes a closer look at what happens when the alternative supporting hyperplanes are actually improper supporting hyperplanes; and also at the conditions that allow them to be exchanged, in their respective representations, without changing the shape of the set of points they describe.

6.2.1 The Terms of Reference and Notation

Every affine subset of $\mathbb{R}^n$ is the intersection of a finite set of hyperplanes [Roc70] [Cor.1.4.1]. Hence, an affine set can be represented as the intersection of two opposing linear inequalities whose shared relative boundary is a hyperplane.
Let $P \in \mathcal{P}(\mathit{Half}^n)$ be a minimal representation of $\cap P$, an m-polyhedra in $\mathbb{R}^n$ where $m \in \{0, \ldots, n\}$. Now consider a partition of P, let $P = A_P \cup B_P$ where $A_P = \{\alpha \in P \mid \operatorname{rb} \alpha \cap \cap P = \cap P\}$ so that $\cap A_P = \operatorname{aff} \cap P$. Then $B_P = P \setminus A_P$, is a set of closed supporting half-spaces with relative boundaries that are proper supporting hyperplanes of $\cap P$. If $\dim \cap P = n$ then the proper supporting hyperplanes of $\cap P$ are the tangent hyperplanes to $\cap P$, but if $\dim \cap P < n$, then the proper supporting hyperplanes of $\cap P$ contain the unique relative tangent hyperplanes to $\cap P$. Recall that a tangent hyperplane to $\cap P$ is the affine hull of a facet of $\cap P$, so $B_P = \{\beta \in P \mid \operatorname{rb} \beta \cap \cap P \in F^c(\cap P)\}$. Hence $\cap A_P$ defines the m-space in which $\cap P$ is embedded and $\cap B_P$ the delineation of $\cap P$ within the m-space. The boundary hyperplanes of $\cap P$ that are the relative boundaries

of the elements of B_P can be thought of, informally, as cutting the m-space in which $\cap P$ is embedded into two disjoint sets. A_P and B_P are referred to as the A and the B *components* of P, respectively.

Alternative Representations in $\mathbb{R}^2$. Note that in a slight abuse of notation, the word *direction* is used to quantify the amount of space in which rotations can take place. Its use in this way is particular to this section and should not be confused with its use elsewhere in this thesis. The following Figure 12 and Table 6 serve to illustrate (i) the distinguishing features of a polyhedron and (ii) the function of the linear inequalities in the representation of a polyhedron. In Table 6, P_1 and P_2 both represent the same

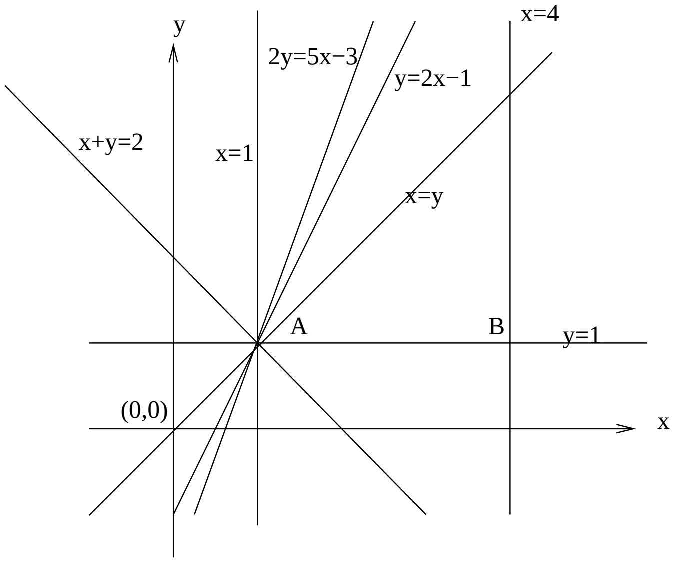


Figure 12: The line $AB \in \mathbb{R}^2$

	P_i	A_{P_i}	B_{P_i}	$H^T_{P_i}$	H_{P_i}
P_1	$\{1 \leq x, x \leq 4, 1 \leq y, y \leq 1\}$	$\{1 \leq y, y \leq 1\}$	$\{1 \leq x, x \leq 4\}$	$\{x = 1, x = 4\}$	$\{y = 1\}$
P_2	$\{y \leq x, x \leq 4, 1 \leq y, y \leq 1\}$	$\{1 \leq y, y \leq 1\}$	$\{y \leq x, x \leq 4\}$	$\{x = 1, x = 4\}$	$\{y = 1\}$

Table 6: The line $AB \in \mathbb{R}^2$

polyhedron, the line segment $y = 1 \in \mathbb{R}^2$ where $1 \leq x \leq 4$, named AB in Figure 12. In the table the second column contains A_{P_i} the half-spaces in P_i that define the affine hull of $\cap P_i$ and the third column B_{P_i} the half spaces in P_i that define $\cap P_i$ within it affine hull. The third column contains the set $H^T_{P_i}$ the set of relative tangent hyperplanes to $\cap P_i$, and the fourth column contains the hyperplanes, the intersection of which defines the affine hull H_{P_i} of $\cap P_i$. In this case the A_{P_i} sets are identical but the B_{P_i} sets are different. The relative tangent hyperplanes to the line segment AB are also identical, since they uniquely define AB within the line $y = 1$. Here, at the bottom end of the dimensional scale, the relative tangent hyperplanes to AB are of dimension zero and they are coincident with the facets of AB. AB is embedded in $\mathbb{R}^2$ and each half-space in the representation of AB is also of dimension 2, so the relative boundary of each half-space in B_{P_i} is a 1-dimensional hyperplane that intersects the affine hull of AB at a relative tangent hyperplane of AB. Clearly there are many hyperplanes that intersect the line $y = 1$ at the point where $x = 1$, for example, the lines $x = 1$ and $x = y$; and more of these lines are illustrated in Figure 12. These alternative hyperplanes are rotations[1] about the facet, the point $\langle 1, 1\rangle$, which allows alternative closed half-spaces in the representation of AB, as, for example, $1 \leq x$ in B_{P_1} is an alternative to $x \leq y$ in B_{P_2}.

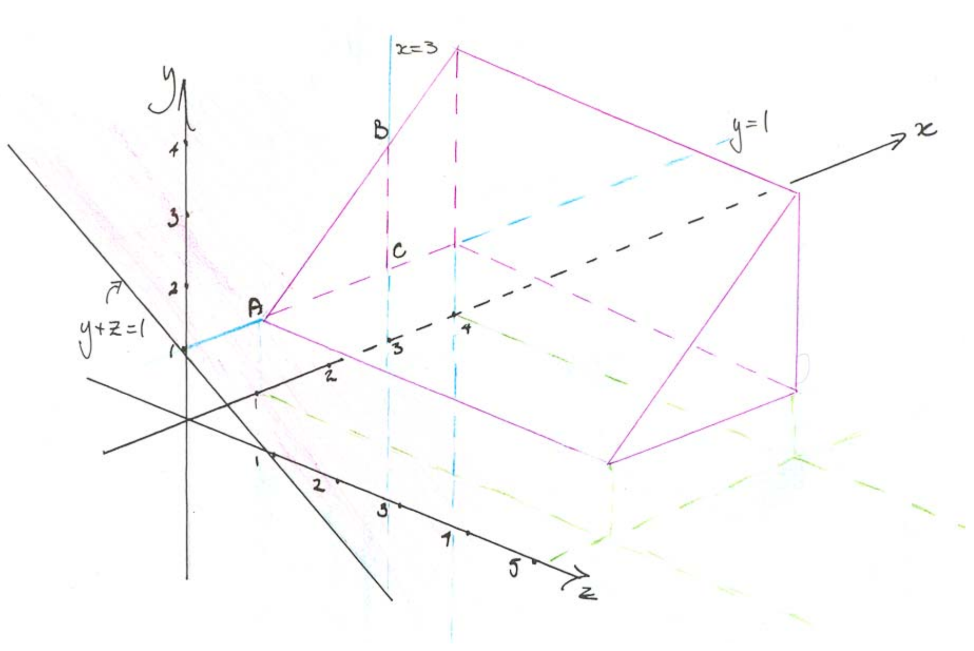


Figure 13: Polyhedra $P_i \in \mathbb{R}^3$ where $\dim \cap P_i = i$

Now consider what might happen in $\mathbb{R}^3$. In Table 7, the polyhedra, illustrated in Figure 13, are in $\mathbb{R}^3$ and this further illustrates how alternative representations of a polyhedron are possible. For example, consider P_2, the triangle ABC embedded in its affine hull the plane, $z = 0$. In P_2 the affine hull is defined by the elements of A_{P_2}. In this case the facets of ABC are line segments, AB, BC and AC, the union of which is the relative boundary of ABC; recall that the relative boundary is the boundary of the polyhedron within the confines of its affine hull. Since ABC is embedded in $z = 0$ the relative tangent hyperplanes of ABC, the lines $\{x = y,\ x = 3,\ y = 1\}$ can each be

[1]The notion of the rotation of a line about a point is well understood and can be extended to the rotation of a hyperplane and in consequence a half-space of $\mathbb{R}^n$ about any space of dimension k, where $0 \leq k < n - 1$, [Lay82] and [Gru67].

	$P_i = A_{P_i} \cup B_{P_i}$	$H^T_{P_i}$	H_{P_i}	$F^c_{P_i}$
P_1	$A_{P_1} = \{1 \leq y,\ y \leq 1\}$ $0 \leq z,\ z \leq 0\}$ $B_{P_1} = \{1 \leq x,\ x \leq 2\}$	$\{x = 1,\ x = 2\}$ dim $= 0$	$\{y = 1,$ $z = 0\}$ dim $= 1$	$\{x = 1,\ x = 2\}$ dim $= 0$
P_2	$A_{P_2} = \{0 \leq z,\ z \leq 0\}$ $B_{P_2} = \{1 \leq y,\ y \leq x,$ $x \leq 3\}$	$\{x = 3,\ y = 1,\ y = x\}$ dim $= 1$	$\{z = 0\}$ dim $= 2$	$\{x = 3,\ 1 \leq y \leq 3\}$ $\{y = 1,\ 1 \leq x \leq 3\}$ $\{y = x,\ 1 \leq x \leq 3\}$ dim $= 1$
P_3	$A_{P_3} = \phi$ $B_{P_3} = \{x \leq 4,$ $1 \leq y,\ y \leq x,$ $0 \leq z,\ z \leq 5\}$	$\{x = 4,\ y = 1,\ y = x,$ $z = 0,\ z = 5\}$ dim $= 2$	ϕ	$\{x = 4, 1 \leq y \leq 4,\ 0 \leq z \leq 5\}$ $\{y = 1,\ 1 \leq x \leq 4,\ 0 \leq z \leq 5\}$ $\{y = x,\ 1 \leq x \leq 4,\ 0 \leq z \leq 5\}$ $\{z = 0,\ 1 \leq x \leq 4,\ 1 \leq y \leq x\}$ $\{z = 5,\ 1 \leq x \leq 4,\ 1 \leq y \leq x\}$ dim $= 2$

Table 7: Polyhedra $P_i \in \mathbb{R}^3$ where dim $\cap P_i = i$

embedded in any of the 2-dimensional hyperplanes that intersect with $z = 0$ at those lines. Each alternative hyperplane containing a particular line is a rotation of all the other hyperplanes about that particular line. For example, the hyperplane $y + z = 1$ (coloured pink in Figure 13) intersects the plane $z = 0$ at the line $y = 1$, so in P_2, $1 \leq y$ could be replaced by $y + z \geq 1$ as ABC is a subset of $y + z \geq 1$ and is bound by the relative boundary of $y + z \geq 1$, namely $y + z = 1$. The prism, $\cap P_3$, illustrates the case where the A component is empty as $\cap P_3$ is a 3-polyhedron in $\mathbb{R}^3$ and the dimension of the relative tangent hyperplanes and of the facets is 2. There is no space to allow rotation so there is only one representation possible for $\cap P_3$.
At the other end of the scale, $\cap P_1$, the line segment AC in the plane $z = 0$, is now a candidate for many alternative representations. Switching back to $\mathbb{R}^2$ for a moment, consider the point $\langle 1,\ 1\rangle$, in $\mathbb{R}^2$ and its alternative boundary hyperplanes that can rotate about $\langle 1,\ 1\rangle$, effectively through π radians as rotations in $\mathbb{R}^2$ through more than π radians will be repeats of those through π radians. In $\mathbb{R}^3$ the alternative supporting hyperplanes that contain aff $\cap P_1$ are of dimension 2 and must all be rotations about aff $\cap P_1$, namely the line $y = 1$. Since they must contain a line of dimension 1 there is space to rotate through π radians in just one direction. However, the hyperplanes of dimension 2 that contain the facets of dimension zero, must intersect the plane of $z = 0$ at the point $\langle 1,\ 1\rangle$, so they can rotate through the full range of π in $z = 0$, but also through π again in each of many planes that intersect $z = 0$ at $y = 1$. So here the rotations of $y = 1$ can rotate through π radians in two directions. The proper supporting hyperplanes of an m-polyhedron in $\mathbb{R}^n$ support the m-polyhedron at k-faces where $0 \leq k < m$ and can rotate through π in $n - 1 - k$ directions about a k-face. Similarly the improper supporting hyperplanes are of dimension m, since they contain the polyhedron and can rotate through π about the m-polyhedron in $n-1-m$ directions. Extending the concept of rotations in this way allows the provision of an intuitive path to

understanding the way in which alternative systems of linear inequalities can represent polyhedral sets.

6.2.2 Exchanging Elements in Alternative Representations

Let the set of relative tangent hyperplanes to $\cap P \subseteq \mathbb{R}^n$ be H_P^T, and let the elements of H_P^T, each associated with a $\beta \in B_P$, be denoted h_β^T. Recall that for any m-polyhedron $\cap P \in \mathbb{R}^n$, $P = A_P \cup B_P$, so, when $m = n$ then $A_P = \phi$. Since when $m = n$, $\forall \beta \in B_P \,.\, [\dim h_\beta^T = n - 1]$, it is clear that there is only one minimal P. On the other hand, if $\cap P$ is a point in $\mathbb{R}^n$ then $B_P = \phi$, and A_P will comprise n pairs of closed supporting half-spaces, each pair represented by a pair of opposing inequalities, one pair for each hyperplane. It is possible to represent an affine space with a set of inequalities that is not made up of pairs of opposing inequalities but this consideration is dealt with later, in Section 6.5.1. As alternative representations of affine sets are possible, what can be asserted about any of these inequalities with regard to their being swapped between alternative systems?

In order to answer this question it is necessary to look closely at what happens when the representation of an affine set is a set of pairs of opposing closed half-spaces. When considering the intersection of closed half-spaces, focus is normally on the space that is included in the intersection, but if the space is affine, the function of each opposing pair of half-spaces is as much to exclude as to include. Consider the line represented by $P = \{1 \leq x, x \leq 1\} \in \mathbb{R}^2$. The affine hull of $\cap P$ is the same as $\cap P$ and $\cap P$ has only one face, the improper face that is $\cap P$. In this case there is only one possible representation; the key point is that the half-spaces that support $\cap P$ have no space in which to rotate around $\cap P$, so there are no alternative representations. The intersection of $\{1 \leq x, x \leq 1\}$ is the line $x = 1$, the dual property of this intersection is that it excludes any value for x that is less than 1 and any value for x that is greater than 1. The observation is then, that the function of the intersection operation on the half-spaces in the representation of $\cap P$ is not only to restrict the value of x but, in consequence, to exclude the two open half-spaces on either side of the $x = 1$. This means that the dimension of $\cap P$ is the dimension of the real space in which it is embedded reduced by 1, as the exclusion of the two open half-spaces has the effect of projecting out x, so that the dimension of $\cap P$ is 1. Hyperplanes are of dimension $n - 1$, so the intersection of two hyperplanes will be an affine space of dimension $n - 2$, the intersection of three hyperplanes is an affine space of dimension $n - 3$ and so on. Thus, in general to represent an affine space in $\mathbb{R}^n$ of dimension m in $\mathbb{R}^n$ then $n - m$ pairs of opposing inequalities are required to delineate the affine space. An example that appeals to intuition is the origin in $\mathbb{R}^n$. The origin is an affine space, the intersection of the hyperplanes that are the variable axes of $\mathbb{R}^n$. Applying the above reasoning, the intersection of the n hyperplanes that are the variable axes is the origin and this gives the dimension of the origin as $n - n = 0$, as expected.

From the previous paragraph it is clear that when there are opposing pairs of inequalities in the representation each pair has a dual function, to include the hyperplane that is their intersection and to exclude the two open half-spaces on either side of the hyperplane. When the dimension, m, of an affine set in $\mathbb{R}^n$ is such that $m \leq n - 2$, there is room for the half spaces to rotate about the delineated affine space and more than one representation is possible.

Consider the point $\langle 1, 1 \rangle$ in Figure 14. Three of the possible different representations of the point $\langle 1, 1\rangle$ in $\mathbb{R}^2$, described over the variables $\{x, y\}$ are given in the accompanying Table 8. The solution to each of the three systems of inequalities is the same, namely that $x = 1$ and $y = 1$ which is reflected most directly in the representation P_0.

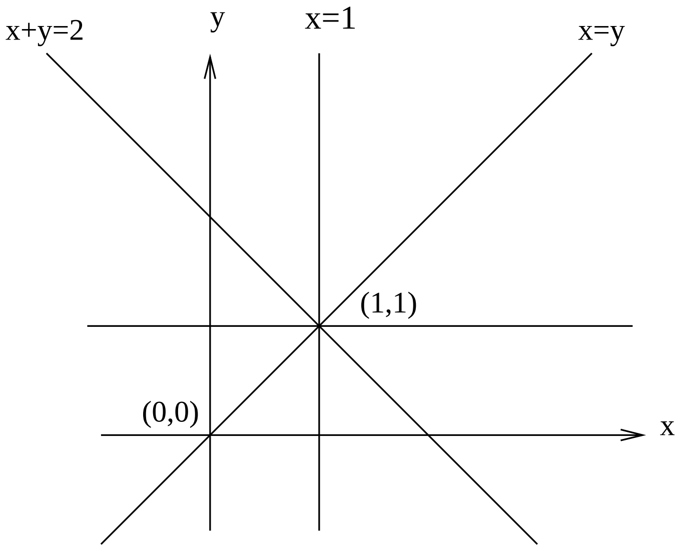


Figure 14: The point $\langle 1, 1\rangle \in \mathbb{R}^2$

	P_i	H_{P_i}
P_0	$\{1 \le x,\ x \le 1$ $1 \le y,\ y \le 1\}$	$\{x = 1,\ y = 1\}$
P_1	$\{2 \le x + y,\ x + y \le 2$ $x \le y,\ y \le x\}$	$\{x + y = 2,\ x = y\}$
P_2	$\{2 \le x + y,\ x + y \le 2$ $x \le y,\ y \le 1\}$	$\{x + y = 2,\ x = y,\ y = 1\}$

Table 8: Different Representations of the point $\langle 1, 1\rangle \in \mathbb{R}^2$

Consider now $\cap P_0$. The intersection of each opposing pair of inequalities is a hyperplane, in this case the lines $x = 1$ and $y = 1$. This representation is minimal as if any linear inequality were to be removed from P_0 that shape of $\cap P_0$ would change. This is confirmed, by observing that each of the open half lines bound at one end by the point, $\cap P_0$, is present in only three out of the four half-spaces associated with the representation. The absence of the open half line from the fourth half-space excludes its presence from the intersection. Consider $\langle 1, 1\rangle$ as depicted in Figure 14. The open half line $x = 1$ such that $y > 1$ is excluded in the opposing half-space $y \le 1$. Therefore any half space that does not include $x = 1$ such that $y > 1$ (or conversely, includes $x = 1$ such that $y < 1$) can replace $y \le 1$ in a representation of $\langle 1, 1\rangle$ without changing its shape. For example, in P_1 the half-space $y \le x$ could replace $y \le 1$ and in P_2 either

$y \leq x$ or $x + y \leq 2$ could replace $y \leq 1$, as each of the three half spaces does not include $x = 1$ such that $y > 1$.
This example concerns the representation of a point of dimension 0 in $\mathbb{R}^2$ and, as has been observed, since alternative representations of affine sets require the affine set to be of dimension at least 2 less than n, points are the only affine sets in $\mathbb{R}^2$ that can have alternative representations. The observations for the point in $\mathbb{R}^2$ can be generalised for any point in $\mathbb{R}^2$ as follows in Lemma 6.2.1 which is then used to generalise the premise for $\mathbb{R}^n$. The motivation for separating this preliminary proof for $\mathbb{R}^2$ from the general case for $\mathbb{R}^n$ is to introduce the proof tactic in the simplest case possible. It should be noted that the representations of affine sets are minimal in accordance with all representations in this discussion. The Tucker representation is a formal definition of such a minimal representation [Roc70], that confirms when the representation is minimal the number of hyperplanes required for an affine set of dimension m is $n - m$. Note that in the proof that follows α is used to denote one of a pair of opposing closed half-spaces and subscripts are employed only when it is necessary to distinguish between elements in a pair.

Lemma 6.2.1 Let A_1, $A_2 \subseteq \mathit{Half}^n$ such that the elements of A_i are opposing pairs of inequalities so that $\cap A_1$, $\cap A_2$ are affine sets in $\mathbb{R}^2$ and $\cap A_1 = \cap A_2$ but $A_1 \neq A_2$. Further let $\dim \cap A_i = m = 0$, where $0 \leq m \leq n-2$. Then $\forall \alpha \in A_1 . \exists \alpha' \in A_2 . \cap (A_1 \setminus \{\alpha\} \cup \{\alpha'\}) = \cap A_1$.

Proof Let $H_1 = \{h \mid h = \operatorname{rb} \alpha \wedge \alpha \in A_1\}$, so that $\cap A_1 = \cap H_1$. Note that H_1 is a set, not a multiset. Since the $\dim \cap A_1 = 0$ it follows that $|H_1| = 2$. Each hyperplane in H_1 is a line and there will be two open halves of that line that are not contained within $\cap H_1$. Let h', h'' be the open halves of a hyperplane h in H_1, so that $\{h'\} \cup \{h''\} \cup \cap H_1 = h$. This means that $\forall \bar{x}' \in h' . \bar{x}' \notin \cap A_1$. Now consider $\alpha \in A_1$. Suppose that $h \neq \operatorname{rb} \alpha$ so that $h' \not\subset \operatorname{rb} \alpha$. Since the union of α and its opposing half-space is $\mathbb{R}^n$, h' must be a subset of either α or its opposing half-space. Let $h' \subset \alpha$. In order for a linear inequality to replace α in A_1 without changing the shape of $\cap A_1$ the half-space it represents must include h' and in consequence exclude h''. Now consider the representation A_2. As A_2 contains $n - m = 2$ pairs of opposing half-spaces $\forall \bar{x} \in \mathbb{R}^n . \bar{x} \notin \cap A_2 . \bar{x} \in \alpha'$ for at least two α's, as $\forall \alpha_1, \alpha_2 \in \mathit{Half}^n . \forall \bar{x} \in \mathbb{R}^n . \bar{x} \in \alpha_1 \vee \bar{x} \in \alpha_2$, where (α_1, α_2) is a pair of opposing half-spaces.

Therefore $\exists \alpha' \in A_2 . h' \subset \alpha'$. There are three cases:

- if $h' \not\subset \operatorname{rb} \alpha'$ then $h' \subset \operatorname{ri} \alpha'$ and α' can replace α in A_1 without changing the shape of $\cap A_1$.
- if $h' \subset \operatorname{rb} \alpha'$ and $\alpha' = \alpha$ then trivially α' can replace $\alpha \in A_1$ without changing the shape of $\cap A_1$.
- if $h' \subset \operatorname{rb} \alpha'$ and $\alpha' \neq \alpha$ then α' and α are opposing pairs of inequalities so that the opposing inequality to α' can trivially replace α in A_1 without changing the shape of $\cap A_1$.

Hence, in all cases there exists a linear inequality in A_1 that can be replaced by a linear inequality in A_2 without changing the shape of $\cap A_2$. ■

The question that follows then is: What happens as alternative representations of affine spaces are considered in higher dimensions? The key observation is that providing there are sufficient degrees of freedom, that is, the dimension of the affine space is at least two less than the real space in which it is embedded, the way the hyperplanes, and in consequence the pairs of opposing inequalities each representing a closed half-space, interact is constant, no matter what the value of m and n are. Hence the proof tactic employed in the previous lemma is not dependent on the fact that $n = 2$.

Lemma 6.2.2 Let A_1, $A_2 \subseteq \mathit{Half}^n$ such that the elements of A_i are opposing pairs of inequalities so that $\cap A_1$, $\cap A_2$ are affine sets in $\mathbb{R}^n$ and $\cap A_1 = \cap A_2$ but $A_1 \neq A_2$. Further let $\dim \cap A_i = m$, where $0 \leq m \leq n-2$. Then $\forall \alpha \in A_1 \,.\, \exists \alpha' \in A_2 \,.\, \cap(A_1 \setminus \{\alpha\} \cup \{\alpha'\}) = \cap A_1$.

Proof The proof tactic is to consider a particular hyperplane in H_1 as in the case where $n = 2$. Consider $H_1 \setminus \{h, g\}$ where $h, g \in H_1$. Both h and g intersect $\cap H_1 \setminus \{h, g\}$ and the function of h is to exclude both g' and g'' Let $H_1 = \{h \mid h = \mathrm{rb}\,\alpha \wedge \alpha \in A_1\}$, so that $\cap A_1 = \cap H_1$. Therefore for every hyperplane in H_1 there will be two open halves of that hyperplane that are not contained within $\cap H_1$. Let h', h'' be the open halves of a hyperplane h in H_1, so that $\{h'\} \cup \{h''\} \cup \cap H_1 = h$. This means that $\forall \bar{x}' \in h' \,.\, \bar{x}' \notin \cap A_1$. Now consider $\alpha \in A_1$. Suppose that $h \neq \mathrm{rb}\,\alpha$ so that $h' \not\subset \mathrm{rb}\,\alpha$. Since the union of α and its opposing half-space is $\mathbb{R}^n$, h' must be a subset of either α or its opposing half-space. Let $h' \subset \alpha$. In order for a linear inequality to replace α in A_1 without changing the shape of $\cap A_1$ the half-space it represents must include h' and in consequence exclude h''. Now consider the representation A_2. As A_2 contains $n - m \geq 2$ pairs of opposing half-spaces $\forall \bar{x} \in \mathbb{R}^n \,.\, \bar{x} \notin \cap A_2 \,.\, \bar{x} \in \alpha'$ for at least two α's. Therefore $\exists \alpha' \in A_2 \,.\, h' \subset \alpha'$. There are three cases:

- if $h' \not\subset \mathrm{rb}\,\alpha'$ then $h' \subset \mathrm{ri}\,\alpha'$ and α' can replace α in A_1 without changing the shape of $\cap A_1$.
- if $h' \subset \mathrm{rb}\,\alpha'$ and $\alpha' = \alpha$ then trivially α' can replace $\alpha \in A_1$ without changing the shape of $\cap A_1$.
- if $h' \subset \mathrm{rb}\,\alpha'$ and $\alpha' \neq \alpha$ then α' and α are opposing pairs of inequalities so that α' can trivially replace the opposing inequality to α in A_1 without changing the shape of $\cap A_1$.

Hence, in all cases there exists a linear inequality in A_1 that can be replaced by a linear inequality in A_2 without changing the shape of $\cap A_2$. ■

The following Corollary 6.2.1 explicitly states that pairs of opposing inequalities can always be exchanged between alternative representations allowing Corollary 6.2.2 which demonstrates that when $\cap A_1 \subset \cap A_2$, all the elements in A_2 are replaceable by elements in A_1, say A_1' without changing the shape of $\cap A_2$, and the elements in A_1' can be replaced in A_1 by those in A_2 without changing the shape of $\cap A_1$.

Corollary 6.2.1 Let $\cap A_1$, $\cap A_2$ be affine sets in $\mathbb{R}^n$ such that $\cap A_1 = \cap A_2$ but $A_1 \neq A_2$. Further let $\dim \cap A_i = m$, where $0 \leq m \leq n-2$. Then, when $\{\alpha_1, \alpha_2\}$ and $\{\alpha_1', \alpha_2'\}$ are opposing pairs of inequalities, $\forall \alpha_1, \alpha_2 \in A_1 \,.\, \exists \alpha_1', \alpha_2' \in A_2 \,.\, \cap(A_1 \setminus \{\alpha\} \cup \{\alpha'\}) = \cap A_1$.

Proof Let $\cap(A_1 \setminus \{\alpha_1\} \cup \{\alpha_1\}') = \cap A_1$, then by Lemma 6.2.1 α_1' performs the same functions with regard to delineating affine space as α_1. Therefore since α_2 performs the opposite function to α_1 it follows that as α_2' performs the opposite function to α_1', α_2' can replace α_2 in A_1 without changing the shape of $\cap A_1$. ■

Corollary 6.2.2 Let $\cap A_1$, $\cap A_2$ be affine sets in $\mathbb{R}^n$ such that $\cap A_1 \subset \cap A_2$ but $A_2 \not\subset A_1$. Further let $\dim \cap A_i = m_i$, where $0 \leq m_1 \leq n-2$, and $0 \leq m_2 \leq n-1$. Then $\forall \alpha' \in A_2 \,.\, \exists \alpha \in A_1 \,.\, \cap(A_1 \setminus \{\alpha\} \cup \{\alpha'\}) = \cap A_1$.

Proof Consider $\cap A_2' = \cap A_1$, such that $A_1 \neq A_2'$. By Corollary 6.2.1 every pair of opposing half-spaces in A_2' can replace a pair of opposing half-spaces in A_1. Suppose that a pair of opposing half spaces is removed from A_2' and the remaining non-empty set is called A_2. Every pair of opposing half-spaces in A_2 can still replace a pair of opposing half-spaces in A_1, as the removal of half-spaces from A_2' does not affect the shape of A_1. However, now $\cap A_1 \subset \cap A_2$, and hence, $\forall \alpha' \in A_2 \,.\, \exists \alpha \in A_1 \,.\, \cap(A_1 \setminus \{\alpha\} \cup \{\alpha'\}) = \cap A_1$. ■

6.3 Representation Independence

This section is concerned only with demonstrating that $\nabla^{\mathcal{H}}$ is representation independent when certain pre-conditions are met. The demonstration is couched in terms of the, now identifiable, unique spatial properties of polyhedra in $\mathbb{R}^n$ rather than in algebraic terms. Therefore, those readers with a primary interest in the observations that are the outcome of this investigation, may wish to proceed to Section 6.4 and the ensuing sections which build on the insights afforded by this investigation.

