Solving polynomial constraints for proving termination of rewriting

Rafael Navarro Marset

Master Thesis
Software Engineering, Formal Methods, and Information Systems
Departamento de Sistemas Informáticos y Computación

December 15, 2008

Supervisor:
Salvador Lucas Alba

Contents

Chapter 1

Introduction

In software verification, one of the most challenging problems is the automatic verification of termination. A program or computation system is said to be terminating whether its computation does not lead to any infinite computation for any possible input data or function calls.

From a *scientific* point of view, termination is an old and classical problem (think of Turing's halting problem, for instance). On the other hand, from an *end-user* point of view, it is also crucial: many times, the unexpected behavior of an application, usually reported as (a never-ending) "processing", or "waiting for an answer" (and also as "abnormal termination"!) usually corresponds to hidden termination problems coming from bugs in the program.

Furthermore, ensuring termination is often a prerequisite for essential program properties like correctness. Thus, being able to *automatically prove* termination of programs is a key issue in modern software development. Unfortunately, termination is, in general, undecidable, so that there exists no algorithm that correctly establishes, for all programs whether they are terminating or not. However, the termination of programs is decidable for some classes of programs.

Term rewriting (see, e.g., [BN98, Ohl02, Ter03]) is a branch of theorical computer science which combines elements of logic, universal algebra, automated theorem proving and functional programming. It is a very powerful method for dealing computationally with equational logic. Equations lie at the foundation of mathematics and sciences. Equational reasoning is based on a restricted class of first order logic, where the only predicate symbol is equality. However, equations are pervasive in many problems. For instance, equations are often used to write programs as sets of equations, define programing languages interpreters, specify requirement to derive correct software or express properties and verify that they are fulfilled.

The main difference between term rewriting and equational logic is that, in term rewriting, equations are used as directed replacements *rules*, i.e. the left-hand side can be replaced by the right-hand side, but not vice versa. The basic ideas date back to Axel Thue. Term rewriting has the computational power of *Markov algorithms* and *Turing machines*. This constitutes to a close model to functional programming. It has applications in algebra (e.g. Boolean algebra, group theory and ring theory), recursion theory (what is and is not computable with certain sets of rewriting rules), software engineering (reasoning about equationally defined data types such as numbers, lists, sets, etc.) and programming languages (specially functional, algebraic, and logic programming). In general, term rewriting applies in any context where efficient methods for reasoning with equations are required.

Since term rewriting formalism is *Turing-complete*, the main idea for proving termination of a program $\mathcal{P}$ is translating into a TRS $\mathcal{R}_{\mathcal{P}}$ whose termination implies termination of $\mathcal{P}$. The notions coming from the already quite mature theory of termination of TRSs provide a basic collection of abstractions, notions, and methods which have been proved useful for treating termination problems in sophisticated programming languages. For some advanced programming languages like Haskell [HPW92], Maude [CDEL$^+$07] and Prolog [NM95], recent papers show that such termination technology may conform an excellent basis (see [GSST06] for Haskell, [DLMMU08] for Maude, and [SGST07] for Prolog).

1.1 Polynomial constraint solving for termination proofs

Proofs of termination in term rewriting involve solving constraints between terms s and t coming from (parts of) the rules of the Term Rewriting System. For instance, in the dependency pairs method [AG00] (which is the basis of most modern termination tools), given a rewrite rule $l \to r$ of a TRS $\mathcal{R}$, we get dependency pairs $L \to S$ for all subterms s of r which are rooted by a defined symbol; the notation T for a given term t means that the root symbol f of t is marked thus becoming F. The dependency pairs associated to $\mathcal{R}$ yield a new $\mathsf{DP}(\mathcal{R})$ which (together with $\mathcal{R}$) determines the so-called dependency chains whose finiteness or infiniteness characterize termination of $\mathcal{R}$.

Basically, we need to impose that $l \gtrsim r$ for all rules in the TRS $\mathcal{R}$ and $u \sqsupset v$ for all the dependency pairs $u \to v \in \mathsf{DP}(\mathcal{R})$. Here, $\gtrsim$ is a quasi-ordering[1] on terms and $\sqsupset$ is a well-founded ordering[2].

Polynomial interpretations are receiving an increasing attention for generating such or-

[1]A quasi-ordering $\gtrsim$ is a reflexive and transitive relation.

[2]An ordering $\sqsupset$ is an irreflexive and transitive relation. It is well-founded if there is no infinite sequence $a_1 \sqsupset a_2 \sqsupset \cdots \sqsupset a_n \sqsupset \cdots$

derings. In fact, many termination tools (AProVE [GST06], C*i*ME 2.03 [CMMU03], MU-TERM [Luc04], TTT [HM07],...) use polynomials as a principal ingredient to achieve termination proofs. In this setting, symbols f of a signature $\mathcal{F}$ are given polynomial functions $[f]$ satisfying appropriate conditions. Constraints $s \succsim t$ and $s \sqsupset t$ are treated as polynomial constraints $P_{s,t} \geq 0$ and $P_{s,t} > 0$, respectively, where $P_{s,t} = [s] - [t]$ is the polynomial obtained from terms s and t by interpreting them as polynomials $[s]$ and $[t]$ whose parameters are the variables occurring in the terms. Contejean et al. have investigated how to mechanize proofs of termination by using polynomials with non-negative integer coefficients [CMTU06], i.e., $[f] \in \mathbb{N}[X_1, \ldots, X_k]$ for each k-ary symbol f. In general, polynomials $P_{s,t} \in \mathbb{Z}[X_1, \ldots, X_n]$ contain integer coefficients. Thus, the corresponding constraints are treated as Diophantine inequalities whose satisfaction is, in general, undecidable [Mat70].

The use of polynomials with *real coefficients* in proofs of polynomial termination of rewriting (no dependency pairs were considered at this time) was proposed by Dershowitz [Der79]. Since the set of real numbers $\mathbb{R}$ furnished with the usual ordering $>_{\mathbb{R}}$ is not well-founded, a subterm property (i.e., $[f](x_1, \ldots, x_i, \ldots, x_k) >_{\mathbb{R}} x_i$ for all k-ary symbols f, arguments i, $1 \leq i \leq k$, and $x_1, \ldots, x_k \in \mathbb{R}$) is explicitly required to ensure well-foundedness. In fact, polynomial interpretations with real or just rational coefficients (i.e., $[f] \in \mathbb{R}[X_1, \ldots, X_n]$ or $[f] \in \mathbb{Q}[X_1, \ldots, X_n]$) and subsets A of $\mathbb{R}$ or $\mathbb{Q}$ as interpretation domains have been recently proved strictly more powerful than polynomial interpretations with integer coefficients in proofs of polynomial termination of rewriting [Luc06]. Following this work, in this thesis, we introduce several conditions for determining the necessity of using rational coefficients in polynomials which are eventually used in termination proofs. In other words, sufficient conditions which state that the termination proof will fail if one uses natural instead of rational interpretations. Dershowitz noticed that, according to Tarski [Tar51], the satisfaction of the corresponding polynomial constraints from polynomial interpretations over the reals then becomes decidable. The termination problem can be encoded as an instance of the general decision problem for the first-order theory of the real closed fields investigated by Tarski. This was the main argument of Dershowitz for proposing the use of polynomials over the reals in proofs of termination. The complexity of Tarski's decision algorithm, however, is not elementary recursive; therefore it was considered not so amenable for inclusion in termination tools.

In practice, polynomial constraints coming from termination problems are solved by using standard algorithms and techniques coming from *Constraint Programming* [RVW06]. Despite of the generality of these constraints, nonlinear constraints seems not to have been studied deeply. The main problem is to find efficient and suitable methods for dealing with the *generated set of constraints*. Up to now, several approaches have been considered for dealing with polynomial constraints in termination proofs. The main techniques are based on *SAT* (Boolean Satisfiability [FGMS$^+$07, FNOG$^+$08]), *CSP* (Constraint Satisfaction Problem [CMTU06, Luc05, Luc07]), *LP* (Linear Programming [Ste94]), and Real Algebraic Geometry [Gie95, Rou91]. Along this thesis, we study in depth some of these approaches and we present some advances in each approach.

1.2 Structure of the thesis

Here is an outline of this thesis. In Chapter 2 the most important concepts and setting from Term Rewriting Systems are reviewed.

In Chapter 3, we describe the basis of proving termination and we study in depth the process of proving termination by polynomial interpretations. We also discuss how to generate polynomial (quasi) orderings by means of polynomial interpretations over the natural, and real numbers and discuss their practical use.

In Chapter 4, we show the limitations of polynomial interpretations over the naturals with respect to the polynomial interpretations over the rationals. We investigate sufficient conditions which are amenable for detecting these situations automatically, on the basis of the structure of the rules in the TRS. This improves the research in [Luc06], where 'ad-hoc' TRSs were introduced to show the limitations of polynomial interpretations over the naturals. These results have been published in [FNOG$^+$08].

Chapter 5 presents the techniques considered in practice for dealing with polynomial constraints coming from termination problems. This chapter is divided in four sections. Section 5.1 focuses on SAT-based techniques, where we propose an efficient SAT encoding for polynomial constraints over $\{0, 1\}$. Next, in Section 5.2, we show how to model our problem as a CSP and, then, we present our contribution to the development of constraint solving algorithms over small domains of rational numbers. Section 5.3 discusses possible combinations of Linear Programming and CSP techniques which can be appropriate for their implementation as part of termination tools. Finally, in Section 5.4, we evaluate the algorithms from Real Algebraic Geometry, especially *quantifier elimination* and we propose techniques that could be applied in termination of Term Rewriting Systems. The results in this Chapter have been partially published in [LN08, LMNS08].

In Chapter 6, we present MULTISOLVER, a prototype of symbolic constraint solving system (see Sections 6.1 and 6.2) and, in Section 6.3, we empirically evaluate the performance of the algorithms presented in Chapter 5 with state-of-the-art solvers using two types of problems.

Finally, in Chapter 7, we give a summary and the conclusions of this thesis and present subjects for further research.

Chapter 2

Term Rewriting Systems

In this chapter, we give a description of the notation and setting important in this thesis. To begin with, we present an introduction to term rewriting. The concepts introduced in this chapter have been taken from [BN98, Ohl02].

2.1 Terms, substitutions and relation properties

A *term rewriting system* consists of rules which are used to transform or rewrite terms. So we first need to formalize the concept of term. Terms are defined over a signature.

Definition 1 (Signature) *A signature $\mathcal{F}$ is a set of function symbols, where each $f \in \mathcal{F}$ is associated with a non-negative integer $ar(f)$, the arity of f (i.e., the number of arguments f can take). For $n \geq 0$, we denote the set of all n-ary elements of $\mathcal{F}$ by $\mathcal{F}^{(n)}$. The elements of $\mathcal{F}^{(0)}$ are also called symbols.*

Terms are string of symbols from an alphabet, consisting of the signature and a set of variables.

Definition 2 (Term) *Let $\mathcal{F}$ be a signature and $\mathcal{X}$ be a set of variables such that $\mathcal{F} \cap \mathcal{X} = \emptyset$. The set $\mathcal{T}(\mathcal{F}, \mathcal{X})$ of all terms defined over $\mathcal{F}$ and $\mathcal{X}$ is inductively defined as the least set satisfying the following:*

- *$\mathcal{X} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$.*
- *If $f \in \mathcal{F}^{(k)}$ and $t_1, ..., t_k \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, then $f(t_1, ..., t_k) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.*

By this notation it is understood (case k=0) that a symbol $c() \in \mathcal{F}^{(0)}$ is in $\mathcal{T}(\mathcal{F}, \mathcal{X})$. We call the term c a constant and we usually write c instead of $c()$.

The terms t_i are often called the immediate subterm of the term $t = f(t_1, \dots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ (or the arguments of the function call represented by $f(t_1, \dots, t_n)$), and the symbol f is called the *head* or *root* symbol; we often write $root(t) = f$ in this case.

For a term t, $\mathcal{V}ar(t)$ denotes the set of all variables occurring in t. If $\mathcal{V}ar(t) = \emptyset$, i.e. t contains no variables, then we say that t is a *ground term*. We can extend the concept of $\mathcal{V}ar(t)$ from terms to sets of terms, if $T \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$, then $\mathcal{V}ar(T) = \bigcup_{t \in T} \mathcal{V}ar(t)$. The set of ground terms is denoted by $\mathcal{T}(\mathcal{F})$ instead of $\mathcal{T}(\mathcal{F}, \varnothing)$. A term t is *linear* if each variable appears at most once in t.

Example 1 *Consider the signature $\mathcal{F} = \{minus, 0\}$ where minus is a binary symbol ($ar(minus) = 2$) and 0 is a constant. Then, $minus(x, 0)$ is a term and $minus(0, 0)$ is a ground term, whereas minus and $0(x)$ are not terms. The term $minus(x, x)$ is not linear.*

Definition 3 (Context) *Let $\Box$ be a new symbol which does not occur in $\mathcal{F} \cup \mathcal{X}$. A context $C[\ ]$ is a term $t \in \mathcal{T}(\mathcal{F} \cup \{\Box\}, \mathcal{X})$ and can be seen as a term with "holes", represented by $\Box$, in it. Particularly, $C[\ ]$ denotes a context with a single hole and $C[\ , \dots,\ ]$ stands for a term with several holes. $C[t_1, ..., t_n]$ is the result of replacing all occurrences of $\Box$ in $C[\ , \dots,\ ]$ by $t_1, ..., t_n$ from left to right.*

Definition 4 (Subterm) *A term s is a subterm (superterm) of term t, if there exists a context $C[\ ]$ such that $s = C[t]$ ($t = C[s]$). We usually write $s \trianglerighteq t$ ($t \trianglerighteq s$) in order to denote that t is a subterm (superterm) of s.*

Example 2 *Consider the following signature $\mathcal{F} = \{minus, quot, 0, s\}$, where 0 is a constant symbol, s is an 1-ary symbols, and minus and quot are binary symbols in prefix notation. Let $s(quot(minus(x, y), s(y)))$ be a term. Then we have the following:*

- $s(quot(minus(x, y), s(y))) \ \triangleright\ minus(x, y)$
- $s(quot(minus(x, y), s(y))) \ \triangleright\ s(y)$
- $s(quot(minus(x, y), s(y))) \ \not\trianglerighteq\ s(quot(x, y))$

The structure of a term can be represented as a labeled tree, where the function symbols are nodes and the arrows point to the arguments of the function (and variables and constants are represented as leaf nodes). Nodes in a term tree are usually called *positions*. This terminology is then transferred to the corresponding term, so that we also speak of positions in terms. By using a standard numbering of the nodes of the tree by strings of indices, we can refer to positions in a term. Let us define formally a position.

Definition 5 (Position) *A position p is a string of positive integers $p \in \mathcal{P}os = \{\Lambda\} \cup \{i.q \mid i \in \mathbb{N}^+ \wedge q \in Pos\}$, where Λ denotes the empty string (root position).*

The length of a position p is denoted by $|p|$. The set of all positions of a term t is defined as follows:

$$\mathcal{P}os(t) = \begin{cases} \{\Lambda\} & \textit{if } t \in \mathcal{X} \\ \{\Lambda\} \cup \bigcup_{1 \leq i \leq k} i.Pos(t_i) & \textit{if } t = f(t_1, ..., t_k) \end{cases}$$

The concept of position is useful in order to define notions like subterm and replacing position. We use the following notation related to the concept of position:

- For $p \in \mathcal{P}os(s)$, the *subterm of s at position p*, denoted by $s|_p$, is defined by induction on the length of p:

$$t|_\Lambda = t$$
$$f(t_1, ..., t_k)|_{i.p} = t_i|_p \quad \text{where } 1 \leq i \leq k$$

- The depth of a subterm $s = t|_p$ of t is the length of the position p: $|p|$.
- For $p \in \mathcal{P}os(s)$, we denote by $t[s]_p$ the term that is obtained from s by replacing the subterm at position p by t, i.e.:

$$t[s]_\Lambda = s$$
$$f(t_1, ..., t_k)[s]_{i.p} = f(t_1, ..., t_i[s]_p, ..., t_k) \quad \text{where } 1 \leq i \leq k$$

- The function root : $(\mathcal{T}(\mathcal{F}, \mathcal{X}) - \{\mathcal{X}\}) \to \mathcal{F}$ maps a non-variable term $t = f(t_1, \ldots, t_n)$ to the symbol f at its root position Λ. We then say that f is the root symbol of t.
- $\mathcal{P}os_\mathcal{F}(t)$ denotes the set of position of non-variable symbol of t, that is:

$$\mathcal{P}os_\mathcal{F}(t) = \{p \in \mathcal{P}os(t) \mid root(t|_p) \in \mathcal{F}\}$$

Example 3 *Following the signature defined in Example 2 and considering the term*

$$t = s(quot(minus(x, y), s(y))),$$

the root symbol of t is $root(t) = s$. The labeled tree representing t is shown at figure 2.1. Then, using this representation it is easy to find all positions in t, $Pos(t) = \{\Lambda, 1, 1.1, 1.2, 1.1.1, 1.1.2, 1.2.1\}$. Moreover, there are two occurrences of the variable y in t at the 1.1.2 and 1.2.1 positions.

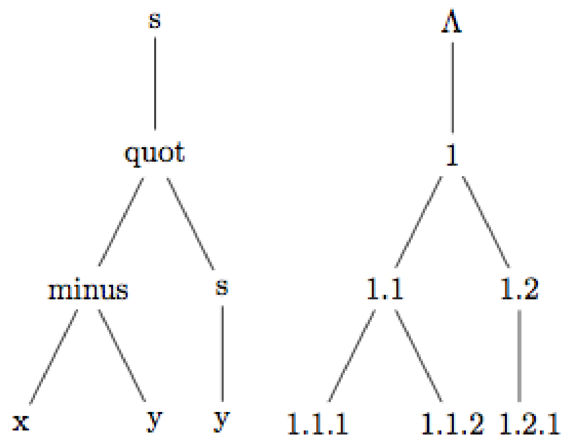


Figure 2.1: Positions of term $s(quot(minus(x, y), s(y)))$

Definition 6 (Substitution) *A substitution is a function $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $\sigma(x) \neq x$ for only finitely many xs. The (finite) set of variables that σ does not map to themselves is call the domain of σ : $\mathcal{D}om(\sigma) = \{x \in \mathcal{X} \mid \sigma(x) \neq x\}$. We denote the set of all substitutions over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ by $\mathcal{S}ub(\mathcal{T}(\mathcal{F}, \mathcal{X}))$.*

A substitution σ is often represented by the set $\sigma = \{x_1 \mapsto t_1, ..., x_n \mapsto t_n\}$, where the domain variables $\{x_1, ..., x_n\} \in \mathcal{D}om(\sigma)$ are linked with the corresponding term $(x_i \mapsto t_i)$. We denote the empty substitution by the greek symbol ϵ (i.e., $\epsilon(x) = x$ for all variable x). The function σ is homomorphically extended to a mapping $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}, \mathcal{X})$, so that $\sigma(f(t_1, ..., t_k)) = f(\sigma(t_1), ..., \sigma(t_k))$.