The following definition is taken from [CC92a], but it should be noted that in [Hal79] the definition given in association with this widening makes the assumption that equalities are represented as pairs of opposing inequalities. A discussion concerning the possible outcomes if this assumption is not adhered to, concludes this section. The widening is defined as follows:

Definition 6.3.1 Let $P_i \in \mathcal{P}(\mathit{Half}^n)$, and $\cap P_0, \ldots, \cap P_i, \ldots$ be an increasing sequence such that $\forall i \leq j \,.\, \cap P_i \subseteq \cap P_j$. Then the sequence $Q_i = Q_{i-1} \nabla^{\mathcal{H}} P_i$ is defined: $Q_i = P_i$ when $i = 0$ and $Q_i = Q_{i-1}' \cup P_i'$, otherwise, where

$$\begin{array}{lcl} Q_{i-1}' & = & \{\beta \in Q_{i-1} \mid \cap P_i \subseteq \beta\} \\ P_i' & = & \{\gamma \in P_i \mid \exists \beta \in Q_{i-1} \wedge \cap(Q_{i-1} \setminus \{\beta\} \cup \{\gamma\}) = \cap Q_{i-1}\} \end{array}$$

■

To support the proof that the widening $\nabla^{\mathcal{H}}$ is representation independent it is necessary to affirm certain properties of the iterates that are the outcome of applying the widening to a sequence of increasing polyhedra. The proof of the supporting Lemma 6.3.2 is by induction and it depends on the fact that in the widening sequence it can be asserted that whenever an inequality is in $B'_{Q_{i-1}}$, the B component of Q_{i-1}' it means that the inequality supports $\cap P_i$. This last is shown formally in the following Lemma 6.3.1.

Lemma 6.3.1 Consider Q_{i-1} in the sequence defined by Definition 6.3.1, so that $Q_{i-1} = Q_{i-2} \nabla^{\mathcal{H}} P_{i-1}$, for all $i \geq 2$. Assume that $\forall \beta \in B_{Q_{i-1}} \,.\, \beta$ constrains $\cap P_{i-1}$, then $\forall \beta \in B_{Q_{i-1}} \,.\, [\cap P_i \subseteq \beta \rightarrow \beta$ supports $\cap P_i]$, and if further, the affine hulls of $\cap P_{i-1}$ and $\cap P_i$ are the same then β also constrains $\cap P_i$.

Proof (i) The proof is by contradiction. Consider $\beta \in B_{Q_{i-1}} . \cap P_i \subseteq \beta$.

suppose	$\cap P_i \subseteq \text{ri}\,\beta$	
then	$\forall \bar{x} \in \text{rb}\,\beta\,[\bar{x} \notin \cap P_i]$	
but	$\exists \bar{y} \in \text{rb}\,\beta\,[\bar{y} \in \text{rb} \cap P_{i-1}]$	as β constrains P_{i-1}
and	$\forall \bar{x} \in \text{rb} \cap P_{i-1}\,[\bar{x} \in \cap P_i]]$	as $P_{i-1} \subseteq \cap P_i$ by defn P_i
hence	$\cap P_i \not\subseteq \text{ri}\,\beta$	by contradiction
therefore	$\exists \bar{x} \in \cap P_i\,[\bar{x} \in \text{rb}\,\beta]$	as $\cap P_i \subseteq \beta$
and	β supports $\cap P_i$	by Definition 6.1.7 of supports

(ii) Let aff $\cap Q_{i-1} =$ aff $\cap P_{i-1} =$ aff $\cap P_i$, and their dimension be m. Since $\forall \beta \in B_{Q_{i-1}} . \beta$ constrains both $\cap Q_{i-1}$ and $\cap P_{i-1}$, and aff $\cap\, Q_{i-1} =$ aff $\cap\, P_{i-1}$, it follows that β constrains both $\cap Q_{i-1}$ and $\cap P_{i-1}$ at a facet.

	$\forall \beta \in Q_{i-1}\,[\beta$ constrains $\cap P_{i-1}]$	assumption
therefore	$\text{rb}\,\beta \cap \cap P_{i-1} \subseteq \cap P_i$	as $\cap P_{i-1} \subseteq \cap P_i$
suppose	$\exists \bar{x} \in \text{rb}\,\beta \cap \cap P_{i-1}\,[\bar{x} \in \text{ri} \cap P_i]$	
then	$\exists \bar{y} \in \cap P_i\,.\,\bar{y} \notin \beta$	
but	$\forall \bar{x} \in \cap P_i\,.\,\bar{x} \in \beta$	as $\cap P_i \subseteq \beta$
therefore	$\nexists \bar{x} \in \text{rb}\,\beta \cap \cap P_{i-1}\,.\,[\bar{x} \in \text{ri} \cap P_i]$	by contradiction
and	$\forall \bar{x} \in \text{rb}\,\beta \cap \cap P_{i-1}\,.\,[\bar{x} \in \text{rb} \cap P_i]$	as $\cap P_{i-1} \subseteq \cap P_i$
now	$\dim \text{rb}\,\beta \cap \cap P_{i-1} = m - 1$	as β constrains P_{i-1} at a facet of $\cap P_{i-1}$
but	$\forall f^c \in F^c(\cap P_i)\,.\,\dim f^c = m - 1$	
therefore	$\text{rb}\,\beta \cap \cap P_i \in F^c(\cap P_i)$	
and	β constrains $\cap P_i$	by Definition 6.1.14 of constrains

■

Corollary 6.3.1 It follows from Lemma 6.3.1 that aff $\cap\, Q_{1-1} =$ aff $\cap\, P_i \;\rightarrow\; Q'_{i-1} = \{\beta \in Q_{i-1} \mid \beta$ constrains $\cap P_i\}$ and aff $\cap\, Q_{1-1} \neq$ aff $\cap\, P_i \;\rightarrow\; Q'_{i-1} = \{\beta \in Q_{i-1} \mid \beta$ supports $\cap P_i\}$, where Q'_{i-1} is defined as in Definition 6.3.1. ■

The following Lemma 6.3.2 demonstrates:

- how the affine hull and therefore the dimension of successive iterates is dictated by that of $\cap P_i$, and
- how the elements of the A and B components of Q_{i-1} and P_i, are placed in the A and B components of $Q_{i-1} \triangledown^{\mathcal{H}} P_i$.

It is this result that allows the proof strategy in Proposition 6.3.1 to safely focus on the A and B components of the operands independently and to pin-point the area where an element from an A component in Q_{i-1} will become an element of the B component in $Q_{i-1} \triangledown^{\mathcal{H}} P_i$. In an abuse of notation the space $\cap\{\beta\}$, is abbreviated to β for reason of clarity; it will be clear from the context whether β is an inequality or the set of points that satisfy β.

Lemma 6.3.2 Let Q_i, $P_i \in \mathcal{P}(\mathit{Half}^n)$ and $Q_i = Q_{i-1} \triangledown^{\mathcal{H}} P_i$ as in Definition 6.3.1. Then

- aff $\cap\, Q_i =$ aff $\cap\, P_i$
- $Q_i = \{\beta \mid \beta$ constrains $\cap Q_{i-1} \wedge \beta$ constrains $\cap P_i\}$

- aff $\cap Q_{i-1} =$ aff $\cap P_i \ \rightarrow \ \cap Q'_{i-1} = \cap P'_i$
- aff $\cap Q_{i-1} \subsetneq$ aff $\cap P_i \ \rightarrow \ \cap P'_i \subseteq \cap Q'_{i-1}$

Proof Let $Q_{i-1} = A_{Q_{i-1}} \cup B_{Q_{i-1}}$, and $P_i = A_{P_i} \cup B_{P_i}$. Hence, $Q_i = A'_{Q_{i-1}} \cup B'_{Q_{i-1}} \cup A'_{P_i} \cup B'_{P_i}$, by definition of $\bigtriangledown^{\mathcal{H}}$. The proof is by induction on the number of representations of polyhedra, i in the sequence. The tactic employed is to consider the A and B components of each operand separately and then to consider how they interact with one another to define the new polyhedron $\cap Q_i$. So for each widening operation, $Q'_{i-1} = A'_{Q_{i-1}} \cup B'_{Q_{i-1}}$ and $P'_i = A'_{P_i} \cup B'_{P_i}$.

- The base step:
  - Case 1: aff $\cap P_0 =$ aff $\cap P_1$
    * consider $A'_{P_0} = \{\alpha \in A_{P_0} \mid \cap P_1 \subseteq \alpha\}$
      $\forall \alpha \in A_{P_0} . [\cap P_1 \subseteq \alpha]$ as $\cap A_{P_0} = \cap A_{P_1}$. Hence, $A'_{P_0} = A_{P_0}$ and $\cap A'_{P_0} = \cap A_{P_0}$.
    * consider $A'_{P_1} = \{\alpha' \in A_{P_1} \mid \exists \beta \in P_0 . [\cap(P_0 \setminus \{\beta\} \cup \{\alpha\}) = \cap P_0]\}$
      $\forall \alpha' \in A_{P_1} . [\exists \alpha \in A_{P_0} . [\cap(P_0 \setminus \{\alpha\} \cup \{\alpha'\}) = \cap P_0]]$, by Corollary 6.2.1. Hence $A'_{P_1} = A_{P_1}$ and $\cap A'_{P_1} = \cap A_{P_1}$.
    * consider $B'_{P_0} = \{\beta \in B_{P_0} \mid \cap P_1 \subseteq \beta\}$. Therefore, by Corollary 6.3.1:

$$\begin{aligned} B'_{P_0} &= \{\beta \in B_{P_0} \mid \beta \text{ constrains } \cap P_1\} \\ &= \{\beta \in B_{P_0} \mid \exists \gamma \in B_{P_1} . [h^T_\beta \in H^T_{P_0}, h^T_\gamma \in H^T_{P_1} . [h^T_\beta = h^T_\gamma]]\} \end{aligned}$$

    * consider $B'_{P_1} = \{\gamma \in B_{P_1} \mid \gamma \text{ constrains } \cap P_0\}$. This means that:

$$B'_{P_1} = \{\gamma \in B_{P_1} \mid \exists \beta \in B_{P_0} . [h^T_\gamma \in H^T_{P_1}, h^T_\beta \in H^T_{P_0} . [h^T_\gamma = h^T_\beta]]\}$$

    Therefore, if $\beta \in B'_{P_0}$ then there is a $\gamma \in B'_{P_1}$ such that their intersection with the affine hull of both P_0 and P_1 is the same, and vice versa. Hence, $\cap(A'_{P_0} \cup B'_{P_0}) = \cap(A'_{P_1} \cup B'_{P_1})$ as $\cap A'_{P_0} = \cap A'_{P_1}$ and $\cap B'_{P_0} = \cap B'_{P_1}$, that is $\cap P'_0 = \cap P'_1$.
  - Case 2: aff $\cap P_0 \subset$ aff $\cap P_1$
    * consider $A'_{P_0} = \{\alpha \in A_{P_0} \mid \cap P_1 \subset \alpha\}$. As aff $\cap P_0 \subset$ aff $\cap P_1$ this means that $\alpha \in A'_{P_0}$ either supports $\cap P_1$ at an improper face or at a proper face of $\cap P_1$. Therefore:

$$\begin{aligned} A'_{P_0} \ = \ & \{\alpha \in A_{P_0} \mid \cap P_1 \subseteq \text{rb}\,\alpha\} \cup \\ & \{\alpha \in A_{P_0} \mid \cap P_1 \subseteq \alpha \wedge \cap P_1 \not\subseteq \text{rb}\,\alpha\} \end{aligned}$$

      The intersection of those elements of A'_{P_0} that contribute improper supporting hyperplanes of $\cap P_1$ will be aff $\cap P_1$. The remaining elements of A'_{P_0} contribute proper supporting hyperplanes of $\cap P_1$ and therefore will contribute to the definition of $\cap Q_1$ within aff $\cap Q_1$. It follows that if α contributes a proper supporting hyperplane to $\cap P_1$ then α supports $\cap P_1$ and therefore $\exists \gamma \in B_{P_1} . [\exists h^T_\gamma \in H^T_{P_1} . [h^T_\gamma \subseteq \text{rb}\,\alpha]]$.

Therefore, $\cap A'_{P_0}$ defines either an affine space equal to the affine hull of P_1, or a space within the same affine hull as $\cap P_1$ that is defined by some elements that support $\cap P_1$. (2)

* consider $A'_{P_1} = \{\alpha' \in A_{P_1} \mid \alpha' \text{ constrains } \cap P_0\}$. By Corollary 6.2.2: $\forall \alpha' \in A_{P_1} . [\exists \alpha \in A_{P_0} . [\cap(A_{P_0} \setminus \{\alpha\} \cup \{\alpha'\}) = \cap A_{P_0}]]$, therefore $A'_{P_1} = A_{P_1}$ and $\cap A'_{P_1} = \cap A_{P_1}$.
* consider $B'_{P_0} = \{\beta \in B_{P_0} \mid \cap P_1 \subseteq \beta\}$. Therefore, by Corollary 6.3.1:

$$\begin{aligned} &= \{\beta \in B_{P_0} \mid \beta \text{ supports } \cap P_1\} \\ &= \{\beta \in B_{P_0} \mid \exists \gamma \in B_{P_1} . [h^T_\beta \in H^T_{P_0}, h^T_\gamma \in H^T_{P_1} . [h^T_\beta \subset h^T_\gamma]]\} \end{aligned}$$

Since $\dim h^T_\beta < \dim h^T_\gamma$, and $\cap P_0 \subset \cap P_1$ it follows that the relative tangent hyperplane to $\cap P_0$ contained within the boundary of β is a subset of the relative tangent hyperplane to $\cap P_1$ contained within the boundary of γ. If $h^T_\gamma \subseteq \operatorname{rb} \beta$ then β constrains $\cap P_1$ as it contains a relative tangent hyperplane of $\cap P_1$. However, in general it is not possible to assert that any element of B'_{P_0} constrains $\cap P_1$, only that it supports $\cap P_1$.

* consider $B'_{P_1} = \{\gamma \in B_{P_1} \mid \gamma \text{ constrains } \cap P_0\}$. This means that:

$$B'_{P_1} = \{\gamma \in B_{P_1} \mid \exists \beta \in B_{P_0} . [h^T_\gamma \in H^T_{P_1}, h^T_\beta \in H^T_{P_0} . [h^T_\gamma \supset h^T_\beta]]\}$$

From the above $\operatorname{aff} \cap Q_1 = \operatorname{aff} \cap P_1$, and the definition of Q_1 within its affine hull is described by the conjunction of (i) any element of A'_{P_0} that does not contain $\cap P_1$ within its relative boundary, and (ii) the elements of B'_{P_0} and B'_{P_1}. Since the elements of B'_{P_1} constrain $\cap P_0$ by definition, but those in A'_{P_0} or B'_{P_0} only support $\cap P_1$ any α or β that is not a constraint on P_1 will be redundant in the intersection of the union of those two sets. On the other hand, if any α or β is not redundant it must constrain $\cap P_1$ (that is, it intersects the affine hull of $\cap P_1$) at the same place as some γ from B'_{P_1}. Therefore, if all αs in A'_{P_0} and βs in B'_{P_0} constrain $\cap P_1$ it follows that $\cap P'_1 = \cap P'_0$. Hence, in general $\cap(A'_{P_1} \cup B'_{P_1}) \subseteq \cap(A'_{P_0} \cup B'_{P_0})$, that is $\cap P'_1 \subseteq \cap P'_0$ and as $Q_0 = P_0$ then $\cap P'_1 \subseteq \cap Q'_0$.

- The induction step:

  By the induction hypothesis:

  - $\operatorname{aff} \cap Q_{i-1} = \operatorname{aff} \cap P_{i-1}$
  - $Q_{i-1} = \{\beta \mid \beta \text{ constrains } \cap P_{i-2} \wedge \beta \text{ constrains } \cap P_{i-1}\}$
  - $\operatorname{aff} \cap Q_{i-1} = \operatorname{aff} \cap P_{i-1} \rightarrow \cap Q'_{i-2} = \cap P'_{i-1}$
  - $\operatorname{aff} \cap Q_{i-2} \subseteq \operatorname{aff} \cap P_{i-1} \rightarrow \cap P'_{i-1} \subseteq \cap Q'_{i-2}$

  Since

  - Case 1: $\operatorname{aff} \cap Q_{i-1} = \operatorname{aff} \cap P_i$
    * consider $A'_{Q_{i-1}} = \{\alpha \in A_{Q_{i-1}} \mid \cap P_i \subseteq \alpha\}$
      $\forall \alpha \in A_{Q_{i-1}} . [\cap P_i \subseteq \alpha]$ as $\operatorname{aff} \cap Q_{i-1} = \operatorname{aff} \cap P_i$. Hence $A'_{Q_{i-1}} = A_{Q_{i-1}}$ and $\cap A'_{Q_{i-1}} = \cap A_{Q_{i-1}}$.

  - consider $A'_{P_i} = \{\alpha' \in P_i \mid \exists\beta \in Q_{i-1}\,[\cap(Q_{i-1} \setminus \{\beta\} \cup \{\alpha\}) = \cap Q_{i-1}]\}$ $\forall\alpha' \in A_{P_i} \,.\, [\exists\alpha \in A_{Q_{i-1}} \,.\, [\cap(Q_{i-1} \setminus \{\alpha\} \cup \{\alpha'\}) = \cap Q_{i-1}]]$ by Corollary 6.2.1. Hence $A'_{P_i} = A_{P_i}$ and $\cap A'_{P_i} = \cap A_{P_i}$.
  - consider $B'_{Q_{i-1}} = \{\beta \in B_{Q_{i-1}} \mid \cap P_i \subseteq \beta\}$. Therefore, by Corollary 6.3.1:

$$\begin{aligned} B'_{Q_{i-1}} = & \{\beta \in B_{Q_{i-1}} \mid \beta \text{ constrains } \cap P_i\} \\ = & \{\beta \in Q_{i-1} \mid \exists\gamma \in B_{P_i} \,.\, [\exists h^T_\beta \in H^T_{Q_{i-1}} \,.\, \exists h^T_\gamma \in H^T_{P_i} \,.\, [h^T_\beta = h^T_\gamma]]\} \end{aligned}$$

  - consider $B'_{P_i} = \{\gamma \in B_{P_i} \mid \gamma \text{ constrains } \cap P_{i-1}\}$. That is:

$$B'_{P_i} = \{\gamma \in B_{P_i} \,.\, \exists\beta \in B_{Q_{i-1}} \,.\, [\exists h^T_\gamma \in H^T_{P_i} \,.\, \exists h^T_\beta \in H^T_{Q_{i-1}} \,.\, [h^T_\gamma = h^T_\beta]]\}$$

  Therefore, if β is in $B'_{Q_{i-1}}$ then there is a γ is in B'_{P_i} such that the intersection with the affine hull of both $\cap Q_{i-1}$ and $\cap P_i$ is the same and vice versa which means that $\cap B'_{Q_{i-1}} = \cap B'_{P_i}$. Further, aff $\cap\, Q_{i-1} =$ aff $\cap\, Q'_{i-1} =$ aff $\cap\, P_i =$ aff $\cap\, P'_i$. Therefore $\cap(A'_{Q_{i-1}} \cup B'_{Q_{i-1}}) = \cap(A'_{P_i} \cup B'_{P_i})$ as $\cap A'_{Q_{i-1}} = \cap A'_{P_i}$ and $\cap B'_{Q_{i-1}} = \cap B'_{P_i}$, that is $\cap Q'_{i-1} = \cap P'_i$.

- Case 2: aff $\cap\, Q_{i-1} \subset$ aff $\cap\, P_i$
  - consider $A'_{Q_{i-1}} = \{\alpha \in A_{Q_{i-1}} \mid \cap P_i \subseteq \alpha\}$
    Following the same reasoning as in the base step Case 2: for A'_{P_0} (see 2), aff $\cap\, A'_{Q_{i-1}} = \cap A_{P_i}$, and $A'_{Q_{i-1}}$ may contain elements that do not contribute to an improper supporting hyperplane, but that do support $\cap P_i$, and possibly constrain $\cap P_i$. Therefore:

$$\begin{aligned} A'_{Q_{i-1}} = & \{\alpha \in A_{Q_{i-1}} \mid \cap P_i \subseteq \text{rb}\,\alpha\} \cup \\ & \{\alpha \in A_{Q_{i-1}} \mid \cap P_i \subseteq \alpha \wedge \cap P_i \not\subseteq \text{rb}\,\alpha\} \end{aligned}$$

    The intersection of the relative boundaries of those elements of $A'_{Q_{i-1}}$ that contribute improper supporting hyperplanes of $\cap P_i$ will be equal to aff $\cap\, P_i$. Those elements of $A'_{Q_{i-1}}$ that contribute improper supporting hyperplanes of $\cap P_i$ will contribute to the definition of $\cap Q_i$ within aff $\cap$ Q_i. It follows that if α contributes a proper supporting hyperplane to $\cap P_i$ then α supports $\cap P_i$ and therefore $\exists\gamma \in B_{P_i} \,.\, [h^T_\gamma \in H^T_{P_i}\,[h^T_\gamma \subseteq \text{rb}\,\alpha]]$. Therefore, $\cap A'_{Q_{i-1}}$ defines either an affine space equal to the affine hull of P_i, or a space within the same affine hull as $\cap P_i$ that is defined by some element(s) that support $\cap P_i$. (2)
  - consider $A'_{P_i} = \{\alpha \in A'_{P_i} \mid \alpha \text{ constrains } Q_{i-1}\}$

$$\begin{array}{lll} \text{now} & \cap A_{Q_{i-1}} \subset \cap A_{P_i} & \text{as aff } \cap Q_{i-1} \subset \text{aff } \cap P_i \\ \text{and} & \forall\alpha' \in A_{P_i} \,.\, [\exists\alpha \in A_{Q_{i-1}} \,. & \\ & [\cap(A_{Q_{i-1}} \setminus \{\alpha\} \cup \{\alpha'\}) = \cap A_{Q_{i-1}}]] & \text{by Corollary 6.2.2} \\ \text{therefore} & A'_{P_i} = A_{P_i} & \end{array}$$

  - consider $B'_{Q_{i-1}} = \{\beta \in Q_{i-1} \mid \cap P_i \subseteq \beta\}$. Therefore, by Corollary 6.3.1

$$\begin{aligned} B'_{Q_{i-1}} & = \{\beta \in B_{Q_{i-1}} \mid \beta \text{ supports } \cap P_i\} \\ & = \{\beta \in B_{Q_{i-1}} \mid \exists\gamma \in B_{P_i} \,.\, [h^T_\beta \in H^T_{Q_{i-1}},\, h^T_\gamma \in H^T_{P_i} \,.\, [h^T_\beta \subset h^T_\gamma]]\} \end{aligned}$$

From the above $\mathrm{aff} \cap Q_i = \mathrm{aff} \cap P_i$, and the definition of Q_i within its affine hull is described by the conjunction of (i) any element of $A'_{Q_{i-1}}$ that does not contain $\cap P_i$ within its relative boundary, and (ii) the elements of $B'_{Q_{i-1}}$ and B'_{P_i}. Since the elements of B'_{P_i} constrain $\cap P_0$ by definition, but those in $A'_{Q_{i-1}}$ or $B'_{Q_{i-1}}$ only support $\cap P_1$ any α or β that is not a constraint on P_i will be redundant in the intersection of the union of those two sets. On the other hand, if α or β is not redundant it must constrain $\cap P_i$ (that is, it intersects the affine hull of $\cap P_i$) at the same place as some γ from B'_{P_i}. Therefore, if all αs in $A'_{Q_{i-1}}$ and βs in $B'_{Q_{i-1}}$ constrain $\cap P_i$ it follows that $\cap P'_i = \cap Q'_{i-1}$. Hence, in general $\cap(A'_{P_i} \cup B'_{P_i}) \subseteq \cap(A'_{Q_{i-1}} \cup B'_{P_i})$, that is $\cap P'_i \subseteq \cap Q'_{i-1}$.

* consider B'_{P_i}

$$\begin{aligned} B'_{P_i} &= \{\gamma \in B_{P_i} \mid \gamma \text{ constrains } \cap Q_{i-1}\} \\ &= \{\gamma \in B_{P_i} \mid \exists \beta \in B_{Q_{i-1}} . [h_\gamma^T \in H_{P_i}^T,\, h_\beta^T \in H_{Q_{i-1}}^T . [h_\gamma^T \subset h_\beta^T]]\} \end{aligned}$$

Therefore, as before, when $\beta \in B'_{Q_{i-1}}$ or $\alpha \in A'_{Q_{i-1}}$ support $\cap P_i$ but do not constrain $\cap P_i$, then in $\cap(A'_{Q_{i-1}} \cup B'_{Q_{i-1}} \cup A'_{P_i} \cup B'_{P_i})$ it follows that β and α will be redundant. This means that $\cap P'_i \subseteq \cap Q'_{i-1}$ and that all elements of $P'_i \cup Q'_{i-1}$ constrain $\cap P_i$.

Hence, by induction the required properties are proven for Q_i, for all integers, $i \geq 0$.

■

The following proposition builds on the previous lemmas allowing a significant reduction in the number of cases that need to be considered in order to confirm that $\nabla^{\mathcal{H}}$ is representation independent. In the proof that follows, recall that when a linear inequality in a representation of a polyhedron *constrains* that polyhedron (Definition 6.1.14), it means that the linear inequality makes a contribution to the shape of the polyhedron by contributing either to the delineation of the affine hull of the polyhedron or to the delineation of the polyhedron within its affine hull.

Proposition 6.3.1 Let $P_i \in \mathcal{P}(\mathit{Half}^n)$ and $\cap P_i$ be an increasing sequence of polyhedra. Further, let $\cap P_i = \cap R_i$ and $Q_i = Q_{i-1} \nabla^{\mathcal{H}} P_i$ and $K_i = K_{i-1} \nabla^{\mathcal{H}} R_i$. Then, providing equalities are represented as pairs of opposing inequalities, $\cap Q_i = \cap K_i$, that is, $\nabla^{\mathcal{H}}$ is representation independent. Put $\cap Q = \cap Q_{i-1} = \cap K_{i-1}$ and $\cap P = \cap P_i = \cap R_i$.

Proof Recall that $\nabla^{\mathcal{H}}$ is defined over the elements of the sets of inequalities in the representation of polyhedra in $\mathbb{R}^n$. When both polyhedra are of dimension n since the representation of an n-polyhedron in $\mathbb{R}^n$ is unique the result follows. It is sufficient to consider a) $\mathrm{aff} \cap P_{i-1} \subseteq \mathrm{aff} \cap Q_{i-1}$, as by Lemma 6.3.2 $\mathrm{aff} \cap Q_i = \mathrm{aff} \cap P_i$, b) the B components of P'_i and R'_i, as by Lemma 6.3.2 $\mathrm{aff} \cap Q_i = \mathrm{aff} \cap P_i$, and $\cap P'_i \subseteq \cap Q'_{i-1}$.

* Case 1. $\mathrm{aff} \cap Q_{i-1} = \mathrm{aff} \cap P_i$ and $\dim \cap Q_{i-1} < n$.

  $\forall \beta \in B_{P_i} . \exists \beta' \in B_{R_i} . \beta \cap \mathrm{aff} \cap Q = \beta' \cap \mathrm{aff} \cap Q$, as $\mathrm{aff} \cap Q = \mathrm{aff} \cap P$. Therefore, whenever β constrains $\cap Q$, β' is a constraint on $\cap Q$. Hence, $\cap P'_i = \cap R'_i$ and $\cap Q_i = \cap K_i$.

- Case 2. aff $\cap Q_{i-1} \subset$ aff $\cap P_i$

  By Definition 6.3.1, $\forall \beta \in P'_i \,.\, \beta$ constrains $\cap Q$ and β constrains $\cap P$; and $\forall \beta' \in R'_i \,.\, \beta'$ constrains $\cap Q$ and β' constrains $\cap P$.

  $\cap P_i = \cap R_i = \cap P$, therefore $\forall \beta \in B_{P_i} \,.\, \exists \beta' \in B_{R_i} \,.\, \beta \cap$ aff $\cap P = \beta' \cap$ aff $\cap P$ and vice versa. As aff $\cap Q_i =$ aff $\cap K_i =$ aff $\cap P$, and aff $\cap Q \subset$ aff $\cap P$ it follows that whenever $\beta \in B_{P_i}$ constrains both $\cap Q$ and $\cap P$, there is a $\beta' \in B_{R_i}$ that also constrains both $\cap Q$ and $\cap P$, and vice versa.

  Therefore, $\cap P'_i = \cap R'_i$ and hence $\cap Q_i = \cap K_i$.

Therefore, providing equalities are represented as pairs of opposing inequalities, since in all cases $\cap Q_i = \cap K_i$, the widening $\bigtriangledown^{\mathcal{H}}$ is representation independent. ■

6.4 When is this Widening Useful?

Properly, the first question that arises when considering polyhedral approximation (or any other spatial approximation) is: When can increasing sets of points be sensibly approximated? Sensible approximations of increasing sequences can only be made when there is a repeating pattern associated with an identifiable subset of the whole domain. The widening described here is useful and it is, perhaps, simpler to start with what is useful rather than what is not. At the outset of this discussion the widening could be informally described as returning the common bounds of its operands. It is now clear precisely what this means. The widening returns a set of half-spaces, where each element of that set is a supporting half-space to and that constrains both of the polyhedra that are the operands. Therefore, this widening will return $\mathbb{R}^n$ if there are no half-spaces that constrain both operands, that is, if the increasing sequence is such that there are no bounds common to successive polyhedra then the widening will return $\mathbb{R}^n$. This means that the widening will be most useful when the monotonic function that generates the increasing sequence of polyhedra is not only monotone on the polyhedra, but also, over each variable in the set X, of n variables over which the polyhedra in $\mathbb{R}^n$ are described. That is, if $\cap P \in \mathbb{R}^n$, and $\mathcal{F}$ is a monotonic and increasing function over polyhedra in $\mathbb{R}^n$ then for every $\langle x_1, \ldots, x_n \rangle \in \cap P$ and every $i \in [1, \ldots n]$, there is no $\langle x'_1, \ldots, x'_n \rangle \in \mathcal{F}(\cap P)$ such that $x'_i < x_i$. (This covers the case where the polyhedra increase in a positive way over the variable set, but would similarly apply if they were to increase in a negative way over the variable set. That is, if $\cap P \in \mathbb{R}^n$, and $\mathcal{F}$ is a monotonic and increasing function over polyhedra in $\mathbb{R}^n$ then for every $\langle x_1, \ldots, x_n \rangle \in \cap P$ and every $i \in [1, \ldots n]$, there is no $\langle x'_1, \ldots, x'_n \rangle \in \mathcal{F}(\cap P)$ such that $x'_i > x_i$.) This is confirmed by the example in [BJT99] referred to in Chapter 3, Figure 5 where there was only one supporting half-space that constrained every polyhedra in the sequence and the increasing sequence was not monotone over the variables x and y. Now consider Figure 15, that depicts a sequence of increasing squares with no common bounds, a sequence of irregular unbound polyhedra also with no common bounds where the widening would return $\mathbb{R}^2$, and finally a sequence of polyhedra with common bounds, where the widening would return the shaded space.