Let s, t be $\mathcal{T}(\mathcal{F}, \mathcal{X})$, we say that s *matches* t, if there exists a substitution σ such that $t = \sigma(s)$. The substitution σ is called a *matching substitution*.

Let s, t be $\mathcal{T}(\mathcal{F}, \mathcal{X})$, we say that s *unifies* t, if there exists a substitution σ such that $\sigma(t) = \sigma(s)$. We call *unifier* of t and s to the substitution σ.

Finally, we present several definition concerning properties of relations between terms.

Definition 7 (Monotonicity) *Let R be a relation over $\mathcal{T}(\mathcal{F}, \mathcal{X})$. We call R monotonic or compatible with $\mathcal{F}$-operations, if for all $s_1, s_2 \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, all $k > 0$, and all $f \in \mathcal{F}^{(k)}$, $s_1 \; R \; s_2$ implies*

$$f(t_1, ..., t_{i-1}, s_1, t_{i+1}, ..., t_k) \; R \; f(t_1, ..., t_{i-1}, s_2, t_{i+1}, ..., t_k)$$

for all i, $1 \leq i \leq k$, and all $t_1, ..., t_{i-1}, t_{i+1}, ..., t_k \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

Definition 8 (Stability) *Let R be a relation over $\mathcal{T}(\mathcal{F},\mathcal{X})$. We call R stable or closed under subtitutions, if for all $s_1, s_2 \in \mathcal{T}(\mathcal{F},\mathcal{X})$ and all substitutions $\sigma \in \mathcal{S}ub(\mathcal{T}(\mathcal{F},\mathcal{X}))$, $s_1 \; R \; s_2$ implies $\sigma(s_1) \; R \; \sigma(s_2)$.*

2.2 Term Rewriting Systems

Term rewriting systems are defined as follows:

Definition 9 (Term rewriting systems) *A Term Rewriting System (TRS) is a pair $(\mathcal{F}, R)$, where $\mathcal{F}$ is a signature and R is a set $R \subseteq \mathcal{T}(\mathcal{F},\mathcal{X}) \times \mathcal{T}(\mathcal{F},\mathcal{X})$ of rewriting rules. A TRS $\mathcal{R} = (\mathcal{F}, R)$ is finite if both sets $\mathcal{F}$ and R are finite.*

Each rewriting rule $l \to r \in \mathcal{R}$ has to satisfy the following:

- $l \notin \mathcal{X}$, i.e., the left hand side of the rule l is not a variable, and
- $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$, i.e., all variables in the right hand side (rhs) r of the rule also occur in the left hand side (lhs) l.

Here we only consider finite term rewriting systems.

For a TRS $\mathcal{R}$, the *(term) rewriting relation* $\to_{\mathcal{R}}$ over terms in $\mathcal{T}(\mathcal{F},\mathcal{X})$ is defined by $s \to_{\mathcal{R}} t$ if there is a rewriting rule $l \to r$ in R, a substitution σ, and a context $C[\,]$ such that $s = C[\sigma(l)]$ and $t = C[\sigma(r)]$. $s \to_{\mathcal{R}} t$ denotes a *rewriting step* or *reduction step.*

A *redex* (reducible expression) is an instance of the lhs of a rewriting rule. Moreover, it is said that s rewrites to t by means of contracting a redex $\sigma(l)$, and that $\sigma(r)$ is the *contractum.*

The signature $\mathcal{F}$ of a TRS $\mathcal{R}$ is divided into two disjoint sets $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$:

- The set of *defines symbols* $\mathcal{D} = \{root(l) \mid l \to r \in R\}$, and
- The set of *constructor symbols* $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$.

Example 4 *Consider the following TRS $\mathcal{R}$ from [AG00, Example 1]:*

$$\begin{array}{rcl}
minus(x,0) & \to & x \\
minus(s(x),s(y)) & \to & minus(x,y) \\
quot(0,s(y)) & \to & 0 \\
quot(s(x),s(y)) & \to & s(quot(minus(x,y),s(y)))
\end{array}$$

Here, $\mathcal{D} = \{minus, quot\}$ is the set of defined symbols, whereas $\mathcal{C} = \{0, s\}$ is the set of constructor symbols.

Chapter 3

Termination of Term Rewriting Systems

In this chapter, we describe some methods for proving termination of term rewriting systems. The concepts introduced in this chapter have been taken from [BN98, Ohl02, CMTU06].

3.1 Termination and orderings

A central notion in termination analysis is that of *well-foundedness*, which we introduce in the following definition.

Definition 10 (Well-Founded Relation) *Let $\mathcal{A}$ be a set. A relation $\rightarrow \subseteq \mathcal{A} \times \mathcal{A}$ is said to be well-founded (also noetherian or terminating) if there is no infinite reduction sequence $a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow \cdots$ where $a_1, a_2, \ldots$ are elements of $\mathcal{A}$.*

Definition 11 (Quasi-ordering) *Let A be a set. A quasi-ordering is a transitive and reflexive relation $\succeq$ over A.*

Definition 12 (Ordering) *Let A be a set. A (strict) ordering is a transitive and irreflexive relation $>$ over A.*

Recall that the termination problem is undecidable. Nevertheless, it is often necessary to prove that a particular system terminates, and it is possible to envisage methods to address this task. The basic idea for proving termination is to employ a well-founded ordering as shown the following definition [MN70].

Definition 13 (Termination of TRS) *Let $\mathcal{R}$ be a finite term rewriting system, and $>$ be a well-founded (strict) order on $\mathcal{T}(\mathcal{F}, \mathcal{X})$. Then, $\mathcal{R}$ is terminating if, for all terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $s \rightarrow_{\mathcal{R}} t$ implies $s > t$.*

By instead of deciding $s > t$ for the (infinitely many) pairs s, t with $s \rightarrow_{\mathcal{R}} t$, we would rather like to check whether $l > r$ for the (finitely many) rules $l \rightarrow r \in \mathcal{R}$. However, in order to guarantee that this approach is correct, the order $>$ must satisfy some additional properties. This motivates the definition of *reduction orders*.

Definition 14 (Rewrite quasi-order) *Let $\mathcal{F}$ be a signature and $\mathcal{X}$ be a (countably infinite) set of variables. A rewrite quasi-order $>$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is a quasi-ordering $\succeq$ stable and (weakly) monotonic.*

Definition 15 (Rewrite order) *Let $\mathcal{F}$ be a signature and $\mathcal{X}$ be a (countably infinite) set of variables. A strict order $>$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is called a rewrite order if it is stable and monotonic.*

Definition 16 (Reduction Ordering) *A reduction order is a well-founded rewrite order.*

The name "rewrite order" is motivated by the fact that the relations $\rightarrow_{\mathcal{R}}^{*}$ and $\rightarrow_{\mathcal{R}}^{+}$ for a term rewriting system $\mathcal{R}$ are rewrite quasi-orderings and rewrite orderings, respectively.

We say that a relation $>$ is compatible with a rule $l \rightarrow r$ if $l > r$ holds. Then, by using reduction orderings one can indeed show termination of term rewriting systems:

Theorem 1 (Termination with reduction orderings) *A term rewriting system $\mathcal{R}$ terminates if and only if there exists a reduction order $>$ such that satisfies $l > r$ for all $l \rightarrow r \in \mathcal{R}$.*

Given a term rewriting system $\mathcal{R}$, a *direct termination proof* is a termination proof in which a concrete reduction ordering which is compatible with the rules of $\mathcal{R}$ is given.

3.2 The interpretation method

In order to construct a reduction order on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, we can utilize the interpretation method. This method does not look directly at the terms over $\mathcal{F}$. Instead it considers their *interpretation* in an $\mathcal{F}$-algebra that is equipped with a well-founded order.

Definition 17 (Algebra) *Let $\mathcal{F}$ be a signature. An $\mathcal{F}$-algebra $\mathcal{A} = (A, \mathcal{F}_A)$ consists of a set A, called the algebra domain or carrier set, and a set $\mathcal{F}_A$ of mappings $f_A : A^k \rightarrow A$ for all $f \in \mathcal{F}^{(k)}$ that interpret the function symbols $f \in \mathcal{F}$.*

In the following, we assume that the carrier of an $\mathcal{F}$-algebra is not empty.

A special algebra is a *term algebra* $\mathcal{T}(\mathcal{F},\mathcal{X})$ consisting of the carrier $\mathcal{T}(\mathcal{F},\mathcal{X})$ and the operations $f_{\mathcal{T}(\mathcal{F},\mathcal{X})}(t_1,...,t_k) = f(t_1,...,t_k)$. The ground-term $\mathcal{F}$-algebra is defined analogously.

Given an $\mathcal{F}$-algebra, we assign to every ground term t its interpretation $[t]_{\mathcal{A}}$ in $\mathcal{A}$.

Definition 18 (Ground-term interpretation) *Let $\mathcal{A}$ be an $\mathcal{F}$-algebra. The mapping $[\cdot]_{\mathcal{A}}$ from $\mathcal{T}(\mathcal{F})$ to A is defined inductively by $[f(t_1,...,t_k)]_{\mathcal{A}} = f_{\mathcal{A}}([t_1]_{\mathcal{A}},...,[t_k]_{\mathcal{A}})$. In particular, if t is a constant, then $[t]_{\mathcal{A}} = t_{\mathcal{A}}$.*

If a term t contains variables, then its interpretation $[\cdot]_{\mathcal{A}}$ in $\mathcal{A}$ depends on the assignment of values in A to the variables in t.

Definition 19 (Valuation) *Let $\mathcal{A} = (A,\mathcal{F}_A)$ be an $\mathcal{F}$-algebra. A mapping $\alpha : \mathcal{X} \to A$ is value called an assignment. For every assignment α, a valuation mapping $[\alpha]_{\mathcal{A}} : \mathcal{T}(\mathcal{F},\mathcal{X}) \to A$ is defined inductively by*

$$[\alpha]_{\mathcal{A}}(t) = \begin{cases} \alpha(t) & \textit{if } t \in \mathcal{X} \\ f_{\mathcal{A}}([\alpha]_{\mathcal{A}}(t_1),...,[\alpha]_{\mathcal{A}}(t_k)) & \textit{if } t = f(t_1,...,t_k) \end{cases}$$

where $f \in \mathcal{F}$ and $t_1,\ldots,t_n \in \mathcal{T}(\mathcal{F},\mathcal{X})$.

For ground terms, we would say that a term s is bigger than a term t if the interpretation of s in $\mathcal{A}$ is bigger than the interpretation of t in $\mathcal{A}$. For arbitrary terms (possibly with variables), the *interpretation* of a term with variables only makes sense for a given valuation of the variables. Thus we have the following definition.

Definition 20 (Ordering induced by an algebra) *Let $\mathcal{F}$ be a signature, $\mathcal{A} = (A,\mathcal{F}_A)$ an $\mathcal{F}$-algebra, and $>$ a well-founded (strict) ordering on its carrier set A. The binary relation $>_{\mathcal{A}}$ on $\mathcal{T}(\mathcal{F},\mathcal{X})$ induced by $\mathcal{A}$ is defined as follows*

$$s >_{\mathcal{A}} t \textit{ if } [\alpha]_{\mathcal{A}}(s) > [\alpha]_{\mathcal{A}}(t) \textit{ for all value assignment } [\alpha] : \alpha \to A$$

Stability for the ordering $>_{\mathcal{A}}$ on terms is a consequence of the fact that we consider all possible valuations of the variables in its definition. To satisfy compatibility with function symbols operations (monotonicity), we must require that all interpretations of function symbols are monotone.

Definition 21 (Monotone functions) *Let R be a valuation on a set A. A function $F : A^n \to A$ is called monotone (w.r.t. R) if*

$$a \; R \; b \quad \Rightarrow \quad F(a_1, ..., a_{i-1}, a, a_{i+1}, ..., a_n) \; R \; F(a_1, ..., a_{i-1}, b, a_{i+1}, ..., a_n)$$

for all i, $1 \leq i \leq n$, and all $a, b, a_1, ..., a_{i-1}, a_{i+1}, ..., a_n \in A$.

Definition 22 (Well-founded $\mathcal{F}$-algebra) *Let $\mathcal{F}$ be a signature, $(A, \mathcal{F}_A)$ an $\mathcal{F}$-algebra and $>_A$ a well-founded (strict) ordering on A. Then, a triple $(A, \mathcal{F}_A, >_A)$ is an well-founded (ordered) $\mathcal{F}$-algebra.*

By means of the next theorem, it is possible to use algebras as a tool for proving termination.

Theorem 2 *Let $\mathcal{F}$ be a signature, and $\mathcal{A} = (A, \mathcal{F}_A, >_A)$ an well-founded $\mathcal{F}$-algebra. If the interpretations f_A of all function symbols $f \in \mathcal{F}$ are monotone w.r.t. $>_A$, then we say that $\mathcal{A}$ is a well-founded monotone $\mathcal{F}$-algebra and the induced ordering $>_{\mathcal{A}}$ is a reduction order on $\mathcal{T}(\mathcal{F}, \mathcal{X})$.*

We say that a TRS $(\mathcal{F}, R)$ is compatible with a well-founded monotone $\mathcal{F}$-algebra $\mathcal{A} = (A, \mathcal{F}_A, >_A)$ if $(\mathcal{F}, R)$ is compatible with the reduction ordering $>_{\mathcal{A}}$ induced by $\mathcal{A}$.

Corollary 1 *A TRS $(\mathcal{F}, R)$ is terminating if it is compatible with a well-founded monotone $\mathcal{F}$-algebra $\mathcal{A}$.*

3.3 Polynomial orderings over the naturals

In this section, we introduce a particular class of reduction orders of the form $>_{\mathcal{A}}$, in which the carrier set is the set $\mathbb{N}$ of natural numbers and function symbols are interpreted as polynomial functions over the naturals (polynomial interpretations over the naturals).

Polynomial interpretations over the naturals are special well-founded algebras. Lankford [Lan75, Lan79] first studied them in their final form but the ideas on which the polynomial interpretation method is based can be tracked back to the work of Manna and Ness [MN70].

The basic idea is to employ algebras based on polynomial interpretations which *interpret the terms as polynomials.* Then, the comparison between terms is done according to the Definition 20. In other words, a term s is greater than a term t w.r.t. a polynomial interpretation, if the polynomial $[s]$ interpreting to s is greater than the polynomial $[t]$ interpreting to t. Recall that given two polynomials P and Q on variables $x_1, ..., x_n$, we say that $P > Q$ over a subset A of the numerical domain of the variables $x_1, ..., x_n$, if $\forall x_1, ..., x_k \in A$, $P(x_1, ..., x_k) > Q(x_1, ..., x_k)$.

Definition 23 (Polynomial Algebra over the naturals) *Let $\mathcal{F}$ be a signature. A polynomial algebra over the naturals is an $\mathcal{F}$-algebra $\mathcal{A} = (A, \mathcal{F}_A)$ with the carrier set $A \subseteq \mathbb{N}$ such that* *for all function symbol $f \in \mathcal{F}^{(n)}$, the interpretation function $[f]$ is a polynomial $[f](x_1, \dots, x_n) \in \mathbb{N}[x_1, \dots, x_n]$ satisfying the algebraicity property: for all $a_1, ..., a_n \in A$, $[f](a_1, ..., a_n) \in A$.*

Given a polynomial algebra, the polynomial $[t]$ associated to the term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is obtained as follows:

$$[t] = \begin{cases} x & \text{if } t = x \in \mathcal{X} \\ [f]([t_1], ..., [t_k]) & \text{if } t = f(t_1, ..., t_k) \end{cases}$$

Note that x above is used as a syntactic variable $x \in \mathcal{X}$ and as a *numeric* variable ranging on $\mathbb{N}$ (or better, as the identity polynomial).

If $\mathcal{A} = (A, \mathcal{F}_A, >_A)$ is a monotone polynomial interpretation, then this interpretation is a monotone (well-founded) algebra and the induced ordering $>_\mathcal{A}$ is a reduction ordering on $\mathcal{T}(\mathcal{F}, \mathcal{X})$. Such a reduction ordering $>_\mathcal{A}$ is called a *polynomial ordering.* This motivates the following definition:

Definition 24 (Polynomial Termination) *A TRS $\mathcal{R}$ is called polynomially terminating if it is compatible with a polynomial ordering.*

Therefore, in order to show (polynomial) termination of a given TRS, one must find monotone polynomials as interpretations of function symbols. The following proposition [Ohl02] states a sufficient criterion for that:

Proposition 1 *Let $\mathcal{A} = (A, \mathcal{F}_A, >_A)$ be a ordered polynomial interpretation such that $A \subseteq \mathbb{N}_{>0}$. Then, $\mathcal{A}$ is a monotone polynomial interpretation if and only if every function symbol $f \in \mathcal{F}^{(n)}$, $n > 0$, is interpreted by a polynomial $[f](x_1, \dots, x_n) \in \mathbb{N}[x_1, \dots, x_n]$ that for every $i \in \{1, ..., n\}$ contains a monomial $m \cdot x_1^{k_1} \dots x_i^{k_i} \dots x_n^{k_n}$ with $m, k_i \in \mathbb{N}_{>0}$.*

Example 5 *Considering the TRS $\mathcal{R}$ containing the following rules:*

$$f(g(x, y)) \to g(f(x), f(y))$$
$$g(g(x, y), z) \to g(x, g(y, z))$$

We define a polynomial interpretation $\mathcal{A}$ as follows:

The carrier set is $\mathbb{N}_{>0}$ and the interpretation functions are the monotone polynomials

$$[f](x) = x^2$$
$$[g](x, y) = xy + x$$

In order to show that $\mathcal{R}$ is (polynomially) terminating, one has to prove that $l >_{\mathcal{A}} r$ for every $l \to r \in \mathcal{R}$, that is,

$$f(g(x,y)) >_{\mathcal{A}} g(f(x), f(y))$$
$$g(g(x,y),z) >_{\mathcal{A}} g(x, g(y,z))$$

These inequalities are indeed satisfied for all $x, y, z \in \mathbb{N}^+$

$$[f]([g](x,y)) = x^2y^2 + 2x^2y + x^2 > x^2y^2 + x^2 = [g]([f](x), [f](y)))$$
$$[g]([g](x,y),z) = xyz + xz + xy + x > xyz + xz + x = [g](x, [g](y,z))$$

Given a polynomial ordering $>_{\mathcal{A}}$, it would be desirable to have a procedure that for terms $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ decides whether $l >_{\mathcal{A}} r$ holds. Consequently, it would be decidable whether a finite TRS $\mathcal{R}$ is compatible with the ordering $>_{\mathcal{A}}$. However, according to Matjasevich [Mat70], who proved the undecidability of the satisfaction problem for Diophantine constraints (Hilbert's 10th Problem), such a decision procedure does not exist.

Another disadvantage of polynomial order is that they cannot show termination of rewriting systems with "very long" reduction chains [HL89]:

Proposition 2 *Assume that termination of the finite term rewriting $\mathcal{R}$ can be shown with a polynomial order. Then there exists a constant $c > 0$ in $\mathbb{R}$ such that for all terms t the length of every $\mathcal{R}$-derivation sequence starting with t is bounded by $2^{2^{c|t|}}$.*

Thus, terminating term rewriting systems that have reduction chains whose length exceeds this doubly-exponential bound cannot be shown terminating by means of a polynomial order.