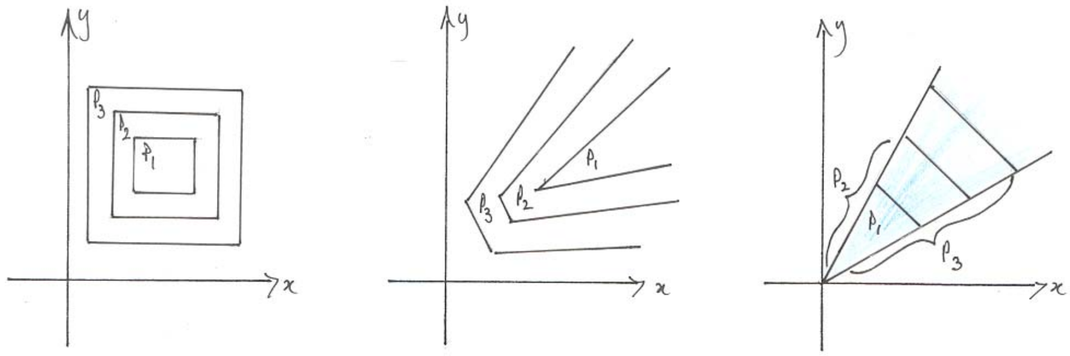


Figure 15: Increasing sequences of polyhedra.

6.5 Representation of affine hulls

$\nabla^{\mathcal{H}}$ is defined with representations of polyhedra as sets of inequalities. If the inequalities that delineate the affine hull of a polyhedron are not pairs of opposing inequalities then in certain circumstances the outcome of the widening may not be as expected. This is demonstrated in Section 6.5.1. In section 6.5.2 the consequence of lifting the restriction on equalities in the representation of a polyhedron is considered.

6.5.1 Inequalities

The pre-condition that affine hulls be represented as sets of opposing inequalities suggests that in an automated system some kind of re-write system would have to be employed as no solver would return equalities as opposing inequalities. In fact this is suggested in [HPR94] where this widening is proposed in an abstract interpretation that models reachable states of hybrid automata. Given the previous discussion it is clear that this pre-condition is in place so that when the dimension increases in the sequence of P_is entailed information is not lost. However, this pre-condition masks the fact that an affine space can be represented as a set of linear inequalities that do not contain any pairs of opposing inequalities and if this is the case then it is possible for the widening to return different outcomes for different representations of the same polyhedra. Representation independence relies on the inductive proof of Lemma 6.3.2 which is supported by Corollaries 6.2.1 and 6.2.2 to auxiliary Lemma 6.2.2 that only holds when affine sets are represented as pairs of opposing inequalities. Consider Example 6.5.1 and the associated diagram, Figure 15a (at the end of this chapter).

Example 6.5.1 Let $P_{0a} = \{y \geq 1 - x, y \geq x + 1, y \leq 1\}$, $P_{0b} = \{0 \leq x, x \leq 0, 1 \leq y, y \leq 1\}$ and $P_1 = \{0 \leq x, x \leq 3, 1 \leq y, y \leq 1\}$. Both P_{0a} and P_{0b} represent the point $\langle 0, 1\rangle$; P_1 represents the line segment $y = 1$ between and including $x = 0$ and $x = 3$; and P_2 represents the triangle bound by $y \leq x, 1 \leq y$ and $x \leq 3$. Consider the sequence of Q_is generated by $\bigtriangledown^{\mathcal{H}}$, first using P_{0a} then the sequence of Q_is using P_{0b}.

$Q_0 = P_{0a}$			$Q_0 = P_{0b}$		
$Q_1 = Q_0 \bigtriangledown^{\mathcal{H}} P_1$	$=$	$Q'_0 \cup P'_1$	$Q_1 = Q_0 \bigtriangledown^{\mathcal{H}} P_1$	$=$	$Q'_0 \cup P'_1$
Q'_0	$=$	$\{y \geq 1 - x, y \leq 1\}$	Q'_0	$=$	$\{0 \leq x, 1 \leq y, y \leq 1\}$
P'_1	$=$	$\{0 \leq x, y \leq 1\}$	P'_1	$=$	$\{0 \leq x, 1 \leq y, y \leq 1\}$
$Q'_0 \cup P'_1$	$=$	$\{y \geq 1 - x, y \leq 1\}$	$Q'_0 \cup P'_1$	$=$	$\{0 \leq x, 1 \leq y, y \leq 1\}$
$Q_2 = Q_1 \bigtriangledown^{\mathcal{H}} P_2$	$=$	$Q'_1 \cup P'_2$	$Q_2 = Q_1 \bigtriangledown^{\mathcal{H}} P_2$	$=$	$Q'_1 \cup P'_2$
Q'_1	$=$	$\{y \geq 1 - x\}$	Q'_1	$=$	$\{0 \leq x, 1 \leq y\}$
P'_2	$=$	ϕ	P'_2	$=$	$\{1 \leq y, y \leq x + 1\}$
$Q'_1 \cup P'_2$	$=$	$\{y \geq 1 - x\}$	$Q'_1 \cup P'_2$	$=$	$\{1 \leq y, y \leq x + 1\}$

■

Minimal representations of affine sets do not necessarily have the same number of elements in the way that B_P component of the representation of an n-polyhedron $\cap P$ does. Recall that in the representation of an m-polyhedron, $m \leq n$, the number of facets is constant, therefore, as each facet is contained within a single relative tangent hyperplane, the number of relative tangent hyperplanes is constant. Similarly, each relative tangent hyperplane is contained within a the relative boundary of a half-space in the representation. Hence $|B_P|$ is constant for all representations of $\cap P$. However, the situation is not so straightforward with affine spaces as the example illustrates. In the example $\cap Q_0$ is represented by just three inequalities, although there are many minimal representations of $\cap Q_0$ as two pairs of opposing inequalities. The problem occurs because only one of the inequalities that delineate the affine hull of $\cap P_1$ can actually replace any element of Q_0 without changing the shape of $\cap Q_0$. The example sequence with P_{0a} converges to a single half-space within two iterations and it can never grow, as a single half-space has a unique representation so no new element of any P_i could constrain it, no matter how $\cap P_i$ grows. In this example if $\cap P_i$'s growth is restricted to the increase in an upper bound for x then $\cap Q_i$ will be a safe approximation.

This prompts the question: Is it simply the odd number of constraints with no opposing pairs that causes this phenomenon? The answer is that it is not simply the lack of opposing pairs, as the following Example 6.5.2 illustrates. Here P_{0a} has no opposing pairs, but the widened Q_is with $Q_0 = P_{0a}$ are coincident with those that that arise from the sequence with $Q_0 = P_{0b}$.

Example 6.5.2 Let $P_{0a} = \{1 \leq x, x \leq y, x + y \leq 2\}$, $P_{0b} = \{1 \leq x, x \leq 1, 1 \leq y, y \leq 1\}$; $P_1 = \{0 \leq x, x \leq 3, 1 \leq y, y \leq 1\}$. Both P_{0a} and P_{0b} represent the point $\langle 1, 1\rangle$; P_1 represents the line segment $y = 1$ between and including $x = 0$ and $x = 2$; and P_2 represents the triangle bound by $y \leq x$, $1 \leq y$, $x \leq 3$. Consider the sequence of Q_is generated by $\bigtriangledown^{\mathcal{H}}$, first using P_{0a} then the sequence of Q_is using P_{0b}.

$Q_0 = P_{0a}$			$Q_0 = P_{0b}$		
$Q_1 = Q_0 \bigtriangledown^{\mathcal{H}} P_1$	$=$	$Q'_0 \cup P'_1$	$Q_1 = Q_0 \bigtriangledown^{\mathcal{H}} P_1$	$=$	$Q'_0 \cup P'_1$
Q'_0	$=$	$\{1 \leq x\}$	Q'_0	$=$	$\{1 \leq x,\, 1 \leq y,\, y \leq 1\}$
P'_1	$=$	$\{1 \leq x,\, 1 \leq y,\, y \leq 1\}$	P'_1	$=$	$\{1 \leq x,\, 1 \leq y,\, y \leq 1\}$
$Q'_0 \cup P'_1$	$=$	$\{1 \leq x,\, 1 \leq y,\, y \leq 1\}$	$Q'_0 \cup P'_1$	$=$	$\{1 \leq x,\, 1 \leq y,\, y \leq 1\}$
$Q_2 = Q_1 \bigtriangledown^{\mathcal{H}} P_2$	$=$	$Q'_1 \cup P'_2$	$Q_2 = Q_1 \bigtriangledown^{\mathcal{H}} P_2$	$=$	$Q'_1 \cup P'_2$
Q'_1	$=$	$\{1 \leq x,\, 1 \leq y,\, \}$	Q'_1	$=$	$\{1 \leq x,\, 1 \leq y\}$
P'_2	$=$	$\{y \leq x,\, 1 \leq y\}$	P'_2	$=$	$\{y \leq x,\, 1 \leq y\}$
$Q'_1 \cup P'_2$	$=$	$\{y \leq x,\, 1 \leq y, \}$	$Q'_1 \cup P'_2$	$=$	$\{y \leq x,\, y \leq 1\}$

■

Further questions arise: Is there any way of deducing that a representation without opposing-pairs is going to cause this type of problem? Is it worth the trouble to find out? Answers to both of these questions are pre-empted by the next section, however, beyond these two questions is a further consideration that is important. The post conditions for this widening are dependent on a pre-condition and although the outcome in Example 6.5.1, where the pre-condition is not met, is a safe approximation it does not return the expected outcome, the common bounds. In a situation where the use of the outcome depends on it being the expected outcome this may have serious consequences.

6.5.2 Equalities

Recall that in the widening, $Q_{i-1} \bigtriangledown^{\mathcal{H}} P_i = Q'_{i-1} \cup P'_i$, $P'_i = \beta' \in P_i \,.\, \exists \beta \in Q'_{i-1} \,.\, [\cap(Q_{i-1} \setminus \{\beta\} \cup \{\beta'\}) = \cap Q_{i-1}]$. The aim of the widening is to retain the common bounds and these are in effect the relative tangent hyperplanes within the relative boundaries of the elements of each representation. The idea is to focus on the relative boundaries as these are instrumental in delineation of the polyhedra. Suppose that affine hulls are represented as equalities, so that the representation of a polyhedron is a set of inequalities and equalities. In this case, then given that by definition, the relative boundary of an affine space is the affine space itself, P'_i could easily be computed from a set of mixed equalities and inequalities. Given also that by Lemma 6.3.2 $\cap P'_i \subseteq \cap Q'_{i-1}$ it follows that a widening defined:

$$Q_{i-1} \bigtriangledown P_i = P'_i \text{ where } P'_i = \{\beta' \in P_i \mid \forall \beta \in Q_{i-1} \,.\, \mathrm{rb}\, \beta' \cap \mathrm{rb}\, \beta \neq \phi\}$$

could be computed (in SICStus Prolog, for example) by posting the relative boundaries of each element in Q_{i-1} to the store in turn and then posting the relative boundary of each element in P_i also in turn onto the store (by simply replacing inequalities with equalities). As each relative boundary of an element of P_i is posted to the store, if the outcome evaluates to *true* then retain that element and continue, if not just continue until all the elements of P_i have been tested. Since Q_i is made up of elements from P_i, the rotations that occur when there is a dimension change need to be kept and the criterion here will allow this. However, this set of tests will have the effect of throwing out the translations of any previous bound as there will be an instance when a relative boundary of an element in Q_{i-1} has an empty intersection with the relative boundary of an element of P_i.

Since no re-writing is required and solvers handle equalities faster than inequalities such a widening based on P_i would be computationally advantageous. Further, this widening would adequately cover dimension changes when they occur so could be applied at any point in the increasing sequence.

Example 6.5.3 Let an increasing sequence of polyhedra be defined by $P_0 = \{x = 1, y = 1\}$, $P_1 = \{1 \leq x, x \leq 3, y = 1\}$, $P_2 = \{1 \leq y, y \leq x, x \leq 4\}$, and let $Q_0 = P_0$. In the table that follows the first and fourth columns show the initial posting to the store for the computation of Q_1 and Q_2 respectively. The second and fifth columns show each relative boundary that is posted individually to the store. If the intersection of an element in these columns with each of those in the first and fourth respectively is non-empty, then the linear inequality that represents the half-space that has that relative boundary is retained, in columns three and six respectively, for Q_1 and Q_2.

$\mathrm{rb}\,\beta\,.\,\beta \in Q_0$	$\mathrm{rb}\,\beta'\,.\,\beta' \in P_1$	$\beta' \in Q_1$	$\mathrm{rb}\,\beta\,.\,\beta \in Q_1$	$\mathrm{rb}\,\beta'\,.\,\beta' \in P_2$	$\beta' \in Q_2$
$x = 1, y = 1$	$x = 1$	$1 \leq x$	$x = 1, y = 1$	$y = 1$	$1 \leq y$
	$y = 1$	$y = 1$		$y = x$	$y \leq x$
	$x = 3$	–		$x = 4$	–

Hence the computation of the widening:

$$\begin{aligned} Q_1 = Q_0 \bigtriangledown P_1 &= \{1 \leq x, y = 1\} \\ Q_2 = Q_1 \bigtriangledown P_2 &= \{1 \leq y, y \leq x\} \end{aligned}$$

■

A further observation is that an initial step of stripping out like equalities, that is focusing the computation on the smallest affine space that contains both polyhedra, will further simplify the computation. In the example above, that would mean retaining $y = 1$ for Q_i, but removing $y = 1$ from the tests carried out by the solver. This last technique can be exploited in other circumstances as described in the next Section, 6.6.

6.6 Reducing Operations to the Affine Hull of the Union of Both Operands

Two observations were made early on in this chapter: i) that considering a polyhedron within its affine hull rather than just within $\mathbb{R}^n$ allowed useful insight into alternative representations of a polyhedron and ii) that the representation $P \in \mathcal{P}(\mathit{Half}^n)$ of a polyhedron may be considered in two distinct components: the A_P component that constitutes the delineation of $\mathrm{aff}\, P$ and the B_P component that delineates $\cap P$ within its affine hull. Consider the operations over polyhedra, namely, entailment, the closure of the convex hull and intersection. It is clear that for the first two of these operations (this discussion has focussed on binary versions of intersection and the closure of the convex hull, but since these operations are both commutative and associative, there is no loss of generality) then the computation can be restricted to the affine hull of the union of the operands. In the case of program analyses of the kind proposed here these operations are conducted over polyhedra P_1, P_2 where $\mathrm{aff}\, P_1 \subseteq \mathrm{aff}\, P_2$, so there may be computational advantages to be gained by reducing the amount of work required by the constraint solvers. The techniques described below will be most efficient when the output from the solvers is syntactically consistent between iterations.

Entailment Consider $\cap P_1$, $\cap P_2$ where their respective dimensions, m_1, m_2 are such that $m_i < n$, and $\cap P_1 \models \cap P_2$ is to be computed. When $\mathrm{aff}\, P_1 \subseteq \mathrm{aff}\, P_2$ entailment can be reduced to entailment within $\mathrm{aff}\, P_2$ which means that like equalities can be discarded from both P_1 and P_2. This means that the entailment test will in consequence be cheaper, as removing the equalities will be the equivalent of projecting out some variables and reducing the dimension, **before** the solvers begin computation. For example:

Example 6.6.1 Let $\cap P_1, \cap P_2 \subseteq \mathbb{R}^2$ such that $P_1 = \{x = 1, y = 1\}$, $P_2 = \{1 \leq x, x \leq 2, y = 1\}$. Suppose that the computer representation in the analysis of a logic program is such that P_1 `= [equals(var(1),1.0),equals(var(2),1.0)]` and P_2 `= [lesseq(1.0,var(1)),lesseq(var(1),2.0),equals(var(2),1.0)]`. Since $\mathrm{aff} \cap P_1 \subseteq \mathrm{aff} \cap P_2$, where the computer representation of P_1 and P_2 are lists containing identical elements of the form `equals(arg1, arg2)` before any elements are decoded and posted to the store the element that denotes $y = 1$, namely `equals(var(2),1.0)` is removed from both representations. This will simplify solver computation by projecting out y and reducing the entailment check to verifying whether or not $\cap\{x = 1\}$ entails $\cap\{1 \leq x, x \leq 2\}$. ▪

Closure of the Convex Hull In this case the common affine constraints should be retained for inclusion in the outcome of the closure of the convex hull operation, but otherwise computation can be reduced to that within the affine hull of the union of the operands. In the context of the analyses described here this is simply the affine hull of the operand with the greater dimension.

Intersection The situation is a little more complex here, as some pre-computation manipulation of the representations is required, and it is a moot point as to whether the time taken by the pre-processing would outweigh the advantage gained by posting a simpler computation to the constraint store.

Example 6.6.2 Let $\cap P_1, \cap P_2 \subseteq \mathbb{R}^3$ such that $P_1 = \{1 \leq x, x \leq 3, y \leq x, \underline{z = 0}\}$, $P_2 = \{0 \leq x, x \leq 4, \underline{y = 2}, -1 \leq z, z \leq 1\}$. In a similar way to that described in the previous example, the underlined constraints that are equalities can be removed from those to be posted to the constraint store, but retained for the solution. When the equalities equate a variable with a constant then the variable should be replaced with the constant wherever it occurs in the rest of the constraints. In this example, this would leave the constraints store to compute $\{2 \leq x, 1 \leq x, x \leq 3\} \cap \{0 \leq x, x \leq 4\}$, and then the outcome, $\{2 \leq x, x \leq 3\}$ is added to the equality constraints $\{y = 2, z = 0\}$, for the solution, $\{2 \leq x, x \leq 3, y = 2, z = 0\}$. ▪

Two further considerations are i) that in the case of equalities between variables, a choice of which variable to eliminate would have to be made and ii) as happens in the case above with the substitution $z = 0$ into the constraints on P_2, it would be sensible to discard inequalities like $-1 \leq 0$, but of course, this would require their detection.

6.7 Summary

The outcome of this investigation has yielded a precise understanding of when it is possible to have alternative representations of the same polyhedron as a set of linear inequalities. Further, as a result of this understanding it has been demonstrated that the widening of [Hal79] is representation independent in terms of the spatial attributes that uniquely define a polyhedron providing equalities are represented as pairs of opposing inequalities. A new less complex computation approach is suggested and some further ideas for reducing the computation involved in some frequently used operations over polyhedra are explored.

A constraint is a closed supporting half-space that either i) contributes an improper supporting hyperplane to the definition of the affine hull of an n-polyhedron in $\mathbb{R}^n$, or ii) contributes a relative tangent hyperplane to the definition of an m-polyhedron in $\mathbb{R}^n$ within its affine hull. Hence, a set of constraints defines the bounds of a polyhedron and the widening, $\bigtriangledown^{\mathcal{H}}$, returns the common bounds of its operands, also a polyhedron. The choice criteria for $\bigtriangledown^{\mathcal{H}}$ that are applied to the constraints in Q_{i-1} and P_i relate to the unique properties of the polyhedra, and this is the reason why $\bigtriangledown^{\mathcal{H}}$ is representation independent, providing the pre-condition that each equality is represented as a pair of opposing inequalities is met. It has been shown that $\forall \beta \in Q_i \,.\, [\beta \text{ constrains } \cap Q_{i-1} \wedge \beta \text{ constrains } \cap P_i]$. Since Q_0 is initialised to P_0 and in consequence, Q_1 is computed in terms of P_0 and P_1, it follows by Lemma 6.3.1 that the above statement can be couched in terms of P_i's as follows $Q_i = \{\beta \mid \forall i \geq 0 \,.\, \beta \text{ constrains } \cap P_i\}$. There are two principle cases. In Case 1, when the affine hulls of the operands are the same then $\cap Q'_{i-1} = \cap P'_i$. In Case 2 when the affine hulls of the operands are different, then $\cap P'_i \subseteq \cap Q'_{i-1}$ so that the constraints from Q_{i-1} are superfluous. This means that the widening operation could be reduced to computing only P'_i. However, the choice criterion for $\cap Q'_{i-1}$ is only less precise than that for $\cap P'_i$ when the dimensions of the operands are different. Thus if the widening is considered as applicable at any point in an increasing sequence, if it is delayed until the dimension has stabilised then $\cap Q'_{i-1}$ is as precise as $\cap P'_i$; and this technique is employed by [Sa97] who further observes that the nature of the increasing polyhedra is such that the two way entailment test can be reduced to a one way test. However, with an appropriate delay, $Q'_{i-1} = \{\beta \in Q_{i-1} \mid \cap P_i \subseteq \beta\}$ is also a suitable alternative widening. This tactic, employed over equalities as well as inequalities is the widening suggested in Chapter 5. Since the widening is delayed to allow relationships to stabilise, the dimension of the polyhedra also stabilises and Q'_{i-1} matches P'_i in precision.

The clarification of precisely what happens spatially has allowed a further adaptation of the original widening devised for representations of polyhedra that are mixtures of equalities and inequalities. This widening is based on P'_i as defined before, but with a prescribed computation method that exploits what is known about the way in which the different representations of a polyhedron interact and uniquely define a polyhedron. This widening (Section 6.5.2) is potentially the most attractive as it i) is computationally less complex, and ii) accommodates dimension changes in the sequence of increasing polyhedra and therefore it can be applied at any point in the sequence. A further benefit of the spatial insight afforded by this investigation is a reduced computation technique for both entailment and the closure of the convex hull proposed in Section 6.6 that reduces solver computation to the affine hull of the union of the operands.

Finally, there remains an insight into choosing the propitious moment to widen. If what has been observed here is associated with the observations towards the end of Chapter 5 regarding the stabilising of relationships with different classes of predicate, it is clear that i) the stabilising of the dimension and ii) the appearance of a constraint that is a translate of one in the previous iteration, are both good indicators of the stabilising of relationships. Dimension stability is likely to be simple to evaluate through tracking equalities in representations. Translates can be recognised when there is an empty intersection of the boundary hyperplanes of any two constraints from consecutive iterations. Both of these techniques are relatively cheap indicators as to when to widen.

Future investigations might focus on identifying and dealing with increasing sequences with the potential for approximation like those identified by [BJT99] in Figure 5 and sequences that although monotonic and increasing have a decreasing rate of increase that indicates a convergence to some limit.

Note, with reference to Figure 15a, the P_i's are in the middle column, the Q_i's commencing with $Q_0 = P_{0a}$ are in the left-most column and the Q_i's commencing with $Q_0 = P_{0b}$ are in the right-most column.

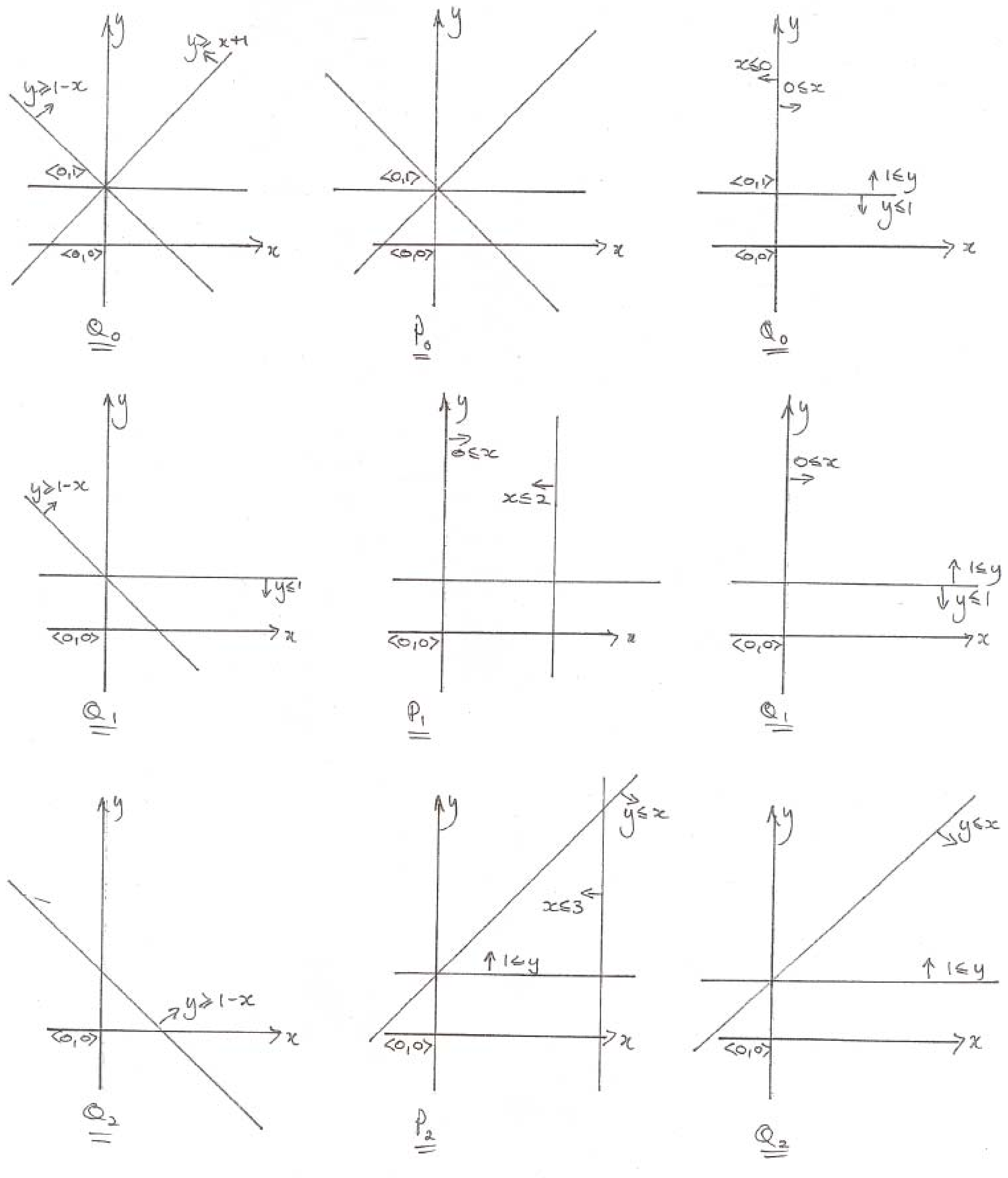


Figure 15a. Increasing sequence of polyhedra.

Chapter 7

φ : an Isomorphism

An Isomorphism between Abstract Polyhedral Cones and Definite Boolean Functions

7.1 Introduction

In logic programming Boolean functions are a well-known medium for capturing dependencies, typically between program variables or terms. Many of the representations of Boolean functions are not unique and manipulation with meet and join operations can prove computationally very expensive. In CLP systems the solvers required to manipulate linear inequalities are already in place. Therefore, the motivation was to explore the possibility of representing Boolean functions with linear inequalities, so that analyses using Boolean functions in CLP environments might be carried out without the cost of setting up complex computational machinery.

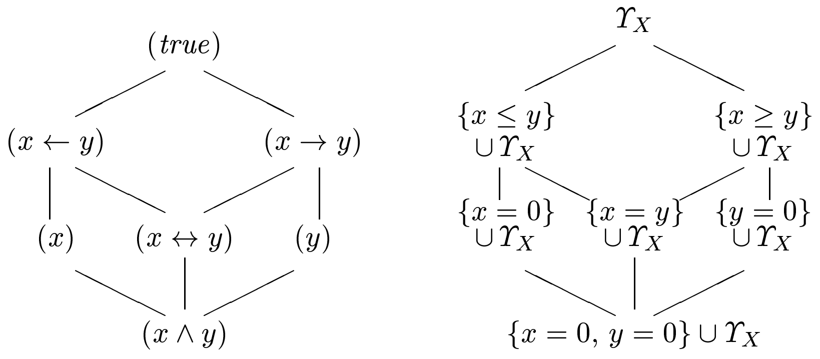
Consider the Boolean functions over variables $\{x, y, z\}$ expressing information about a particular property p and represented by propositional formulae. For example:

formula	dependency information
$x \leftarrow y$	if y has property p then x has property p
$x \wedge (y \leftarrow z)$	x has property p and if z has property p then y has property p

Similarly, given the observation above, this information could equally well be expressed by sets of linear inequalities over a set of non-negative variables $\{x, y, z\}$ in this way:

linear inequalities	dependency information
$\{0 \leq x, x \leq y\}$	if y has property p then x has property p
$\{x = 0, 0 \leq y, y \leq z\}$	x has property p and if z has property p then y has property p

However, linear inequalities cannot express all Boolean functions, for example, no linear inequality can express $x \vee y$, conveying the information either that x has property p or that y has property p. Linear inequalities can be viewed algebraically or spatially; so, by considering the conjunction of its elements, a set of linear inequalities can represent a set of points in $\mathbb{R}^n$. Such spaces are, by definition, polyhedra. In Figure 16 (ii) the set of points represented by each vertex in the Hasse diagram can be further distinguished as a cone since each space is closed under positive scalar multiplication. Since polyhedral cones are ordered by set inclusion this allows a semantic ordering between sets of


$(true)$
$(x \leftarrow y)$
$(x \rightarrow y)$
(x)
$(x \leftrightarrow y)$
(y)
$(x \wedge y)$
Υ_X
$\{x \leq y\} \cup \Upsilon_X$
$\{x \geq y\} \cup \Upsilon_X$
$\{x = 0\} \cup \Upsilon_X$
$\{x = y\} \cup \Upsilon_X$
$\{y = 0\} \cup \Upsilon_X$
$\{x = 0,\, y = 0\} \cup \Upsilon_X$

(i) Boolean functions.

(ii) Polyhedral cones. $\Upsilon_X = \{0 \leq x, 0 \leq y\}$

Figure 16: Hasse diagrams confirming an isomorphism in the dyadic case

inequalities with respect to the polyhedral cone that each set of inequalities represents. The Boolean functions in (i) are ordered by logical consequence. The Hasse diagrams in Figure 16, illustrate the association, an isomorphism, between certain Boolean functions and certain polyhedral cones in the case with two variables. In the two variable case identical diagrams confirm the isomorphism [DP90][Proposition 1.15]. In the general case (when there are more than two variables) the proof strategy is to demonstrate that the map is both bijective and a strict order embedding. The mapping requires a normal form for both domains.

The remainder of this chapter is divided as follows. Section 7.2 provides an informal view of the analogy between certain cones and certain Boolean functions. Section 7.3 demonstrates a normal form representation for Boolean functions in Def_X. Section 7.4 describes the mapping on cones that identifies the particular cones that capture dependencies. This mapping is known as the abstraction. Section 7.4.2 demonstrates that the abstraction mapping is both total and well-defined and further, that the mapping induces a normal form representation for abstract cones in $\alpha(Cone^n)$. Section 7.5 confirms that the relation between $Cone^n$ and $\alpha(Cone^n)$ is abstraction and that $\alpha(Cone^n)$ cannot be derived from $Cone^n$ through the introduction of equivalence classes. Section 7.6 introduces φ_X, the mapping between Def_X and $\alpha(Cone^n)$ and confirms the isomorphism. Section 7.7 concludes. This work is based on [BK99].

7.2 Polyhedral Cones and Boolean Functions

Recall that cones are closed under positive scalar multiplication and with the exception of the origin, a special case, they are unbound. Concern here is only with polyhedral cones, that is, cones that are the intersection of a set of closed half spaces. A closed half space can be represented as a non-strict linear inequality and, as illustrated in the previous Section 7.1, it is the linear inequalities in the representation of a polyhedral cone that allow the characterisation of dependencies.