3.4 Polynomial orderings over the reals

In previous sections, we have considered polynomial interpretations over the naturals, but polynomial interpretations *over the reals* are also possible. Polynomials over the reals were proposed by Dershowitz [Der79] as an alternative to Lankford's polynomials over the naturals [Lan79]. Dershowitz [Der79] noticed that according to Tarski [Tar51], the satisfaction of polynomial constraints over the reals then becomes decidable. Recall that the satisfaction of polynomial constraints over the naturals is not decidable. In order to guarantee the *well-foundedness* of the induced ordering, which does not follow from that the ordering $>_{\mathbb{R}}$ over the reals (because it is not well-founded), Dershowitz requires a *subterm property.* i.e., $[f](x_1, \ldots, x_i, \ldots, x_k) >_{\mathbb{R}} x_i$ for all k-ary symbols f, arguments i, $1 \leq i \leq k$, and $x_1, \ldots, x_k \in \mathbb{R}$.

In [Luc05], Lucas presented a framework for using polynomial interpretations over the reals without requiring the subterm property. The idea is to consider the following well-founded ordering:

$$\text{Given } \delta \in \mathbb{R}_{>0},\ \forall x, y \in \mathbb{R},\ x >_{\mathbb{R},\delta} y \text{ if } x - y \geq_{\mathbb{R}} \delta$$

Definition 25 (Ordering induced by an algebra) *Let $\mathcal{F}$ be a signature, $A \subseteq \mathbb{R}$, and $\mathcal{A} = (A, \mathcal{F}_A)$ be an $\mathcal{F}$-algebra. Given $\delta \in \mathbb{R}_{>0}$ we define the relation $>_\delta$ on terms by*

$$t >_\delta s \textit{ if for all assignments } \alpha : \mathcal{X} \to A, [\alpha](t) - [\alpha](s) \geq_{\mathbb{R}} \delta$$

Given $m \in \mathbb{R}$ and $A \subseteq \mathbb{R}$, we say that $f_{\mathcal{A}} : A^k \to A$ is a m-bounded if $f_{\mathcal{A}}(x_1, \ldots, x_n) \geq m$ for all $x_1, \ldots, x_k \in A$. If there exists an $m \in \mathbb{R}$ such that $f_{\mathcal{A}}$ is m-bounded for all $f \in \mathcal{F}$, then we say that $\mathcal{A} = (A, f_A)$ is m-bounded. Thus, we have the following:

Theorem 3 *Let $\mathcal{F}$ be a signature and $A \subseteq [m, +\infty)$ for some $m \in \mathbb{R}$, and $\mathcal{F}_A$ be a set of polynomial functions $f_{\mathcal{A}} : A^k \to A$ for each k-ary $f \in \mathcal{F}$. Then, for every $\delta > 0$, $>_\delta$ is a well-founded and stable (strict) ordering on $\mathcal{T}(\mathcal{F},\mathcal{X})$.*

In order to use an ordering $>_\delta$ (induced by an m-bounded algebra) for proving termination of rewriting, we have to further ensure that $>_\delta$ is monotonic. The following proposition provides a sufficient condition to ensure monotonicity of $>_\delta$ for an arbitrary δ.

Theorem 4 *Let $\delta > 0$, $P \in \mathbb{R}[X_1, \ldots, X_n]$ and $A = [0, +\infty)$. Then, P is $>_\delta$-monotone if $\frac{\partial P}{\partial X_i} \geq 1$ on A for all $1 \leq i \leq n$.*

The following proposition provides a basis to avoid the explicit specification of δ when checking compatibility of the ordering with a set of pairs of terms (e.g., the rules of a TRS).

Theorem 5 *Let $T \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}, \mathcal{X})$ be a finite set of pairs of terms and $\mathcal{A} = (A, \mathcal{F}_A, >_\delta)$ be an ordered $\mathcal{F}$-algebra over the reals. If for each $(t, s) \in T$, there is $\delta_{t,s} \in \mathbb{R}_{>0}$ such that $P_{t,s} = [t] - [s]$ is $\delta_{t,s}$-bounded in A, then $\delta = min(\{\delta_{t,s} | (t, s) \in T\})$ is positive and $t >_\delta s$ for all $(t, s) \in T$.*

The results in this section provide a general framework to prove termination by using orderings induced by algebras over the reals.

Example 6 *Let us consider the following TRS $\mathcal{R}$ coming from a conditional TRS:*

$$\begin{aligned} f(x) &\to u1(x,x) \\ u1(h(y),x) &\to u2(i(x),x,y) \\ u2(y,x,y) &\to g(y) \\ i(x) &\to a \\ u(d) &\to b \\ a &\to u(c) \\ c &\to d \end{aligned}$$

Then, we can prove termination of $\mathcal{R}$ using the following polynomial interpretation over $\mathbb{Q}$:

$$\begin{aligned} [f](x) &= 3x+1 \\ [h](x) &= x+3 \\ u[x] &= x+1/2 \\ [u1](x,y) &= x+2y \\ [i](x) &= x+3/2 \\ [u2](x,y,z) &= x+y+z+1 \\ [g](x) &= x \\ [d] &= 0 \\ [b] &= 0 \\ [a] &= 1 \\ [c] &= 1/3 \end{aligned}$$

Here, we can check that the ordering $>_\delta$ induced by the polynomial interpretation is monotonic by using the Theorem 4:

$$\begin{array}{ccccc} \frac{\partial[f]}{\partial x} \geq 3 & \frac{\partial[h]}{\partial x} \geq 1 & \frac{\partial[u]}{\partial x} \geq 1 & \frac{\partial[u1]}{\partial x} \geq 1 & \frac{\partial[u1]}{\partial y} \geq 2 \\ \frac{\partial[i]}{\partial x} \geq 1 & \frac{\partial[u2]}{\partial x} \geq 1 & \frac{\partial[u2]}{\partial y} \geq 1 & \frac{\partial[u2]}{\partial z} \geq 1 & \frac{\partial[g]}{\partial x} \geq 1 \end{array}$$

Now, we can obtain the 'hidden' δ which is implicitly used for ensuring that the polynomial *interpretation computed actually proves termination of $\mathcal{R}$. According to Theorem 5 (with $\alpha = 0$):*

$$\begin{aligned}
\delta_{f(x),u1(x,x)} &= 1\\
\delta_{u1(h(y),x),u2(i(x),x,y)} &= 3/2\\
\delta_{u2(y,x,y),g(y)} &= 1\\
\delta_{i(x),a} &= 1/2\\
\delta_{u(d),b} &= 1/2\\
\delta_{a,u(c)} &= 1/6\\
\delta_{c,d} &= 1/3
\end{aligned}$$

Therefore,

$$\delta = min(\{1, 3/2, 1, 1/2, 1/2, 1/6, 1/3\}) = 1/6$$

3.5 Dependency Pairs termination

The polynomial interpretation method is well-suited for automatically proving termination of term rewriting systems, because polynomial orderings can be automatically generated in an efficient way. Unfortunately, there are many terminating TRSs which cannot be proved terminating by using polynomial orderings, i.e., they are not polynomially terminating.

The *dependency pairs method* proposed by Arts and Giesl [AG00] improves this situation because this method relaxes the necessary conditions for proving termination. With the dependency pairs method we are able to use polynomials for proving termination of TRSs which cannot be proved terminating by using polynomial orders.

The main idea of the dependency pairs method is that in an infinite reduction sequence, some rules are applied infinitely often.

Roughly speaking, the application of this method provides a set of inequalities, each one of the form $s \gtrsim t$ or $s \sqsupset t$. An essential task in the practical use of this method is finding reduction pairs $(\gtrsim, \sqsupset)$ satisfying all constraint. The notion of *reduction pair* was introduced by Kusakari et al. [KNT99].

Definition 26 (Reduction pair) *A reduction pair $(\gtrsim, \sqsupset)$ consists of a rewrite quasi-order $\gtrsim$ and a stable and well-founded partial order $\sqsupset$ such that satisfies the following conditions of compatibility:*

$$\gtrsim \circ \sqsupset \subseteq \sqsupset \quad \textit{or} \quad \sqsupset \circ \gtrsim \subseteq \sqsupset$$

Note that $\sqsupset$ need not be monotonic.

Definition 27 (Dependency Pairs) *If $f(s_1, ..., s_n) \to C[g(t_1, ..., t_m)]$ is a rewrite rule of a TRS $\mathcal{R}$ and g is a defined symbol, then $F(s_1, ..., s_n) \to G(t_1, ..., t_m)$ is a dependency pair of $\mathcal{R}$. The set of all dependency pairs of $\mathcal{R}$ is called* $\mathsf{DP}(\mathcal{R})$.

Example 7 *Consider the TRS presented in Example 4. Then, we obtain the following dependency pairs:*

$$\begin{array}{rcl} MINUS(s(x), s(y)) & \to & MINUS(x, y) \\ QUOT(s(x), s(y)) & \to & QUOT(minus(x, y), s(y)) \\ QUOT(s(x), s(y)) & \to & MINUS(x, y) \end{array}$$

To trace the evolution of function calls, we examine special sequences of dependency pairs, called *chains.*

Definition 28 (Dependency Pair Chain) *Let $\mathcal{R}$ be a TRS. A sequence of dependency pairs $s_1 \to t_1, s_2 \to t_2, \ldots$ of $\mathcal{R}$ is an $\mathcal{R}$-chain if there exists a substitution σ such that $\sigma(t_j) \to_{\mathcal{R}}^* \sigma(s_{j+1})$ holds for every two consecutive pairs $s_j \to t_j$ and $s_{j+1} \to t_{j+1}$ in a sequence. Furthermore, if every $\sigma(t_j)$ is terminating, then the $\mathcal{R}$-chain is said to be minimal.*

Example 8 *According to the TRS presented in Example 7, there is a chain*

$$\begin{array}{c} MINUS(s(x_1), s(y_1)) \to MINUS(x_1, y_1) \\ MINUS(s(x_2), s(y_2)) \to MINUS(x_2, y_2) \\ MINUS(s(x_3), s(y_3)) \to MINUS(x_3, y_3) \end{array}$$

because with $\sigma = \{x_1 \mapsto s(s(x_3)), x_2 \mapsto s(x_3), y_1 \mapsto s(s(y_3)), y_2 \mapsto s(y_3)\}$, we have

$$\begin{array}{c} \sigma(MINUS(x_1, y_1)) \to_{\mathcal{R}}^* \sigma(MINUS(s(x_2), s(y_2))) \text{ and} \\ \sigma(MINUS(x_2, y_2)) \to_{\mathcal{R}}^* \sigma(MINUS(s(x_3), s(y_3))). \end{array}$$

The essence of the dependency pairs method is to *characterize* the termination of TRS as the absence of infinite dependency pair chains.

Theorem 6 *[AG00] A TRS $\mathcal{R}$ is terminating if only if there is no infinite $\mathcal{R}$-chain.*

According to the previous result, the termination of TRS is ensured by the absence of infinite $\mathcal{R}$-chains. But, how do we prove it? The following result can be used for automation of the process.

Theorem 7 *[AG00] A TRS $\mathcal{R}$ is terminating if only if there is a reduction pair $(\succsim, \sqsupset)$ such that*

- *$l \succsim r$ for every rule $l \to r$ in $\mathcal{R}$ and*
- *$u \sqsupset v$ for every dependency pair $u \to v$ in $\mathsf{DP}(\mathcal{R})$.*

Example 9 *According to the TRS $\mathcal{R}$ in Example 7, in order to show termination of $\mathcal{R}$ it suffices to find a reduction pair $(\succsim, \sqsupset)$ satisfying the constraints*

$$\begin{aligned}
minus(x, 0) &\succsim x \\
minus(s(x), s(y)) &\succsim minus(x, y) \\
quot(0, s(y)) &\succsim 0 \\
quot(s(x), s(y)) &\succsim s(quot(minus(x, y), s(y))) \\
MINUS(s(x), s(y)) &\sqsupset MINUS(x, y) \\
QUOT(s(x), s(y)) &\sqsupset QUOT(minus(x, y), s(y)) \\
QUOT(s(x), s(y)) &\sqsupset MINUS(x, y)
\end{aligned}$$

In fact, for finite systems we can do even better. This is because some dependency pairs cannot occur twice in any chain and hence they need not be taken into account in a proof that no infinite chain exists. In order to identify these insignificant dependency pairs, the notion of *dependency graph* has been introduced by Arts and Giesl [AG00].

Definition 29 (Dependency Graph) *Let $\mathcal{R}$ be a TRS. The dependency graph of $\mathcal{R}$ is the directed graph whose nodes are the dependency pairs of $\mathcal{R}$ and there is an arc from $s \to t$ to $u \to v$ if $s \to t, u \to v$ is a chain.*

Example 10 *The dependency graph for the system from Example 7 is given in figure 3.1.*

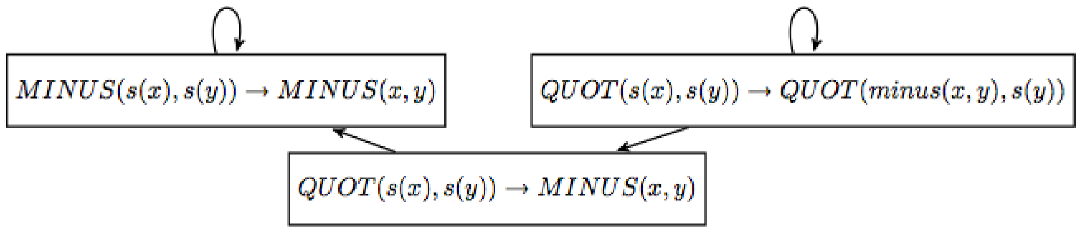

$MINUS(s(x), s(y)) \to MINUS(x, y)$
$QUOT(s(x), s(y)) \to QUOT(minus(x, y), s(y))$
$QUOT(s(x), s(y)) \to MINUS(x, y)$

Figure 3.1: Dependency Graph from Example 7

If a TRS is finite, then every infinite chain corresponds to a cycle in the dependency graph. In other words, those dependency pairs not contained in a cycle need not be taken into account in a termination proof. It should be pointed out that for finite TRS there is only a finite number of cycles because the number of dependency pairs is finite.

Definition 30 (Dependency Graph Cycle) *A nonempty set $\mathcal{P}$ of dependency pairs is called a cycle if, for any two dependency pairs $s \to t, u \to v \in \mathcal{P}$, there is a nonempty path from $s \to t$ to $u \to v$ and from $u \to v$ to $s \to t$ in the dependency graph that traverses dependency pairs only from $\mathcal{P}$.*

Theorem 8 *A TRS $\mathcal{R}$ is terminating if only if for each cycle $\mathcal{P}$ in the dependency graph of $\mathcal{R}$ there exists no infinite $\mathcal{R}$-chain of dependency pairs from $\mathcal{P}$.*

Example 11 *According to the dependency graph in Example 10, there are two cycles:*

$$\{MINUS(s(x), s(y)) \to MINUS(x, y)\} \text{ and}$$
$$\{QUOT(s(x), s(y)) \to QUOT(minus(x, y), s(y))\}$$

As a result of the Theorem 8, the dependency pair $QUOT(s(x), s(y)) \to MINUS(x, y)$ is irrelevant.

Similar to Theorem 7, Theorem 8 can be used to obtain a termination criterion that can be tested automatically.

Theorem 9 *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$. $\mathcal{R}$ is terminating if only if for every cycle $\mathcal{P}$ in the dependency graph of $\mathcal{R}$ there is a reduction pair $(\succsim, \sqsupset)$ such that*

- *$l \succsim r$ for every rule $l \to r$ in $\mathcal{R}$,*
- *$u \succsim v$ for every dependency pair $u \to v$ in $\mathcal{P}$, and*
- *$u \sqsupset v$ for at least one dependency pair $u \to v$ in $\mathcal{P}$.*

According to this Theorem, one may use different reduction pairs for different cycles in a termination proof. This is in contrast to Theorem 7, where one reduction ordering has to satisfy all the constraints simultaneously. Therefore, this method leads to modularized termination proofs.

For an automatic approach the usage of dependency graphs is impractical because it is in general undecidable whether two dependency pairs form a chain. However, in order to obtain a sound technique for termination proofs, we can safety use any approximation of the dependency graph that preserves all its cycles. In case we use a rough approximation, we might not be able to determine all the insignificant dependency pairs that we could find with a more accurate approximation.

In order to estimate which dependency pairs may occur consecutively in a chain, the estimated dependency graph has been introduced [AG00]. We consider the following notions:

- $\textsc{Cap}(t)$ results from replacing all subterms of t that have a defined symbol by different fresh variables, and
- $\textsc{Ren}(t)$ results from replacing all variables in t by different fresh variables.

Then, in order to determine if $u \rightarrow v$ can follow $s \rightarrow t$ in a chain, one checks whether $\textsc{Ren}(\textsc{Cap}(t))$ *unifies* with u. The function $\textsc{Ren}$ is needed to rename multiples occurrences of the same variable x in t since two different occurrences of $\sigma(x)$ in $\sigma(t)$ could reduce to different terms.

Definition 31 (Estimated dependency graph) *Let $\mathcal{R}$ be a TRS. The estimated dependency graph of $\mathcal{R}$, denoted by $\mathsf{EDG}(\mathcal{R})$, is the directed graph whose nodes are the dependency pairs of $\mathcal{R}$ and there is an arc from $s \rightarrow t$ to $u \rightarrow v$ if $\textsc{Ren}(\textsc{Cap}(t))$ and u are unifiable.*

Corollary 2 *A TRS $\mathcal{R}$ is terminating if only if for each cycle $\mathcal{P}$ in $\mathsf{EDG}(\mathcal{R})$ there is a reduction pair $(\succsim, \sqsupset)$ such that*

- *$l \succsim r$ for all rules $l \rightarrow r$ from $\mathcal{R}$,*
- *$u \succsim v$ for all dependency pairs $u \rightarrow v$ from $\mathcal{P}$, and*
- *$u \sqsupset v$ for at least one dependency pair $u \rightarrow v$ from $\mathcal{P}$.*

3.6 Reduction Pairs based on polynomial orderings

In previous sections, we have shown how to transform a termination problem into a set of inequalities. But, at this point, the question is how to deal with these constraints. The idea is to adapt the existing termination proving techniques to work on the set of constraints generated by the dependency pair method. In this section, we show how to adapt polynomial orders.

Definition 32 *Let $\mathcal{F}$ be a signature, $\mathcal{A} = (A, \mathcal{F}_{\mathcal{A}}, \succsim)$ an ordered $\mathcal{F}$-algebra where $\succsim$ a quasi-ordering in A,*

1. *We say that $\mathcal{A}$ is weakly monotonic if every algebra operation is weakly monotone in its arguments. More precisely, for all $f \in \mathcal{F}$ and $a, b \in A$ with $a \succsim b$ we have $f_{\mathcal{A}}(..., a, ...) \succsim f_{\mathcal{A}}(..., b, ...)$.*

2. *$\mathcal{A}$ is call well-founded if $\succ$ is well-founded. Here, $\succ$ is the strict part of the quasi-order $\succsim$: $\succ = \succsim - \precsim$, that is, $\forall a, b \in A$, $a \succ b$ if $a \succsim b \wedge b \not\succsim a$.*

3. *The quasi-ordering $\succsim_{\mathcal{A}}$ over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ induced by $\mathcal{A}$ is $t \succsim_{\mathcal{A}} u$ if $[\alpha]_{\mathcal{A}}(t) \succsim [\alpha]_{\mathcal{A}}(u)$ for all assignments $\alpha : \mathcal{X} \to A$.*

4. *The strict-ordering $\sqsupset_{\mathcal{A}}$ over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ induced by $\mathcal{A}$ is: $t \sqsupset_{\mathcal{A}} u$ if $[\alpha]_{\mathcal{A}}(t) \succ [\alpha]_{\mathcal{A}}(u)$ for all assignments $\alpha : \mathcal{X} \to A$.*

Note that every polynomial $P(x_1, ..., x_n)$ with non-negative coefficients is weakly monotonic over non-negative numbers. In other words, if $a \geq b$, then $P(a_1, ..., a_{i-1}, a, a_{i+1}, ..., a_n) \geq P(a_1, ..., a_{i-1}, b, a_{i+1}, ..., a_n)$, for all $a, b, a_1, ..., a_{i-1}, a_{i+1}, ..., a_n \in [0, +\infty)$ and $i \in \{1, ..., n\}$.

3.6.1 Polynomials over the naturals

Each polynomial interpretation over the naturals of $\mathcal{F}$ together with the natural order $\geq$ in $\mathbb{N}$ is a well-founded and weakly monotone $\mathcal{F}$-Algebra. According to [CMTU06], the pair $(\succsim, \sqsupset)$ induced by this algebra is a reduction pair.

By interpreting rules of $\mathcal{R}$ and the dependency pairs of $\mathcal{P}$, and according to the inequalities presented in Theorem 9, a set of polynomial constraints is obtained. This set of constraints leads to a *constraint solving problem.* Hence, the termination problem has been transformed into a constraint solving problem.

Corollary 3 *A TRS $\mathcal{R}$ is terminating if for each cycle $\mathcal{P}$ in $\mathsf{EDG}(\mathcal{R})$, there exists a deduction pair $(\geq, >)$ such that*

- *$[l] \geq [r]$ for every rule $l \to r$ from $\mathcal{R}$,*
- *$[u] \geq [v]$ for every dependency pair $u \to v$ from $\mathcal{P}$, and*
- *$[u] > [v]$ for at least one dependency pair $u \to v$ from $\mathcal{P}$.*

Example 12 *Consider the dependency graph presented in Example 10. According to Theorem 9, to check termination of the cycle $MINUS(s(x), s(y)) \to MINUS(x, y)$, the following inequalities must be oriented by a reduction pair $(\succsim, \sqsupset)$:*

$$\begin{aligned}
minus(x, 0) &\succsim x \\
minus(s(x), s(y)) &\succsim minus(x, y) \\
quot(0, s(y)) &\succsim 0 \\
quot(s(x), s(y)) &\succsim s(quot(minus(x, y), s(y))) \\
MINUS(s(x), s(y)) &\sqsupset MINUS(x, y)
\end{aligned}$$

These constraints are satisfied by the reduction pair induced by the following polynomial interpretation:

$$\begin{aligned}[minus](x, y) &= x \\ [quot](x, y) &= x \\ [s](x) &= x + 1 \\ [0] &= 0 \\ [MINUS](x, y) &= x\end{aligned}$$

As a remark, for an $\mathcal{F}$-algebra $\mathcal{A} = (A, \mathcal{F}_A)$ such that $A = \mathbb{N}$ and $\mathcal{F}_A$ is a set of polynomials $[f]$ with non-negative coefficients, these conditions guarantee that:

1. each induced polynomial fulfils the algebraicity property,
2. the quasi-ordering $\succsim$ on terms induced by the ordering $\geq_{\mathbb{N}}$ over the naturals, is stable and monotonic,
3. the strict ordering $\sqsupset$ induced by the strict part $>_{\mathbb{N}}$ of the ordering over the naturals, is stable and well-founded, and
4. $(\succsim, \sqsupset)$ is a reduction pair.

3.6.2 Polynomials over the reals

We are interested in using polynomial interpretations over the real number, since they are strictly more powerful for proving termination than those over the naturals [FNOG$^+$08, Luc06]. In [Luc05, Luc07], Lucas showed how to obtain reduction pairs based on polynomial orders over the reals.

Definition 33 *Let $\mathcal{F}$ be a signature. An ordered $\mathcal{F}$-algebra $\mathcal{A} = (A, \mathcal{F}_A, \geq_A)$ is a quasi-order in A is said to be a weak polynomial $\mathcal{F}$-algebra if $A \subseteq [0, \infty)$, $\geq_A$ is the restriction of $\geq_{\mathbb{R}}$ to A, and $\mathcal{F}_A$ consists of $\geq_{\mathbb{R}}$-monotone polynomials $[f] \in \mathbb{R}[X_1, \ldots, X_n]$.*

Now, given a polynomial algebra $\mathcal{A}$, $\delta > 0$, and terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, we write: $s >_\delta t$ if $[s] - [t] - \delta \geq 0$.

At this point, we can use a quasi-ordering $\geq$ induced by the usual partial ordering $\geq_{\mathbb{R}}$ together with the strict ordering $>_\delta$ to define reduction pairs.

Theorem 10 *Let $\mathcal{F}$ be a signature, $\mathcal{A} = (A, \mathcal{F}_{\mathcal{A}}, \geq_A)$ be a weak polynomial $\mathcal{F}$-algebra, and $\delta > 0$. Then $(\geq, >_\delta)$ is a reduction pair.*

Similarly to Corollary 3, we obtain a criterion for proving termination using reduction orders induced by polynomial interpretations over reals.

Corollary 4 *A TRS $\mathcal{R}$ is terminating if for each cycle $\mathcal{P}$ in $\mathsf{EDG}(\mathcal{R})$, there exist a deduction pair $(\geq, >_\delta)$ such that*

- $[l] \geq [r]$ *for every rule* $l \to r$ *from* $\mathcal{R}$,
- $[u] \geq [v]$ *for every dependency pair* $u \to v$ *from* $\mathcal{P}$, *and*
- $[u] >_\delta [v]$ *for at least one dependency pair* $u \to v$ *from* $\mathcal{P}$.

Note that for an ordered $\mathcal{F}$-algebra $\mathcal{A} = (A, \mathcal{F}_A, \succsim_A)$ such that $\delta > 0$, $A = [0, +\infty) \subseteq \mathbb{R}$ and $\mathcal{F}_A$ is a set of polynomials $[f]$ with non-negative coefficients, then these conditions guarantee that:

1. each induced polynomial fulfils the algebraicity property,
2. the induced quasi-ordering $\geq$ is stable and monotonic,
3. the strict ordering $>_\delta$ is stable and well-founded, and
4. $(\geq, >_\delta)$ is a reduction pair.

Therefore, the polynomial contraints over the reals are the same as those for polynomial with natural coefficients.

3.7 Generating polynomial constraints

In the previous section, we have assumed that polynomial interpretations were given to the function symbols, so that the constraints obtained form (e.g.) the dependency pairs method are satisfied. However, it is not obvious how to choose polynomial interpretations. In the following, we use the approach in [CMTU06, Luc07] to address this problem.

Then polynomial interpretations are considered to solve constraints, each constraint over terms is transformed into a polynomial constraint by means of the interpretation of each k-ary $f \in \mathcal{F}$ in the signature $\mathcal{F}$ as a *parametric* polynomial $[f]$. A *parametric* polynomial is a polynomial where the coefficients are variables whose values have to be found.

In fact, each one of the polynomials is represented as an instance of a unique parametric polynomial $[f]$ with $M_f + k$ variables where k is the arity of f (and it corresponds with the domain of the variable X_i which the polynomial takes as a parameter) and M_f is the number of allowed parametric coefficients $C_1, ..., C_{M_f}$. Therefore, all coefficient in the parametric polynomial $[f] \in \mathbb{Z}[C_1, ..., C_{M_f}, X_1, ..., X_k]$ are 1, and there is no constant coefficients.

In order to have a finite number of such variables, we need to fix a bound on the degree of the polynomials. Steinbach classified a number of restricted forms of multivariate polynomials [Ste92]:

Linear Contains polynomials of degree 1 at most:

$$[f](x_1, ..., x_k) = a_k x_k + ... + a_1 x_1 + a_0$$

Simple Constains polynomials of at most degree 1 in each variables:

$$[f](x_1, ..., x_k) = \sum_{i_j \in \{0,1\}} a_{i_1,...,1_k} x_1^{i_1} \cdots , x_k^{i_k}$$

Simple-Mixed Contains polynomials whose monomials consist of either a single variables with degree 2, or of several variables of at most degree 1:

$$[f](x_1, ..., x_k) = \sum_{i_j \in \{0,1\}} a_{i_1,...,1_k} x_1^{i_1} \cdots , x_k^{i_k} + \sum_{1 \le i \le n} b_i x_i^2$$

Nowadays termination tools restrict the attention to polynomial interpretations within one of the previous. The main reason for this is keeping the obtained constraints simple and small, i.e., the reason is avoiding the huge search space which is associated to symbolic constraints which are interpreted using polynomials of higher degree. The use of such polynomials lead to a combinatorial explosion not only in the number of variables which have to be considered (stemming from the big amount of parametric coefficients which are necessary in this case) but also in the number of constraints which are generated.

Example 13 *Consider the term $t = a(b(c(x)))$. If we interpret symbols a, b, c using linear (parametric) polynomial interpretation, we obtain the following polynomial*

$$a_1 b_1 c_1 x + a_1 b_1 c_0 + a_1 b_0 + a_0$$

containing 4 monomials. On the other hand, if we consider a simple-mixed interpretation for each function symbol, we obtain a polynomial with 52 monomials.

In general, given an upper bound N for the degree of the polynomials $[f] \in D[X_1, ..., X_k]$ (where D is a suitable domain of coefficients), the maximum number of monomials in the interpretation $[f]$ of f is $M_f = \binom{N+ar(f)}{ar(f)} = \frac{(N+ar(f))!}{ar(f)!N!}$.

The next step is to interpret the term s and t in a constraint $s \succsim t$ or $s \sqsupset t$ by using the parametric polynomial interpretations of the symbols. If $[s]$ and $[t]$ are the corresponding polynomials, a constrain $s \sqsupset t$ is transformed into a polynomial constraint $[s] > [t]$. This constraint can be seen as a polynomial constraint (or positiveness condition) $P_{s,t} > 0$ where $P_{s,t} = [s] - [t] \in \mathbb{Z}[C_1, ..., C_{M_f}, X_1, ..., X_k]$ is a polynomial. Thus, we are faced to the problem of deciding whether polinomial $P_{s,t}$ is positive or not (respectively, non-negative when a constraint $s \succsim t$ is considered and a polynomial constraint $P_{s,t} \geq 0$ is obtained in a similar way). A constraint $P(C_1, ..., C_M, X_1, ..., X_K) \geq 0$ can be represented as:

$$P(C_1, ..., C_M, X_1, ..., X_k) = \sum_{r \in \mathbb{N}^K} B_r(C_1, ..., C_M) X^r \geq 0$$

where X^r means $X_1^{r_1} ... X_K^{r_K}$ and $B_r \in \mathbb{Z}[C_1, ..., C_M]$ are polynomials in the parametric coefficients of the parametric polynomial interpretation of the function symbols. Since each X_i, for $1 \leq i \leq K$, takes values in a subset of A of non-negative numbers, using the Hong and Jakuš criterion [HJ98] according to [CMTU06], the constraint is satisfied if the following existencial constraint can be solved

$$\exists C_1, ..., \exists C_M \bigwedge_{r \in \mathbb{N}^K} B_r \geq 0$$

Therefore, the problem becomes a problem of resolution of existentially quantified variables over a domain D of coefficients. Strict constraints are handled in a similar way, but the monomial representing the constant coefficient (B_r such that $r = (0, ..., 0)$) is required to be positive.

Now, the termination problem has been transformed into a standard *constraint solving* problem which can be treated by using standard algorithms and techniques.

Example 14 *Consider again the set of constraints presented in Example 12. The following parametric polynomials interpretations are given to the symbols:*

$$\begin{aligned}
[\mathtt{0}] &= a_0 \\
[\mathtt{s}](X) &= s_1 X + s_0 \\
[\mathtt{minus}](X, Y) &= m_1 X + m_2 Y + m_0 \\
[\mathtt{quot}](X, Y) &= q_1 X + q_2 Y + q_0 \\
[\mathtt{QUOT}](X, Y) &= Q_1 X + Q_2 Y + Q_0
\end{aligned}$$

Then, for instance, the first constraint is translated into the polynomial constraint:

$$(m_1 - 1)x + m_2 a_0 + m_0 \geq 0$$

According to the approach described in this section, we have to solve the following (conjunc-

tion of) polynomial constraints:

(1) $a_0m_2 + m_0 \geq 0$

(2) $m_1 - 1 \geq 0$

(3) $m_1s_0 + m_2s_0 \geq 0$

(4) $m_1s_1 - m_1 \geq 0$

(5) $m_2s_1 - m_2 \geq 0$

(6) $a_0q_1 + q_2s_0 + q_0 - a_0 \geq 0$

(7) $q_2s_1 \geq 0$

(8) $q_1s_0 + q_2s_0 + q_0 - m_0q_1s_1 - q_2s_0s_1 - q_0s_1 - s_0 \geq 0$

(9) $q_1s_1 - m_1q_1s_1 \geq 0$

(10) $q_2s_1 - m_2q_1s_1 - q_2s_1s_1 \geq 0$

(11) $Q_1s_0 - m_0Q_1 > 0$

(12) $Q_1s_1 - m_1Q_1 \geq 0$

(13) $-m_2Q_1 \geq 0$

Note that the variables *which have to be* solved *here are the* coefficients *of the parametric polynomials. The previous set of constraints is sound regarding* DG*-based termination proofs (i.e., its satisfaction implies that the cycle is harmless) provided that* all such variables/parametric coefficients take non-negative values.

Chapter 4

Conditions for selecting rational polynomial orders

In [Luc06], Lucas proved that polynomial interpretation over the rationals are strictly more powerful than polynomial interpretation over the naturals. The proof is constructive, i.e., he *synthesized* examples showing these deficiencies of polynomial interpretations over naturals. Continuing his work, in Section 4.1, we introduce several criteria for determining the necessity of rational coefficients in termination proofs.

The material in this chapter has been published in [FNOG$^+$08].

4.1 Sufficient conditions

In the following, we introduce some simple facts regarding polynomials and related properties.

Definition 34 (Monotonicity in the i-th argument) *A polynomial $P \in \mathbb{R}[X_1, \ldots, X_n]$ is called (R, i)-monotonic w.r.t $A \subseteq \mathbb{R}$ if*

$$x \; R \; y \quad \Rightarrow \quad P(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_n) \; R \; P(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)$$

for i, $1 \leq i \leq n$, and all $x, y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n \in A$.

Lemma 1 *Let $P \in \mathbb{R}[X_1, \ldots, X_n]$ be $(\geq, i)$-monotonic on $A \subseteq \mathbb{R}$. Then, for all $x_1, \ldots, x_n, x, y \in A$, if $P(x_1, \ldots, x_{i-1}, x, \ldots, x_n) > P(x_1, \ldots, x_{i-1}, y, \ldots, x_n)$, then $x > y$.*

Proof. By contradiction: if $x \leq y$, then, by $(\geq, i)$-monotonicity, $P(x_1, \ldots, x_{i-1}, x, \ldots, x_n) \leq P(x_1, \ldots, x_{i-1}, y, \ldots, x_n)$ leading to a contradiction. □

Lemma 2 *Let $P \in \mathbb{R}_0[X_1, \ldots, X_n]$ be a polynomial with nonnegative coefficients. Then, for all i, $1 \leq i \leq n$:*

1. *If there is a linear monomial $a_i x_i$ in P such that $a_i \geq 1$, then P has the $(\geq, i)$-subterm property on every $A \subseteq \mathbb{R}_0$, i.e., $P(x_1, \ldots, x_{i-1}, x, x_{i+1} \ldots, x_n) \geq x$.*
2. *If there is a linear monomial $a_i x_i$ in P such that $a_i > 0$, then P is $(>, i)$-monotonic on every $A \subseteq \mathbb{R}_0$.*
3. *P is $(\geq, i)$-monotonic on every $A \subseteq \mathbb{R}_0$.*

Definition 35 (Variable visibility) *A variable $x \in \mathcal{X}$ occurring in $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is visible by a polynomial interpretation $[\cdot]$ if $[t]$ contains a monomial in x.*

The following proposition is obvious.

Proposition 3 *Let $\mathcal{F}$ be a signature, $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, and let $[\cdot]$ be a polynomial interpretation of $\mathcal{F}$ such that $[s] \geq [t]$. If $x \in \mathcal{V}ar(t)$ is visible in t by $[\cdot]$, then x is also visible in s by $[\cdot]$.*

Let $\mathcal{E}mb(\mathcal{F})$ be a set of projection rules for a signature $\mathcal{F}$, such that

$$\mathcal{E}mb(\mathcal{F}) = \{f(x_1, \ldots, x_i, \ldots, x_n) \to x_i \mid f \in \mathcal{F}^{(n)}, 1 \leq i \leq n\}$$

Given a subset $\mathcal{E} \subseteq \mathcal{E}mb(\mathcal{F})$ and terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, we write $s \trianglerighteq_{\mathcal{E}} t$ if $s \to^*_{\mathcal{E}} t$ (we write $s \triangleright_{\mathcal{E}} t$ if $s \trianglerighteq_{\mathcal{E}} t$ and $s \neq t$). We also denote as $\mathcal{F}(\mathcal{E})$ the set of symbols with projection rules in $\mathcal{E}$: $\mathcal{F}(\mathcal{E}) = \{root(l) \mid l \to r \in \mathcal{E}\}$.

Proposition 4 *Let $\mathcal{F}$ be a signature, $\mathcal{E} \subseteq \mathcal{E}mb(\mathcal{F})$ and let $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be such that $t \trianglerighteq_{\mathcal{E}} s$. Let $(A, [\cdot])$ be a linear polynomial interpretation such that $A \subseteq \mathbb{R}_0$ and $[f] \in \mathbb{R}_0[X_1, \ldots, X_n]$ for all n-ary symbol $f \in \mathcal{F}$. If $[s] > [t]$ on A then there is $f \in \mathcal{F}(\mathcal{E})$ which is interpreted as a polynomial $[f](X_1, \ldots, X_k)$ containing a monomial $a_i X_i$ with $a_i < 1$ for some i such that $f(X_1, \ldots, X_i, \ldots, X_k) \to X_i \in \mathcal{E}$.*

Proof. First we note that, since $[s] > [t]$ on A, we actually have $t \triangleright_{\mathcal{E}} s$. We proceed by contradiction. Assume that for all $f \in \mathcal{F}(\mathcal{E})$ and i such that $f(X_1, \ldots, X_i, \ldots, X_k) \to X_i \in \mathcal{E}$, $[f](X_1, \ldots, X_k)$ contains no monomial $a_i X_i$ with $a_i < 1$. Since $[f]$ is linear, there must be $a_i \geq 1$ in that case. Now we prove by induction on the length n of the rewrite sequence $t \to^+_{\mathcal{E}} s$ (remember that $t \triangleright_{\mathcal{E}} s$) that $[t] \geq [s]$ on A. If $n = 1$, then $t = t[f(t_1, \ldots, t_i, \ldots, t_k)]_p$ for some position $p \in \mathcal{P}os(t)$

and $s = t[t_i]_p$ due to the application of a projection rule $f(X_1, \ldots, X_i, \ldots, X_k) \to X_i \in \mathcal{E}$. By hypothesis and by Lemma 2(1), $[f(t_1, \ldots, t_i, \ldots, t_k)] \geq [t_i]$ on A. Since $[f]$ only contains non-negative coefficients, by a repeated application of Lemma 2(3), we conclude that $[t] \geq [s]$ as desired.