In the discussion that follows only non-strict linear inequalities are considered but hereafter, for reasons of readability, **non-strict linear inequalities** are referred to as **linear inequalities**. Consider the polyhedral cone represented by the set: $\{0 \leq x,\, 3x \leq$

$2y\}$. Each linear inequality describes an inter-variable relationship over a numeric domain that is scaled by the ratio between the variable coefficients. In the example above the upper bound on x is two thirds of that on y, so that the relationship between x and y is scaled by the ratio of the coefficients of x and y, namely 3 and 2 respectively. In general a scaled relationship can only relate to variables with numeric domains. However, if all variable coefficients are unitary, then the relationships involve only one-to-one ratios, for example as in $\{x \leq y\}$, where the coefficient both of x and of y is one. Therefore, if variables are constrained to be either non-negative or zero and only unit variable coefficients are allowed then linear inequalities can characterise inter-variable dependency relationships that are valid in non-numeric domains in precisely the same way as certain Boolean functions.
Different polyhedral cones may encode the same unscaled inter-variable dependencies. For example, $\{0 \leq x, 3x \leq 4y\}$, $\{0 \leq x, 1.2x \leq 0.01y\}$ and $\{0 \leq x, x \leq y\}$, all express scaled inter-variable dependencies with a degree of precision relevant only in a numerical domain, but they all have the property that if y is zero then then it can be inferred that x is zero. This property is an abstract property in that it is an approximation of what is happening in the wider numeric domain. Therefore, by defining an abstraction operator that captures the means to infer that variables can be assigned the value zero, the three sets above can each be generalised to $\{0 \leq x, x \leq y\}$. Before abstraction the linear inequalities describe scaled dependencies between variables. Hence, after abstraction the linear inequalities are said to describe *unscaled* dependencies. It is these unscaled dependencies that map to the dependencies that can be captured by functions in Def_X. By defining an abstraction operator that effectively puts aside the non-unit coefficients whilst retaining only the dependency aspect of the inter-variable relationships, the abstract polyhedral cones can express the same dependency information as Def_X. The abstract polyhedral cones can be manipulated with the usual meet and join operators from the more expressive domain as any cone that is generated by their operations can be abstracted to a cone represented by linear inequalities describing unscaled inter-variable dependencies. It will be shown that abstraction collapses the infinite domain of convex cones to a finite subset of itself which, ordered by set inclusion, forms a complete lattice. In order to qualify the mapping between Boolean functions in Def_X and the abstract cones in $\alpha(Cone^n)$ a normal form representation of Boolean functions in Def_X is required. A Boolean function encodes, for each variable, inference rules that are a means of inferring that the variable is true when certain other variables are true. The next section derives a unique representation for Boolean functions as propositional formulae. For each function, this representation is in the form of conjoined implications where the implications are an explicit description of all of the inference rules for each variable.

7.3 Representation in Def_X

Various representations of Boolean functions can be derived from the conjunctive normal form of a definite sentence, $\wedge_{i=1}^{m}(\vee_{i=1}^{n} x_{ij})$ where each x_{ij} is either a propositional variable or the negation of a propositional variable. Reduced Monotonic Body Form [AMSS98, Dar91] is such a variant where each variable occurs exactly once as a head and each body is not only monotonic, but the variable in the head does not occur in the body. A function $f \in Def_X$ iff f can be represented in Reduced Monotonic Body Form

[AMSS98, Dar91].

Definition 7.3.1 A formula:

$$\bigwedge_{i=1}^{n} x_i \leftarrow M_i$$

is in *Reduced Monotonic Body Form* (RMBF) iff each $M_i \in Mon_{X\backslash\{x_i\}}$. ▪

Definition 7.3.2 A Boolean function $f = \bigwedge F$, described over a set of variables X, is in *Definite Monotonic Body Form* (DMBF), iff the following conditions on F hold:

$$F = \{y \leftarrow \bigwedge Y \mid \forall y \in X \,.\, \forall Y \subseteq X/\{y\} \,.\, [f \models y \leftarrow \bigwedge Y \ \wedge \ \forall Y' \subset Y \,.\, [f \not\models y \leftarrow \bigwedge Y']]\}$$ ▪

The definition of DMBF is such that F contains all of the implications for a particular variable that are entailed by f and including explicit transitive dependencies and no implication in F will entail any other in F. As transitive dependencies are explicit, DMBF is said to be in orthogonal form.

Example 7.3.1 Consider $f_1 = \bigwedge\{x \leftarrow y,\, y \leftarrow z,\}$ and $f_2 = \bigwedge\{x \leftarrow y,\, x \leftarrow z,\, y \leftarrow z\}$. Clearly $f_1 \equiv f_2$, but the representation of f_1 is not in orthogonal form whereas that of f_2 is. ▪

Proposition 7.3.1 If a formula $f \in Def_X$ then f can be represented in Definite Monotonic Body Form.

Proof A function $f \in Def_X$ iff f can be represented in RMBF [AMSS98]. Let $f = \bigwedge MB$ where

$$MB = \textstyle\bigcup_{i=1}^{n}\{x_i \leftarrow M_i \mid M_i \in Mon_{X\backslash\{x_i\}}\}$$

be in RMBF. The proof is in two parts.
Part 1. By relaxing the reduced condition, that each variable occurs at least and only once as a head, a set of implications F' can be derived from those in MB such that $\bigwedge F' \equiv \bigwedge MB$. The aim is to derive for each variable x_i, that is not implied only by *false*, a set of implications, F_i, such that the body of each implication in F_i is a conjunction of propositional variables and then $F' = \bigcup_{i \in \mathcal{I}} F_i$.
The proof is by induction on the depth, k, of the formulae that compose M_i.

- Base Step: Consider $x_i \leftarrow M_i$ where $k = 0$, that is, there are no connectives.
  - $x_i \leftarrow x_j$ where $i \neq j$, is in the required form.
  - $x_i \leftarrow \mathit{true} \ \equiv \ x_i \leftarrow \bigwedge \phi$ [Dar91], which is in the required form.
  - $x_i \leftarrow \mathit{false}$ is a tautology and can be discarded.
- Induction Step: Now consider $x_i \leftarrow M_i$, where $k > 0$, and by the induction hypothesis, let every formula M_i of depth $k-1$ or less be expressible as a conjunction of variables in the set X_i, such that $x_i \notin X_i$. Since M_i can only be constructed from Mon_X there are two cases to consider, $M_i = f_1 \wedge f_2$ and $M_i = f_1 \vee f_2$:
  - Case 1: $x_i \leftarrow f_1 \wedge f_2$. By the induction hypothesis $f_1 = \bigwedge X_i^1$ and $f_2 = \bigwedge X_i^2$. Therefore, $F_i = \{x_i \leftarrow \bigwedge(X_i^1 \cup X_i^2)\}$ and is in the required form.

– Case 2: $x_i \leftarrow f_1 \vee f_2$. By the induction hypothesis $f_1 = \bigwedge X_i^1$ and $f_2 = \bigwedge X_i^2$. Therefore, since $x_i \leftarrow \bigwedge X_i^1 \vee \bigwedge X_i^2 \equiv x_i \leftarrow \bigwedge X_i^1 \wedge x_i \leftarrow \bigwedge X_i^2$, it follows that $x_i \leftarrow f_1 \vee f_2$ can be replaced by the conjunction of the two implications on the right hand side of the equivalence, that are both in the required form. Hence $F_i = \{x_i \leftarrow \bigwedge X_i^1, x_i \leftarrow \bigwedge X_i^2\}$.

Hence as the hypothesis holds when $k = 1$ and when $k > 0$, by the principle of mathematical induction a function $f \in Def_X$ iff f can be represented as $\bigwedge_{i \in \mathcal{I}} x_i \leftarrow \bigwedge X_i$ where $X_i \subseteq X \setminus \{x_i\}$. The condition on X_i arises as in RMBF each $M_i \in Mon_{X \setminus \{x_i\}}$. Let $\bigcup_{i \in \{1...n\}} F_i = F'$ and then $\bigwedge F' \equiv \bigwedge MB$. Providing F' is derived from MB, in the manner of the inductive proof, transitive dependencies will be explicit. Further, $y \leftarrow \bigwedge Y \in F' \leftrightarrow f \models y \leftarrow \bigwedge Y$ and since no single implication in F' can entail another with a different single variable in the head, $y \leftarrow \bigwedge Y \in F' \rightarrow \forall Y' \subset Y . [f \not\models y \leftarrow \bigwedge Y']$.
Part 2. Now consider the definition of DMBF and a Boolean function $f = \bigwedge F$ where:

$$F = \{y \leftarrow \bigwedge Y \mid f \models y \leftarrow \bigwedge Y \ \wedge \ \forall Y' \subset Y . [f \not\models y \leftarrow \bigwedge Y']\}$$

It follows that if $f \in Def_X$ and f can be represented as $\bigwedge F'$ the f can also be represented as $\bigwedge F$. $F' \subseteq F$, and $\bigwedge F' \equiv \bigwedge F$, therefore $f \in Def_X$ iff f can be represented in DMBF.

■

7.4 The Abstraction of $Cone^n$

This section is divided into subsections as follows:

- 7.4.1 defines the mapping between polyhedral cones and abstract polyhedral cones.
- 7.4.2 comprises a discussion and proofs that demonstrate that the mapping is well-defined and prescribes a normal form for abstract cones.
- 7.4.3 defines the meet and join operators for abstract polyhedral cones and demonstrates that abstract cones form a complete lattice.

A polyhedral cone is the set of points contained by the intersection of a set of closed half spaces. As cones are closed under positive scalar multiplication this means that the closed half spaces have boundary hyperplanes that pass through the origin. Therefore the linear inequalities that represent such half spaces are of the form $\bar{b}\bar{x} \leq 0$. For example, see Chapter 4 Example 4.1.3. By constraining variables to non-negative values and restricting variable assignment only to zero, non-numeric dependencies can be described by linear inequalities with variables that have only unitary coefficients[1]. Recall that different sets of inequalities can encode the same unscaled dependencies between variables; for example, $x \leq 3y$, and $x \leq 0.6y$ express the same dependency as $x \leq y$, as in each case if $y = 0$ then it can be inferred that $x = 0$. Since it is only the dependencies that are of concern here, not the degree of dependency, the abstraction operator is defined solely with respect to the facility to infer that a variable can be assigned the value zero.

[1]It is assumed that the linear inequalities are such that like variables of the same order are coalesced. For example, neither $0 \leq x + y + x$ nor $0 \leq x^2 - xy + x + xy - x^2$ could occur in the representation of an abstract cone.

Since the abstraction maps from $Cone^n$ to $Cone^n$ the abstract cones are ordered by set inclusion and will retain their relative ordering as in $Cone^n$.

7.4.1 The Abstraction Operator

Hereafter the non-negative constraints on the totally ordered set of variables X are denoted by $\Upsilon_X = \{0 \le x \mid x \in X\}$, and E^α denotes a set of non-strict inequalities as mapped by the abstraction. Hence, the union, $\Upsilon_X \cup E^\alpha$, represents an abstract polyhedral cone in the non-negative orthant of n-dimensional space.

Definition 7.4.1 α: $Cone^n \rightarrow Cone^n$ Let $C \in Cone^n$, $y \in X$, $Y \subseteq (X \backslash \{y\})$.

$$\alpha(C) = \Upsilon_X \cup E^\alpha$$

where,

$$E^{\alpha'} = \{y \le \Sigma Y \mid (C \wedge \textstyle\bigcup_{y' \in Y} y' = 0) \models y = 0\}$$

and

$$E^\alpha = \{y \le \Sigma Y \in E^{\alpha'} \mid \nexists Y' . [Y' \subset Y \wedge (C \wedge \textstyle\bigcup_{y' \in Y'} y' = 0) \models y = 0]\}$$

■

This means that E^α in the representation of an abstract cone is minimal. Since α is defined over the whole of $Cone^n$ it is a total mapping and in the next Section 7.4.2, that follows, it is demonstrated that α is well-defined.

Example 7.4.1 Let $X = \{x_1, x_2, x_3\}$ and $C = \Upsilon_X \cup \{3x_1 \le x_2\}$. Now $(\Upsilon_X \cup \{x_1 \le x_2\} \cup \{x_2 = 0\}) \models (x_1 = 0)$, and $(\Upsilon_X \cup \{x_1 \le x_2\} \cup \{x_2 = 0, x_3 = 0\}) \models (x_1 = 0)$, but $\{x_1\} \subset \{x_2, x_3\}$, therefore $\alpha(C) = \Upsilon_X \cup \{x_1 \le x_2\}$. ■

In E^α there may be more than one non-strict inequality with the same variable on the left hand side of the inequality sign. In this case, all such non-strict inequalities are incomparable, for example:

Example 7.4.2 Let $C = \Upsilon_X \cup \{3x \le y, 0.5x \le 2z\}$, then $\alpha(C) = \Upsilon_X \cup \{x \le y, x \le z\}$, and $\{x \le y\} \not\parallel \{x \le z\}$. ■

Overall, the abstraction operator has the effect of relaxing the scaled dependencies by (i) making all coefficients unitary and (ii) relaxing equations of the form $\Sigma \bar{b} Y \le \Sigma \bar{c} Y'$, where $\bar{b}$ and $\bar{c}$ are $|Y|$ and $|Y'|$-vectors, respectively, in $\mathbb{R}$, to $\{y \le \Sigma Y' \mid \forall y \in Y\}$. The Example 7.4.3, that follows, illustrates both (i) and (ii):

Example 7.4.3 Let $X = \{x_1, x_2, x_3, x_4\}$, and $C \in Cone^n$, where $C = \Upsilon_X \cup \{2x_1 + x_2 \le 4x_3 + 1.5x_4\}$.

$$\alpha(C) = \Upsilon_X \cup \{x_1 \le x_3 + x_4, x_2 \le x_3 + x_4\}$$

■

The abstraction operator is such that α is a many-to-one, idempotent mapping with no inverse and it follows that $\alpha(Cone^n)$ is a strict subset of $Cone^n$ that is finite. Hereafter cones in $\alpha(Cone^n)$ are denoted C^α to distinguish them from cones that are not in $\alpha(Cone^n)$.

7.4.2 Representation in $\alpha(Cone^n)$

In general, different sets of inequalities may represent the same set of points, and linear combination of those inequalities can disclose entailed dependencies. The question that arises naturally, then, is this: "Is it possible for there to be dependencies in the representations of abstract cones that are not explicit?". The answer is "No.", but precisely what is meant by the technique known as linear combination when applied to linear inequalities and how the use of scalar multipliers might facilitate this technique must be considered first. This section is divided into subsections as follows:

- 7.4.2.1 comprises a discussion on the rationale for linear combination and definitions that allow the process of linear combination to be formally qualified.
- 7.4.2.2 focuses on how the use of scalar multiplication in linear combination facilitates the disclosure of entailed dependencies and demonstrates that although it is a useful tactic in the wider mathematical context, it does not serve the same purpose in the context of abstract cones.

The definition of α has a significant impact on the usefulness of linear combination. A cone in $\alpha(Cone^n)$ is represented by a set of inequalities, $\Upsilon_X \cup E^\alpha$. Each non-strict inequality in E^α is of the form $y \leq \Sigma Y$, that describes an inference rule for y with respect to the assignment of zero. By definition of α every such rule is explicit including those derived from transitive dependencies. Direct dependencies and those derived from transitive dependencies are considered to be non-redundant (the formal definition follows). Finally, in the two lemmas that follow these subsections, it is shown that:

- any non-redundant linear combination of the set of non-strict inequalities that represents a cone in $\alpha(Cone^n)$ will already be explicit in the representation,
- the representation of any cone in $\alpha(Cone^n)$ as prescribed by α, is unique up to ordering of the elements in $\Upsilon_X \cup E^\alpha$.

7.4.2.1 Linear Combination in $\alpha(Cone^n)$

Throughout the discussion that follows the index i associated with a variable, indicates its position in the ordered set of variables over which the non-strict inequalities are considered, and the index j indicates the jth element of an ordered set of m non-strict inequalities.

In computations involving linear inequalities information about a particular variable may be entailed in the conjunction of those linear inequalities rather than being explicit. Linear combination is a tactic that is used to elicit entailed information. The information that is sought is invariably the same as that required by any algorithm designed to solve, that is find solutions for, a system of linear inequalities, namely, anything that makes a contribution towards minimising the range of possible values for each variable. This means the primary goal is to assign a variable a single value and failing that to minimise its possible range of values. Now consider just how the motives for linear combination in the wider mathematical context differ from those in the context of $\alpha(Cone^n)$:

- assigning a value in $\mathbb{R}$ to a variable - in $\alpha(Cone^n)$ the only value ever assigned to a variable is zero.

- inferring a range of constants for a variable - not applicable in $\alpha(Cone^n)$, since variables are only assigned to zero, and all variables are greater than or equal to zero.
- inference of inter-variable dependencies - in $\alpha(Cone^n)$ the dependencies are strictly confined to those that facilitate variable assignment to zero.

There now follow formal definitions that allow a more precise expression of linear combination. In the following definition, the λ_i's are the coefficients for the non-negative constraints on the n variables and the λ_j's for the remaining linear inequalities. The non-negative constraints are treated separately because they play no real part in linear combination in this context as will become apparent as the discussion proceeds.

Definition 7.4.2 The *linear combination* of a set of m linear inequalities in $\alpha(Cone^n)$ with n variables constrained to non-negative values, where λ_i and λ_j are non-negative scalars, such that at least two of these scalars are greater than zero, is:

$$\Sigma_{i=1}^{n}\lambda_i.0 + \Sigma_{j=n+1}^{n+m}\lambda_j y_j \ \leq\ \Sigma_{i=1}^{n}\lambda_i y_i + \Sigma_{j=n+1}^{n+m}\lambda_j(\Sigma Y_j)$$ ■

In this discussion all but one of the examples are binary linear combinations; since addition is associative and commutative, binary combinations may be considered without loss of generality. Linear combinations are denoted in the following way: let l_i be a linear inequality and the symbol $+\!\!+$ denote any binary linear combination of inequalities. Then the linear combination, denoted $+\!\!+$, of k linear inequalities, $1 < k$, is itself a linear inequality l' where $l' = l_1 +\!\!+ \ldots +\!\!+ l_k$.

Definition 7.4.3 A linear combination, $\Sigma Z \leq \Sigma Z'$ of the $n+m$ non-strict linear inequalities $\Upsilon_X \cup E^\alpha \in \alpha(Cone^n)$, is considered *redundant* iff,

$$\exists(Z \cup Z').(\Upsilon_X \cup \{\Sigma Z \leq \Sigma Z'\}) \not\models \exists(Z \cup Z').(\Upsilon_X \cup E^\alpha)$$ ■

Examples of a redundant and a non-redundant linear combination follow.

Example 7.4.4 Consider $X = \{x_1, x_2, x_3, x_4\}$ and $C^\alpha = \Upsilon_X \cup \{x_1 \leq x_2, x_2 \leq x_3, x_1 \leq x_3\}$, and the linear combination:

$$\begin{array}{rcll}
0.0 & \leq & 0.x_1 & +\!\!+ \\
0.0 & \leq & 0.x_2 & +\!\!+ \\
0.0 & \leq & 0.x_3 & +\!\!+ \\
x_1 & \leq & x_2 & +\!\!+ \\
2x_2 & \leq & 2x_3 & +\!\!+ \\
0.x_1 & \leq & 0.x_3 & \\
\hline
x_1 + 2x_2 & \leq & 2x_3 &
\end{array}$$

Since, $\exists\{x_1, x_2, x_3\}\,.\,[\Upsilon_X \cup \{x_1 + 2x_2 \leq 2x_3\}] \not\models \exists\{x_1, x_2, x_3\}\,.\,[\Upsilon_X \cup \{x_1 \leq x_2, x_2 \leq x_3, x_1 \leq x_3\}]$, it follows that $\{x_1 + 2x_2 \leq 2x_3\}$ is *redundant.* ■

Example 7.4.5 Consider $X = \{x_1, x_2, x_3, x_4\}$ and $C^\alpha = \Upsilon_X \cup \{x_1 \leq x_2, x_2 \leq x_3, x_1 \leq x_3\}$, and the linear combination:

$$\begin{array}{rcll} 0.0 & \leq & 0.x_1 & \text{++} \\ 0.0 & \leq & 0.x_2 & \text{++} \\ 0.0 & \leq & 0.x_3 & \text{++} \\ 0.0 & \leq & 0.x_4 & \text{++} \\ x_1 & \leq & x_2 & \text{++} \\ x_2 & \leq & x_3 & \\ \hline x_1 & \leq & x_3 & \end{array}$$

Since, $\exists\{x_1, x_3\} . [\Upsilon_X \cup \{x_1 \leq x_3\}] \models \exists\{x_1, x_3\} . [\Upsilon_X \cup \{x_1 \leq x_2, x_2 \leq x_3, x_1 \leq x_3\}]$, it follows that $\{x_1 \leq x_3\}$ is *non-redundant.* ■

Definition 7.4.4 S_d is a *system of non-cyclic transitive dependencies* iff

$$S_d = \bigcup_{i=1}^{d}\{y_i \leq \Sigma Y_i \mid y_i \notin Y_i \wedge \forall i, d\ 2 \leq i \leq d . [\exists p\ 1 \leq p < i . [y_i \in Y_p]]\}$$ ■

The previous Definition 7.4.4 means that the linear combination of such a system will be of the form $y_1 \leq \Sigma Y'$ where $Y' \subset \bigcup_{i=1}^{d} Y_i$. In the context of $\alpha(\mathit{Cone}^n)$, when the variables in Y' are zero, y_1 is also zero, and therefore this expresses a variable dependency, or inference rule for y_1, and y_1 is said to be the *root* of the system. An example follows:

Example 7.4.6 Let $\{x, y, z, p, q, r, s\} \subseteq X$, and Υ_X hold,

$$\begin{array}{rcll} x & \leq & y + z & \text{++} \\ y & \leq & p + q & \text{++} \\ p & \leq & r & \text{++} \\ q & \leq & s & \\ \hline x & \leq & z + r + s & \end{array}$$

In Example 7.4.6 linear combination of the system is a variable dependency, $x \leq z+r+s$ for the root of the system, x. ■

Given the representation of an abstract cone the circumstances in which a linear combination of that representation is non-redundant must be considered. Linear combination with variable elimination allows the explicit representation of entailed constraints on variables, as illustrated in Example 7.4.5 above. If no variables are eliminated in the combination process then the linear combination is a relaxation of the explicit constraints, as illustrated in Example 7.4.4 above. This occurs because the inequalities in E^α are of the form $y_j \leq \Sigma Y_j$, and any linear combination of the form $\Sigma Y \leq \Sigma Y'$, where $y_j \in Y$ and $Y_j \subset Y'$ will be redundant, since it is a relaxation of $y_j \leq \Sigma Y_j$. Therefore a non-redundant linear combination must be of the form $y \leq \Sigma Y$. In the wider context, positive scalar multiplication aids variable elimination so the next step is to examine its effectiveness in the context of $\alpha(\mathit{Cone}^n)$.

1.4.2.2 Scalar multiplication in $\alpha(\mathit{Cone}^n)$

Consider how scalar multiplication facilitates variable elimination in the general mathematical context.

1. A variable can be eliminated if it occurs on the same side of each inequality with coefficients of the same cardinality, but different signs. An example follows:

   Example 7.4.7

$$\begin{array}{rcl} x & \leq & 3y + 2z \;{+}\!\!{+} \\ q & \leq & p - 2z \\ \hline x + q & \leq & 3y + p \end{array}$$

∎

2. A variable can be eliminated if it occurs on both sides of the resulting inequality with coefficients of the same cardinality and sign. This can only occur if there is at least one transitive dependency. See Example 7.4.6.

Generally then, positive scalar multipliers are applied in linear combination to equate the cardinality of like variables. However, since all variables in any system of linear inequalities have positive unit coefficients, further positive scalar multiplication is ineffective, as the following lemma demonstrates. It will be shown that variable elimination can only be effected when (i) there is a system of non-cyclic transitive dependencies and (ii) if at each step the current coefficient of the root is equated with that of the next linear inequality to be combined (a system of non-cyclic transitive dependencies is ordered by definition). Therefore, the only possible non-redundant linear combination can, can actually be obtained without using non-unit positive scalar multipliers.

Lemma 7.4.1 Let $C^\alpha \in \alpha(Cone^n)$ and $C^\alpha = \Upsilon_X \cup E^\alpha$. Further, let S_d be a non-cyclic system of transitive dependencies such that $S_d \subseteq E^\alpha$. Then the only non-redundant linear combination,

$$\Sigma_{i=1}^{n} \lambda_i.0 + \Sigma_{j=n+1}^{n+m} \lambda_j y_j \leq \Sigma_{i=1}^{n} \lambda_i y_i + \Sigma_{j=n+1}^{n+m} \lambda_j (\Sigma Y_j),$$

can be derived with all non-zero $\lambda_j = 1$. (Note that in all cases $\lambda_i = 0$, as otherwise the linear combination will be redundant.)

Proof A system of non-cyclic transitive dependencies is ordered, by definition. Since the linear combination is of an ordered system, the proof is by induction on the depth d of the system.
Let S_d be a system of non-cyclic transitive dependencies, then given $0 \leq \lambda_i$, $S_d = \{\lambda_1 y_1 \leq \lambda_1 \Sigma Y, \ldots, \lambda_d y_d \leq \lambda_d \Sigma Y_d\}$.

- Base step: Consider the linear combination of a system of non-cyclic transitive dependencies of depth $d = 2$.

$$\begin{array}{rcrcl} & & \lambda_1 y_1 & \leq & \lambda_1 Y_1 \;{+}\!\!{+} \\ & & \lambda_2 y_2 & \leq & \lambda_2 Y_2 \\ \hline \lambda_1 y_1 & + & \lambda_2 y_2 & \leq & \lambda_1 Y_1 + \lambda_2 Y_2 \end{array}$$

  By Definition 7.4.4 y_1 is the root and $y_2 \in Y_1$. Therefore the linear combination can be expressed as,

$$\lambda_1 y_1 + \lambda_2 y_2 \;\leq\; \lambda_1 (Y_1/\{y_2\}) + \lambda_1 y_2 + \lambda_2 Y_2$$

In order to facilitate the elimination of y_2 from both sides of the inequality λ_1 must equal λ_2.

- Induction step: Consider the linear combination of a system of non-cyclic transitive dependencies of depth $d > 2$. Let the linear combination of a system of non-cyclic transitive dependencies of depth $d-1$ be $R = \lambda_1 y_1 \;\leq\; \lambda_1(\Sigma Y_1') + \ldots + \lambda_{d-1}(\Sigma Y_{d-1}')$, where Y_i' is the residue of Y_i after variable elimination. By the induction hypothesis,

$$R = \lambda_1 y_1 \;\leq\; \lambda_1(\Sigma(Y_1' \cup \ldots \cup Y_{d-1}')).$$

Now consider the linear combination of R with the dth element in the system:

$$\begin{array}{rcl} \lambda_1 y_1 & \leq & \lambda_1(\Sigma(Y_1' \cup \ldots \cup Y_{d-1}')) +\!\!+ \\ \lambda_d y_d & \leq & \lambda_d Y_d \\ \hline \lambda_1 y_1 \;+\; \lambda_d y_d & \leq & \lambda_1(\Sigma(Y_1' \cup \ldots \cup Y_{d-1}')) + \lambda_d Y_d \end{array}$$

By definition, $y_d \in (Y_1' \cup \ldots \cup Y_{d-1}')$, therefore the the linear combination of R with the dth element in the system can also be expressed as:

$$\lambda_1 y_1 + \lambda_d y_d \;\leq\; \lambda_1(\Sigma(Y_1' \cup \ldots \cup Y_{d-1}')/\{y_d\}) + \lambda_1 y_d + \lambda_d Y_d$$

As before, in order to facilitate elimination, here of y_d, λ_1 must equal λ_d.

Therefore, by the principle of mathematical induction, in order to facilitate variable elimination in a linear combination of a system of non-cyclic transitive dependencies, the scalar multipliers employed at each combination step must be the same. Since variables throughout the system have unit coefficients to begin with, the only linear combination of a system of non-cyclic transitive dependencies that is non-redundant can be derived with all non-zero $\lambda_j = 1$. ■

It can now be shown that linear combination of the representation of a cone in $\alpha(\mathit{Cone}^n)$ cannot yield a non-redundant result that is not already explicit in its representation.

Lemma 7.4.2 Let $\Upsilon_X \cup E^\alpha$ represent a polyhedral cone, in $\alpha(\mathit{Cone}^n)$, then,

$$\forall \text{ non-redundant } y \leq \Sigma Y \,.\, [\Upsilon_X \cup E^\alpha \models y \leq \Sigma Y \;\leftrightarrow\; \exists y \leq \Sigma Y \in E^\alpha]$$

Proof The proof is in two parts:

1. $\Upsilon_X \cup E^\alpha \models y \leq \Sigma Y \;\leftarrow\; \exists y \leq \Sigma Y \in E^\alpha$ Elementary.
2. $\Upsilon_X \cup E^\alpha \models y \leq \Sigma Y \;\rightarrow\; \exists y \leq \Sigma Y \in E^\alpha$

   The proof is by contradiction. Let $\Upsilon_X \cup E^\alpha$ represent a cone in $\alpha(\mathit{Cone}^n)$, suppose that $\Upsilon_X \cup E^\alpha \models y \leq \Sigma Y$, where $y \leq \Sigma Y$ is non-redundant and assume that $\nexists y \leq \Sigma Y \in E^\alpha$. This assumption asserts that there exists a linear combination of elements in $\Upsilon_X \cup E^\alpha$ that is equal to $y \leq \Sigma Y$. Consider the possible linear combinations of the elements in $\Upsilon_X \cup E^\alpha$:

(a) In all cases a linear combination that includes a linear inequality that is a non-negative constraint will be redundant.

$$\begin{array}{rcl} \text{(i)} & & \\ 0 & \leq & y_1 \quad ++ \\ y_1 & \leq & \Sigma Y_1 \\ \hline 0 & \leq & \Sigma Y_1 \end{array} \qquad \begin{array}{rcl} \text{(ii)} & & \\ 0 & \leq & y_1 \quad ++ \\ y_2 & \leq & \Sigma Y_2 \\ \hline y_2 & \leq & y_1 + \Sigma Y_2 \end{array}$$

In case (i), $0 \leq \Sigma Y_1$ is redundant. In case (ii) $y_2 \leq y_1 + \Sigma Y_2$ is redundant.

(b) *Without variable elimination*

The non-strict inequalities are of the form $y \leq \Sigma Y$. Hence, any linear combination of these non-strict inequalities without variable elimination, will be of the form $\Sigma Y \leq \Sigma Y'$. This will relax the upper bound on each of the variables in Y, and therefore be redundant.

(c) *With variable elimination*

i. A variable can be eliminated if it occurs on the same side of each inequality with coefficients of the same cardinality, but different signs. This method is inapplicable, since in any linear combination,

$$\Sigma_{i=1}^{n}\lambda_i.0 + \Sigma_{j=n+1}^{n+m}\lambda_j y_j \;\leq\; \Sigma_{i=1}^{n}\lambda_i y_i + \Sigma_{j=n+1}^{m+n}\lambda_j(\Sigma Y_j)$$

there are no variables with negative coefficients on either side of the inequality.

ii. Variable elimination can occur when facilitated by a system of non-cyclic transitive dependencies, however, by Lemma 7.4.1, any linear combination of a system of non-cyclic transitive dependencies that is non-redundant can be derived with unit multipliers (λ_j).

Hence, the only non-redundant linear combinations are derived through transitive dependencies, and all transitive dependencies are explicit in E^α, by definition of α. That is, where $y \leq \Sigma Y$ is non-redundant, $\Upsilon_X \cup E^\alpha \models y \leq \Sigma Y \rightarrow \exists y \leq \Sigma Y \in E^\alpha$.

Hence,

$$\Upsilon_X \cup E^\alpha \models y \leq \Sigma Y \leftrightarrow \exists y \leq \Sigma Y \in E^\alpha \qquad \blacksquare$$

Corollary 7.4.1 Let $Y \subset X$, then, given the non-negative constraints on the variables in X, when $y_i \in X$ and $y_i \leq \Sigma Y$, it follows that $\forall Y' \supseteq Y \,.\, [y_i \leq \Sigma Y \models y_i \leq \Sigma Y']$. Therefore:

$$(\Upsilon_X \cup E^\alpha \models y \leq \Sigma Y') \leftrightarrow \exists (y \leq \Sigma Y) \in E^\alpha \,.\, [Y \subseteq Y'] \qquad \blacksquare$$

Corollary 7.4.2 Let $C_1^\alpha = \Upsilon_X \cup E_1^\alpha$ and $C_2^\alpha = \Upsilon_X \cup E_2^\alpha$ then

$$C_1^\alpha = C_2^\alpha \leftrightarrow \Upsilon_X \cup E_1^\alpha = \Upsilon_X \cup E_2^\alpha$$

Hence α prescribes a normal form for abstract cones that is unique up to the ordering of the elements in E^α. $\blacksquare$

Since the abstraction prescribes an orthogonal normal form for abstract cones in the form of a set of inference rules like those for representations of functions in Def_X, at an intuitive level, the very strong association between abstract cones and definite Boolean functions that is asserted at the end of this chapter becomes feasible.

7.4.3 Joins and Meets in $\alpha(Cone^n)$

Since $\alpha(Cone^n)$ is a subset of $Cone^n$ the join and meet operators in $Cone^n$, can be applied to cones in $\alpha(Cone^n)$. The join of two cones is their convex hull and the meet their intersection. However, although these operators can be applied to cones in $\alpha(Cone^n)$ it is possible for the outcome to be outside of $\alpha(Cone^n)$. An example follows:

Example 7.4.8 Consider C_1^α, $C_2^\alpha \in \alpha(Cone^n)$ where $X = \{x, y, z, w\}$, $C_1^\alpha = \Upsilon_X \cup \{x \leq y+z\}$ and $C_2^\alpha = \Upsilon_X \cup \{y \leq w,\, z \leq w\}$. $C_1^\alpha \cap C_2^\alpha = \Upsilon_X \cup \{x \leq y+z,\, y \leq w,\, z \leq w\}$, but $\alpha(C_1 \cap C_2) = \Upsilon_X \cup \{x \leq y+z,\, y \leq w,\, z \leq w,\, x \leq w\}$. ▪

This phenomenon occurs because in the wider context, although $x \leq 2w$ is entailed by this intersection, it is only when $w = 0$ that it is possible to infer that both y and z are 0 and hence, as x is non-negative, that $x = 0$ as well. The abstraction of $C_1^\alpha \cap C_2^\alpha$ which includes $x \leq w$, is a strict subset of $C_1^\alpha \cap C_2^\alpha$ as $x \leq w$ introduces another facet. A slightly different example in $\mathbb{R}^3$:

Example 7.4.9 Consider C_1^α, $C_2^\alpha \in \alpha(Cone^n)$ where $X = \{x, y, z\}$, $C_1^\alpha = \Upsilon_X \cup \{x \leq y+z\}$ and $C_2^\alpha = \Upsilon_X \cup \{y = z\}$. $C_1^\alpha \cap C_2^\alpha = \Upsilon_X \cup \{y = z, x \leq 2y\}$, but $\alpha(C_1^\alpha \cap C_2^\alpha) = \Upsilon_X \cup \{x \leq y,\, x \leq z,\, y \leq z,\, z \leq y\} = \Upsilon_X \cup \{x \leq y,\, x \leq z,\, y = z\} = \Upsilon_X \cup \{x \leq y,\, y = z\}$. ▪

Again, the abstraction of $C_1^\alpha \cap C_2^\alpha$ is a strict subset of $C_1^\alpha \cap C_2^\alpha$. ($\Upsilon_X \cup \{x \leq y,\, x \leq z,\, y = z\}$ is not a minimal representation of this 2 dimensional space, the planes $x = y$ and $x = z$, the boundary hyperplanes of the half spaces $x \leq y$ and $x \leq z$, repectively are orthogonal to one another and the half spaces each intersect the plane $y = z$ at the same place.) In both of the above examples the abstraction has a tightening effect on the shape of the cone as it explicitly introduces transitive dependencies with respect to the assignment of zero and replaces non-unit coefficients with unit coefficients. Therefore, the abstraction of intersection cannot be a relaxation of the original intersection it must be either the same or tighter than the original intersection. However, since all cones in $Cone^n$ can be abstracted to $\alpha(Cone^n)$, the application of α to the outcome of set intersection on cones means that $\alpha(Cone^n)$ is closed under this operation, allowing the definition of a meet operation for $\alpha(Cone^n)$:

Definition 7.4.5 Let $C_i^\alpha \in \alpha(Cone^n)$, then $C_1^\alpha \cap^\alpha C_2^\alpha = \alpha(C_1^\alpha \cap C_2^\alpha)$. ▪

Since set intersection is commutative it remains only to demonstrate $\cap^\alpha$, is associative. The proof is more easily assimilated with some insight into what the abstraction means spatially. Recall that when $C \in Cone^n$ then $\alpha(C)$ returns the non-negative constraints with a set of inequalities denoted E^α specified in this way:

$$\alpha(C) = \Upsilon_X \cup \{y \leq \Sigma Y \mid (C \wedge \bigcup_{y' \in Y} y' = 0) \models y = 0\}$$

such that each $y \leq \Sigma Y$ is the least element of that form for which the conditions specified are true. Consider how this might be viewed spatially. By the definition of $Cone^n$, every cone in $Cone^n$ is the intersection of a set of closed half spaces with boundary hyperplanes that pass through the origin. The specification for the inequality is couched in terms of equality, it is specifying that $y \leq \Sigma Y$ should be included in the set E^α only if the intersection of C with hyperplanes $\{y' = 0 \mid y' \in Y\}$ is a subset of the hyperplane $y = 0$. This intersection may be a boundary hyperplane of C or it may be embedded in the relative interior, $\mathrm{ri}\, C$, if so then this is when the abstraction is tighter than the intersection. In the proof that follows, note that C_i^α denotes a set of points and $\Upsilon_X \cup E_i^\alpha$ the set of linear inequalities that delineate C_i^α.

Lemma 7.4.3 Let $C_i^\alpha = \Upsilon_X \cup E_i^\alpha \in \alpha(Cone^n)$, then $(C_1^\alpha \cap^\alpha C_2^\alpha) \cap^\alpha C_3^\alpha = C_1^\alpha \cap^\alpha (C_2^\alpha \cap^\alpha C_3^\alpha)$.

Proof Now, $C_1^\alpha \cap^\alpha C_2^\alpha = \cap(\Upsilon_X \cup \{y \leq \Sigma Y \mid ((C_1^\alpha \cap C_2^\alpha) \wedge \bigcup_{y' \in Y} y' = 0) \models y = 0\})$. Let $\{y \leq \Sigma Y \mid ((C_1^\alpha \cap C_2^\alpha) \wedge \bigcup_{y' \in Y} y' = 0) \models y = 0\} \setminus (E_1^\alpha \cup E_2^\alpha) = E_{12}^\alpha$, and note that E_{12}^α may be empty. Similarly let $\{y \leq \Sigma Y \mid ((C_2^\alpha \cap C_3^\alpha) \wedge \bigwedge_{y' \in Y} y' = 0) \models y = 0\} \setminus (E_2^\alpha \cup E_3^\alpha) = E_{23}^\alpha$.

- Consider $((C_1^\alpha \cap^\alpha C_2^\alpha) \cap^\alpha C_3^\alpha)$:
  $C_1^\alpha \cap^\alpha C_2^\alpha = \cap(\Upsilon_X \cup E_1^\alpha \cup E_2^\alpha \cup E_{12}^\alpha)$, and therefore
  $(C_1^\alpha \cap^\alpha C_2^\alpha) \cap C_3^\alpha = \cap((\Upsilon_X \cup E_1^\alpha \cup E_2^\alpha \cup E_{12}^\alpha) \cup E_3^\alpha)$. Since set union (of inequalities) and set intersection (of sets of points) are both associative and commutative, it follows that:
  $(C_1^\alpha \cap^\alpha C_2^\alpha) \cap C_3^\alpha = \cap(\Upsilon_X \cup E_1^\alpha \cup E_2^\alpha \cup E_{12}^\alpha \cup E_3^\alpha)$.
- Similarly, $C_1^\alpha \cap (C_2^\alpha \cap^\alpha C_3^\alpha) = \cap(\Upsilon_X \cup E_1^\alpha \cup E_2^\alpha \cup E_3^\alpha \cup E_{23}^\alpha)$.

Consider: (1) $\alpha(\cap(\Upsilon_X \cup E_1^\alpha \cup E_2^\alpha \cup E_3^\alpha \cup E_{12}^\alpha))$ and (2) $\alpha(\cap(\Upsilon_X \cup E_1^\alpha \cup E_2^\alpha \cup E_3^\alpha \cup E_{23}^\alpha))$. In (1) α will return, amongst other inequalities E_{23}^α as both E_2^α and E_3^α are present. Similarly in (2), α will return, amongst other inequalities, E_{12}^α. Hence, by the definition of α, in both cases the sets of inequalities returned will be the same, up to the ordering of the elements, and hence the intersection of the inequalities will be the same.
Therefore: $\alpha((C_1^\alpha \cap^\alpha C_2^\alpha) \cap C_3^\alpha) = \alpha(C_1^\alpha \cap (C_2^\alpha \cap^\alpha C_3^\alpha))$. Hence, $\cap^\alpha$ is associative. ■

There is a similar case with the join operator. The closure of the convex hull of two or more cones in $\alpha(Cone^n)$ will be a polyhedral cone, but it may not be in $\alpha(Cone^n)$, as demonstrated in the following Example 7.4.10. However, an abstraction of this cone preserves the properties associated with assignment of zero and returns a cone in $\alpha(Cone^n)$.

Example 7.4.10 Let $C_1^\alpha, C_2^\alpha \in \alpha(Cone^n)$, where $X = \{x_1, x_2, x_3, x_4\}$, $C_1^\alpha = \Upsilon_X \cup \{x_1 \leq x_3, x_2 \leq x_4\}$ and $C_2^\alpha = \Upsilon_X \cup \{x_1 \leq x_4, x_2 \leq x_3\}$. Hence,

$$C_1^\alpha \mathbin{\overline{\cup}} C_2^\alpha = \Upsilon_X \cup \{x_1 + x_2 \leq x_3 + x_4\},$$

but $\Upsilon_X \cup \{x_1 + x_2 \leq x_3 + x_4\} \notin \alpha(Cone^n)$. Now, $\alpha(C_1^\alpha \mathbin{\overline{\cup}} C_2^\alpha) = \Upsilon_X \cup \{x_1 \leq x_3 + x_4, x_2 \leq x_3 + x_4\}$, and although $C_1^\alpha \mathbin{\overline{\cup}} C_2^\alpha \subset \alpha(C_1^\alpha \mathbin{\overline{\cup}} C_2^\alpha)$, if $x_3 = 0$ and $x_4 = 0$ then in both cases it can be deduced that both $x_1 = 0$ and $x_2 = 0$, and abstraction preserves the propagation of variable assignment to zero. ■

Employing the same tactic as with the meet, the application of α to the output of the $\overline{\cup}$ operator (closure of the convex hull) on cones means that $\alpha(Cone^n)$ is closed under this operation, allowing the definition of a join operation for $\alpha(Cone^n)$. The definition of both the join and meet in this more general domain means that $\alpha(Cone^n)$ is a lattice in its own right, but since neither $C_1^\alpha \cap C_2^\alpha$ nor $C_1^\alpha \overline{\cup} C_2^\alpha$ is always in $\alpha(Cone^n)$, it is not a sublattice of $Cone^n$.

Definition 7.4.6 $\overline{\cup}^{\alpha}: \alpha(\mathit{Cone}^n) \times \alpha(\mathit{Cone}^n) \to \alpha(\mathit{Cone}^n)$ is defined:

$$C_1^{\alpha} \,\overline{\cup}^{\alpha}\, C_2^{\alpha} = \alpha(C_1^{\alpha} \,\overline{\cup}\, C_2^{\alpha})$$

where C_1^{α}, $C_2^{\alpha} \in \alpha(\mathit{Cone}^n)$, and $\overline{\cup}$ is the closure of the convex hull, the join for Cone^n. ▪

In order to demonstrate that, the abstraction of the join for cones is a join for abstract cones, it is necessary to take a closer look at the computation method of the convex hull for cones and the effect abstraction has on its output. $C_i \in \mathit{Cone}^n$ is represented by $\Upsilon_X \cup E_i$. For the purpose of computing the convex hull of two cones in Cone^n, consider the inequalities in $\Upsilon_X \cup E_i$ in a matrix format so that $\Upsilon_X \cup E_i = A_i\bar{X}_i \leq \bar{0}$, where A_i is an m by n matrix of coefficients in $\{-1, 0, 1\}$ and X_i is an n-vector of variables. Now, $C_1 \,\overline{\cup}\, C_2 = C_1 + C_2 = \{\bar{x_1} + \bar{x_2} \mid \bar{x_i} \in C_i\}$, by Definition 4.1.17. Let $C_i = A_i\bar{X}_i \leq \bar{0}$, then $C_1 + C_2 = \{\bar{X} \mid \bar{X} = \bar{X}_1 + \bar{X}_2,\ A_1\bar{X}_1 \leq \bar{0},\ A_2\bar{X}_2 \leq \bar{0}\}$.
The next step is to qualify the circumstances when the convex hull of two abstract cones is not an abstract cone. The following definition is required.

Definition 7.4.7 Let $C_1^{\alpha} = \Upsilon_X \cup E_1^{\alpha}$, $C_2^{\alpha} = \Upsilon_X \cup E_2^{\alpha} \in \alpha(\mathit{Cone}^n)$. If $\{y \leq \Sigma Y,\ z \leq \Sigma Z\} \subseteq E_1^{\alpha}$ and $\{y \leq \Sigma Z,\ z \leq \Sigma Y\} \subseteq E_2^{\alpha}$ then y and z are a pair of *interchangeable* variables. ▪

In Example 7.4.10, x_1 and x_2 are *interchangeable.* If there are pairs of interchangeable variables in the inequalities in the operands then the convex hull of the two abstract cones may not be an abstract cone. In the discussion that follows if, for example, $\{x \leq y + z\}$ then (the value of) x is said to be bound from above by the sum of the variables in the set $\{y,\ z\}$. Consider the computation required for the following Example 7.4.11:

Example 7.4.11 Let $C_i^{\alpha} \in \alpha(\mathit{Cone}^n)$ and $X = \{x,\ y,\ z,\ w\}$ where $C_1^{\alpha} = \Upsilon_X \cup \{x \leq y,\ y \leq w\}$, and $C_2^{\alpha} = \Upsilon_X \cup \{x \leq z\}$, then the matrix computation reduces to this:

$$C_1^{\alpha} + C_2^{\alpha} = \left\{ \langle x,y,z,w\rangle \;\middle|\; \begin{array}{ll} x = x_1 + x_2, & x_1 \leq y_1, \\ & x_2 \leq z_2, \\ y = y_1 + y_2, & y_1 \leq w_1, \\ z = z_1 + z_2, & \\ w = w_1 + w_2, & \\ \text{where } \Upsilon_X,\ \Upsilon_{X1},\ \Upsilon_{X2} & \end{array} \right\}$$

The solution to the above system of inequations is:

$$C_1^{\alpha} + C_2^{\alpha} = \Upsilon_X \cup \{\langle x,y,z,w\rangle \mid x \leq y + z\}$$

Inspection of the example computation above indicates, informally, that:

- if a variable y is bound from above by the sum of a set of variables Y_1 in one cone and by the sum of a set of variables Y_2, in the other cone, then in their convex hull y will be bound from above by the sum of the union $Y_1 \cup Y_2$,
- if a variable y is unbound from above in either or both of the two cones, then in their convex hull y will be unbound from above.

The lemma that follows confirms these observations.

Lemma 7.4.4 Let $C_1^\alpha \bar{\cup} C_2^\alpha = \Upsilon_X \cup E_h$, $\alpha(\Upsilon_X \cup E_h) = \Upsilon_X \cup E_h^\alpha$, and $C_i^\alpha = \Upsilon_{X\,i} \cup E_i^\alpha$, then:

$$y \leq \Sigma Y \in E_h^\alpha \;\leftrightarrow\; \exists y \leq \Sigma Y_1 \in E_1^\alpha \,.\, [\exists y \leq \Sigma Y_2 \in E_2^\alpha \,.\, [Y = Y_1 \cup Y_2]]$$

Proof $C_1^\alpha \bar{\cup} C_2^\alpha = \{\bar{X} \mid \bar{X} = \bar{X}_1 + \bar{X}_2,\ \Upsilon_{X1} \cup E_1^\alpha,\ \Upsilon_{X2} \cup E_2^\alpha\}$. Let $y \in X$, $y_1 \in X_1$ and $y_2 \in X_2$ such that $y = y_1 + y_2$. It is sufficient to consider the pairwise combinations of constraints on any $y \in \bar{X}$. The proof is in two parts:

Part 1. $y \leq \Sigma Y \in E_h^\alpha \;\rightarrow\; \exists y \leq \Sigma Y_1 \in E_1^\alpha \,.\, [\exists y \leq \Sigma Y_2 \in E_2^\alpha \,.\, [Y = Y_1 \cup Y_2]]$

The convex hull of two polyhedral cones reduces to their sum as defined above, and each variable $y = y_1 + y_2$. Suppose that either y_1 or y_2 are not bound from above. Then y is equal to the sum of two variables, one of which has no upper bound and therefore y will have no upper bound in the convex hull. Hence, if y is bound from above in the convex hull then both of y_1 and y_2 must be bound from above, that is: $y \leq \Sigma Y \in E_h^\alpha \;\rightarrow\; \exists y \leq \Sigma Y_1 \in E_1^\alpha \,.\, [\exists y \leq \Sigma Y_2 \in E_2^\alpha \,.\, [Y = Y_1 \cup Y_2]]$

Part 2. $y \leq \Sigma Y_i \in E_h^\alpha \;\leftarrow\; \exists y \leq \Sigma Y_1 \in E_1^\alpha \,.\, [\exists y \leq \Sigma Y_2 \in E_2^\alpha \,.\, [Y_i = Y_1 \cup Y_2]]$

If y is bound from above in both C_1^α and C_2^α then y is bound from above in $C_1^\alpha \bar{\cup}^\alpha C_2^\alpha$. Since $y \leq \Sigma Y_1 \in E_1^\alpha$ and $y \leq \Sigma Y_2 \in E_2^\alpha$, $y \leq \Sigma Y_1 + \Sigma Y_2$, as $y = y_1 + y_2$. There are two cases to consider, where y is not one of a pair of interchangeable variables and where it is.

- Case 2.1. y is not interchangeable with another variable.

  Let $k \in [1, \ldots, n]$ such that $k \neq i$. Now consider $y \leq \Sigma Y_1 + \Sigma Y_2$. Whenever $z_1 \in Y_1 \wedge z_2 \in Y_2$, their sum can be replaced by z, as $z = z_1 + z_2$. Whenever $z_1 \in Y_1 \wedge z_2 \notin Y_2$, or vice versa, z_1 or z_2 can be replaced by z in the sum on the right hand side of the inequality, as $z = z_1 + z_2$.

  Therefore, projecting out $\bar{X}_1$ and $\bar{X}_2$, if $y \leq \Sigma Y_1 \in E_1^\alpha$ and $y \leq \Sigma Y_2 \in E_2^\alpha$ then $y \leq \Sigma(Y_1 \cup Y_2) \in E_h^\alpha$. By the definition of α, if $y \leq \Sigma(Y_1 \cup Y_2) \in E_h$ then $y \leq \Sigma(Y_1 \cup Y_2) \in E_h^\alpha$.

- Case 2.2 Let y and z, be interchangeable variables, then $y \leq \Sigma Y_1$, $z \leq \Sigma Z_1 \in E_1^\alpha$ and $y \leq \Sigma Y_2$, $z \leq \Sigma Z_2 \in E_2^\alpha$, where $Y_1 = Z_2$ and $Z_1 = Y_2$.

  There are three cases to consider, in the first when $Y_1 = Z_1$, the convex hull is the same as its abstraction, otherwise it is not.

  1. $Y_1 = Z_1$
     $Y_1 = Y_2 = Z_1 = Z_2$ as $Y_1 = Z_2$ and $Z_1 = Y_2$, by definition of interchangeable. Hence, as $y = y_1 + y_2$ it follows that $y \leq \Sigma Y_1$, $z \leq \Sigma Z_1 \in E_h$ where $Y_1 = Z_1$. Put $Y = Y_1 = Z_1$, then by definition of α, $y \leq \Sigma Y$, $z \leq \Sigma Y \in E_h^\alpha$.
  2. $Y_1 \neq Z_1$ and $Y_1 \cap Z_1 \neq \phi$
     If there is a pair of interchangeable variables, y and z but $Y_1 \neq Z_1$, then in $C_1^\alpha \bar{\cup} C_2^\alpha$, the variables y and z will be constrained as follows: $y + z \leq \Sigma\bar{\lambda}(Y_1 \cup Z_1)$, $y_b \leq \Sigma(Y_1 \cup Z_1)\, y_c \leq \Sigma(Y_1 \cup Z_1)$ where, in an abuse of notation, $\bar{\lambda}$ is a vector of variable coefficients in $\{1, 2\}$, and the set intersection of variables is viewed as a vector of variables. A variable coefficient in $\bar{\lambda}$ will be

2 only when the variable is in both Y_1 and Z_1. By Definition 7.4.1 of α, the abstraction will relax these constraints to $y \leq \Sigma(Y_1 \cup Z_1)$, $z \leq \Sigma(Y_1 \cup Z_1)$

3. $Y_1 \cap Z_1 = \phi$:
   If there is a pair of interchangeable variables, y and z such that $Y_1 \cap Z_1 = \phi$, then in $C_1^\alpha \overline{\cup} C_2^\alpha$, the variables y and z will be constrained as follows: $y + z \leq \Sigma(Y_1 \cup Z_1)$.
   However, by Definition 7.4.1 of α, the abstraction will relax this constraint to $y \leq \Sigma(Y_1 \cup Z_1)$, $z \leq \Sigma(Y_1 \cup Z_1)$

Hence, in all cases: $y \leq \Sigma Y \in E_h^\alpha \;\leftarrow\; \exists y \leq \Sigma Y_1 \in E_1^\alpha \,.\, [\exists y \leq \Sigma Y_2 \in E_2^\alpha \,.\, [Y = Y_1 \cup Y_2]]$

Therefore $y \leq \Sigma Y \in E_h^\alpha \;\leftrightarrow\; \exists y \leq \Sigma Y_1 \in E_1^\alpha \,.\, [\exists y \leq \Sigma Y_2 \in E_2^\alpha \,.\, [Y = Y_1 \cup Y_2]]$ ■

Lemma 7.4.5 $\overline{\cup}^\alpha$ is associative, that is, $\forall C_1^\alpha, C_2^\alpha, C_3^\alpha \in \alpha(\mathit{Cone}^n)$,

$$C_1^\alpha \overline{\cup}^\alpha (C_2^\alpha \overline{\cup}^\alpha C_3^\alpha) = (C_1^\alpha \overline{\cup}^\alpha C_2^\alpha) \overline{\cup}^\alpha C_3^\alpha$$

Proof Let $C_i^\alpha = \Upsilon_X \cup E_i^\alpha$. By Lemma 7.4.4, for each convex hull operation there are two cases to consider, let $y \in X$, when (1) y is bound from above in both operands and (2) when y is bound from above in only one or neither operand.

1. Let $C_2^\alpha \overline{\cup}^\alpha C_3^\alpha = C_{h1}^\alpha$ where $C_{h1}^\alpha = \Upsilon_X \cup E_{h1}^\alpha$ and $C_1^\alpha \overline{\cup}^\alpha C_{h1}^\alpha = C_{h2}^\alpha$ where $C_{h2}^\alpha = \Upsilon_X \cup E_{h2}^\alpha$.
   Consider $C_2^\alpha \overline{\cup}^\alpha C_3^\alpha$. $\forall y \in X \,.\, [y \leq \Sigma Y_2 \in E_2^\alpha \wedge y \leq \Sigma Y_3 \in E_3^\alpha \;\leftrightarrow\; y \leq \Sigma(Y_2 \cup Y_3) \in E_{h1}^\alpha]$, by Lemma 7.4.4.
   Now consider $C_1^\alpha \overline{\cup}^\alpha C_{h1}^\alpha$. Similarly, $\forall y \in X \,.\, [y \leq \Sigma Y_1 \in E_1^\alpha \wedge y \leq \Sigma Y_{h1} \in E_{h1}^\alpha \;\leftrightarrow\; y \leq \Sigma(Y_1 \cup (Y_2 \cup Y_3) \in E_{h2}^\alpha)]$, by Lemma 7.4.4.
   Hence, $\forall y \in X \,.\, [y \leq \Sigma Y_{h2} \in E_{h2}^\alpha \;\leftrightarrow\; \exists y \leq \Sigma Y_1 \in E_1^\alpha \,.\, \exists y \leq \Sigma Y_2 \in E_2^\alpha \,.\, \exists y \leq \Sigma Y_3 \in E_3^\alpha \,.\, [Y_{h2} = (Y_1 \cup Y_2 \cup Y_3)]$.

2. Let $C_1^\alpha \overline{\cup}^\alpha C_2^\alpha = C_{h3}^\alpha$ where $C_{h3}^\alpha = \Upsilon_X \cup E_{h3}^\alpha$ and $C_{h3}^\alpha \overline{\cup}^\alpha C_3^\alpha = C_{h4}^\alpha$ where $C_{h4}^\alpha = \Upsilon_X \cup E_{h4}^\alpha$.
   Consider $C_1^\alpha \overline{\cup}^\alpha C_2^\alpha$. $\forall y_i \in X \,.\, [y \leq \Sigma Y_{i1} \in E_1^\alpha \wedge y_i \leq \Sigma Y_{i2} \in E_2^\alpha \;\leftrightarrow\; y \leq \Sigma(Y_1 \cup Y_2) \in E_{h3}^\alpha]$, by Lemma 7.4.4.
   Now consider $C_{h3}^\alpha \overline{\cup}^\alpha C_3^\alpha$. Similarly, $\forall y \in X \,.\, [y \leq \Sigma Y_{h3} \in E_{h3}^\alpha \wedge y \leq \Sigma Y_3 \in E_3^\alpha \;\leftrightarrow\; y \leq \Sigma(Y_{h4} \cup (Y_{h3} \cup Y_3) \in E_{h4}^\alpha)]$, by Lemma 7.4.4.
   Hence, $\forall y \in X \,.\, [y \leq \Sigma Y_{h4} \in E_{h4}^\alpha \;\leftrightarrow\; \exists y \leq \Sigma Y_1 \in E_1^\alpha \,.\, \exists y \leq \Sigma Y_2 \in E_2^\alpha \,.\, \exists y \leq \Sigma Y_3 \in E_3^\alpha \,.\, [Y_{h4} = (Y_1 \cup Y_2 \cup Y_3)]$.

Hence, y is bound from above in $C_1^\alpha \overline{\cup}^\alpha (C_2^\alpha \overline{\cup}^\alpha C_3^\alpha)$ iff it is bound from above by $Y_{h2} = (Y_1 \cup Y_2 \cup Y_3)$ and y_i is bound from above in $(C_1^\alpha \overline{\cup}^\alpha C_2^\alpha) \overline{\cup}^\alpha C_3^\alpha$ iff it is bound from above by $Y_{h4} = (Y_1 \cup Y_2 \cup Y_3)$. Since in all cases $Y_{h2} = Y_{h4}$, it follows that

$$C_1^\alpha \overline{\cup}^\alpha (C_2^\alpha \overline{\cup}^\alpha C_3^\alpha) = (C_1^\alpha \overline{\cup}^\alpha C_2^\alpha) \overline{\cup}^\alpha C_3^\alpha$$

■

Proposition 7.4.1 $\langle \alpha(\mathit{Cone}^n), \subseteq, \perp, \top, \overline{\cup}^\alpha, \cap^\alpha \rangle$ is a complete lattice.

Proof $\overline{\cup}^{\alpha}$ and $\cap^{\alpha}$ are associative by Lemmas 7.4.5 and 7.4.3 respectively. Therefore $\overline{\cup}^{\alpha}$ and $\cap^{\alpha}$ are the join and meet, respectively for $\alpha(Cone^n)$ and $\alpha(Cone^n)$ is a lattice. Since $\alpha(Cone^n)$ is finite, it is a complete lattice, [Sza63], with top element: the non-negative orthant in n-dimensional space, and bottom element: the origin. ■

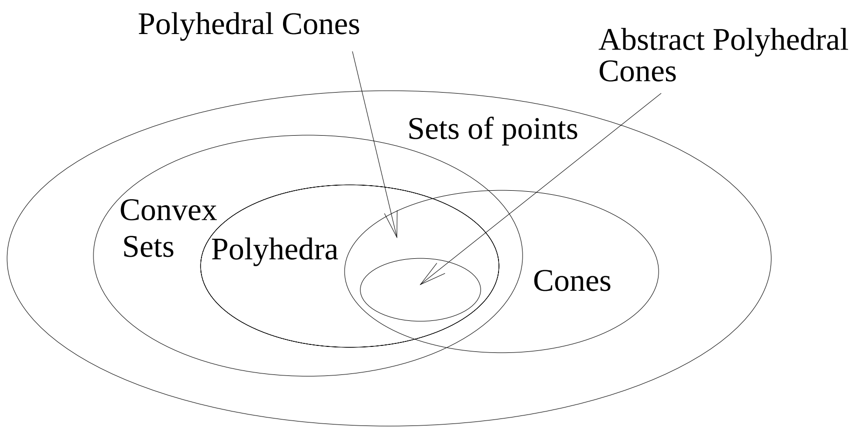


Figure 17: Sets of points in $\mathbb{R}^n$

Figure 17 demonstrates how the various sets of points in $\mathbb{R}^n$ fit together. There are cones that are neither polyhedral nor convex, but all abstract cones are polyhedral.