On the other hand, if $n > 1$, we can write the $\mathcal{E}$-rewrite sequence as follows: $t \to_{\mathcal{E}} t' \to^{+}_{\mathcal{E}} s$. By the induction hypothesis, $[t'] \geq [s]$ and reasoning as in the base case, we have that $[t] \geq [t']$. Again, we conclude that $[t] \geq [s]$. This contradicts that $[s] > [t]$. □

Given a term t in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and a position $p \in \mathcal{P}os(t)$, the chain of symbols lying on positions above/on $p \in \mathcal{P}os(t)$ is $\mathit{prefix}_t(\Lambda) = \mathit{root}(t)$, $\mathit{prefix}_t(i.p) = \mathit{root}(t).\mathit{prefix}_{t_i}(p)$. The strict prefix *sprefix* is $\mathit{sprefix}_t(\Lambda) = \Lambda$, $\mathit{sprefix}_t(p.i) = \mathit{prefix}_t(p)$. The following proposition shows that, in some cases, the use of real coefficients which are not integer is necessary in termination proofs.

Theorem 11 *Let $\mathcal{F}$ be a signature and $l, r, s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be such that $t \trianglerighteq_{\mathcal{E}} s$ for some $\mathcal{E} \subseteq \mathcal{E}mb(\mathcal{F})$. Let $(A, [\cdot])$ be a linear polynomial interpretation such that $A \subseteq \mathbb{R}_0$ and $[f] \in \mathbb{R}_0[X_1, \ldots, X_n]$ for all n-ary symbol $f \in \mathcal{F}$. If $[s] > [t]$, $[l] \geq [r]$ and there is $x \in \mathcal{V}ar(r)$ which is visible in r by $[\cdot]$ and $\forall p \in \mathcal{P}os_x(l), \mathcal{F}(\mathcal{E}) \subseteq prefix_l(p)$, then there is $f \in \mathcal{F}(\mathcal{E})$ and $i \in \mathbb{N}$ such that $f(X_1, \ldots, X_i, \ldots, X_k) \to X_i \in \mathcal{E}$ satisfying that $[f_i](X_1, \ldots, X_i, \ldots, X_k) = a_1 X_1 + \cdots + a_i.X_i + \cdots + a_k X_k + a_0$ and $a_i \in (0, 1)$.*

Proof. By Proposition 4, there is $f \in \mathcal{F}(\mathcal{E})$ and $i \in \mathbb{N}$ such that $f(X_1, \ldots, X_i, \ldots, X_k) \to X_i \in \mathcal{E}$ and $a_i \in [0, 1)$. We show that $a_i \neq 0$. Since $f \in prefix_l(p)$ for all $p \in \mathcal{P}os_x(l)$, we proceed by contradiction and assume that $a_i = 0$. Then (since $[f]$ is a linear polynomial), $[l]$ contains no monomial in x. However, $[r]$ contains a monomial in x. This means that $[l] - [r]$ contains a negative monomial in x. Thus, we cannot have $[l] \geq [r]$. □

The following corollary is interesting to treat *collapsing* rules, i.e., rules $l \to r$ such that r is a variable.

Corollary 5 *Let $\mathcal{F}$ be a signature and $l, s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be such that $t \trianglerighteq_{\mathcal{E}} s$ for some $\mathcal{E} \subseteq \mathcal{E}mb(\mathcal{F})$. Let $(A, [\cdot])$ be a linear polynomial interpretation such that $A \subseteq \mathbb{R}_0$ and $[f] \in \mathbb{R}_0[X_1, \ldots, X_n]$ for all n-ary symbol $f \in \mathcal{F}$. If $[s] > [t]$, $x \in \mathcal{V}ar(l)$, $[l] \geq [x]$, and $\forall p \in \mathcal{P}os_x(l), \mathcal{F}(\mathcal{E}) \subseteq prefix_l(p)$, then there is $f \in \mathcal{F}(\mathcal{E})$ and $i \in \mathbb{N}$ such that $f(X_1, \ldots, X_i, \ldots, X_k) \to X_i \in \mathcal{E}$ satisfying that $[f_i](X_1, \ldots, X_i, \ldots, X_k) = a_1 X_1 + \cdots + a_i.X_i + \cdots + a_k X_k + a_0$ and $a_i \in (0, 1)$.*

Example 15 *In order to illustrate the use of Corollary 5, consider the following TRS from the*

TPDB (secret05-tpa2):

$$
\begin{aligned}
minus(x,0) &\rightarrow x && (4.1)\\
minus(s(x),s(y)) &\rightarrow minus(x,y) && (4.2)\\
p(s(x)) &\rightarrow x && (4.3)\\
f(s(x),y) &\rightarrow f(p(minus(s(x),y)),p(minus(y,s(x)))) && (4.4)\\
f(x,s(y)) &\rightarrow f(p(minus(x,s(y))),p(minus(s(y),x))) && (4.5)
\end{aligned}
$$

This TRS has eleven dependency pairs, but there are only two minimal cycles in the dependency graph: $\{(4.6)\}$ *and* $\{(4.7),(4.8)\}$*, where*

$$
\begin{aligned}
MINUS(s(x),s(y)) &\rightarrow MINUS(x,y) && (4.6)\\
F(s(x),y) &\rightarrow F(p(minus(s(x),y)),p(minus(y,s(x)))) && (4.7)\\
F(x,s(y)) &\rightarrow F(p(minus(x,s(y))),p(minus(s(y),x))) && (4.8)
\end{aligned}
$$

The dependency pair (4.6) can immediately be removed by the subterm criterion (see [HM04]). It remains to find a polynomial interpretation such that one of the dependency pairs (4.7) *and* (4.8) *is oriented strictly and the other dependency pair and the rules* $\{(4.1),(4.2),(4.3)\}$ *are oriented weakly. For both* (4.7) *and* (4.8)*, the left-hand side is embedded in the right-hand side. For instance for* (4.7)*, we have* $F(p(minus(s(x),y)),p(minus(y,s(x)))) \rightarrow^*_{\mathcal{E}} F(s(x),y)$ *with* $\mathcal{E} = \{p(x_1) \rightarrow x_1,\ minus(x_1,x_2) \rightarrow x_1\}$*. So by Corollary 5, the interpretation of* p *or* $minus$ *contains a rational coefficient* a *in* $(0,1)$*. For instance, the following polynomial interpretation is computed automatically by* MU-TERM *for the cycle* $\{(4.7),(4.8)\}$*:*

$$
\begin{aligned}
[\mathtt{minus}](X_1,X_2) &= X_1\\
[\mathtt{0}] &= 0\\
[\mathtt{s}](X) &= 2X+2\\
[\mathtt{p}](X) &= \tfrac{1}{2}X\\
[\mathtt{f}](X_1,X_2) &= 0\\
[\mathtt{F}](X_1,X_2) &= 2X_1
\end{aligned}
$$

Note the coefficient $\frac{1}{2}$ *in the interpretation of* p*.*

The following result is a simple corollary of Theorem 11.

Corollary 6 *Let* $\mathcal{R}$ *be a TRS, let* $\mathcal{F}$ *be a signature,* $\mathcal{E} \subseteq \mathcal{E}mb(\mathcal{F})$*,* $\mathcal{P}$ *be a sequence* $s_1 \rightarrow t_1,\ldots,s_n \rightarrow t_n$ *of pairs for some* $n \geq 1$*, and let* σ *be a substitution such that* $\sigma(t_i) \rightarrow^*_{\mathcal{R}} \sigma(s_{i+1})$ *for all* i*,* $1 \leq i < n$ *and* $\sigma(t_n) \trianglerighteq_{\mathcal{E}} \sigma(s_1)$*. Let* $(A,[\cdot])$ *be a linear polynomial interpretation such that* $A \subseteq \mathbb{R}_0$ *and* $[f] \in \mathbb{R}_0[X_1,\ldots,X_k]$ *for all* k*-ary symbols* $f \in \mathcal{F}$*. If* $[l] \geq [r]$ *for all* $l \rightarrow r$ *in* $\mathcal{R}$*,* $[s] \geq [t]$ *for all* $s \rightarrow t \in \mathcal{P}$*, and* $[s] > [t]$ *for at least one* $s \rightarrow t \in \mathcal{P}$*, then there is* $f \in \mathcal{F}(\mathcal{E})$ *which is interpreted as a polynomial* $[f](X_1,\ldots,X_k)$ *containing a monomial* a_iX_i *with* $a_i < 1$ *for some* i *such that* $f(X_1,\ldots,X_i,\ldots,X_k) \rightarrow X_i \in \mathcal{E}$*.*

PROOF. According to our assumptions, we have that $[s_i] > [t_i]$ for some i, $1 \leq i < n$. Hence, $[\sigma(s_i)] > [\sigma(t_i)]$. Since $\geq$ is stable and monotonic, then $[\sigma(s_1)] \geq [\sigma(t_1)] \geq \cdots \geq [\sigma(s_i)] > [\sigma(t_i)] \geq \ldots \geq [\sigma(s_n)] \geq [\sigma(t_n)]$ holds, so that $[\sigma(s_1)] > [\sigma(t_n)]$. By Proposition 4, since $\sigma(t_n) \trianglerighteq_{\mathcal{E}} \sigma(s_1)$, there is $f \in \mathcal{F}(\mathcal{E})$ which is interpreted as a polynomial $[f](X_1, \ldots, X_k)$ containing a monomial $a_i X_i$ with $a_i < 1$ for some i such that $f(X_1, \ldots, X_i, \ldots, X_k) \to X_i \in \mathcal{E}$. □

Example 16 *Consider the following TRS $\mathcal{R}$ from the TPDB (SchneiderKamp-TRS-thiemann40), where $random(x)$ computes a random number between 0 and x:*

$$\begin{aligned}
nonZero(0) &\to false && (4.9)\\
nonZero(s(x)) &\to true && (4.10)\\
p(0) &\to 0 && (4.11)\\
p(s(x)) &\to x && (4.12)\\
id\_inc(x) &\to x && (4.13)\\
id\_inc(x) &\to s(x) && (4.14)\\
random(x) &\to rand(x, 0) && (4.15)\\
rand(x, y) &\to if(nonZero(x), x, y) && (4.16)\\
if(false, x, y) &\to y && (4.17)\\
if(true, x, y) &\to rand(p(x), id\_inc(y)) && (4.18)
\end{aligned}$$

Here, there are six dependency pairs, but there are only one minimal cycle $\mathfrak{C} = \{(4.19), (4.20)\}$:

$$\begin{aligned}
RAND(x, y) &\to IF(nonZero(x), x, y) && (4.19)\\
IF(true, x, y) &\to RAND(p(x), id\_inc(y)) && (4.20)\\
& && (4.21)
\end{aligned}$$

The following variant of such dependency pairs in the cycle

$$\begin{aligned}
RAND(x, y) &\to IF(nonZero(x), x, y) && (4.22)\\
IF(true, x', y') &\to RAND(p(x'), id\_inc(y')) && (4.23)
\end{aligned}$$

satisfies the conditions in Corollary 6 for the substitution $\sigma = \{x \mapsto s(X), y \mapsto Y, x' \mapsto s(X), y' \mapsto Y\}$:

$$\begin{aligned}
\sigma(RAND(x, y)) \;=\;& RAND(s(X), Y)\\
& \to IF(\underline{nonZero(s(X))}, s(X), Y)\\
& \to IF(\mathsf{true}, s(X), Y)\\
& \to RAND(p(s(X)), id\_inc(Y)) = \sigma(RAND(\mathsf{p}(x'), id\_inc(y')))
\end{aligned}$$

For every polynomial interpretation $[\cdot]$ which is compatible with $\mathfrak{C}$, we have $[RAND(s(X), Y)] > [RAND(p(s(X)), id\_inc(Y))]$. Since $RAND(p(s(X)), id\_inc(Y)) \trianglerighteq_{\mathcal{E}} RAND(s(X), Y)$ for $\mathcal{E} =$

$\{p(x) \to x, \mathsf{id\_inc}(x) \to x\}$, either $[p](x)$ or $[id\_inc](x)$ contain a monomial $a_i x$ with $a_i \in [0,1)$. In fact, considering the rules defining p and id_inc and using now Proposition 5, we further conclude that either $[p](x)$ or $[id\_inc](x)$ contain a monomial $a_i x$ with $a_i \in (0,1)$, i.e., $[\cdot]$ cannot *be a polynomial interpretation with non-negative integer coefficients only.*

As a final remark, in this example, the substitution σ could be computed by narrowing $nonZero(x)$ in the first dependency pair (see [AG00, Definition 21]).

Now we end this section with a last result concerning binary symbols.

Theorem 12 *Let $A \subseteq \mathbb{R}_0$, $\mathcal{F}$ be a signature, $f \in \mathcal{F}$ be a binary symbol, $x \in \mathcal{V}ar$, and let $s,t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ be such that $x \notin \mathcal{V}ar(s) \cup \mathcal{V}ar(t)$. Let $[\cdot]$ be a polynomial interpretation such that $[g] \in \mathbb{R}_0[X_1,\ldots,X_n]$ for all n-ary symbol $g \in \mathcal{F}$. If $[f]$ is a linear polynomial $[f](x,y) = f_1x + f_2y + f_0$ and $[f(x,s)] > [f(t,x)]$ on A (resp. $[f(s,x)] > [f(x,t)]$ on A), then $f_1 \geq f_2 > 0$ (resp. $f_2 \geq f_1 > 0$) and $[s] > [t]$ on A.*

Proof. We prove the first case; the second one is analogous. Since f is interpreted as a linear polynomial $[f](x,y) = f_1x + f_2y + f_0$, the inequality $[f](x,[s]) > [f]([t],x)$ can be written as $f_1x+f_2[s]+f_0 > f_1[t]+f_2x+f_0$. Furthermore, since $x \notin \mathcal{V}ar(s)\cup\mathcal{V}ar(t)$, this is equivalent to $f_1 \geq f_2 \wedge f_2[s] > f_1[t]$. Note that $f_2[s] > f_1[t]$ implies that $f_2 \neq 0$. Hence, $f_1 \geq f_2 > 0$. Now, we prove that $[s] > [t]$ on A by contradiction. Assume that $[s] = P(X_1,\ldots,X_n)$, $[t] = Q(X_1,\ldots,X_n)$ and there are numbers $x_1,\ldots,x_n \in A$ such that $P(x_1,\ldots,x_n) \leq Q(x_1,\ldots,x_n)$. Then, since $P \geq 0$ on A and $f_2 \geq 0$, we conclude that $f_1Q(x_1,\ldots,x_n) \geq f_2Q(x_1,\ldots,x_n) \geq f_2P(x_1,\ldots,x_n)$, contradicting that $f_2[s] > f_1[t]$ on A. □

Example 17 *To illustrate the criterion in Theorem 12, we consider the following TRS from the TPDB (Zantema-jw05):*

$$f(\mathsf{f}(a,x),a) \to f(f(x,f(a,a)),a) \tag{4.24}$$

This TRS has three dependency pairs:

$$F(f(a,x),a) \to F(f(x,f(a,a)),a) \tag{4.25}$$
$$F(f(a,x),a) \to F(x,f(a,a)) \tag{4.26}$$
$$F(f(a,x),a) \to F(a,a) \tag{4.27}$$

We focus on cycle $\mathfrak{C} = \{(4.25)\}$. We have to find a reduction pair $(\succeq,>)$ which satisfies that $F(f(a,x),a) > F(f(x,f(a,a)),a)$. Assume that we search for the reduction pair using a linear polynomial interpretation where only non-negative coefficients are used. By Lemma 1, $F(f(a,x),a) > F(f(x,f(a,a)),a)$ implies that $f(a,x) > f(x,f(a,a))$. By Proposition 12, $f(a,x) > f(x,f(a,a))$ implies that $a > f(a,a)$ and the coefficients f_1 and f_2 of the first and *second arguments of the linear polynomial* $[f]$ *are positive:* $f_2 \geq f_1 > 0$*. Note that* $f(a,a) \trianglerighteq_{\mathcal{E}} a$ *for* $\mathcal{E}$ *consisting of either the first projection* $f(x,y) \to x$ *for* f *or the second one* $f(x,y) \to y$*. Thus, by Proposition 4,* $f_1 < 1$ *and* $f_2 < 1$*, i.e.,* $f_1, f_2 \in (0,1)$*.*

Furthermore, since $[a] > f_1[a]+f_2[a]+f_0 = [a](f_1+f_2)+f_0 \geq 0$ *this means that* $[a] = a_0 > 0$ *and* $0 < f_1 + f_2 < \frac{a_0-f_0}{a_0} = 1 - \frac{f_0}{a_0} \leq 1$*, i.e.,* $0 < f_1 + f_2 < 1$ *and we can say that* $f_1 \in (0, \frac{1}{2})$*. For instance, the following polynomial interpretation is computed automatically by* MU-TERM *for solving cycle* $\{(4.25)\}$*:*

$$\begin{array}{lcl} [\mathtt{f}](X_1,X_2) & = & \frac{1}{4}X_1 + \frac{1}{4}X_2 \\ [\mathtt{a}] & = & 4 \\ [\mathtt{F}](X_1,X_2) & = & 4X_1 \end{array}$$

Chapter 5

Polynomial Constraint Satisfaction Techniques

Many termination tools (AProVE [GST06], C*i*ME 2.03 [CMMU03], MU-TERM [Luc04], TTT [HM07],...) use the dependency pair method together with polynomials as a principal ingredient to achieve termination proofs. As we have studied in previous chapters, the termination problem can be transformed into a *constraint solving* problems which can be handled by using standard algorithms and techniques. Despite of the generality of the polynomial constraints, nonlinear constraints seems not to have been studied deeply. Researchers from Constraint Programming area usually restrict to binary constraints. Consequently, the main problem is to find efficient and suitable methods for dealing with the generated set of constraints.

In this chapter, we introduce some new advances from different approaches for solving polynomial constraints. In Section 5.1, we present SAT-based techniques for small domains of natural numbers. We also introduce CSP-based techniques for finite domains of rational numbers (see Section 5.2). In Section 5.3, we discuss about Linear programming and Hybrid techniques. Finally, in Section 5.4, we talk about Computer Algebra techniques.