7.5 Is $\alpha(Cone^n)$ really an abstraction?

Every cone in $Cone^n$ can be mapped to an abstract cone in $\alpha(Cone^n)$ and the mapping of an infinite domain to a finite one might suggest the notion of equivalence classes. However, $\alpha(Cone^n)$ is not a sublattice of $Cone^n$ and it is this characteristic of the relation between the two domains that prevents the expression of the relation in terms of equivalence classes. Equivalence classes are usually defined in terms of some common property held by the elements of each class, but in this case equivalence would be couched in terms of an approximation that is not precise enough, to preserve the integrity of the meet operation, the abstraction of set intersection. Note that in the example that follows the abstraction of set intersection is coincident with set intersection. This is demonstrated in Example 7.5.1.

Example 7.5.1 Let $X = \{x, y\}$ where $\{0 \leq x, y \leq 0\}$. Consider the notion of equivalence defined with respect to the propagation of the assignment of zero and the two classes:

Class A	$[\ldots, \{3x \leq 2y\}, \ldots, \{x \leq y\} \ldots]_{\approx}$	if $y = 0$ then $x = 0$
Class B	$[\ldots, \{3y \leq 2x\}, \ldots, \{y \leq x\} \ldots]_{\approx}$	if $x = 0$ then $y = 0$
and the abstract intersections:		$\{3x \leq 2y\} \cap^{\alpha} \{3y \leq 2x\} = \{x = 0, y = 0\}$
		$\{x \leq y\} \cap^{\alpha} \{y \leq x\} = \{x = y\}$
Class C	$[\ldots, \{x = 0, y = 0\}, \ldots]_{\approx}$	$x = 0$ and $y = 0$
Class D	$[\ldots, \{x = y\}, \ldots]_{\approx}$	$x = 0$ iff $y = 0$

The conjunction of $\{3x \leq 2y\} \in$ Class A with $\{3y \leq 2x\} \in$ Class B, is in Class C, but the conjunction of $\{x \leq y\} \in$ Class A with $\{y \leq x\} \in$ Class B is in Class D. ▪

The failure of the equivalence class model is interesting in itself. The relation as prescribed above qualifies as an equivalence relation, as it is both reflexive and transitive, every element in the domain falls into a single class and the intersection of any two classes is empty. The problem lies in the preservation of ordering between classes and this occurs because this relation is an approximation that puts aside the notion of scalars (as coefficients or as constants) which have a significant impact on the ordering of polyhedral cones. In this case the notion of equivalence is not strong enough to ensure that the order of the original elements in the domain is maintained over the classes.

The abstraction of an object is a mapping from the object to a domain that can represent a certain property(s) of that object and this is precisely the function of the abstraction operator, α. Therefore $\alpha(Cone^n)$ is a valid abstraction of $Cone^n$. To recap, inference in $\alpha(Cone^n)$ is driven by these aims:

- to infer that as many variables as possible are zero
- to infer dependency relationships, in the form $y \leq \Sigma Y$, for as many variables as possible in order to facilitate their assignment to zero.

Since no further non-redundant linear inequality can be derived from linear combination of the prescribed representations, it follows that each representation is a complete set of inference rules that are entailed by the delineation of the abstract cones. Further, in $\alpha(Cone^n)$, the only inference rule is of the form $y \leq \Sigma Y$, and the prescribed representation of a cone in $\alpha(Cone^n)$ as the union of non-negative variable constraints with a set of inference rules is unique.

7.6 A Lattice Isomorphism

Abstract cones are uniquely defined as sets of points and functions in Def_X are uniquely defined by their sets of models. In both cases set intersection is the meet operator but in order that both domains are closed under their respective join operators, a generalisation of set union is required. The significant characteristic of the abstract cones that indicates the necessity for generalisation is their convexity. This characteristic disallows a straightforward join that comprises the union of those points that are in either or both operands, as the resulting set must be convex. There are instances when the union of two particular polyhedra cannot be represented in terms of a single polyhedra, that is, the union can only be represented as two distinct, albeit possibly abutting, polyhedra. Hence, the union cannot always be represented as a single polyhedra without generalisation and this is precisely what the closure of the convex hull is, a generalisation that is the smallest convex space that contains all the points in both operands. Similarly, there are instances when the join of two functions in Def_X that in terms of models is the union of the sets of models for each function, is such that the union of models represents a function that is not itself in Def_X, since its models are not closed under intersection. The generalisation required is analogous to that required in $\alpha(Cone^n)$.
Representations in $\alpha(Cone^n)$ are a set of linear inequalities that are inference rules. Each linear inequality prescribes the means of inferring that the variable on the left hand side of the inequality is zero. There may be more than one such rule for each

variable, or there may be none for a particular variable. This is consistent with DMBF representation in Def_X. DMBF is such that where $f \in Def_X$ and $f = \bigwedge F$, each implication in F is a inference rule for the variable that occurs in the head. Every possible rule is explicit, but no rule is entailed by any other, and if nothing is known about a particular variable it will not occur in the head of any implication. Inference and propagation follow one from the other and Dart [Dar91] considers that such implications can be thought of, alternatively as propagation rules. However these rules are viewed, the inequalities in the representation of elements in the abstraction of $Cone^n$ serve the same purpose as the implications in DMBF representations of functions in Def_X.

7.6.1 From Polyhedral Cones to Boolean Functions

Since both abstract polyhedral cones and Boolean functions have normal form representations (by Definition 7.3.2 and Corollary 7.4.2) the mapping from abstract polyhedral cones to Boolean functions is described in those terms.

Definition 7.6.1 The mapping $\varphi'_X : \alpha(Cone^n) \to Bool_X$, is defined:

$$\varphi'_X\left(\Upsilon_X \cup \bigcup_{j=1}^{m}\{y_j \leq \Sigma Y_j\}\right) = \begin{cases} true & \text{when } m = 0 \\ \bigwedge_{j=1}^{m}(y_j \leftarrow \bigwedge Y_j) & \text{otherwise} \end{cases}$$

■

Although φ' is syntactic it maps from one normal form to another and therefore, operates at the semantic level as well. Further, since each element in the domain is uniquely identified by the normal form and each element in the co-domain is uniquely identified by the normal form the mapping is well-defined. It will be shown that φ'_X maps a cone $C^\alpha \in \alpha(Cone^n)$ to a set of implications, definite clauses, the conjunction of which represents a function in Def_X.

Example 7.6.1 Let $X = \{x_1, x_2, x_3\}$,

$$\varphi'_X(\Upsilon_X \cup \{x_1 \leq x_2,\ x_3 \leq 0\}) \ = \ (x_1 \leftarrow x_2) \wedge (x_3 \leftarrow true)$$

■

Note that for any variable $y \in X$, when $y \leq 0 \in E^\alpha$ this is equivalent to $y \leq \Sigma Y$, where $Y = \phi$ and φ'_X maps $y \leq \Sigma\phi$ to the implication $y \leftarrow \bigwedge \phi$ which is equivalent to $y \leftarrow true$ [Dar91]. Since all variables are constrained to non-negative values whenever $y \leq 0$, it can be inferred that $y = 0$, giving a mapping from $y = 0$ to $y \leftarrow true$ as expected.

The implications in $\varphi'_X(\alpha(Cone^n))$ represent Boolean functions and it will be shown that the representations of cones in $\alpha(Cone^n)$, as prescribed by α, are such that the definite clauses in the image conform to DMBF and therefore the Boolean function so represented is in Def_X.

Proposition 7.6.1 $\varphi'_X(\alpha(Cone^n)) = Def_X$.

Proof The proof is in two parts:

- $\varphi'_X(\alpha(Cone^n)) \subseteq Def_X$

  A cone in $\alpha(Cone^n)$ can be represented by $\Upsilon_X \cup E^\alpha$. By definition of α, E^α is in orthogonal form, that is, transitive dependencies are explicit and $(y \leq \Sigma Y) \in$

$E^\alpha \rightarrow (\forall Y' \subset Y \,.\, [(y \leq \Sigma Y') \notin E^\alpha]$, by Lemma 7.4.2. Therefore, by definition of φ'_X, F is also in orthogonal form and $(y \leftarrow \bigwedge Y) \in F \rightarrow \forall Y' \subset Y \,.\, [(y \leftarrow \bigwedge Y') \notin F]$. Since these are precisely the conditions that describe DMBF, by Definition 7.3.2, it follows that $\varphi'_X(C^\alpha) \subseteq \mathit{Def}_X$.

- $\mathit{Def}_X \subseteq \varphi'_X(\alpha(\mathit{Cone}^n))$

  Similarly, a formula $f \in \mathit{Def}_X$ can be represented in DMBF and $f = \bigwedge F$. By definition of DMBF, F is in orthogonal form and $(y \leftarrow \bigwedge Y) \in F \;\rightarrow\; (\forall Y' \subset Y \,.\, [\bigwedge F \not\models (y \leftarrow \bigwedge Y')]$. Since the elements of F are definite clauses it follows that $(y \leftarrow \bigwedge Y) \in F \;\rightarrow\; \forall Y' \subset Y \,.\, [(y \leftarrow \bigwedge Y') \notin F]$. By definition of α and φ'_X these conditions also apply to every element of $\varphi'_X(C^\alpha)$. Therefore, $\mathit{Def}_X \subseteq \varphi_X(\alpha(\mathit{Cone}^n))$.

Hence, $\varphi'_X(\alpha(\mathit{Cone}^n)) = \mathit{Def}_X$ as $\varphi'_X(\alpha(\mathit{Cone}^n)) \subseteq \mathit{Def}_X$ and $\mathit{Def}_X \subseteq \varphi'_X(\alpha(\mathit{Cone}^n))$. ■

Proposition 7.6.1 allows φ_X to be defined:

Definition 7.6.2 $\varphi_X\colon \alpha(\mathit{Cone}^n) \rightarrow \mathit{Def}_X$.

$$\varphi_X\,(\Upsilon_X \,\cup\, E^\alpha) = \varphi'_X\,(\Upsilon_X \,\cup\, E^\alpha)$$

■

Proposition 7.6.2 φ_X is bijective.

Proof Let $\varphi_X(C_1^\alpha) = \bigwedge F_1$, $\varphi_X(C_2^\alpha) = \bigwedge F_2$, where $C_1^\alpha = \Upsilon_X \cup E_1^\alpha$, $C_2^\alpha = \Upsilon_X \cup E_2^\alpha \in \alpha(\mathit{Cone}^n)$. Let $\varphi_X(C_1^\alpha) = \varphi_X(C_2^\alpha)$, then $\bigwedge F_1 = \bigwedge F_2$, and $F_1 = F_2$, by definition of DMBF. Therefore, $E_1^\alpha = E_2^\alpha$, by definition of φ_X. Hence:

$$\varphi_X(C_1^\alpha) = \varphi_X(C_2^\alpha) \;\rightarrow\; C_1^\alpha = C_2^\alpha.$$

and φ_X is injective. Since φ_X is injective and by Definition 7.6.2 $\varphi_X(\alpha(\mathit{Cone}^n)) = \mathit{Def}_X$ and is therefore surjective, it follows that φ_X is bijective. ■

Note that in the proposition that follows it is necessary to distinguish between a set of inequalities and the set of points it represents, so where $C_i^\alpha = \Upsilon_X \,\cup\, E_i^\alpha$, in the usual way, C_i^α is a set of points and $\Upsilon_X \,\cup\, E_i^\alpha$, is the set of implicitly conjoined inequalities that represent C_i^α. Further the relation $\prec$ over Boolean functions defines a strict ordering such that when $f_i \in \mathit{Def}_X$ if $f_1 \prec f_2$ then $f_1 \models f_2$ but $f_1 \neq f_2$.

Proposition 7.6.3 φ_X is a lattice isomorphism.

Proof Let $C_i^\alpha \in \alpha(\mathit{Cone}^n)$, then $C_1^\alpha \subset C_2^\alpha \;\leftrightarrow\; \varphi_X(C_1^\alpha) \prec \varphi_X(C_2^\alpha)$.
Let $C_1^\alpha = \Upsilon_X \,\cup\, E_1^\alpha$, $C_2^\alpha = \Upsilon_X \,\cup\, E_2^\alpha$, $\varphi_X(C_1^\alpha) = \bigwedge F_1$ and $\varphi_X(C_2^\alpha) = \bigwedge F_2$. The proof is in two parts:

- $C_1^\alpha \subset C_2^\alpha \;\rightarrow\; \varphi_X(C_1^\alpha) \prec \varphi_X(C_2^\alpha)$

  Let $C_1^\alpha \subset C_2^\alpha$. There are two cases, where $E_2^\alpha = \phi$ and where $E_2^\alpha \neq \phi$.

  Case 1. If $E_2^\alpha = \phi$, then $\varphi_X(C_2^\alpha) = \mathit{true}$ and $\varphi_X(C_1^\alpha) \prec \varphi_X(C_2^\alpha)$.

  Case 2. If $E_2^\alpha \neq \phi$, then by Corollary 7.4.1:

$\forall(y \leq \Sigma Y) \in E_2^\alpha . [\exists(y \leq \Sigma Y') \in E_1^\alpha . [Y' \subseteq Y]]$ and
$\exists(y \leq \Sigma Y') \in E_1^\alpha . [\forall(y \leq \Sigma Y) \in E_2^\alpha . [Y' \subset Y]]$

therefore, by definition of φ_X :

$\forall(y \leftarrow \bigwedge Y) \in F_2 . [\exists(y \leftarrow \bigwedge Y') \in F_1 . [Y' \subseteq Y]]$ and thus, $\varphi_X(C_1^\alpha) \prec \varphi_X(C_2^\alpha)$.

$\exists(y \leftarrow \bigwedge Y') \in F_1 . [\forall(y \leftarrow \bigwedge Y) \in F_2 . [Y' \subset Y]]$ and thus, $\varphi_X(C_2^\alpha) \not\prec \varphi_X(C_1^\alpha)$.

Hence, $\varphi_X(C_1^\alpha) \prec \varphi_X(C_2^\alpha)$ and $C_1^\alpha \subset C_2^\alpha \rightarrow \varphi_X(C_1^\alpha) \prec \varphi_X(C_2^\alpha)$.

- $C_1^\alpha \subset C_2^\alpha \leftarrow \varphi_X(C_1^\alpha) \prec \varphi_X(C_2^\alpha)$

  Let $\varphi_X(C_1^\alpha) \prec \varphi_X(C_2^\alpha)$. There are two cases, where $F_2 = \phi$ and where $F_2 \neq \phi$.

  Case 1. If $F_2 = \phi$, then $C_2^\alpha = \Upsilon_X$ and $C_1^\alpha \subset C_2^\alpha$.

  Case 2. If $F_2 \neq \phi$, then by the definition of DMBF:

  $\forall(y \leftarrow \bigwedge Y) \in F_2 . [\exists(y \leftarrow \bigwedge Y') \in F_1 . [Y' \subseteq Y]]$ and
  $\exists(y \leftarrow \bigwedge Y') \in F_1 . [\forall(y \leftarrow \bigwedge Y) \in F_2 . [Y' \subset Y]]$

  therefore by definition of φ_X :

  $\forall(y \leq \Sigma Y) \in E_2^\alpha . [\exists(y \leq \Sigma Y') \in E_1^\alpha . [Y' \subseteq Y]]$ and thus $C_1^\alpha \subset C_2^\alpha$.
  $\exists(y \leq \Sigma Y') \in E_1^\alpha . [\forall(y \leq \Sigma Y) \in E_2^\alpha . [Y' \subset Y]]$ and thus $C_2^\alpha \not\subset C_1^\alpha$.

  Hence $C_1^\alpha \subset C_2^\alpha$ and $C_1^\alpha \subset C_2^\alpha \leftarrow \varphi_X(C_1^\alpha) \prec \varphi_X(C_2^\alpha)$.

Hence,

$$C_1^\alpha \subset C_2^\alpha \leftrightarrow \varphi_X(C_1^\alpha) \prec \varphi_X(C_2^\alpha)$$

Since φ_X is bijective, by Proposition 7.6.2 and $C_1^\alpha \subset C_2^\alpha \leftrightarrow \varphi_X(C_1^\alpha) \prec \varphi_X(C_2^\alpha)$ then by [Prop.5.11][DP90], φ_X is a lattice isomorphism. ■

7.7 Summary

Def$_X$ allows a representation in the form of a set of implications, effectively a complete set of inference rules for each variable, namely DMBF. The abstraction of *Cone*^{n} prescribes a representation that is unique and in the form of a complete set of inference rules entailed by the delineation of the cone. It is not surprising then, that there is an isomorphism between these abstract cones and Boolean functions in *Def*$_X$.

The reason this phenomenon occurs is not due to a property of bijective maps on complete lattices *per se* as a bijective map between lattices is not a sufficient condition for a lattice isomorphism. The proof of the lattice (order) isomorphism is based directly on the definition of φ which maps the normal form representations of abstract cones to normal form representations of Boolean functions in *Def*$_X$.

Inspection of φ shows that whilst φ is syntactic in its nature, it is also semantic because the representation is a normal form. The crucial characteristic of the normal forms lies in their expression as a set of inference rules for the variables over which the elements of both domain and co-domain are described. The inference rules are induced by the meaning of the ordering that is imposed on the elements of both domains and it is for this reason and this reason alone that φ is a lattice isomorphism.

Consider Figure 16 that depicts the Hasse diagrams for the lattices of Def_X and $\alpha(Cone^n)$ over two variables. It is the interpretation of the symbols that denote the elements of the lattice that induces the ordering on the elements and in consequence the meaning of the least upper and greatest lower bounds. The normal forms for both lattices are sets of inference rules for each variable. What is the common characteristic of the two domains that makes it possible to express an element of either domain as a complete set of inference rules for each variable? The answer to this question lies with the fact that Def_X is a domain where variables are assumed *false* or *true* until proven to be *true* and its counterpart, $\alpha(Cone^n)$, where variables are assumed to be non-negative until proven to be zero. Hence, not only are these domains very simple, but the motivation for computation in either domain can be matched exactly in its counterpart. It is, therefore, very straightforward to take each element of these domains and reduce it to a set of inference rules in the manner prescribed by DMBF and α.

The conjectured generalisation is then, that if there is a bijective map between two ordered sets from one normal form to another and the normal forms constitute a complete set of inference rules then there will always be an order isomorphism. In consequence, if the ordered sets are lattices then the map is a lattice isomorphism. It is a moot point as to whether the generic class to which it applies will ever have more than one element and whether as such it actually constitutes a class.

The discovery of abstract cones was driven by a combination of the observation concerning the propagation of zero assignment and the need to find a useful abstract domain for representing dependency information. It should be noted that although abstract cones are formally couched in terms of an orthogonal form with explicit transitive dependencies, in practise the representations do not need the orthogonal form. For example, the representation of the abstract cone $\{x \leq y, y \leq z\}$ in any computation does not require the inclusion of the transitive dependency of x on z. Similarly the non-negative constraint on a variable is only required if the relative boundary of the half-space that represents such a constraint includes a facet of the abstract cone, otherwise the non-negativity of that variable will be implicit in the representation.

It has already been noted, in Chapter 3, that the choice of abstract domain is crucial to the success of an analysis implemented as an abstract interpretation. This relationship between a class of polyhedral cone and a class of Boolean function suggests that there may be unexpected relationships between other apparently disparate subdomains of well-understood mathematical domains. The advantage of such associations is that the behaviour of the elements of these domains and the machinery for the manipulation of their elements is well understood.

Chapter 8

Groundness Analysis with SICStus clp(QR)

8.1 Introduction

Groundness analysis is one of the most useful analyses, in its own right and as a facilitator of other types of analysis. The aim of groundness analysis is to answer the question: At a certain program point is the variable x always bound to a term that contains no variables? Groundness analysis in its simplest form infers whether a variable is definitely ground or its groundness status is unknown – this is typically extended to deducing groundness dependencies between variables. The dependencies are useful for information propagation between predicates during the analysis but also as an expression of definite dependency between variables. Optimising compilers can exploit information of this kind to speed up operations, for example, unification. Initially the lattices used to represent this information ranged from the simplest linear domain representing variable status as either ground or unknown, to more complex structures capable of representing a variable bound to a compound structure with non-ground arguments [Mel81]. A systematic approach to lattice construction was adopted in [lCMH90] with status elements ground, unbound and unknown.

Much work has been done in this area and some well understood domains for capturing such properties have been established that encompass relational information as well. In [Gal95] as a part of an analysis toolkit a groundness analysis is presented that employs a pre-interpretation domain that associates either the status g - ground or ng - non-ground with program variables. This analysis infers the minimal Herbrand model with respect to the groundness status of program variables and in this way groundness dependencies can be deduced as well as instances of variables that are definitely ground. In another approach to inferring relational information, the domain *Prop* of propositional formula was introduced by Marriott and Søndergaard in [MS90] as a suitable abstract domain for groundness information in the form of dependencies rather than simply assigning variables some particular status.

However, currently Boolean functions are probably the best known medium for encoding dependency information. At the implementation level the representation of Boolean functions has become a crucial issue with regard to program analyses, as it has in the other areas of application for Boolean functions. In the analysis of large programs there may be hundreds of variables which has a considerable impact on the size of the representation and tractability of the required operations. Directed acyclic graphs known variously as Binary Decision Diagrams, Reduced Order Binary Decision Diagrams have been devised for representing Boolean functions.

The representation of dependency information with Boolean functions is refined in [AMSS98][1] with the introduction, in particular, of the positive Boolean functions, *Pos*, and a subclass of *Pos*, the definite Boolean functions, *Def*. They present bottom-up groundness analyses demonstrating that *Pos* expresses greater accuracy. They note and illustrate the condensing/additive property of *Pos* that allows *Pos* a greater precision than *Def* in bottom-up analyses even when the dependency relation can be expressed in *Def*. An illustrative example follows in Section 8.2. Several implementation techniques are investigated: variations on conjunctive and disjunctive normal forms for both *Pos* and *Def* are tested along with ROBDD's on a benchmark suite that included programs like `bryant`, a prolog implementation of BDD's and `analyze`, the analyser itself. The overall outcome was that whilst *Pos* based analyses were the most precise, the Dual Blake Canonical Form representation of *Def* was the fastest. They note however, that a top-down analysis employing *Def* would fare better, with regard to precision, in relation to *Pos* based analyses, since this different approach would compensate for *Def*'s lack of the condensing property.

The work presented by Codish and Demoen, [CD95] employs a representation of the domain *Prop* where programs are transformed to include dependency information encoded in `iff/n` predicates that are added to the original program. They employ a magic-set transform and then a bottom-up analysis. The transform allows call and answer patterns to be taken into account and analysis results are commensurate with those for *Pos* but overall analysis times are slow, for example, by comparison to the work of [AMSS98].

Under certain circumstances both *Pos* and *Def*-based analyses may require iterations that are exponential in the number of symbols in a program. Codish in [Cod99] has shown that for certain predicates, analyses using BDD's for *Pos*-based abstractions can generate unmanageably large representations. The point of the paper is to highlight that there is a problem with *Pos* under some circumstances and that the solution is widening. However, the nature of a suitable widening is not explored. More recently [GHC01], similar circumstances have been identified for *Def*-based analyses that also generate unmanageably large representations. In [HAZCK00] a very simple technique, that imposes a threshold on the size of representations is introduced to keep their size manageable. Experimental results draw comparisons with *Pos*-based analyses and these confirm the existence of classes of programs for which a *Pos*-based analysis will not terminate. So although *Pos*-based analyses are considered more precise than those based on *Def*, it is interesting to note that *Def* can cope with these cases and further, in many cases where *Pos* is faster than *Def*, the difference is negligible, less than 0.1sec.

Howe and King [HK01] present a definitive, methodical approach to the implementation of groundness analysis in Prolog. They first investigate the frequency of operations and then proceed to an implementation that favours representations that allow fast execution of the those operations that occur most frequently. The analyser is a goal-dependent bottom-up implementation based on a refinement of the induced magic set transform that avoids the recomputation of literals inherent in the original transform. The outcome is a hybrid analyser that performs a precise analysis commensurate in

[1]This work has been available as a technical report since 1994.

efficiency terms with BDD based analyses. This work adds fuel to the overriding message of Getzinger's implementation of abstract domains and analysis of performance [Get94], which was that simple analysis domains are the most effective, and that there is a threshold beyond which increasing precision is invariably bought at too high a cost to be of practical use.

Boolean functions are clearly the front runner for capturing dependency information. Both *Pos* and *Def* afford a useful degree of precision, but the recent work of Codish, King and Howe has brought their levels of precision and efficiency more closely in line with one another. What distinguishes these two domains now? In certain circumstances, which may not arise very often, *Pos* is more precise, but this increase in precision brings with it the possibility of an eventually-terminating analysis, as shown by Codish in [Cod99]. If it is difficult to widen *Pos* simply to ensure termination in a reasonable time and or to pre-empt analysis for predicates that "break" *Pos* then *Def* may well be the more viable option for dependency analyses.

Recall that positive Boolean functions are those that are *true* when all their variables are *true*. Definite Boolean functions are the subset of the positive Boolean functions that each has a set of models which is closed under intersection. *Pos* is closed under classical disjunction whereas *Def* is not, therefore the join for *Def* is sometimes an upper approximation of disjunction. This means that under certain circumstances *Def* will lose or approximate dependency relationships that *Pos* would be able to express. A simple example illustrates the point. Since *Def* is a subclass of *Pos*, every function in *Def* is also in *Pos*. Let $f_i \in Def$ where $f_1 = x$, $f_2 = y$. The join in *Pos* is classical disjunction denoted $\vee$, so that $f_1 \vee f_2$ is simply $x \vee y$. However the join in *Def* is the least Boolean function in *Def* entailed by both f_1 and f_2, that is, $\wedge f : (f_1 \models f \wedge f_2 \models f)$. The join in *Def* is denoted $\dot{\vee}$ and $f_1 \dot{\vee} f_2 = true$, as in this case *true* is the only function in *Def* (over just two variables) that entails both of the Boolean functions $f_1 = x$ and $f_2 = y$. It is here that the association between the join for abstract polyhedral cones and for functions in *Def* becomes clear. The polyhedral join for two polyhedra, P_1 and P_2, (abstract cones or otherwise) is simply the smallest convex set in $Poly^n$ that is a superset of P_1 and P_2, that is, $\cap P : (P_1 \subseteq P \wedge P_2 \subseteq P)$. The convexity constraint on all polyhedra disallows the disjoint sets that may be the straightforward union of any collection of convex sets, in the same way that the constraint that sets of models be closed under intersection - the formal constraint that distinguishes functions in *Pos* that are also in *Def* from those that are not, disallows classical disjunction. In both cases, intuitively, it is the notion of "either...or" that is disallowed.

From this discussion it can be surmised that although *Pos* affords, what in practice amounts to, slightly better precision than *Def*, i) *Def* based analyses can handle certain classes of program that are troublesome for those based on *Pos*, ii) precision in *Def* often matches that in *Pos*, iii) the disparity in precision, where it exists, is minimal. Therefore, *Def* based analyses are a viable option. Operations carried out in CLP domains may not always be cheap, but in an environment with constraint support they are available *on tap* and bearing in mind that the rate of increases in the efficiency of chip and processing technology are forecasted as continuing for at least another 15 years, then a *Def* based analysis that terminates in a reasonable, rather than the fastest, time and with useful precision may well be acceptable. Hence, the prototypes undertaken here. The aim of the investigation that follows, was i) to confirm the theory, that an analysis based on abstract cones could capture the same dependency information as *Def*, and ii) to see if the speed of the analyses based on abstract cones was acceptable. There are two implementations of groundness analysis, the first[2] used the same analyser as that of the argument size analysis of Chapter 5. The second analysis used a hybrid analyser due to Howe and King [HK01], but due to technical problems[3] and time constraints this work is incomplete.

The rest of this chapter is as follows. The next Section 8.2 provides an example analysis. Section 8.3 illustrates the condensing property of *Pos* but also demonstrates how the magic transform allows a commensurate analysis with *Def*. Sections 8.4, 8.5 and 8.6 cover the semantics, operations over abstract cones and the implementations respectively. Section 8.8 summarises.

8.2 Worked Example

The example below is based on the same iteration scheme of the argument size analysis in Chapter 5. The initial iterate $I_0 = \phi$ and the iteration sequence proceeds as described below converging when $I_2 = I_3$ (without the need for widening).

$\alpha(Cone^n)$			Def_X		
I_1	$=$	$\{\langle r,s,t\rangle \mid r=0,\ s=t,\ 0\leq s\}$	I_1	$=$	$r \wedge (s \leftrightarrow t)$
I_2	$=$	$\{\langle r,s,t\rangle \mid r=0,\ s=t,\ 0\leq s\}\ \overline{\cup}^{\alpha}$	I_2	$=$	$r \wedge (s \leftrightarrow t)\ \dot{\vee}\ t \leftrightarrow (r \wedge s)$
		$\{\langle r,s,t\rangle \mid t=r+s,\ 0\leq s\}$		$=$	$t \leftrightarrow (r \wedge s)$
	$=$	$\{\langle r,s,t\rangle \mid t=r+s,\ 0\leq s\}$			
I_3	$=$	$\{\langle r,s,t\rangle \mid r=0,\ s=t,\ 0\leq s\}\ \overline{\cup}^{\alpha}$	I_3	$=$	$r \wedge (s \leftrightarrow t)\ \dot{\vee}\ t \leftrightarrow (r \wedge s)$
		$\{\langle r,s,t\rangle \mid t=r+s,\ 0\leq s\}$		$=$	$t \leftrightarrow (r \wedge s)$
	$=$	$\{\langle r,s,t\rangle \mid t=r+s,\ 0\leq s\}$			

Figure 18: Groundness dependency analysis for `append(R,S,T)`

8.3 *Def* vs. *Pos* - the Condensing Property

Abstract interpretations for groundness analysis have employed both Pos_X and Def_X, but Pos_X is distributive whilst Def_X is not, and this lack means that analyses in Def_X may not be so precise as their Pos_X counterparts. The Example 8.3.1 that follows is due to [AMSS98], the analysis output is from the implementation described in subsection 8.6.1.

Example 8.3.1 Consider the predicate $q/2$:

[2]This implementation was curtailed prematurely as the author was obliged to intermit for three months for personal reasons. When work resumed various considerations dictated that the isomorphism proof, on which the implementation was based, be placed at the top of the agenda.

[3]A projection problem in SICStus prevented implementation beyond the point detailed in this chapter.

Before unfolding `r`.	After unfolding `r`.

`q(X, Y) :- p(X, Y), r(X,Y).`
`p(X, X).`
`r(a, Y).`
`r(X, a).`
	`q(a, Y) :- p(a, Y).`
	`q(X, a) :- p(X, a).`
	`p(X, X).`

A Def_X based analysis for $q/2$ yields $x \leftrightarrow y$, whereas a Pos_X based analysis yields $x \wedge y$. However, if $r/2$ is unfolded, then a Def_X analysis is able to match the precision of Pos_X and will return $x \wedge y$. ▪

This problem can be overcome by using a magic transform, due to [CD95], on the program, that avoids the necessity of unfolding during program abstraction, as shown in Figure 19, where the addition of `_c` and of `_a` to a predicate name indicates a call and an answer respectively. The outcome of a Def_X analysis for the magic'd version of $q/2$

Magic'd $q/2$	Analysis of $q/2$ in $\alpha(Cone^n)$	Interpretation in Def_X
`q_c(_, _).`	`q_c(x, y) :-` $\{0 \leq x, 0 \leq y\}$.	*true*

`p_c(X, X) :-`
`   q_c(X, X).`
	`p_c(x, y) :-` $\{0 \leq x, 0 \leq y\}$.	*true*

`r_c(X, Y) :-`
`   q_c(X, Y),`
`   p_a(X, Y).`
	`r_c(x, y) :-` $\{x = y\}$.	$x \leftrightarrow y$

`q_a(X, Y) :-`
`   q_c(X, Y),`
`   p_a(X, Y),`
`   r_a(X, Y).`
	`q_a(x, y) :-` $\{x = 0, y = 0\}$.	$\underline{x \wedge y}$

`p_a(X, X) :-`
`   p_c(X, X).`
	`p_a(x, y) :-` $\{x = y\}$.	$x \leftrightarrow y$

`r_a(a, Y) :-`
`   r_c(a, Y).`
	`r_a(x, y) :-` $\{x = 0, y = 0\}$.	$x \wedge y$

`r_a(X, a) :-`
`   r_c(X, a).`		

Figure 19: The magic version of $q/2$ and its analysis in Def_X.

using abstract cones to represent functions in Def_X is on the left hand side of Figure 19.

8.4 Semantics

The concrete domain and abstract domains in this analysis are the same as those for the argument size analysis presented in Chapter 5. However, whereas there is a total mapping between Lin^n and $Poly^n$, there is a partial map between Lin^n and $\alpha(Cone^n)$, since abstract polyhedral cones, denoted $\alpha(Cone^n)$, are a finite subset of $Poly^n$.

Hence, the Galois connection between these two domains is also the same as for the argument size analysis, as are the fixpoint semantics over both domains. The abstraction, however, is different, but couched in similar terms.

8.4.1 Program Abstraction

Program abstraction for groundness analysis is dealt with in a similar way to that for argument size analysis. As defined in Section 5.3.3, *Eqn* denotes the set of Herbrand equations and Eqn_{Lin} denotes the set of linear equations. Recall the mapping *sol*, (repeated for convenience) required to ensure that constraints over like terms are mapped to constraints over like variables in the numerical domain.

Definition 8.4.1 Recall the mapping $sol : Eqn \rightarrow \mathcal{P}(Eqn)$ defined by: $sol(E) = \{E' \mid E \rightsquigarrow^* E'\}$ where * is transitive closure and the relation, $Eqn \rightsquigarrow Eqn$ is the least binary relation defined by:

- $\{x = t\} \cup E \rightsquigarrow \{x = s\} \cup E \quad$ if $t = s \wedge s \notin X$
- $\{x = t\} \cup E \rightsquigarrow \{x = y\} \cup E \quad$ if $t = y \wedge y \in X$

■

In order to derive abstract programs from concrete programs a mapping from Herbrand equations to linear equations is required, so $\mathfrak{a}_{gr}$ is defined with respect to the groundness status of the terms: This mapping is fundamental to program abstraction and includes the insertion of the non-negative constraint on all variables, since the analysis domain is confined to the non-negative orthant of $\mathbb{R}^n$.

Definition 8.4.2 Let c be an Herbrand equation, then $\mathfrak{a}_{gr} : Eqn \rightarrow Eqn_{Lin}$ is defined by:

$$\mathfrak{a}_{gr}(c) = \begin{cases} \{x = 0\} & \text{if } c = \{x = t\} \text{ and } t \text{ is ground} \\ \{0 \leq x\} & \text{if } c = \{x = t\} \text{ and } t \text{ is a variable} \\ \{0 \leq x, x \leq \Sigma var(t)\} & \text{if } c = \{x = t\} \text{ and } t \text{ is non-ground} \\ \{\mathfrak{a}_{gr}(sol(c_1)), \ldots, \mathfrak{a}_{gr}(sol(c_n))\} & \text{if } c = \{c_1, \ldots, c_n\} \end{cases}$$

■

Note that in the first case above, when t is a ground term $\mathfrak{a}_{gr}$ returns $\{0 \leq x, x \leq 0\}$ which is equivalent to $\{x = 0\}$.

Definition 8.4.3 The abstraction of a program P, to its abstract counterpart, $P^{\mathcal{A}}$, is defined:

$$P^{\mathcal{A}} = \left\{ p(\bar{x}') \leftarrow \mathfrak{a}_{gr}(E), p_1(\bar{x}'_1), \ldots p_n(\bar{x}'_n) \;\middle|\; \begin{array}{l} w \in P, \\ w = p(\bar{t}) \rightarrow p_1(\bar{t}_1), \ldots, p_n(\bar{t}_n), \\ E \in sol(\bar{x}'_i = \bar{t}_i), \\ \cup_{i=0}^{n} \bar{x}'_i \cap var(w) = \phi \end{array} \right\}$$

■

Example 8.4.1 Consider the program `append/3`:

```
w1  append([], Y, Y).
w2  append([X|Xs], Y, [X|Z]) :-
      append(Xs, Y, Z).
```

and the application of both *sol* and α_{gr} to each clause as prescribed by the abstraction:

w_1			
		$sol(\{x_{0i} = t_{0i}\})$	$\alpha_{gr}(sol(\{x_{0i} = t_{0i}\}))$
$\{x_{01} = t_{01},$	$t_{01} = []\}$	$\rightsquigarrow \{x_{01} = []\}$	$\{x_{01} = 0\}$
$\{x_{02} = t_{02},$	$t_{02} = y\}$	$\rightsquigarrow \{x_{02} = y\}$	$\{x_{02} = y\}$
$\{x_{03} = t_{03},$	$t_{03} = y\}$	$\rightsquigarrow \{x_{03} = y\}$	$\{x_{03} = y\}$
w_2			
		$sol(\{x_{0i} = t_{0i}\})$	$\alpha_{gr}(sol(\{x_{0i} = t_{0i}\}))$
$\{x_{01} = t_{01},$	$t_{01} = [x\|xs]\}$	$\rightsquigarrow \{x_{01} = [x\|xs]\}$	$\{x_{01} \leq \Sigma var([x\|xs])\}$
			$\{x_{01} \leq x + xs\}$
$\{x_{02} = t_{02},$	$t_{02} = y\}$	$\rightsquigarrow \{x_{02} = y\}$	$\{x_{02} = y\}$
$\{x_{03} = t_{03},$	$t_{03} = [y\|z]\}$	$\rightsquigarrow \{x_{03} = [y\|z]\}$	$\{x_{03} \leq \Sigma var([y\|z])\}$
			$\{x_{03} \leq y + z\}$
		$sol(\{x_{1i} = t_{1i}\})$	$\alpha_{gr}(sol(\{x_{1i} = t_{1i}\}))$
$\{x_{11} = t_{11},$	$t_{11} = xs\}$	$\rightsquigarrow \{x_{11} = xs\}$	$\{x_{11} = xs\}$
$\{x_{12} = t_{12},$	$t_{12} = y\}$	$\rightsquigarrow \{x_{12} = y\}$	$\{x_{12} = y\}$
$\{x_{13} = t_{13},$	$t_{13} = z\}$	$\rightsquigarrow \{x_{13} = z\}$	$\{x_{13} = z\}$

Hence, an abstract version of `append/3` is derived:

	equivalent to:	
w_1 append$^{\mathcal{A}}$(X$_{01}$, X$_{02}$, X$_{03}$). X$_{01}$ = 0, X$_{02}$ = Y, X$_{03}$ = Y. w_2 append$^{\mathcal{A}}$(X$_{01}$, X$_{02}$, X$_{03}$) :- X$_{01}$ ≤ X + Xs, X$_{02}$ = Y, X$_{03}$ ≤ X + Z, X$_{11}$ = Xs, X$_{12}$ = Y, X$_{13}$ = Z, append$^{\mathcal{A}}$(X$_{11}$, X$_{12}$, X$_{13}$).		w_1 append$^{\mathcal{A}}$(0, Y, Y). w_2 append$^{\mathcal{A}}$(XXs, Y, XZ) :- XXs ≤ X + Xs, XZ ≤ X + Z, append$^{\mathcal{A}}$(Xs, Y, Z).

■

8.5 Operations over Abstract Cones

Recall from Chapter 4, that polyhedral cones always include the origin, therefore the convex hull of an arbitrary number of polyhedral cones is always closed. The convex hull computation over polyhedral cones is very straightforward as it simply reduces to

addition. This means that as no λ multipliers are needed the operation is linear and can be expressed in matrix terms without recourse to a relaxation technique. That is, where C_1, $C_2 \in Cone^n$ such that $C_1 = A_1\bar{x_1} \leq B_1$ and $C_2 = A_2\bar{x_2} \leq B_2$ then the closure of their convex hull , $C_1 \bar{\cup} C_2 = \{\bar{x} \mid \bar{x} = \bar{x_1} + \bar{x_2}, A_1\bar{x_1} \leq B_1, A_2\bar{x_2} \leq B_2\}$. The operation is further simplified as the elements of A_i will be in $\{-1, 0, 1\}$ as will the elements of the vectors B_i. The process of abstraction amounts to replacing inequalities of the form $\Sigma Y \leq \Sigma Y'$, where both Y and Y' are sets of variables in X, with the set of inequalities $\{y \leq Y' \mid y \in Y\}$.
Should scalar multipliers not in $\{-1, 1\}$ arise in any operation they are simply discarded when the operation output is coded in whatever form the representation takes.
Abstract intersection is not so straightforward to deal with and no simple way has been devised to detect when abstract intersection is not the same as intersection. This does not seem to occur very often and in the second implementation it is addressed by incorporating abstraction into the entailment test that always follows the meet operation. For example, if the test is: Does $C_1^\alpha \models C_2^\alpha$? this means: Are the inference rules with respect to the assignment of zero in C_2^α entailed by those in C_1^α? An example follows that illustrates the process.

Example 8.5.1 Consider an intermediate entailment test in the process of the second implementation for the analysis of `append/3`, where $C_1^\alpha = \Upsilon_X \cup \{a \leq d + f,\ d \leq a,\ f \leq a,\ c \leq d+e,\ d \leq c,\ e \leq c,\ b \leq e, e \leq f+b,\ f \leq e\}$ and $C_2^\alpha = \Upsilon_X \cup \{a \leq c,\ b \leq c\}$. To test $C_1^\alpha \models C_2^\alpha$ requires affirmation that $(C_1^\alpha \cap c = 0) \models a = 0$ and $(C_1^\alpha \cap c = 0) \models b = 0$. ▪

This is not a satisfactory solution and needs rigorous attention in any further study.

8.6 Implementation

The implementations that have been carried out have been in the form of prototypes. The first used the same analyser as the argument size analysis presented in Chapter 5, with preliminary transforms for program abstraction, normalisation (as in the left hand column of the abstract version of `append/3`) and the calculation of SCCs. The second used a hybrid analyser that induces the necessary magic transform as the analysis proceeds. The analyses also differed in the implementation of the abstraction as detailed in the following Sections 8.6.1 and 8.6.2.

8.6.1 Analyser I

The analyser is the same bottom-up analyser with ground representation. Apart from the different abstraction this implementation differs from that of argument size analysis in the following ways:

- abstraction and normalisation were automated
- the normalised programs were passed through a magic transform and then abstracted
- the SCCs were derived and incorporated into the iteration strategy

The abstraction theory has been covered in Section 8.4.2 and the normalisation and magic transform are standard. The SCCs were derived in a single pass using an edge template that kept track of those predicates that had already been encountered.

The implemented abstraction Various factors encouraged the use of an implementation shortcut with regard to the abstraction. Empirical evidence during the argument size analysis suggested that large numbers of inequalities slow the solver down even if their solution does not require excessive computation. In this initial stage the widening was still in place as it wasn't until investigations into the nature of the abstract domain had been completed that it was realised that the widening was unnecessary. It was clear that excessive use of the solver would hold the analysis up. Two way implication of the form $x \leftrightarrow y$ is modelled as $\{x \leq y, \, y \leq x\}$ which is equivalent to $x = y$. It was apparent that modelling $x \leftrightarrow y \wedge z$ as $x = y + z$ instead of $\{x \leq y + z, \, y \leq x, \, z \leq x\}$ would substantially reduce solver activity. Such a shortcut would clearly preserve the propagation facility required of the prescribed abstraction, but could it ever be too precise, that is, could an analysis ever infer a variable's groundness or dependency that might not always, or never be true?
One approach to answering that question is to consider the circumstances when it may happen and then consider whether the identified circumstance can happen in the context of the analysis of a logic program. Consider the following example:

Example 8.6.1 Let $X = \{x, \, y, \, z\}$ and $C_1^\alpha = \{x \leq y + z, \, y \leq x, \, z \leq x\}$, $C_2^\alpha = \{z \leq y + x, \, y \leq z, \, x \leq z\}$, $C_3^\alpha = \{x = y + z\}$, $C_4^\alpha = \{z = y + x\}$, and Υ_X. C_1^α and C_3^α both represent the same propagation facility with respect to zero, in that in both cases if x were assigned zero then it can be inferred that both y and z are zero; and if both y and z are zero then it can be inferred that x is zero. Similarly with C_2^α and C_4^α, if z were assigned zero then it can be inferred that both y and x are zero; and if both y and x are zero then it can be inferred that z is zero. Now, $C_1^\alpha \cap C_2^\alpha = \{y \leq x, \, x = z\}$, but $C_3^\alpha \cap C_4^\alpha = \{y = 0, x = z\}$. $\varphi(C_1^\alpha) = \varphi(C_3^\alpha) = x \leftrightarrow (y \wedge z)$ and $\varphi(C_2^\alpha) = \varphi(C_4^\alpha) = z \leftrightarrow (y \wedge x)$, but $\varphi(C_1^\alpha \cap C_2^\alpha) = (y \leftarrow x) \wedge (x \leftrightarrow z)$ and $\varphi(C_3^\alpha \cap C_4^\alpha) = y \wedge (x \leftrightarrow z)$. Hence, although the propagation facility is the same for C_1^α and C_3^α and is the same for C_2^α and C_4^α, the meet for C_3^α and C_4^α is more precise than the meet for C_1^α and C_2^α. ▪

However, inspection of C_1^α and C_2^α, (and C_3^α and C_4^α,) discloses a cyclic dependency in that x is dependent on both y and z in the first, but z is dependent on y and x in the second. There is a constraint on substitutions in logic programs that they be idempotent and dependencies such as these would break that constraint. Therefore providing the logic programs are legitimate in this sense it is not an unreasonable conjecture that the afore mentioned implementation shortcut might be safe. The first automated abstraction employed this shortcut.

8.6.2 Analyser II

The analyser is due to King and Howe [HK01]. A query dependent analysis is a top-down analysis as it commences from a known goal. A bottom-up analysis proceeds from what is known at the basest level and is usually based on the T_P operator which embodies an algorithm for finding the minimal model or meaning of a program. The analyser employed here is a hybrid, a query dependent bottom-up analyser. This is not the contradiction in terms that it appears to be. For an analysis to be useful it should be as general as possible, so the query is as general as possible, that is, a query with no instantiated arguments. Concern is with the possible computed answers to such a query. Since the computed answers constitute the success set, this concern extends to the call and answer patterns for the success set. If a query is to succeed it must satisfy the answer pattern for the success set, and a pre-requisite of this condition is that it must also satisfy the call pattern for the success set. In order to satisfy the call pattern, the call pattern must be known and in order to infer the call pattern, analysis must begin at the bottom and work its way up. The analyser references a database of abstract facts derived from program abstraction and also maintains and references a dynamic database of facts that fall into two classes. The first consists of current facts known about a particular predicate, the second acts as a place marker. The need for a place marker arises because the iteration scheme works its way through the program SCCs on the fly. This means that there is no need for the pre-processing phase that would compute the SCCs, but the number of iterations is reduced as auxilliary predicates are completely analysed before those at the top level. The principles employed are built on those of [Cod99] and the technique is termed by the authors *ordered induced magic*. Hence the only pre-analysis phase required is abstraction.

8.7 Example Analyses

Implementation I In Figure 21 that follows, the first column displays the original program and the second the set of groundness dependency relationships derived by analysis. These results are from the first implementation which experimentally employed a tighter abstraction than that prescribed by α. The dependencies can be interpreted as formulae representing Def_X functions as follows: i) $x = 0$ means that x is *true* (definitely ground) ii) $x = y + z$ means that $x \leftrightarrow (y \wedge z)$ and iii) $0 \leq x$ means that nothing is known about x, it simply confines the computation to the positive orthant of n-space. The abstracted magic'd programs are not included for reasons of space.

The first two programs are taken from [AMSS98], and the analyses are those expected from a top-down call-pattern analysis using Def_X. Similarly the output for the analysis of both the `quicksort/2` and the `queens/2` programs along with their predicates is commensurate with that expected. The analysis results for the game program `kalah` can also be found in Appendix A, and the results are also commensurate with those expected with Def_X analysis.

Implementation II The analysis results for the second implementation, in Figure 21, are very sparse as there was a problem with the projection machinery in SICStus that prevented further experimentation. The question mark in the table indicates where the analysis terminated with a system error. Again the analysis output was as expected with a *Def*-based analysis. The abstraction was as prescribed in $\alpha(Cone^n)$: i) $x = 0$ is interpreted as x is *true*, ii) $x \leq y$ as $x \leftarrow y$ and iii) $x \leq y + z$ as $x \leftarrow (y \wedge z)$.

8.8 Summary

Both implementations were curtailed prematurely and the fact that neither analysis was complete, even as a prototype is vexing to say the least. However, there are some observations that can be made and the opportunities for further investigation are many.

The aims then of the implementations were:

- to confirm the theory, that an analysis based on abstract cones could capture the same dependency information as *Def* - This has undoubtedly been achieved, the analysis results in both implementations, although sparse have covered various predicate structures and addressed some of the likely problems with *Def* analyses as identified in the literature.

- to see if the speed of the analyses based on abstract cones was acceptable. This is currently still unknown. The timings for the first analysis were in seconds, but this implementation was largely experimental and was curtailed before it had reached an optimisation phase. The second implementation did not reach the point of speed considerations, but as far as it went, analysis results were those expected.

The first implementation was a groundness analysis based on the previous argument size analysis - a goal dependent analysis that was bottom up, courtesy of the magic transform. The implementation used the same iterative technique that had been used before with the same ground representation. The second implementation was designed as a plug-in facility to a hybrid analyser that used different domains within one analysis [HK01]. Some observations can be made from the experience of these two implementations, despite their inconclusive nature with regard to timing.

It should be noted that although the orthogonal forms of both Def_X and $\alpha(Cone^n)$ are employed in the proof strategy, computation does not require the orthogonal form, allowing more compact form with transitive dependencies implicit in the representation.

Further, the use of an equality: $x = y + z$, to capture the dependency of the form $x \leftarrow y \wedge z$ rather than the inequalities prescribed by the abstraction is a fertile area for investigation, not only because empirical information and conjecture suggest that it is sound, but, because if it is a sound optimisation, then there must be an identifiable subclass of *Def*, that represents legitimate dependencies between the variables of logic programs.

In summary, $\alpha(Cone^n)$ is a viable abstract domain for representing dependency information. Further investigation, especially with the implementation of the abstract meet, is required on the implementation front, but the evidence is promising and with the facility to capture this information at hand, as is the case in a constraint solving environment, providing the analysis times are acceptable this domain could prove most useful.

Program	*Groundness Dependencies*
`q(X, Y):-`	`q`$^{\mathcal{A}}(x,\ y)$ `:-`
`   p(X, Y),`	$\{x = 0,\ y = 0\}$
`   r(X, Y),`	`p`$^{\mathcal{A}}(x,\ y)$ `:-`
`   s(X).`	$\{x = 0,\ y = 0,\}$
	`r`$^{\mathcal{A}}(x,\ y)$ `:-`
`p(X, X).`	$\{x = 0,\ y = 0\}$
`r(a, Y).`	`s`$^{\mathcal{A}}(x)$ `:-`
`r(X, a).`	$\{x = 0\}$
`s(X).`	
`or(X, Y, Z):-`	`or`$^{\mathcal{A}}(x,\ y,\ z)$ `:-`
`   and(X, Y, U),`	$\{x = 0,\ y = 0, z = 0\}$
`   xor(X, Y, V),`	`and`$^{\mathcal{A}}(x,\ y,\ z)$ `:-`
`   xor(U, V, Z).`	$\{z = x + y,\ 0 \leq x,\ 0 \leq y\}$
	`xor`$^{\mathcal{A}}(x,\ y,\ z)$ `:-`
`and(true, Y, Y).`	$\{x = y,\ z = 0,\ 0 \leq z\}$
`and(X, true, X).`	
`and(false, false, false).`	
`xor(X, X, false).`	
`xor(true, false, true).`	
`xor(false, true, true).`	
`quicksort([], []).`	`quicksort`$^{\mathcal{A}}(x,\ y)$ `:-`
`quicksort([X|Xs], S):-`	$\{x = y,$
`   part(X, Xs, L, G),`	$0 \leq x\}$
`   quicksort(L, SL),`	
`   quicksort(G, SG),`	`part`$^{\mathcal{A}}(w,\ x,\ y,\ z)$ `:-`
`   append(SL, [X|SG], S).`	$\{x = 0,\ y = 0,\ z = 0,$
	$0 \leq w,$
`part(_, [], [], []).`	
`part(X, [Y|Ys], L, [Y|G]):-`	`append`$^{\mathcal{A}}(x,\ y,\ z)$ `:-`
`   X =< Y,`	$\{z = x + y,$
`   part(X, Ys, L, G).`	$0 \leq x,\ 0 \leq y\}$
`part(X, [Y|Ys], [Y|L], G):-`	
`   Y =< X,`	
`   part(X, Ys, L, G).`	
`append([], Y, Y).`	
`append([X|Xs], Y, [X|Zs]):-`	
`   append(Xs, Y, Zs).`	
`queens(X, Y).`	`queens`$^{\mathcal{A}}(x,\ y)$ `:-`
`delete(X, Y, Z).`	$\{x = y, 0 \leq x\}$
`perm(X, Y).`	`delete`$^{\mathcal{A}}(x,\ y,\ z)$ `:-`
`safe(X).`	$\{y = x + z,\ z \leq y,\ 0 \leq z\}$
`no_attack(X,Y,Z).`	`perm`$^{\mathcal{A}}(x,\ y)$ `:-`
	$\{x = y,\ 0 \leq x\}$
	`safe`$^{\mathcal{A}}(x)$ `:-`
	$\{0 \leq x\}$
	`no_attack`$^{\mathcal{A}}(x,\ y,\ z)$ `:-`
	$\{0 \leq x,\ y = 0,\ z = 0\}$

Figure 20: Analysis Results for Implementation I

Program	*Groundness Dependencies*
`reverse([], []).`	$\texttt{reverse}^{\mathcal{A}}(x,\, y)$ `:-`
`reverse([X|Xs],Zs) :-`	$\{x \leq y,\, y \leq x\}$
`   reverse(Xs, Ys),`	
`   append(Ys, [X], Zs).`	$\texttt{append}^{\mathcal{A}}(x,\, y,\, z)$ `:-`
	$\{z \leq x + y,\, x \leq z,\, y \leq z\}$
`append([], Y, Y).`	
`append([X|Xs], Ys, [X|Zs]):-`	
`   append(Xs,Y, Zs).`	
`main(X, Y):-`	$\texttt{main}^{\mathcal{A}}(x,\, y)$ `:-`
`   quicksort(X, Y, Z).`	`?`
`quicksort([], Y, Y).`	$\texttt{quicksort}^{\mathcal{A}}(x,\, y,\, z)$ `:-`
`quicksort([X|Xs], Ys, Zs):-`	$\{x = 0,\, y = z\}$
`   part(X, Xs, Xs1, Xs2),`	
`   quicksort(Xs1, Ys, [X|U]),`	$\texttt{part}^{\mathcal{A}}(x,\, y,\, z,\, w)$ `:-`
`   quicksort(Xs2, U, Zs).`	$\{x = 0,\, z = 0,\, w = 0\}$
`part([], _, [], []).`	
`part([Y|Ys], X, L, [Y|G]):-`	
`   X =< Y,`	
`   part(X, Ys, L, G).`	
`part([Y|Ys], X, [Y|L], G):-`	
`   Y < X,`	
`   part(X, Ys, L, G).`	

Figure 21: Analysis Results for Implementation II

Chapter 9

Summary

"As you set out for Ithaka
hope the voyage is a long one,
full of adventure, full of discovery.
...
Ithaka gave you the marvelous journey.
Without her you would not have set out.
She has nothing left to give you now..."
- C. P. Cavafy, *Ithaka*

...except the beginning of the next journey.

The contributions of this thesis are four-fold, two are theoretical and two are practical; they are all concerned with polyhedra:

The Isomorphism between $\alpha(Cone^n)$ and Def The work in Chapter 7 amounts to the identification of a subclass of polyhedra, namely abstract polyhedral cones. The abstract polyhedral cones are derived by an abstraction mapping that is couched in terms of the facility to infer that a variable is zero. The abstraction is a total map on the domain of all polyhedral cones that effectively collapses an infinite domain to a finite one. The notion of reducing an infinite domain to a finite one by defining a relation that is a many to one map, suggests the possibility of equivalence classes. However, in this case the equivalence relation does not preserve the ordering between potential classes and the integrity of the meet is not preserved. The abstraction induces a finite lattice that although a lattice in its own right is not a sublattice of the infinite lattice of polyhedral cones.

These abstract polyhedral cones can capture dependency relationships in the same way as a subclass of the positive Boolean functions known as Def. A lattice isomorphism is established between these two apparently disparate domains. The isomorphism required the definition of a normal form for both domains to facilitate an injective map. The representation of an abstract polyhedral cone is as a set of implicitly conjoined linear inequalities. Since linear combination of linear inequalities can disclose entailed information, the establishment of the normal form for the abstract polyhedral cones includes a discussion on the motivation for, and possible outcomes of, the linear combination of linear inequalities. The discussion concludes with a formal demonstration that the normal form is a complete set of inference rules for each variable, and that no further inference rule is entailed by the representation of any abstract polyhedral cone. The establishment of analogous normal forms for these two domains, in that they both constitute a complete set of inference rules for each variable, allows a bijective map that is both semantic and syntactic. It is precisely this peculiar attribute of this mapping that allows the proof of the order isomorphism and with it the lattice isomorphism.

This work concludes with a conjecture that whenever there is a mapping between sets of inference rules of this kind, then the order isomorphism follows and therefore in the case where both domains are complete lattices the lattice isomorphism follows too.

Representations of Polyhedra In Chapter 6 the precise conditions that allow different representations of the same polyhedron are established - the outcome of an investigation of the properties that uniquely identify a polyhedron. The impetus for this work was to gain a better understanding of how linear spaces interact and to establish a spatial rationale for a well known widening technique. It is demonstrated that the widening is representation independent only when the criteria that equalities be represented as pairs of inequalities is met.

The body of this work consists of the establishment of the unique properties of polyhedra with respect to their representation as sets of implicitly conjoined linear inequalities. This entails the definition of the *relative tangent hyperplanes* which constitute the unique bounds of a polyhedron that allow its delineation in n-space. In any minimal representation of a polyhedron as a set of implicitly conjoined linear inequalities, each linear inequality represents a closed half-space and the relative boundary of each half-space is a hyperplane of n-space. So each half-space in a representation has a boundary hyperplane that contains the relative tangent hyperplane that is one of the unique bounds of the polyhedron. Hence the relative tangent hyperplanes that delineate a polyhedron within n-space are each embedded in the boundary hyperplanes of the half-spaces in the representation of the polyhedron. Alternative representations are possible only when a polyhedron is of dimension less than the n-dimensional space in which it is embedded. The reason this is possible is because then the polyhedron is bound by half spaces that have the space in which to rotate about the relative tangent hyperplanes or about the polyhedron itself. Each minute rotation, since it contains the relative tangent hyperplane of the polyhedron, is an alternative candidate for the set of inequalities in a representation of that polyhedron.

Computational Tactics for Polyhedra In the process of the widening investigation recounted in the previous paragraph, it became apparent that the representation of a polyhedron could properly be considered in two parts: i) the representation of the affine hull of the polyhedron, and ii) the representation of the half-spaces that intersect the affine hull and so delineate the polyhedron within its affine hull. This leads to the observation that operations, particularly the convex hull, and entailment, over polyhedra can be considered within the affine hull that is the least upper bound of the affine hulls of the operands, rather than in n-space. This may not seem particularly advantageous but in increasing sequences when such an operation might be required, the affine hulls of the operands are often either the same or ordered by subset inclusion.

This premise with regard to affine hulls also holds for the technique known as widening. The improved understanding that this discussion affords has also allowed the suggestion of a computationally less expensive interpretation of this widening tactic which is aimed at identifying the common bounds of polyhedra in an increasing sequence.

Polyhedra for Analyses Two analyses are presented. The argument size analysis of Chapter 5 introduced and established as valid, a relaxation technique used in disjunctive constraint programming for computing convex hulls. This technique avoids the cost of dual representations and switching between them, that has been established practice. This analysis also served to illustrate the very useful information that can be distilled with analyses employing polyhedral domains. Further, it was noted that in logic programming, where the recursive definition of procedure is the most usual, increasing sequences based on program definitions might display different kinds of behaviour. Some different behaviours are classified and it is noted that they might warrant different tactics with regard to introducing a widening to enforce analysis convergence.

The two prototypes for groundness analysis have separately employed a number of different tactics that are recounted in Chapter 8. Unfortunately neither implementation was completed and therefore efforts to improve analysis speed had not been commenced, so in this respect the work is inconclusive. It should be noted that analyses should be completed within acceptable time frames that are not necessarily limited to the smallest time frame. Whilst there will be circumstances when the smallest time frame is the only acceptable one, chip and processor performances are still improving and the convenience of having solvers able to perform analyses, without the set-up and maintenance costs of of analysis representation and manipulation machinery, may outweigh the need for the fastest analysis time, providing the analysis time is acceptable. In any event, preliminary results confirm the theory that abstract polyhedral cones can capture dependency information in the same way as Boolean functions in *Def*.

Overall therefore, there is a positive indication that both polyhedra and abstract cones are useful abstract domains for the analysis of logic programs, both to capture analysis with respect to some measure of size and to capture groundness dependency analysis. However, abstract domains are not paradigm-dependent and the use of these polyhedral domains and widening techniques may also serve analyses in the imperative paradigm.