5.1 SAT-based techniques

In this section, we describe methods for dealing with polynomial constraint from termination problems by means of SAT-based encodings. In Section 5.1.1, we give a short introduction to Propositional Logic and Boolean satisfiability. Finally, in Section 5.1.2, we present a SAT-based solver for polynomial constraints over the domain $\{0, 1\}$. The material in this section has been published in [LN08].

5.1.1 Propositional Logic and SAT

The *Boolean satisfiability* (SAT) problem is a well-known constraint satisfaction problem with many applications. The last few years have seen an enormous progress in the performance of SAT solvers.

Definition 36 (Propositional Logic) *Let $\mathcal{A}$ be propositional signature, a nonempty set of symbols called proposition symbols or propositional variables. Then, propositional formulas of a propositional signature $\mathcal{A}$ are formed from propositional variables and the 0-ary connective $\{True, False\}$ called truth or boolean values, the unary connective $\neg$, and the binary connectives $\{\wedge, \vee, \rightarrow, \leftrightarrow\}$.*

An *interpretation* of a propositional signature $\mathcal{A}$ is an *assignment* $\alpha : \mathcal{A} \mapsto \{true, false\}$ from propositional variables to truth values. The assignment is extended to formulas (semantics of propositional formulas) as follows:

- $\alpha(True) = True$ and $\alpha(False) = False$,
- $\alpha(\neg x) = False$, if $\alpha(x) = True$ or $\alpha(\neg x) = False$, if $\alpha(x) = True$,

and similarly for the other propositional connectives.

Given a propositional formula φ, the Boolean satisfiability problem posed on φ is to determine whether there exists an assignment under which φ evaluates to $True$. If this is the case, we say that the formula φ is satisfiable. Modern SAT solvers provide a "black-box" procedure that can often solve hard structured problems with over a million variables and several millions constraints.

SAT solving is not only interesting from a scientific point of view, but also from a commercial point of view. SAT solvers have been used as a target language for many applications related to theorem proving, verification, artificial intelligence (AI), electronic design automation (EDA) and even challenging problems from algebra (see [PBG05] for a recent survey). In our research area, SAT solvers are enjoying an increasing interest as well (see, e.g, [CLS06, FGMS$^+$07, ZM07]).

Boolean Satisfiability and Constraint Programming seem to be two independent research threads, but they have a lot in common. For an overview of the two areas in a comparative way, we refer to [BHZ06].

In our setting, the basic idea is to encode the polynomial constraints as a propositional formula according to the definition of an upper bound for the domain of the coefficients. Most SAT solvers only take propositional formulas in CNF (Conjunctive Normal Form) as input, i.e., one has to convert the encoded formula to CNF. Next we use a state-of-the-art SAT solver to

Coeff. Range:	1	2	3
# Success	421	431	434
% Success	49,1	49,8	50,2
Time	45.5 s.	91.8 s.	118.6 s.

Table 5.1: AProVE-SAT benchmarks

find an assignment which evaluates the formula to $True$. Finally, we transform the propositional assignment into a diophantine assignment.

5.1.2 SAT-solving for polynomial constraints over $\{0, 1\}$

In [FGMS+07], Fuhs et al. propose the use of SAT techniques for solving polynomial constraints. Fuhs et al.'s (extensive) benchmarks show that, indeed, using $D = \{0, 1\}$ as the domain for coefficients in polynomial interpretations is already a very powerful option in comparison to bigger domains. The information in Table 5.1 has been taken from [FGMS+07]. It corresponds to the benchmarks performed with the new version of AProVE which implements a SAT-based solver for polynomial constraints (AProVE-SAT), where 865 examples and three different ranges for coefficients (corresponding to $N_1 = \{0, 1\}$, $N_2 = \{0, 1, 2\}$, and $N_3 = \{0, 1, 2, 3\}$) were considered. The termination problems come from the 2006 Termination Problem Data Base (TPDB, version 3.2)[1].

Table 5.1 shows that AProVE-SAT increments the ratio of solved examples in $0,7\%$ when using coefficients from N_2 instead of N_1, but the time for achieving the proofs is *duplicated!*. So this suggests that polynomials with coefficients from $\{0, 1\}$ are sufficient in most examples. Thus, specifically considering the (small) domain N_1 to obtain an efficient solver on this particular domain still makes sense.

In this section, we develop an easy SAT-based encoding for polynomial orders over the domain of coefficients N_1.

Considering this domain, we can perform a simplification in the polynomial representation: since for all $x \in N_1$ and all $n > 0$ we have $x^n = x$, when considering the representation of a polynomial P, we can *replace* each monomial $\mathsf{m} = cX_1^{\alpha_1} \cdots X_n^{\alpha_n}$ in P by the monomial $\rho(\mathsf{m}) = cX_1^{\beta_1} \cdots X_n^{\beta_n}$ where $\beta_i = 1$ if $\alpha_i \neq 0$ and $\beta_i = 0$ if $\alpha_i = 0$. Then, we obtain a simpler representation $\rho(P)$ of P:

[1]see `http://www.lri.fr/~marche/tpdb/`

$$\rho(P) = \begin{cases} K \text{ if } P = K \in \mathbb{R} \\ cX_1^{\beta_1} \cdots X_n^{\beta_n} + \rho(Q) \text{ if } P = cX_1^{\alpha_1} \cdots X_n^{\alpha_n} + Q \\ \quad \text{and there is } i, 1 \leq i \leq n, \alpha_i > 0 \end{cases}$$

The following result will be used to justify the correctness (and completeness) of this approach regarding constraint solving over N_1.

Proposition 5 *Let $P \in \mathbb{R}[X_1, \ldots, X_n]$ be a polynomial. For all $x_1, \ldots, x_n \in N_1$, $P(x_1, \ldots, x_n) = \rho(P)(x_1, \ldots, x_n)$.*

Proof. By induction on the number N of monomials with variables in P. If $N = 0$, then P is a constant monomial which is obviously identic to $\rho(P)$. If $N > 0$, let $P = cX_1^{\alpha_1} \cdots X_n^{\alpha_n} + Q$ where Q consists of all monomials in P but $cX_1^{\alpha_1} \cdots X_n^{\alpha_n}$. Since $x_i \in N_1$, we have that $x_i^{\alpha_i} = x_i^{\beta_i}$ for $\beta_i = 1$ if $\alpha_i \neq 0$ and $\beta_i = 0$ if $\alpha_i = 0$. Thus, for all $x_1, \ldots, x_n \in N_1$, $cx_1^{\alpha_1} \cdots x_n^{\alpha_n} = cx_1^{\beta_1} \cdots x_n^{\beta_n}$. By Induction Hypothesis., $Q(x_1, \ldots, x_n) = \rho(Q)(x_1, \ldots, x_n)$. Since $P(x_1, \ldots, x_n) = cx_1^{\alpha_1} \cdots x_n^{\alpha_n} + Q = cx_1^{\beta_1} \cdots x_n^{\beta_n} + \rho(Q) = \rho(P)(x_1, \ldots, x_n)$, the conclusion follows. □

According to this, when solving constraints over N_1, we can use $\rho(P) \geq 0$ (or $\rho(P) > 0$) instead of $P \geq 0$ (resp. $P > 0$) without loosing anything.

Example 18 *The polynomial constraint*

$$q_2 s_1 - m_2 q_1 s_1 - q_2 s_1 s_1 \geq 0$$

would be equivalently transformed into

$$-m_2 q_1 s_1 \geq 0$$

if we are going to solve it over N_1.

Along this section, we often use 0 and 1 instead of $False$ and $True$, respectively, to denote boolean values. Furthermore, given a finite set $\{X_1, \ldots, X_n\}$ of (boolean) variables where some arbitrary total order is assumed, boolean assignments are represented as sequences $(x_1, \ldots, x_n) \in \{0,1\}^n$ where variable X_i is instantiated by x_i for $1 \leq i \leq n$.

When considering polynomial constraints over N_1, the arithmetics on N_1 becomes very close to boolean operations when 0 is interpreted as $False$ and 1 as $True$, respectively. In particular, the product of values in N_1 corresponds to conjunction. Following this intuition, we have developed a simple encoding of polynomial constraints as propositional formulas.

$$
\begin{array}{rcl}
\mathsf{rmM}_{\mathcal{X}}(K) & = & K \text{ if } K \text{ is a constant} \\
\mathsf{rmM}_{\mathcal{X}}(cX_1^{\alpha_1}\cdots X_n^{\alpha_n}+P) & = & \left\{\begin{array}{ll} \mathsf{rmM}_{\mathcal{X}}(P) & \text{if } \mathcal{X}\subseteq\{X_1,\ldots,X_n\} \\ cX_1^{\alpha_1}\cdots X_n^{\alpha_n}+\mathsf{rmM}_{\mathcal{X}}(P) & \text{otherwise}\end{array}\right. \\
\mathsf{rmV}_{\mathcal{X}}(K) & = & K \text{ if } K \text{ is a constant} \\
\mathsf{rmV}_{\mathcal{X}}(cX_1^{\alpha_1}\cdots X_n^{\alpha_n}+P) & = & cX_1^{\beta_1}\cdots X_n^{\beta_n}+\mathsf{rmV}_{\mathcal{X}}(P) \\
 & & \text{where, for all } i, 1\leq i\leq n, \beta_i = \left\{\begin{array}{ll} 0 & \text{if } X_i\in\mathcal{X} \text{ and} \\ \alpha_i & \text{otherwise}\end{array}\right.
\end{array}
$$

Figure 5.1: Definition of rmM and rmV

$$
\begin{array}{rcl}
\tau(K\geq 0) & = & True, \quad \text{if } K\geq 0 \\
\tau(K\geq 0) & = & False, \quad \text{if } K< 0 \\
\tau(K> 0) & = & True, \quad \text{if } K> 0 \\
\tau(K> 0) & = & False, \quad \text{if } K\leq 0 \\
\tau(cX_1\ldots X_n+Q\geq 0) & = & \Big(\Big(\bigvee_{1\leq i\leq n}\neg X_i\Big)\wedge\tau(\mathsf{rmM}_{\{X_1,\ldots,X_n\}}(Q)\geq 0)\Big)\vee \\
 & & \Big(\Big(\bigwedge_{1\leq i\leq n}X_i\Big)\wedge\tau(\mathsf{rmV}_{\{X_1,\ldots,X_n\}}(Q)+c\geq 0)\Big) \\
\tau(cX_1\ldots X_n+Q> 0) & = & \Big(\Big(\bigvee_{1\leq i\leq n}\neg X_i\Big)\wedge\tau(\mathsf{rmM}_{\{X_1,\ldots,X_n\}}(Q)> 0)\Big)\vee \\
 & & \Big(\Big(\bigwedge_{1\leq i\leq n}X_i\Big)\wedge\tau(\mathsf{rmV}_{\{X_1,\ldots,X_n\}}(Q)+c> 0)\Big) \\
\tau(C\wedge C') & = & \tau(C)\wedge\tau(C')
\end{array}
$$

Figure 5.2: SAT encoding of polynomial constraints over N_1

The translation function τ is given in Figure 5.2, where Q is a polynomial, c and K are numeric constants (with $c\neq 0$), $\mathcal{X}$ is a set of variables, $\mathsf{rmM}_{\mathcal{X}}(P)$ removes all monomials in P which include *all* variables in $\mathcal{X}$, and $\mathsf{rmV}_{\mathcal{X}}(P)$ removes from P all occurrences of variables in $\mathcal{X}$. The formal definitions are given in Figure 5.1. The following results are used later. Their proofs are straightforward from the definitions in Figure 5.1.

Lemma 3 *Let $P\in\mathbb{R}[X_1,\ldots,X_n]$ be a polynomial, such that $P=cX_1^{\alpha_1}\cdots X_n^{\alpha_n}+Q$ for some $c\in\mathbb{R}$, $\alpha_i\in\mathbb{N}$ for $1\leq i\leq n$, and $Q\in\mathbb{R}[X_1,\ldots,X_n]$. Let $\mathcal{X}=\{X_i\mid\alpha_i>0\}$ and $x_1,\ldots,x_n\in N_1$ be such that $x_i=0$ for at least one $X_i\in\mathcal{X}$. Then, $\mathsf{rmM}_{\mathcal{X}}(P)(x_1,\ldots,x_n)=\mathsf{rmM}_{\mathcal{X}}(Q)(x_1,\ldots,x_n)=Q(x_1,\ldots,x_n)$.*

Lemma 4 *Let $P\in\mathbb{R}[X_1,\ldots,X_n]$ be a polynomial, such that $P=cX_1^{\alpha_1}\cdots X_n^{\alpha_n}+Q$ for some $c\in\mathbb{R}$, $\alpha_i\in\mathbb{N}$ for $1\leq i\leq n$, and $Q\in\mathbb{R}[X_1,\ldots,X_n]$. Let $\mathcal{X}=\{X_i\mid\alpha_i>0\}$ and $x_1,\ldots,x_n\in N_1$ be such that $x_i=1$ whenever $X_i\in\mathcal{X}$. Then, $\mathsf{rmV}_{\mathcal{X}}(Q)(x_1,\ldots,x_n)=Q(x_1,\ldots,x_n)$.*

According to Proposition 5, we also assume that we only have to deal with polynomials consisting of monomials like $cX_1\cdots X_n$ (i.e., *without* any power greater than 1).

Example 19 *Consider the constraint $q_1s_1-m_1q_1s_1\geq 0$, which is translated into a propositional* *formula as follows:*

$$\begin{aligned}&\tau(q_1s_1 - m_1q_1s_1 \geq 0)\\&\quad = ((\neg q_1 \vee \neg s_1) \wedge \tau(0 \geq 0)) \vee ((q_1 \wedge s_1) \wedge \tau(-m_1 + 1 \geq 0))\\&\quad = ((\neg q_1 \vee \neg s_1) \wedge True) \vee ((q_1 \wedge s_1) \wedge \tau(-m_1 + 1 \geq 0))\end{aligned}$$

Since we have:

$$\begin{aligned}&\tau(-m_1 + 1 \geq 0) = (\neg m_1 \wedge \tau(1 \geq 0)) \vee (m_1 \wedge \tau(0 \geq 0))\\&\quad = (\neg m_1 \wedge True) \vee (m_1 \wedge True)\\&\quad \Leftrightarrow \neg m_1 \vee m_1\\&\quad \Leftrightarrow True\end{aligned}$$

we conclude:

$$\begin{aligned}&\tau(q_1s_1 - m_1q_1s_1 \geq 0)\\&\quad = ((\neg q_1 \vee \neg s_1) \wedge True) \vee ((q_1 \wedge s_1) \wedge ((\neg m_1 \wedge True) \vee (m_1 \wedge True)))\\&\quad \Leftrightarrow (\neg q_1 \vee \neg s_1) \vee (q_1 \wedge s_1)\\&\quad \Leftrightarrow (\neg q_1 \vee \neg s_1) \vee \neg(\neg q_1 \vee \neg s_1)\\&\quad \Leftrightarrow True\end{aligned}$$

The following results establish the correctness and completeness of our technique.

Theorem 13 (Correctness) *Let $P \in \mathbb{R}[X_1, \ldots, X_n]$ be a polynomial. If $\tau(\rho(P) \geq 0)$ (resp. $\tau(\rho(P) > 0)$) holds for some $x_1, \ldots, x_n \in N_1$, then $P(x_1, \ldots, x_n) \geq 0$ (resp. $P(x_1, \ldots, x_n) > 0$).*

Proof. We give the proof for the 'weak' case; the strict one is analogous. We prove, by induction on the number N of monomials with variables in $\rho(P)$, that whenever $\tau(\rho(P) \geq 0)$ holds for some $x_1, \ldots, x_n \in N_1$, then $\rho(P)(x_1, \ldots, x_n) \geq 0$. By Proposition 5, this implies that $P(x_1, \ldots, x_n) \geq 0$.

If $N = 0$, then $\rho(P)$ is a constant polynomial $\rho(P) = K \in \mathbb{R}$. Thus, $\tau(\rho(P) \geq 0)$ is either $True$ or $False$ depending on the value of K. In particular, if $\tau(\rho(P) \geq 0)$ holds this means that $K \geq 0$. Hence $\rho(P) = K \geq 0$.

If $N > 0$, then we can write $\rho(P) = cX_1 \cdots X_m + Q$ for some $c \in \mathbb{R}$, $0 < m \leq n$, and $Q \in \mathbb{R}[X_1, \ldots, X_n]$. Let $\mathcal{X} = \{X_1, \ldots, X_m\}$. If $\tau(\rho(P) \geq 0)$ holds for the sequence $(x_1, \ldots, x_n) \in N_1^n$ viewed as a propositional assignment, then we have the following (mutually exclusive) cases:

1. $\bigvee_{1 \leq i \leq m} \neg X_i$ holds for the sequence $(x_1, \ldots, x_n)$. Hence, since $\tau(\rho(P) \geq 0)$ holds, by definition of τ we know that $\tau(\mathsf{rmM}_{\mathcal{X}}(Q) \geq 0)$ also holds for $(x_1, \ldots, x_n)$. Since $\mathsf{rmM}_{\mathcal{X}}(Q)$ contains at most as many monomials as Q, by the Induction Hypothesis, we know that $\mathsf{rmM}_{\mathcal{X}}(Q)(x_1, \ldots, x_n) \geq 0$. Since $\bigvee_{1 \leq i \leq m} \neg X_i$ holds for the sequence $(x_1, \ldots, x_n)$, by Lemma 3, $\mathsf{rmM}_{\mathcal{X}}(Q)(x_1, \ldots, x_n) = Q(x_1, \ldots, x_n)$. Thus, $Q(x_1, \ldots, x_n) \geq 0$. Furthermore, under the considered conditions, we have that $cx_1 \cdots x_n = 0$, hence $\rho(P)(x_1, \ldots, x_n) = Q(x_1, \ldots, x_n)$ and $\rho(P)(x_1, \ldots, x_n) \geq 0$.

2. $\bigwedge_{1 \leq i \leq m} X_i$ holds for the sequence $(x_1, \ldots, x_n)$. Hence, we know that $\tau(\mathsf{rmV}_{\mathcal{X}}(Q) + c \geq 0)$ also holds for the sequence $(x_1, \ldots, x_n)$ and, by the Induction Hypothesis, we can assume that $\mathsf{rmV}_{\mathcal{X}}(Q)(x_1, \ldots, x_n) + c \geq 0$. Since all variables in $\mathcal{X}$ take value 1 in $(x_1, \ldots, x_n)$, by

Lemma 4 $\mathsf{rmV}_{\mathcal{X}}(Q)(x_1,\ldots,x_n) = Q(x_1,\ldots,x_n)$. Therefore, $\mathsf{rmV}_{\mathcal{X}}(Q)(x_1,\ldots,x_n) + c = Q(x_1,\ldots,x_n) + c \geq 0$. Since, under the considered conditions, $\rho(P)(x_1,\ldots,x_n) = c + Q(x_1,\ldots,x_n)$, we conclude that $\rho(P)(x_1,\ldots,x_n) \geq 0$.