Future Work These contributions constitute a stepping stone to further research. Primarily further research must be directed at an efficient implementation of the abstract cone domain for *Def* analyses. Investigations should be directed at the implementation techniques that have arisen from the work on widenings: the alternative method to capturing the common bounds of sequence of increasing polyhedra and the potentially time saving technique of restricting operations to the confines of the smallest affine space that contains all of the operands. Work should be directed at qualifying the circumstances when these techniques are most useful and also whether there are occasions when their application is not sensible, that is, when at best they afford no tangible gain. A deeper understanding of how linear spaces interact with one another provided potentially useful computational insights, but there remains much more to be investigated.

Appendix A

Implementation I

A.1 kalah

Program definition

```
play(Game,Result) :-
   initialize(Game,Position,Player),
   displaygame(Position,Player),
   play(Position,Player,Result).

play(Position,Player,Result) :-
  gameover(Position,Player,Result),
  announce(Result).
play(Position,Player,Result) :-
  choosemove(Position,Player,Move),
  move(Move,Position,Position1),
  displaygame(Position1,Player),
  nextplayer(Player,Player1),
  play(Position1,Player1,Result).

choosemove(Position,computer,Move) :-
   lookahead(Depth),
   alphabeta(Depth,Position,40,40,Move,Value).
choosemove(Position,opponent,Move) :-
   genlegal(Move).

alphabeta(0,Position,Alpha,Beta,Move,Value) :-
   value(Position,Value).
alphabeta(D,Position,Alpha,Beta,Move,Value) :-
  D > 0,
  allmoves(Position,Moves),
  Alpha1 is 0 - Beta,
  Beta1 is 0 - Alpha,
  D1 is D - 1,
  evaluateandchoose(Moves,Position,D1,Alpha1,Beta1,[],p(Move,Value)).
```

```
allmoves(Position,[X]) :-
  move(Position,X).
allmoves(Position,[X|Xs]) :-
  move(Position,X),
  allmoves(Position,Xs).



evaluateandchoose([Move|Moves],Position,D,Alpha,Beta,Record,BestMove) :-
  move(Move,Position,Position1),
  alphabeta(D,Position1,Alpha,Beta,MoveX,Value),
  Value1 is 0 - Value,
  cutoff(Move,Value1,D,Alpha,Beta,Moves,Position,Record,BestMove).
evaluateandchoose([],Position,D,Alpha,Beta,Move,p(Move,Alpha)).

cutoff(Move,Value,D,Alpha,Beta,Moves,Position,Move1,p(Move,Value)) :-
  Value >= Beta.
cutoff(Move,Value,D,Alpha,Beta,Moves,Position,Move1,BestMove) :-
  Alpha < Value,
  Value < Beta,
  evaluateandchoose(Moves,Position,D,Value,Beta,Move,BestMove).
cutoff(Move,Value,D,Alpha,Beta,Moves,Position,Move1,BestMove) :-
  Value =< Alpha,
  evaluateandchoose(Moves,Position,D,Alpha,Beta,Move1,BestMove).

move(Board,[M|Ms]) :-
   member(M,[1,2,3,4,5,6]),
   stonesinhole(M,Board,N),
   extendmove(N,M,Board,Ms).
move(board([0,0,0,0,0,0],K,Ys,L),[]).

member(X,[X|Y]).
member(X,[F|T]) :-
   member(X,T).

stonesinhole(M,board(Hs,K,Ys,L),Stones) :-
  nthmember(M,Hs,Stones),
  Stones > 0.

extendmove(Stones,M,Board,[]) :-
  Stones =\= 7 - M.
extendmove(Stones,M,Board,Ms) :-
  Stones =:= 7 - M,
  distributestones(Stones,M,Board,Board1),
  move(Board1,Ms).

move([N|Ns],Board,FinalBoard) :-
```

```
  stonesinhole(N,Board,Stones),
  distributestones(Stones,N,Board,Board1),
  move(Ns,Board1,FinalBoard).
move([],Board1,Board2) :-
  swap(Board1,Board2).

distributestones(Stones,Hole,Board,FinalBoard) :-
  distributemyholes(Stones,Hole,Board,Board1,Stones1),
  distributeyourholes(Stones1,Board1,FinalBoard).

distributemyholes(Stones,N,board(Hs,K,Ys,L),board(Hs1,K1,Ys,L),Stones1) :-
  Stones > 7 - N,
  pickupanddistribute(N,Stones,Hs,Hs1),
  K1 is K + 1,
  Stones1 is Stones + N - 7.
distributemyholes(Stones,N,board(Hs,K,Ys,L),Board,0) :-
  pickupanddistribute(N,Stones,Hs,Hs1),
  checkcapture(N,Stones,Hs1,Hs2,Ys,Ys1,Pieces),
  updatekalah(Pieces,N,Stones,K,K1),
  checkiffinished(board(Hs2,K1,Ys1,L),Board).

checkcapture(N,Stones,Hs,Hs1,Ys,Ys1,Pieces) :-
  FinishingHole is N + Stones,
  OppositeHole is 7 - FinishingHole,
  nthmember(OppositeHole,Ys,Y),
  Y > 0,
  nsubstitute(OppositeHole,Hs,0,Hs1),
  nsubstitute(FinishingHole,Ys,0,Ys1),
  Pieces is Y + 1.
checkcapture(N,Stones,Hs,Hs,Ys,Ys,0).

checkiffinished(board(Hs,K,Ys,L),board(Hs,K,Hs,L1))   :-
 zero(Hs),
 sumlist(Ys,YsSum),
 L1 is L + YsSum.
checkiffinished(board(Hs,K,Ys,L),board(Ys,K1,Ys,L))   :-
 zero(Ys),
 sumlist(Hs,HsSum),
 K1 is K + HsSum.
checkiffinished(Board,Board).

updatekalah(0,Stones,N,K,K) :-
  Stones < 7 - N.
updatekalah(0,Stones,N,K,K1) :-
  Stones =:= 7 - N,
  K1 is K + 1.
updatekalah(Pieces,Stones,N,K,K1) :-
  Pieces > 0,
  K1 is K + Pieces.

distributeyourholes(0,Board,Board).
distributeyourholes(Stones,board(Hs,K,Ys,L),board(Hs,K,Ys1,L)) :-
   1 =< Stones,
   Stones =< 6,
   nonzero(Hs),
   distribute(Stones,Ys,Ys1).
distributeyourholes(Stones,board(Hs,K,Ys,L),board(Hs,K,Ys1,L)) :-
   Stones > 6,
   distribute(6,Ys,Ys1),
   Stones1 is Stones - 6,
   distributestones(Stones1,1,board(Hs,K,Ys1,L),Board).
distributeyourholes(Stones,board(Hs,K,Ys,L),board(Hs,K,Hs,L1)) :-
   zero(Hs),
   sumlist(Ys,YsSum),
   L1 is Stones + YsSum + L.

pickupanddistribute(1,N,[H|Hs],[0|Hs1]) :-
   distribute(N,Hs,Hs1).
pickupanddistribute(K,N,[H|Hs],[0|Hs1]) :-
   K > 1,
   K1 is K - 1,
   pickupanddistribute(K1,N,Hs,Hs1).

distribute(0,Hs,Hs).
distribute(N,[H|Hs],[H1|Hs1]) :-
   N > 0,
   N1 is N - 1,
   H1 is H + 1,
   distribute(N1,Hs,Hs1).
distribute(N,[],[]).

value(board(H,K,Y,L),Value) :-
  Value is K - L.

gameover(board(0,N,0,N),Player,draw) :-
  pieces(K),
  N =:= 6 * K.
gameover(board(H,K,Y,L),Player,Player) :-
  pieces(N),
  K > 6 * N.
gameover(board(H,K,Y,L),Player,Opponent) :-
  pieces(N),
  L > 6 * N,
  nextplayer(Player,Opponent).

announce(opponent).
announce(computer).
announce(draw).

nthmember(N,[H|Hs],K) :-
  N > 1,
  N1 is N - 1,
  nthmember(N1,Hs,K).
nthmember(1,[H|Hs],H).

nsubstitute(1,[X|Xs],Y,[Y|Xs]).
nsubstitute(N,[X|Xs],Y,[X|Xs1]) :-
   N > 1,
   N1 is N -1,
   nsubstitute(N1,Xs,Y,Xs1).

nextplayer(computer,opponent).
nextplayer(opponent,computer).

legal([N|Ns]) :-
  0 < N,
  N < 7,
  legal(Ns).
legal([]).

genlegal([N|Ns]) :-
  member(N,[1,2,3,4,5,6]),
  genlegal(Ns).
genlegal([]).

swap(board(Hs,K,Ys,L),board(Ys,L,Hs,K)).

displaygame(Position,computer) :-
  show(Position).
displaygame(Position,opponent) :-
  swap(Position,Position1),
  show(Position1).

show(board(H,K,Y,L)) :-
   reverse(H,Hr),
   writestones(Hr),
   writekalahs(K,L),
   writestones(Y).

writestones(H) :-
  displayholes(H).

displayholes([H|Hs]) :-
   writepile(H),
   displayholes(Hs).
displayholes([]).

writepile(N) :-
  N < 10,
  write(N).
writepile(N) :-
  N >= 10,
  write(N).

writekalahs(K,L) :-
  write(K),
  write(L).

zero([0,0,0,0,0,0]).
nonzero(Hs) :-
  Hs \== [0,0,0,0,0,0].

reverse(L,K):-
   rev(L,[],K).

rev([],L,L).
rev([H|T],L,K):-
   rev(T,[H|L],K).

sumlist(Is,Sum) :-
   sumlist(Is,0,Sum).
sumlist([],Sum,Sum).
sumlist([I|Is],Temp,Sum) :-
  Temp1 is Temp + 1,
  sumlist(Is,Temp1,Sum).

lookahead(2).
lookahead(5).

initialize(kalah,board(a,0,a,0),opponent).
initialize(kalah,board(b,1,b,1),computer).

pieces(6).
pieces(1).
```

Analysis Output

```
I = [atom(allmoves_a,2,[equal(var(2),0),lesseq(var(1),0)]),
atom(allmoves_c,2,[lesseq(var(1),0)]),
atom(alphabeta_a,6,[equal(var(1),0),equal(var(3),0),equal(var(4),0)]),
atom(alphabeta_c,6,[equal(var(1),0),equal(var(3),0),equal(var(4),0),
lesseq(var(2),0)]),
atom(announce_a,1,[equal(var(1),0)]),
atom(announce_c,1,[equal(var(1),0)]),
atom(checkcapture_a,7,[equal(var(1),0),equal(var(2),0),equal(var(7),0),
lesseq(0,var(4)),lesseq(minus(var(4),var(3)),0),
lesseq(minus(var(6),var(5)),0)]),
atom(checkcapture_c,7,[equal(var(1),0),equal(var(2),0),lesseq(0,var(3))]),
atom(checkiffinished_a,2,[lesseq(var(1),0),lesseq(minus(var(2),var(1)),0)]),
atom(checkiffinished_c,2,[lesseq(var(1),0)]),
atom(choosemove_a,3,[equal(var(1),0),equal(var(2),0)]),
atom(choosemove_c,3,[equal(var(1),0),equal(var(2),0)]),
atom(cutoff_a,9,[equal(var(1),0),equal(var(2),0),equal(var(3),0),
equal(var(4),0),equal(var(5),0),equal(var(6),0),equal(var(8),0),
equal(var(9),0)]),
atom(cutoff_c,9,[equal(var(1),0),equal(var(2),0),equal(var(3),0),
equal(var(4),0),equal(var(5),0),equal(var(6),0),equal(var(8),0)]),
atom(displaygame_a,2,[equal(var(1),0),equal(var(2),0)]),
atom(displaygame_c,2,[equal(var(2),0),lesseq(var(1),0)]),
atom(displayholes_a,1,[equal(var(1),0)]),
atom(displayholes_c,1,[lesseq(0,var(1))]),
atom(distribute_a,3,[equal(var(1),0),equal(var(2),var(3))]),
atom(distribute_c,3,[equal(var(1),0)]),
atom(distributemyholes_a,5,[equal(var(1),0),equal(var(2),0),
equal(var(5),0),lesseq(var(3),0),lesseq(0,minus(var(3),var(4)))]),
atom(distributemyholes_c,5,[equal(var(1),0),equal(var(2),0),
lesseq(var(3),0)]),
atom(distributestones_a,4,[equal(var(1),0),equal(var(2),0),lesseq(var(3),0),
lesseq(0,minus(var(3),var(4)))]),
atom(distributestones_c,4,[equal(var(1),0),equal(var(2),0),
lesseq(var(3),0)]),
atom(distributeyourholes_a,3,[equal(var(1),0),lesseq(var(2),0),
lesseq(minus(var(3),var(2)),0)]),
atom(distributeyourholes_c,3,[equal(var(1),0),lesseq(var(2),0)]),
atom(evaluateandchoose_a,7,[equal(var(1),0),equal(var(3),0),equal(var(4),0),
equal(var(5),0),equal(var(6),0),equal(var(7),0)]),
atom(evaluateandchoose_c,7,[equal(var(1),0),equal(var(3),0),equal(var(4),0),
equal(var(5),0),equal(var(6),0)]),
atom(extendmove_a,4,[equal(var(1),0),equal(var(2),0),equal(var(4),0),
lesseq(var(3),0)]),
atom(extendmove_c,4,[equal(var(1),0),equal(var(2),0),lesseq(0,var(4)),
lesseq(var(3),0)]),
atom(gameover_a,3,[equal(var(1),0),equal(var(2),0),equal(var(3),0)]),atom(gameover_c,3,
atom(genlegal_a,1,[lesseq(0,var(1))]),
atom(genlegal_c,1,[]),
atom(initialize_a,3,[equal(var(1),0),equal(var(2),0),equal(var(3),0)]),
atom(initialize_c,3,[lesseq(0,var(1))]),
atom(lookahead_a,1,[equal(var(1),0)]),
atom(lookahead_c,1,[]),
atom(member_a,2,[lesseq(0,var(1)),lesseq(minus(var(1),var(2)),0)]),
atom(member_c,2,[lesseq(0,var(1))]),
atom(move_a,2,[equal(var(2),0),lesseq(var(1),0)]),
atom(move_a,3,[equal(var(1),0),lesseq(var(2),0),
lesseq(minus(var(3),var(2)),0)]),
atom(move_c,2,[lesseq(var(1),0)]),
atom(move_c,3,[lesseq(var(2),0)]),
atom(nextplayer_a,2,[equal(var(1),0),equal(var(2),0)]),
atom(nextplayer_c,2,[equal(var(1),0)]),
atom(nonzero_a,1,[equal(var(1),0)]),
atom(nonzero_c,1,[]),
atom(nsubstitute_a,4,[equal(var(1),0),equal(var(3),0),lesseq(0,var(4)),
lesseq(minus(var(4),var(2)),0)]),
atom(nsubstitute_c,4,[equal(var(1),0),equal(var(3),0),lesseq(0,var(2))]),
atom(nthmember_a,3,[equal(var(1),0),lesseq(0,var(3)),
lesseq(minus(var(3),var(2)),0)]),
atom(nthmember_c,3,[lesseq(0,var(1))]),atom(parts,3,[lesseq(0,var(2)),
equal(var(1),plus(minus(var(2)),var(3))),lesseq(minus(var(2),var(3)),0)]),
atom(pickupanddistribute_a,4,[equal(var(1),0),equal(var(2),0),
lesseq(0,var(4)),lesseq(minus(var(4),var(3)),0)]),
atom(pickupanddistribute_c,4,[equal(var(1),0),equal(var(2),0)]),
atom(pieces_a,1,[equal(var(1),0)]),atom(pieces_c,1,[]),
atom(play_a,2,[equal(var(1),0),equal(var(2),0)]),
atom(play_a,3,[equal(var(1),0),equal(var(2),0),equal(var(3),0)]),
atom(play_c,2,[lesseq(0,var(1)),lesseq(0,var(2))]),
atom(play_c,3,[equal(var(1),0),equal(var(2),0),lesseq(0,var(3))]),
atom(rev_a,3,[lesseq(0,var(2)),equal(var(1),minus(var(3),var(2))),
lesseq(0,minus(var(3),var(2)))]),atom(rev_c,3,[lesseq(0,var(2))]),
atom(reverse_a,2,[equal(var(1),var(2)),lesseq(0,var(1))]),
atom(reverse_c,2,[]),atom(show_a,1,[equal(var(1),0)]),
atom(show_c,1,[lesseq(var(1),0)]),
atom(stonesinhole_a,3,[equal(var(1),0),
equal(var(3),0),lesseq(var(2),0)]),
atom(stonesinhole_c,3,[lesseq(0,var(1)),lesseq(var(2),0)]),
atom(sumlist_a,2,[equal(var(2),0),lesseq(0,var(1))]),
atom(sumlist_a,3,[equal(var(2),0),equal(var(3),0),lesseq(0,var(1))]),
atom(sumlist_c,2,[]),
atom(sumlist_c,3,[equal(var(2),0)]),
atom(swap_a,2,[equal(var(1),var(2)),lesseq(var(1),0)]),
atom(swap_c,2,[lesseq(var(1),0)]),
atom(true,0,[]),
atom(updatekalah_a,5,[equal(var(1),0),equal(var(2),0),equal(var(3),0),
equal(var(4),var(5))]),
atom(updatekalah_c,5,[equal(var(1),0),equal(var(2),0),equal(var(3),0)]),
atom(value_a,2,[equal(var(2),0),lesseq(var(1),0)]),
atom(value_c,2,[lesseq(var(1),0)]),
atom(writekalahs_a,2,[equal(var(1),0),equal(var(2),0)]),
atom(writekalahs_c,2,[]),atom(writepile_a,1,[equal(var(1),0)]),
atom(writepile_c,1,[lesseq(0,var(1))]),
atom(writestones_a,1,[equal(var(1),0)]),
atom(writestones_c,1,[]),atom(zero_a,1,[equal(var(1),0)]),
atom(zero_c,1,[])]
```

Bibliography

[AMSS98] T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Two classes of Boolean Function for dependency analysis. In *Science of Computer Programming*, volume 31 (1), pages 3–45, 1998.

[Bag94] R. Bagnara. On the detection of implicit and redundant numeric constraints in CLP programs. In *Joint Conference on Declarative Programming (GULP-PRODE)*, pages 312–326, 1994.

[Bag96] R. Bagnara. A hierarchy of constraint systems for data-flow analysis of constraint logic-based languages. Technical Report TR-96-10, Dipartimento di Informaticá, Universita di Pisa corso Italia 40, 56125 Pisa, Italy, 1996.

[BGL92] R. Bagnara, R. Giacobazzi, and G. Levi. Static Analysis of CLP Programs over Numeric Domains. In *Workshop on Static Analysis*, volume 81-82, pages 43–50, 1992.

[BGLM91] A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-semantics approach: theory and applications. *Journal of Logic Programming, Special Issue: Ten Years of Logic Programming*, 19-20:149–197, 1991.

[Bir48] G. Birkhoff. *Lattice Theory.* American Mathematical Society, 190 Hope Street, Providence, Rhode Island, USA, 1948.

[BJT99] F. Besson, T. Jensen, and J-P. Talpin. Polyhedral analysis for synchronous languages. In *Static Analysis Symposium.* Springer-Verlag, 1999.

[BK89] V. Balasundaram and K. Kennedy. A Technique for Summarizing Data Access and Its Use in Parallelism Enhancing Transformations. In *Programming Language Design and Implementation*, pages 41–53. ACM Press, 1989.

[BK96] F. Benoy and A.M. King. Inferring Argument Size Relationships with CLP($\mathcal{R}$). In *6th International Workshop on Logic Program Synthesis and Transformation.* Springer-Verlag, 1996.

[BK99] F. Benoy and A.M. King. An Isomorphism between Abstract Polyhedral Cones and Definite Boolean Functions. Technical Report 3-99, Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK, 1999.

[Bro83] A. Bronstead. *An Introduction to Convex Polytopes.* Springer Verlag, New York, 1983.

[CC92a] P. Cousot and R. Cousot. Abstract Interpretation and Application to Logic Programs. *Journal of Logic Programming*, 13(2,3):103–179, 1992.

[CC92b] P. Cousot and R. Cousot. Abstract Interpretation Frameworks. In *Journal of Logic Programming*. Springer-Verlag, 1992.

[CC92c] P. Cousot and R. Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In *4th International Symposium on Programming Language Implementation and Logic Programming*, volume LCNS 631, pages 269–295. Springer-Verlag, 1992.

[CC00] P. Cousot and R. Cousot. Abstract interpretation based program testing. In *Proceedings of the SSGRR 2000 Computer & eBusiness International Conference*. Scuola Superiore G. Reiss Romoli, July 31 – August 6 2000.

[CD95] M. Codish and B. Demoen. Analyzing Logic Programs using "Prop"-ositional Logic with a Magic Wand. *Journal of Logic Programming*, 25 (3):249–274, 1995.

[CH78] P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints among Variables of a Program. In *5th Symposium on Principles of Programming Languages*, pages 84–97, 1978.

[CL95] M. Codish and V. Lagoon. Persistent Type Analysis using a Non-Ground Domain. Technical report, Dept of Maths and Computer Science, Ben-Gurion University of the Negev, Israel, 1995.

[Cod99] M. Codish. Worst-Case Groundness Analysis using Positive Boolean Functions. *Journal of Logic Programming*, 41(1):125 –128, 1999.

[Col90] A. Colmerauer. An Introduction to Prolog III. In *CACM*, volume 33, pages 70–90, July 1990.

[Cou00] P. Cousot. Abstract interpretation: Achievements and perspectives. In *Proceedings of the SSGRR 2000 Computer & eBusiness International Conference*. Scuola Superiore G. Reiss Romoli, July 31 – August 6 2000.

[CT97] M. Codish and C. Taboch. A semantic basis for termination analysis of logic programs and its realisation using symbolic norm constraints. In *The Sixth International Conference on Algebraic and Logic Programming*, 1997.

[Dar91] P. Dart. On Derived Dependencies and Connected Databases. *Journal of Logic Programming*, pages 163–188, 1991.

[DB93] B. De Backer and H. Beringer. A CLP language handling disjunctions of linear constraints. In *International Conference on Logic Programming*, pages 550–563. MIT Press, 1993.

[DP90] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[Gal95] J. Gallagher. A Bottom-Up Analysis Toolkit. In *Workshop on Analysis of Logic Languages, Eilat, Israel*, June 1995.

[GBS95] J. Gallagher, D. Boulanger, and H. Sağlam. Practical Model-Based Static Analysis for Definite Logic Programs. In J.W. Lloyd, editor, *Proceedings International Logic Programming Symposium*, pages 351–365. MIT Press, 1995.

[GDL92] R. Giacobazzi, S. Debray, and G. Levi. Generalised Semantics and Abstract Interpretation for Constraint Logic Programs. Technical report, Dipartimento di Informatica, Universitá di Pisa, 1992.

[Get94] T. Getzinger. The Costs and Benefits of Abstract Interpretation- driven Prolog Optimisations. In *Static Analysis Symposium*, pages 1–25. Springer-Verlag, 1994.

[GHC01] S. Genaim, J.M. Howe, and M. Codish. Worst-Case Groundness Analysis using definite Boolean Functions. *Theory and Practice of Logic Programming*, 2001. Forthcoming.

[Gia93] R. Giacobazzi. *Semantic Aspects of Logic Program Analysis.* PhD thesis, Dipartimento di Informaticá, Universita di Pisa corso Italia 40, 56125 Pisa, Italy, 1993.

[Gru67] B. Grunbaum. *Convex Polytopes.* Interscience Publishers - J.Wiley and Sons, London, 1967.

[Hal79] N. Halbwachs. Détermination automatique de relations linéaires vérifiées par les variables d'un programme. Universit'e scientifique et médicale de Grenoble, 1979. Thèse de 3 ème d'informatique.

[Han95] M. Handjieva. Abstract Interpretation of Constraint Logic Programs over Numeric Domains. Technical report, LIX, Ecole Polytechnique, France, 1995.

[HAZCK00] A. Heaton, M. Abo-Zaed, M. Codish, and A.M. King. A Simple Polynomial groundness analysis for logic programs. *Journal of Logic Programming*, 45:143–156, September 2000.

[HBC$^+$99] M. Hermenegildo, F. Bueno, D. Cabeza, M. Carro, M. Garcia de la Banda, P. Lopez, and G. Puebla. The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems. In *Parallelism and Implementation of Logic and Constraint Logic Programming.* Nova Science, 1999.

[HJM$^+$92] N. C. Heintze, J. Jaffar, S. Michaylov, P. J. Stuckey, and R. H. C. Yap. *The CLP($\mathcal{R}$) Programmer's Manual Version 1.2*, 1992.

[HK99] J.M. Howe and A.M. King. A Semantic Basis for Specialising Domain Constraints. Technical Report 21-99, Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK, 1999.

[HK00] J.M. Howe and A.M. King. Specialising finite domain programs using polyhedra. In A. Bossi, editor, *Logic Programming Synthesis and Transformation 1999*, volume 1817 of *Lecture Notes in Computer Science*, pages 118–135. Springer-Verlag, March 2000.

[HK01] J.M. Howe and A.M. King. Efficient Groundness Analysis in Prolog. Under consideration for publication in Theory and Practice of Logic Programming, 2001.

[Hog90] C.J. Hogger. *Essentials of Logic Programming.* Clarendon Press, Oxford, UK, 1990.

[Hor90] R. N. Horspool. Analyzing List Usage in Prolog Code. University of Victoria, March 1990.

[HPR94] N. Halbwachs, Y.E. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *First International Static Analysis Symposium*, volume LCNS 864, pages 223–237. Springer Verlag, September 1994.

[JBE94] G. Janssens, M. Bruynooghe, and V. Englebert. Abstracting numeric values in CLP(H,N). In *Programming Language Implementation and Logic Programming*, pages 400–414. Springer-Verlag, 1994.

[Kar76] M. Karr. Affine Relationships Among Variables of a Program. *Acta Informatica*, 6:133–151, 1976.

[Ker94] Alain Kerbrat. Reachable state space analysis of lotos specifications. In *7th International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols*, pages 181 – 196, October 1994.

[KS93] D. B. Kemp and P. J. Stuckey. Analysis Based Constraint Query Optimisation. In *International Conference on Logic Programming*, pages 666–682, 1993.

[KSB97] A.M. King, K. Shen, and F. Benoy. Lower-bound time-complexity analysis of logic programs. In Jan Maluszynski, editor, *International Symposium on Logic Programming*, pages 261 – 276. MIT Press, November 1997.

[KvE76] R. A. Kowalski and M.H. van Emden. The semantics of predicate logic as a programming language. *Journal of ACM*, 23:733–742, 1976.

[Lay82] S. Lay. *Convex Sets and their Applications.* Wiley Interscience Publication - J.Wiley and Sons, New York, 1982.

[lCMH90] B. le Charlier, K. Musumbu, and P. Van Hentenryck. Efficient and accurate algorithms for the abstract interpretation of prolog programs. In *Research Paper No. RP-90/9.* University of Namur, Belgium, 1990.

[Llo93] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Heidelberg, Germany, 2nd edition, 1993.

[LMM88] J.L. Lassez, M.J. Maher, and K. Marriott. *Unification Revisited*, pages 587–625. Morgan Kauffman Publishers Inc, 95. First Street, Los Altos,California 94022, 1988.

[Mel81] C. Mellish. Abstract Generation of Mode declarations for Prolog Programs (Draft). In *DAI Research Paper.* University of Edinburgh, 1981.

[MG92] F. Mesnard and J.-G. Ganascia. CLP($\mathcal{Q}$) for Proving Interargument Relations. In *META'92*, pages 308–320, Uppsala, Sweden, 1992. Springer-Verlag.

[Mil90] H. Millroth. *Reforming Compilation of Logic Programs.* PhD thesis, Computing Science Department, Uppsala University, 1990.

[MK97] J.C. Martin and A.M. King. Generating efficient and terminating programs. In *TAPSOFT.* Springer-Verlag, 1997.

[MS90] K. Marriott and H. Søndergaard. Abstract interpretation of logic programs: the denotational approach. In *GULP.* A.Bossi ed,Padova, 1990.

[Plu95] L. Plumer. Automatic verification of parallel logic programs: Termination. In *Logic Programming: Formal Methods and Practical Applications*, pages 92–119. Elsevier Science, 1995.

[Roc70] R. Rockafellar. *Convex Analysis.* Princeton University Press, New Jersey, 1970.

[Sa97] H. Sağlam. *A Toolkit for Static Analysis of Constraint Logic Programs.* PhD thesis, Department of Computer Science, University of Bristol, 1997.

[SG97] H. Sağlam and J. Gallagher. Static Analysis of Logic Programs Using CLP as a Meta-Language. Technical Report CSTR-96-003, Department of Computer Science, University of Bristol, June 1997.

[SG98] H. Sağlam and J. Gallagher. Constrained Regular Approximation of Logic Programs. In N. Fuchs, editor, *Program Synthesis and Transformation. 7th International Workshop, LOPSTR'97, Leuven, Belgium, July 1997*, volume 1463, pages 282–299. Springer Verlag, Lecture Notes in Computer Science, June 1998.

[Soh94] K. Sohn. Constraints among Argument Sizes in Logic Programs. In *Principles of Database Systems*, pages 68–74. ACM Press, 1994.

[SSS97] C. Spiers, Z. Somogyi, and H. Søndergaard. Termination analysis for mercury. Technical report, Dept. of Computer Science, University of Melbourne, Australia, 1997.

[Sza63] G. Szasz. *Introduction to Lattice Theory.* Academic Press, New York, London and The Publishing House of the Hungarian Academy of Sciences, Budapest, 1963.

[Tar55] A. Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–310, 1955.

[Tay91] A. Taylor. *High Performance Prolog Implementation.* PhD thesis, Basser Department of Computer Science, Sydney, 1991.

[Ull85] J. D. Ullman. Implementation of Logical Query Languages for Databases. *ACM Transactions on Database Systems*, 10(3):289–321, 1985.

[UvG88] J.D. Ullman and A. van Gelder. Efficient tests for top-down logical rules. In *Journal ACM*, volume 35(2), pages 345–373. Springer-Verlag, 1988.

[van91] A. van Gelder. Deriving Constraints Among Argument Sizes in Logic Programs. *Annals of Mathematics and Artifical Intelligence*, 3:361–392, 1991.

[VD92] K. Verschaetse and D. De Schreye. Derivation of Linear Size Relations by Abstract Interpretation. In *4th International Symposium on Programming Language Implementation and Logic Programming*, volume LCNS 631, pages 296–310. Springer-Verlag, 1992.

[vR90] P. van Roy. *Can Logic Programming Execute as Fast as Imperative Programming?* PhD thesis, Computer Engineering Division, University of Southern California, 1990.

[Wil93] D. Wilde. A Library for doing Polyhedral Operations. Technical Report PI-785, Institut de Recherche en Informatique et Systemes Aleatoires, Campus Universitaire de Beaulieu, 35042 Rennes Cedex, France, 1993.