□

Theorem 14 (Completeness) *Let $P \in \mathbb{R}[X_1,\ldots,X_n]$ be a polynomial and $x_1,\ldots,x_n \in N_1$. If $P(x_1,\ldots,x_n) \geq 0$ (resp. $P(x_1,\ldots,x_n) > 0$), then $\tau(\rho(P)(X_1,\ldots,X_n) \geq 0)$ (resp. $\tau(\rho(P)(X_1,\ldots,X_n) > 0)$) holds for the sequence $(x_1,\ldots,x_n)$ viewed as a truth assignment.*

Proof. Again, we give the proof for the 'weak' case only. Proposition 5 can be used to start with $\rho(P)(x_1,\ldots,x_n) \geq 0$ instead of $P(x_1,\ldots,x_n) \geq 0$. We also proceed by induction on the number N of monomials with variables in $\rho(P)$ to prove that that whenever $\rho(P)(x_1,\ldots,x_n) \geq 0$ for some $x_1,\ldots,x_n \in N_1$, then $\tau(\rho(P) \geq 0)$ holds.

If $N = 0$, then $\rho(P)$ is a constant polynomial $\rho(P) = K \in \mathbb{R}$. Thus, $\rho(P) = K \geq 0$ means that $\tau(\rho(P) \geq 0)$ is $True$ as required.

If $N > 0$, then we write $\rho(P) = cX_1 \cdots X_m + Q$ for some $c \in \mathbb{R}$, $0 < m \leq n$, and $Q \in \mathbb{R}[X_1,\ldots,X_n]$. Let $\mathcal{X} = \{X_1,\ldots,X_m\}$. If $P(x_1,\ldots,x_n) \geq 0$ for some $x_1,\ldots,x_n \in N_1$, then we consider the following (mutually exclusive) cases:

1. $\bigvee_{1\leq i\leq m} \neg X_i$ holds for the sequence $(x_1,\ldots,x_n)$. Thus, $cx_1 \cdots x_m = 0$ and $\rho(P)(x_1,\ldots,x_n) = Q(x_1,\ldots,x_n) \geq 0$. By Lemma 3, $Q(x_1,\ldots,x_n) = \mathsf{rmM}_{\mathcal{X}}(Q)(x_1,\ldots,x_n) \geq 0$. Since $\mathsf{rmM}_{\mathcal{X}}(Q)$ does not contain more monomials than Q, by the Induction Hypothesis, $\tau(\mathsf{rmM}_{\mathcal{X}}(Q) \geq 0)$ holds for $(x_1,\ldots,x_n)$ and, by definition of τ, $\tau(\rho(P) \geq 0)$ also holds for $(x_1,\ldots,x_n)$.
2. $\bigwedge_{1\leq i\leq m} X_i$ holds for the sequence $(x_1,\ldots,x_n)$. Hence, $cx_1 \cdots x_m = c$ and $\rho(P)(x_1,\ldots,x_n) = Q(x_1,\ldots,x_n)+c \geq 0$. By Lemma 4, we can write $\mathsf{rmV}_{\mathcal{X}}(Q)(x_1,\ldots,x_n) = Q(x_1,\ldots,x_n)$. Therefore, $\mathsf{rmV}_{\mathcal{X}}(Q)(x_1,\ldots,x_n)+c = Q(x_1,\ldots,x_n)+c \geq 0$. By the Induction Hypothesis, $\tau(\mathsf{rmV}_{\mathcal{X}}(Q)+c \geq 0)$ holds for the sequence $(x_1,\ldots,x_n)$ and, by definition of τ, $\tau(\rho(P) \geq 0)$ also holds for $(x_1,\ldots,x_n)$.

□

Note that the SAT encoding has a worst-case time complexity (and the size of the encoding) which is exponential in the maximum number of monomials. This is the case when neither the rmM nor the rmV operations are able to decrease the number of monomials of their respective arguments. Thus, one invocation of τ on a constraint of the form $p \geq 0$ where p consists of n monomials leads to two invocations of τ on arguments whose polynomials have $n-1$ monomials each. In the end, the 2^n calls of $\tau(i \geq 0)$ for all $i \in 0, 1, ..., 2^n - 1$ will be invoked. For instance, consider the following constraint:

$$1 * a_0 * a_1 + 2 * a_1 * a_2 + 4 * a_2 * a_3 + ... + 2^{n-1} * a_{n-1} * a_n \geq 0$$

Our translation yields a propositional formula whose size is exponential in the number of monomials

$$\begin{aligned}
&\tau(1 * a_0 * a_1 + 2 * a_1 * a_2 + 4 * a_2 * a_3 + ... + 2^{n-1} * a_{n-1} * a_n \geq 0)\\
&\quad = ((\neg a_0 \vee \neg a_1) \wedge \tau(\mathsf{rmM}_{\{a_0,a_1\}}(2a_1a_2 + 4a_2a_3 + ... + 2^{n-1}a_{n-1}a_n) \geq 0)) \vee\\
&\qquad ((a_1 \wedge a_2) \wedge \tau(\mathsf{rmV}_{\{a_0,a_1\}}(2a_1a_2 + 4a_2a_3 + ... + 2^{n-1}a_{n-1}a_n) + 1 \geq 0))\\
&\quad = ((\neg a_0 \vee \neg a_1) \wedge \tau(2a_1a_2 + 4a_2a_3 + ... + 2^{n-1}a_{n-1}a_n \geq 0)) \vee\\
&\qquad ((a_1 \wedge a_2) \wedge \tau(2a_2 + 4a_2a_3 + ... + 2^{n-1}a_{n-1}a_n + 1 \geq 0))
\end{aligned}$$

The size of the encoding could be taken as an indicator of the efficiency of the whole system, but we have to take into account that the SAT problem is an NP-complete problem. Thus, the performance of the solver depends on the considered heuristics. Moreover, modern SAT solvers take propositional formulas in CNF format as input. In order to do so, it is possible to convert any propositional formula into an equivalent CNF using De Morgan's laws. However, this transformation leads to an exponential blowup in the size of formula. The conversion of propositional formulas to CNF becomes feasible with the transformation presented by Tseitin in [Tse68]. However, such a transformation introduces linearly many new variables, that is, the search space grows exponentially.

On the other hand, it should be necessary to study whether this worst-case behaviour occurs in practice. This is not a trivial task, one first should try to predict the structure of the polynomial constraints coming from termination problems.

5.2 CSP-based techniques

As said before, a termination problem may become a *constraint solving problem*. Hence, the natural way to solve this new problem consists on using techniques coming from the contraint problem community (see, e.g., [RVW06]).

In this section, we start introducing the most fundamental concepts of Constraint Satisfaction Problem. In the remaining, we introduce our contribution to the development of algorithms based on constraint satisfaction techniques for solving polynomial constraint coming from termination problems (see Section 5.2.2). The material in this section has been published in [LNS09].

5.2.1 Constraint Satisfaction Problems

According to [Tsa95], the Constraint Satisfaction Problem (CSP), basically, is a problem composed of a finite set of variables, each of which is associated with a finite domain, and a set of constraints that restricts the values that the variables can simultaneously take. Formally, a CSP $\mathcal{P}$ is characterized by a tuple $\mathcal{P} = (X, D, C)$ where:

- X is an n-tuple of variables $(x_1, x_2, \ldots, x_n)$,
- D is a corresponding n-tuple of domains $(D_1, D_2, \ldots, D_n)$ such that D_i is the domain of x_i, i.e., the possible values for the variables x_i,
- C is a t-tuple of constraints $(C_1, C_2, \ldots, C_t)$, where C_i is a relation between variables.

The task of the constraint satisfaction problem is to find for each variable, a value from the domain of the variable satisfying all the constraints. Formally, a solution to a CSP $\mathcal{P}$ is an n-tuple $S = (s_1, s_2, \ldots, s_n)$ where $s_i \in D_i$ and each C_j from $\mathcal{P}$ is satisfied. Sometimes we denote the assignment of the variable $x_i \in X$ to the value $d \in D_i$ by $x_i \mapsto d$. In some cases, several possible solutions exist, then in a given task one may require to find the set of all solutions or just to find any solution. When the set of solution is empty the CSP is *unsatisfiable*. Solving CSP is trivial using a *generate-and-test* algorithm. However, this approach is inefficient in practice. In order to improve the performance of CSP solvers, several algorithms have been proposed to reduce the search space. These algorithms can be classified in two categories: *inference* and *search*, and combination of both. The basic inference techniques for CSP are the consistency algorithms for constraint propagation. On the other hand, Backtracking [GB65] is the fundamental search method for CSP.

The Constraint Satisfaction Problem is a simple but powerful framework which captures a wide range of significant applications in fields as diverses as artificial intelligence, operational research, scheduling, graph algorithm and computer visions, although new applications appear every year. Therefore, an important strength of the CSP comes from its flexibility.

5.2.2 Solving polynomial constraints over small domains of rational numbers

In [CMTU06], Contejean et al., following the ideas in the literature of *constraint logic programming over finite domains* (see, e.g., [CD96]) developed an algorithm for solving Diophantine constraints over finite domains. During several years, the tool box C*i*ME [CMMU03] in which the algorithm was implemented for the first time, was considered to be the fastest (maybe the unique) Diophantine solver for termination tools. The algorithm (with minor variants) was incorporated in the earlier versions of termination tools like AProVE [GST06], Polytool [TS07], TTT [HM07], and MU-TERM [Luc04].

The C*i*ME algorithm was used in the earlier version of MU-TERM as an extended Diophantine solver [Luc04]. C*i*ME is only able to solve Diophantine inequations yielding nonnegative integers as solutions. As a result of the study of polynomial interpretations over the reals [Luc05], MU-TERM dealt with the task of making the use of rational numbers compatible with this limitation: given a Diophantine constraint $e_1 \geq e_2$ containing one occurrence of p/q in e_1 or e_2, we obtain an equivalent constraint $q \cdot e_1 \geq q \cdot e_2$. Then, the multiplication of q is propagated to all coefficients in e_1 and e_2, thus removing the occurrences of q in the denominator of any rational coefficient involving it. This simplification process is repeated until no rational coefficient is present. This process led to the first automatic termination proofs using polynomial interpretation over the rationals. But, of course, this process leads to introduce new variables, so that this approach is not very efficient in practice.

Therefore, the main problem when attempting to use polynomial interpretation is that one needs efficient an suitable methods to find polynomial interpretations over the rationals automatically. In order to do so, the idea is to develop an algorithm to solve polynomial contraints over finite subsets of appropriate real numbers, i.e., a constraint solving algorithm dealing directly with real coefficients and without introducing more complexity.

An algorithm based on this idea was described in [Luc07]. The algorithm takes benefit from a suitable choice of domains for the coefficients and an appropriate representation of the polynomial constraints which permit both fast arithmetic and the use a number of techniques for safely avoiding a complete exploration of the search space. On the basis of this algorithm, we developed a CSP-based algorithm for solving polynomial constraints over the rational. In the following, we describe this algorithm.

Finite domains of powers of 2

The domain of the variables is traditionally selected by the user among the natural numbers, for instance, $N_1 = \{0, 1\}$, $N_2 = \{0, 1, 2\}$, etc. The algorithm works for subsets D of rational numbers (actually powers of 2): $D \subseteq D_m = \{0\} \cup \{\pm(2^i) \mid i \in \mathbb{Z}, 0 \leq |i| \leq m\}$ for $m \in \mathbb{N}$ or square roots of powers of 2: $D \subseteq \overline{D}_m = \{0\} \cup \{\pm(2^{\frac{i}{2}}) \mid i \in \mathbb{Z}, 0 \leq |i| \leq 2m\}$. We write D_m^+ if we restrict the attention to non-negative numbers. In particular, $D_1^+ = \{0, 2^{-1}, 2^0, 2^1\} = \{0, \frac{1}{2}, 1, 2\}$ includes the non-negative coefficients used (by default) in all currently available termination tools.

Restricting the attention to such kind of domains allows us reducing the costs of polynomial *arithmetics*, see [Luc07] for details.

Representation of polynomial constraints

The algorithm deals with parametric polynomials P under the form $P = (V, M, N, K)$ for sets of variables V and polynomials $M, N \in \mathbb{N}[X_1, \ldots, X_n]$ and $K \in \mathbb{Z}$, i.e., we represent P by considering the variables V appearing in P (which actually correspond to the parametric coefficients whose rational values will be given as a result of the constraint satisfaction process), the positive monomials $\mathsf{m} \in M$ on one side, the negative monomials $\mathsf{n} \in N$, and the constant coefficient K (which can be positive, negative or null): $P = M - N + K$. Let Pol be the set of polynomials as above. Constraints are sets $C \subseteq Pol \times \{weak, strict\}$ of pairs $(P, cond)$ where $cond$ indicates how the (basic) constraint $c = (P, cond)$ compares P to 0: in a *weak* ($P \geq 0$) or *strict* ($P > 0$) way.

Example 20 *The constraint $a_0m_2 + m_0 \geq 0$ is represented as follows:*

$$((\{a_0, m_2, m_0\}, \{a_0m_2, m_0\}, \varnothing, 0), weak)$$

Recall that the variables represent the parametric coefficients of the polynomials which interpret the symbols in the signature $\mathcal{F}$. Let $Var(P) \subseteq \{X_1, \ldots, X_n\}$ be the set of variables occurring in P and $Var(C) = \bigcup_{(P,cond)\in C} Var(P)$ be the set of variables in C. A solution σ of C is a mapping $\sigma : Var(C) \to D$ such that $(\sigma(P), cond)$ holds for all $(P, cond) \in C$, i.e., $P(\sigma(X_1), \ldots, \sigma(X_m)) \geq 0$ (resp. $P(\sigma(X_1), \ldots, \sigma(X_m)) > 0$) if $cond = weak$ (resp. $cond = strict$) and $Var(P) = \{X_1, \ldots, X_m\}$.

Constraint propagation

The constraint solving algorithm makes extensive use of *partial evaluation* of polynomials P w.r.t. one of its variables for doing *constraint propagation*. For instance, given $P(X_1, X_2, \ldots, X_n)$ and $d \in D$, we could need to obtain $P_{1,d}(X_2, \ldots, X_n) = P(d, X_2, \ldots, X_n)$. This involves the partial evaluation of *each* monomial $\mathsf{m} = cX_1^{\alpha_1} \cdots X_n^{\alpha_n}$ in P and the reconfiguration of the obtained polynomial as a tuple (V, M, N, K).

An important aspect of the algorithm is performing frequent *partial checkings* of the constraints in order to cut the search space. This means that we are often able to conclude the truth or falsity of a basic constraint $c = (P, cond)$ *with variables* without instantiating any variable in P. A (three valued) predicate $checkCS$ performs this task. $checkCS(c)$ returns either *true* if c is definitely true, or *false* if c is definitely false, or *??* otherwise. According to the representation $P = (V, M, N, K)$, and since we use domains D of *non-negative* numbers, we have the following cases:

1. $M \equiv 0 \wedge K < 0 \Rightarrow P \not\geq 0 \wedge P \not> 0$.

2. $N \equiv 0 \wedge K > 0 \Rightarrow P \geq 0 \wedge P > 0$.
3. $N \equiv 0 \wedge K = 0 \Rightarrow P \geq 0$.
4. $M \equiv 0 \wedge K = 0 \Rightarrow P \not> 0$.

here $M \equiv 0$ means that M is identically null.

Example 21 *By using rule 3 above we can remove (or definitely replace by True) constraints such as*

$$\begin{aligned} a_0 m_2 + m_0 &\geq 0 \\ m_1 s_0 + m_2 s_0 &\geq 0 \\ q_2 s_1 &\geq 0 \end{aligned}$$

Let β be the maximum element of D. Then, for all $x_1, \ldots, x_n \in D$,

- $K - N(\beta, \ldots, \beta) \leq P(x_1, \ldots, x_n)$, and
- $P(x_1, \ldots, x_n) \leq M(\beta, \ldots, \beta) + K$

This leads to the following:

1. $M(\beta, \ldots, \beta) + K < 0 \Rightarrow P \not\geq 0 \wedge P \not> 0$.
2. $K - N(\beta, \ldots, \beta) > 0 \Rightarrow P \geq 0 \wedge P > 0$.
3. $K - N(\beta, \ldots, \beta) \geq 0 \Rightarrow P \geq 0$.
4. $M(\beta, \ldots, \beta) + K = 0 \Rightarrow P \not> 0$.

In particular, if $\beta = 1 = 2^0$, then we can easily compute $M(\beta, \ldots, \beta) + K$ and $K - N(\beta, \ldots, \beta) + K$ by just adding the *coefficients* of the corresponding monomials, and then adding (or substracting from) the constant K.

The constraint solving algorithm

We describe the algorithm by means of two mutually recursive functions *solveCS* and *solveCSvar* [Luc07]. The initial call is $solveCS(D, [\ ], [\ ], C)$, where D is a domain (of coefficients) where the variables take values and C is the constraint to be solved.

```
$\mathbf{solveCSvar}(D, V, pSol, \{c\} \cup C)$
  **if** $CD_{Pol} = CD_{fail}$ **then** $\varnothing$
  **else** $\bigcup_{(c,d) \in CD_{unknown}} \mathbf{solveCS}(D, x_i : V, d : pSol, \{c\} \cup C(d))$
       $\cup \bigcup_{(c,d) \in CD_{true}} \mathbf{solveCS}(D, x_i : V, d : pSol, C(d))$
**where**
  $(P, cond) = c$
  $(X \cup \{x_i\}, \_, \_, \_) = P$
  $CD_{Pol} = \{((pEval(P, i, d), cond), d) \mid d \in D\}$
  $CD_{fail} = \{(c, \_) \in CD_{Pol} \mid checkCS(c) = false\}$
  $CD_{true} = \{(c, \_) \in CD_{Pol} \mid checkCS(c) = true\}$
  $CD_{unknown} = \{(c, d) \in CD_{Pol} \mid checkCS(c) = ??\}$
  $C(d) = \{pEvalCS(c, x_i, d) \mid c \in C\}$
```

Figure 5.3: Constraint solving algorithm: **solveCSvar**

1. $solveCS(D, V, pSol, C)$ performs an initial checking of all basic constraints $(P, cond)$ in the constraint C by using $checkCS$. Here, D and C are as above, V is a set of (already visited) variables which are expected to take values on D, and $pSol$ is a list of partial solutions to variables in V. If all constraints are true, then a singleton containing a pair $(V, pSol)$ consisting of the list of previously visited variables V and the list $pSol$ of partial solutions for these variables is returned. A partial solution is just a list $d_1, \ldots, d_k$ of values which correspond to the current list of visited variables $x_1, \ldots, x_k$, i.e., $x_i \mapsto d_i$ will be a binding of the final solution of the constraint. When the final solution is returned, variables x which were not instantiated receive a binding $x \mapsto d$ for an arbitrary $d \in D$ (typically $x \mapsto 0$).

2. $solveCSvar(D, V, pSol, C)$ tries values $d \in D$ on a variable x_i occurring in a constraint $c = (P, cond)$ in C. The instantiation of x_i with a value d yields a new constraint $c_{i,d} = (P_{i,d}, cond)$ consisting of the partial evaluation $P_{i,d}$ of P with d on the variable x_i and the same condition $cond$. The constraint $c_{i,d}$ is checked by using $checkCS$ and if the inconsistency of $c_{i,d}$ is shown, then d is discarded as a possible value for solving c on x_i. Otherwise, the variable x_i is recorded as 'visited' and the value d which permits to make progress is registered in the list of tuples which are partial solutions. Also, each constraint in $C - \{c\}$ is partially evaluated w.r.t. x_i and d as above and a new problem $C_{i,d}$ is raised. If $c_{i,d}$ is found true, then the constraint solving process continues with $C_{i,d}$. If nothing can be said about $c_{i,d}$, then the constraint solving process continues with $\{c_{i,d}\} \cup C_{i,d}$.

```
solveCS($D, V, pSol, C$)
  if $C_{fail} \neq \varnothing$ then $\varnothing$
  else if $C_{noTrue} = \varnothing$ then $\{(V, pSol)\}$
    else solveCSvar($D, V, pSol, C'$)
where
  $C_{noTrue} = \{c \in C \mid checkCS(c) \neq true\}$
  $C_{fail} = \{c \in C_{noTrue} \mid checkCS(c) = false\}$
  $C' = \{c \in C_{noTrue} \mid checkCS(c) = ??\}$
```

Figure 5.4: Constraint solving algorithm: **solveCS**

The complete description of the two functions is in Figure 5.8 and 5.4.

Improving the performance using CSP techniques

The algorithm presented in the previous section is considered a non-binary CSP with non-linear and global constraints. Many of the filtering (such as node-consistency and arc-consistency) and solving techniques cannot be directly applied to this problem due to the fact that they are focused to binary CSPs. Although some of these techniques can be extended to non-binary CSPs, they do not maintain the same level of efficiency.

In this section, we present a CSP algorithm for solving this problem. We have modeled the problem as a CSP in such a way that each variable maintains its own domain. Here, we present all filtering techniques and the new algorithm. The new algorithm improves the computational complexity of the previous version and also maintains a good behavior compared to other well known state of the art approaches.

The constraints in the CSP model are polynomials with $p + 1$ monomials with integer coefficients k_i with sign $(-1)^{n_i}$ for $1 \leq i \leq p$ consisting of q variables x_j for $1 \leq j \leq q$ raised to integer exponents $r_{ij} \geq 0$ within the i-th monomial:

$$\sum_{i=1}^{i=p} (-1)^{n_i} k_i \prod_{j=1}^{j=q} x_{ij}^{r_{ij}} + (-1)^{n_0} k_0 \geq 0, \qquad n_i \in \{1,2\}, k_i \in \mathbb{N}, i \geq 0; \tag{5.1}$$

$$\sum_{i=1}^{i=p} (-1)^{n_i} k_i \prod_{j=1}^{j=q} x_{ij}^{r_{ij}} + (-1)^{n_0} k_0 > 0, \qquad n_i \in \{1,2\}, k_i \in \mathbb{N}, i \geq 0; \tag{5.2}$$

Without loss of generality, we consider the constraint of type (5.1) and (5.2) as polynomial representations $P \in \mathbb{Z}[X_1, \ldots, X_p]$, where variables $X_1, \ldots, X_p$ range on D.

We have developed some static and dynamic filtering techniques to prune search space in the resultant CSP. These techniques are classified as preprocessing techniques carried out before search and constraint propagation and filtering techniques carried out during search.

- **Node-Consistency (NC)** The simplest consistency technique is called *node consistency*. If the domain D of a variable X contains a value a that does not satisfy a unary constraint on X (i.e., a constraint occurring an unique variable X), then the instantiation of X to a will always result in immediate failure. Thus, the node inconsistency can be eliminated by simply removing those values from the domain D of each variable X whose instantiation leads to a unsatisfiable constraint.This technique can be directly applied to our problem.

- **Arc-Consistency (AC)** If the CSP is node-consistent, then unary constraints can be removed because they all are satisfied. A CSP is *arc-consistent* if for any pair of constrained variables x_i, x_j, and for every value a in D_i, there is at least one value b in D_j such that the assignments $x_i \mapsto a$ and $x_j \mapsto b$ satisfy the constraints for x_i and x_j. Any value in the domain D_i of variable x_i that is not arc-consistent can be removed from D_i since it cannot be part of any solution. We apply this technique to filter the domain of many variables of the problem. Due to our problem is modeled as a non-binary CSP, we must apply generalized arc-consistency. Thus, a non-binary CSP is Generalized Arc-Consistent (GAC) if and only if for any variable in a constraint and value that it is assigned, there exist compatible values for all the other variables in the constraint [Mor88]. However, due to temporal complexity, we only apply arc consistency to binary constraints and k-ary constraints once $k-2$ variables have been instantiated.

- **Constraint Ordering (CO).** It is well known that a search algorithm for constraint satisfaction requires the order in which variables and constraints are to be considered as well as the order in which the values are assigned to the variable on backtracking. Choosing the right order of variables, values and constraints can noticeably improve the efficiency of constraint satisfaction. Many of the heuristics that improve backtracking-based algorithms are based on *variable ordering*, *value ordering* [SF06] and constraint ordering [Sal08], due to the additivity of the variables, values and constraints.

  Due to the nature of the problem, it can be observed that a CSP composed by only constraints of type (5.1) and with $k_0 = 0$ can be directly solved by assigning value 0 to all variables. Most problems consist of a set of constraint of type (5.1) with $k_0 = 0$ and a single constraint of type (5.2). In this way, this constraint will be studied first. Furthermore some constraint are tightest than other. These constraints are composed mainly by negative monomials $n_i = 1$ so that the positive monomials must be composed by variables assigned to high values. Thus the constraint ordering is:

$$\sum_{i=1}^{i=p}(-1)^{n_i}k_i\prod_{j=1}^{j=q}x_{ij}^{r_{ij}}+(-1)^{n_0}k_0>0, \qquad n_i\in\{1,2\},k_i\in\mathbb{N},i\geq 0; \tag{5.3}$$

$$\sum_{i=1}^{i=p}-k_i\prod_{j=1}^{j=q}x_{ij}^{r_{ij}}+(-1)^{n_0}k_0\geq 0, \qquad n_i\in\{1,2\},k_i\in\mathbb{N},i\geq 0; \tag{5.4}$$

$$\sum_{i=1}^{i=p}(-1)^{n_i}k_i\prod_{j=1}^{j=q}x_{ij}^{r_{ij}}+(-1)^{n_0}k_0\geq 0, \qquad n_0\in\{1,2\},k_i\in\mathbb{N},i\geq 0; \tag{5.5}$$

- **Variable Ordering (varOrd).** We maintain the following variable ordering: choose the variables which participate in most constraints. This heuristic follows the first-fail principle which can be explained as "*To succeed, try first where you are most likely to fail*". This heuristic is complementary with the constraint ordering heuristic.
- **Value Ordering (valueOrd).** Once the decision is made to instantiate a variable, it may have several values available. Again, the order in which these values are considered can have substancial impact on the time to find the first solution. Our heuristic classifies the values of each variable according to the number of positive or negative monomials in which the variables is involved. Thus, the values of variables that participate in more positive monomials than negative ones are classified in decreasing order. Similarly, the values of variables that participate in more negative monomials than positive ones are classified in increasing order.

We describe our new CSP algorithm by means of four functions *preProcess*, *checking*, *solveCS*, and *solveCSvar*. The initial call is *preProcess*(D, V, C) where D is the domain of the variables. Now, we briefly describe these functions:

1. *preProcess*(D, V, C) performs an initial checking (by means of *checking*, see below) of all basic constraints (see Figures 5.5 and 5.6). If all constraints are true, then a possible solution is returned, and each variable $x \in V$ receives a binding $x \mapsto d$ for an arbitrary $d \in D_x$. If there are inconsistent constraints in C, then no solution is returned. Otherwise, a decreasingly ordered list which represents the variable ordering is generated, taking into account the number of constraints involving each variable (*varOrd*). The domain of each variable is also ordered with respect to the number of positive and negative monomials involving each variable (*valueOrd*). Finally, filtering techniques such as node-consistency and arc-consistency are applied in order to reduce the number of elements in the domains. If a variable x has a single element d in its domain, then x is instantiated with d in the constraint and $x \mapsto d$ becomes a binding of the final solution of the constraint. Variables

```
**preProcess**$(D, V, C)$ =
  **if** $C_{fail} \neq \varnothing$ **then** $\varnothing$
  **else if** $C_{noTrue} = \varnothing$ **then** $\{(V, sol)\}$
    **else solveCS**$(D'', V - V_{trivial}, (V_{trivial}, \{v \in V_{trivial} \mid D_v\}), C'', vOrd)$
**where**
  $(C_{noTrue}, C_{fail}, C') = checking(C)$
  $sol = \{SelectFirstValue(D(v)) \mid v \in V\}$
  $D' = Node/Arc - Consistency(D, V, C')$
  $vOrd = defineVariableOrd(D', V, C')$
  $D'' = defineValueOrd(D', V, C')$
  $V_{trivial} = \{v \in V : |D'_v| = 1\}$
  $C'' = \{pEvalCS(c, v, D'_v) \mid c \in C \ \wedge v \in V_{trivial}\}$
```

Figure 5.5: Constraint solving algorithm: **preProcess**

```
**checking**$(C)$ =
  $(C_{noTrue}, C_{fail}, C')$
**where**
  $C_{noTrue} = \{c \in C \mid checkCS(c) \neq true\}$
  $C_{fail} = \{c \in C_{noTrue} \mid checkCS(c) = false\}$
  $C' = \{c \in C_{noTrue} \mid checkCS(c) = ??\}$
```

Figure 5.6: Constraint solving algorithm: **checking**

whose domain is empty make all constraints containing them inconsistent; thus, no solution is returned. When the final solution is returned, variables x which were not instantiated receive a binding $x \mapsto d$ for an arbitrary $d \in D_x$ (typically $x \mapsto 0$).

2. *checking*(C) performs a checking of all basic constraints $(P, cond)$ in the constraint C by using *checkCS* and returning them partitioned as *trivial* (definitely true), *inconsistent* (definitely false) or *??*.

3. *solveCS*$(D, V, (V_{pSol}, pSol), C, vOrd)$ performs an initial checking of all basic constraints (see Figures 5.7 and 5.8). If all constraints are true, then a singleton containing a pair $(V_{pSol}, pSol)$ consisting of the list of previously visited variables V_{pSol} and the list $pSol$ of partial solutions for these variables is returned. Otherwise, over the non-trivial constraints

```
**solveCS**$(D, V, (V_{pSol}, pSol), C, vOrd) =$
  **if** $C_{fail} \neq \varnothing$ **then** $\varnothing$
  **else if** $C_{noTrue} = \varnothing$ **then** $\{VpSol, pSol\}$
    **else solveCSvar**$(D', V - V_{trivial}, (V_{trivial} \cup V_{pSol}, \{D_v \mid v \in V_{trivial}\} \cup pSol), C'', vOrd)$
**where**
  $(C_{noTrue}, C_{fail}, C') = checking(C)$
  $D' = Node/Arc - Consistency(D, V, C')$
  $V_{trivial} = \{v \in V : |D'_v| = 1\}$
  $C'' = \{pEvalCS(c, v, D'_v) \mid c \in C' \ \wedge v \in V_{trivial}\}$
```

Figure 5.7: Constraint solving algorithm: **solveCS**

is performed the node-consistency and arc-consistency filtering in order to prune inconsistent values. Then, trivial variables (a single element in its domain) are instantiated again. A partial solution is just a list $d_1, \ldots, d_k$ of values which correspond to the current list of visited variables $x_1, \ldots, x_k$, i.e., each $x_i \mapsto d_i$ will be a binding of the final solution of the constraint.

4. $solveCSvar(D, V, (V_{pSol}, pSol), C, vOrd)$ tries values $d \in D_{x_i}$ on a variable x_i occurring in a constraint $c = (P, cond)$ in C. The variable x_i is the most involved variable in P with respect to the initial constraint. The instantiation of x_i with all values $d \in D_{x_i}$ yields a new set of constraint CD_{Pol} whose elements $c_{i,d} = (P_{i,d}, cond)$ consisting of the partial evaluation $P_{i,d}$ of P with d on the variable x_i and the same condition *cond*. The constraint $c_{i,d}$ is checked by using *checking* and if the inconsistency of $c_{i,d}$ is shown, then d is discarded as a possible value for solving c on x_i. Otherwise, the variable x_i is removed from V and x_i together with the value d which permits to make progress is registered in the list of tuples which are partial solutions. Also, each constraint in $C - \{c\}$ is partially evaluated w.r.t. x_i and d as above and a new problem $C_{i,d}$ is raised. Then, $C_{i,d}$ is reordered with respect to the constraint ordering defined (CO). If $c_{i,d}$ is found true, then the constraint solving process continues with $C_{i,d}$. If nothing can be said about $c_{i,d}$, then the constraint solving process continues with $\{c_{i,d}\} \cup C_{i,d}$.

In Chapter 6, we evaluate empirically these algorithms and we show that the introduction of standard techniques for modeling and treating constraint solving problems as CSP problems leads to important improvements in the efficiency.

```
**solveCSvar**$(D, V, (V_{pSol}, pSol), \{c\} \cup C, vOrd) =$
    **if** $CD_{Pol} = C_{fail}$ **then** $\varnothing$
    **else** $\bigcup_{(c,d)\in CD'}$ **solveCS**$(D, V - x_i, (x_i \cup V_{pSol}, d \cup pSol), CO(\{c\} \cup C(d)), vOrd) \cup$
         $\bigcup_{(c,d)\in CD_{true}}$ **solveCS**$(D, V - x_i, (x_i \cup V_{pSol}, d \cup pSol), CO(C(d)), vOrd)$
**where**
    $(P, cond) = c$
    $x_i = SelectFirstVar(P, vOrd)$
    $CD_{Pol} = \{((pEval(P, i, d), cond), d) \mid d \in D_{x_i}\}$
    $(CD_{noTrue}, CD_{fail}, CD') = checking(CD_{Pol})$
    $C(d) = \{pEvalCS(c, x_i, d) \mid c \in C\}$
    $CO(Cs) = ConstraintOrdering(Cs)$
```

Figure 5.8: Constraint solving algorithm: **solveCSvar**

5.3 Linear Programming and Hybrid Methods

As far as we know, the first implementation of a completely automatic system for proving termination of TRSs using polynomial interpretations was reported by Steinbach [Ste94]. The basic idea was to transform the termination problem into a set of linear constraints, which were solved using *linear programming* techniques. This section is devoted to the study of methods based on linear programming for proving termination.

First, in Section 5.3.1 we give a short introduction to Linear programming. Afterwards, in Section 5.3.2, we describe methods for dealing with polynomial constraint from termination problems by means of Hybrid methods mixing Linear Programming and CSP solvers. The material in this section has been published in [LMNS08].

5.3.1 Operational Research and Linear Programming

Operational Research (OR) is an interdisciplinary branch of applied mathematics and formal science that uses methods such as mathematical modeling, statistics, and algorithms to arrive at optimal or near optimal solutions to complex problems.

Linear Programming (LP) is a technique for optimization of a linear objective function, subject to linear equality and inequality constraints. Informally, linear programming determines the way to achieve the best outcome in a given mathematical model given some list of requirements represented as linear equations. Formally, linear programming techniques permit a *direct* computation of the whole space of solutions of a constraint solving problem $A\vec{x} \geq \vec{b}$, where $A \in \mathbb{R}^{m\times n}$ is a $m \times n$-matrix of real numbers and $\vec{x}$ and $\vec{b}$ are vectors of n real numbers: $\vec{x}, \vec{b} \in \mathbb{R}^n$. No inspection of the search space of solutions is required.

A number of economic, industrial, financial, and military systems can be modeled by mathematical systems of linear inequalities and equations, so that, linear programming is one of the most successful disciplines within the field of operational research. The classical tool for solving the linear programming problem in practice is the class of *simplex algorithms* proposed and developed by George Dantzig [Dan63].

The most dramatic new development in operational research during the 1980s was the discovery of the *interior-point approach* to solving linear programming problems. This discovery was made in 1984 by Narendra Karmarkar [Kar84]. Although this particular algorithm experienced only mixed success in competing with the simplex method, the key solution concept appeared to have a great potential for solving huge linear programming problems beyond the reach of the simplex method.

Obviously, the key assumption in LP is that all its functions are linear. Although this assumption essentially holds for numerous practical problems, it frequently does not hold. Therefore, it is often necessary to deal directly with nonlinear programming problems. Unfortunately, no algorithm that solves every specific problem fitting this format is available. However, substantial progress has been made for some important special cases by making various assumptions about these functions. This is the case of Geometric Programming (GP) [BKVH07], where new developments lead to efficiently and reliable solutions of large-scale (GP) problems.

5.3.2 Linear programming techniques for termination problems

As said before, Steinbach [Ste94] reported the first implementation of a completely automatic system for proving termination of TRSs using polynomial interpretations. Steinbach proposed the use of *parametric* polynomial interpretations (over the reals) where the shape of polynomials is usually simple (e.g., linear polynomials, but see 3.7) and the coefficients of the polynomials are *variables* which range on real numbers which are greater than or equal to 1. The constraints which are obtained following Steinbach's method are *linear constraints over the reals* which are then solved by the *Simplex* algorithm. The obtained solutions provide concrete values for the variable coefficients of the parametric polynomial interpretation which witness termination of the original TRS.

An appealing aspect of Steinbach's proposal is the use of standard methods from linear programming to solve polynomial constraints over real domains. Unfortunately, Steinbach's method has a number of drawbacks. In particular, due to the use of Simplex, he needs to *linearize all inequalities* which are obtained during the process. His linearization procedure [Ste94, Lemma 4.9] leads to an *incomplete* and rather *unpractical* algorithm.

Furthermore, his constraints are expected to be solved over a domain D of real numbers (actually an interval) which are bigger than or equal to 1: $D = [\mu, +\infty)$ for some $\mu \geq 1$,which is quite a strong restriction in modern, dependency-pair-based proofs of termination (see Chapter 4). In Section 4, it is proved that some termination problems *require* the use of polynomial interpretations with coefficients $c \in [0, 1)$ *below* 1 if polynomial interpretations are used to achieve the proof. Thus, Steinbach's approach could not be used to solve such kind of problems.

In the following section, we discuss possible combinations of *Linear Programming* (LP) and *Constraint Satisfaction Problems* (CSP) techniques which can be appropriate for their implementation as part of termination tools.

The main idea is considering a constraint $\mathcal{C}$ as decomposed into a linear part $\mathcal{L}$ and a nonlinear part $\mathcal{D}$. Then, we can use LP-based techniques to obtain a *complete* and *exact* representation of the space of solutions of $\mathcal{L}$ as a *convex polyhedron* having a finite representation in terms of a finite number of tuples of *rational* numbers (which can be represented and operated by using standard techniques without introducing lack of precision). Then, we can use CSP-like techniques to deal with the nonlinear part $\mathcal{D}$ of the constraint, hopefully taking advantage of the explicit representation of the set of solutions for the linear part. These ideas have been published in [LMNS08].

A complete hybrid constraint solving method

The space of solutions to a linear constraint-solving problem $A\vec{x} \geq \vec{b}$ is a convex polyhedron P. A convex polyhedron $P \subseteq \mathbb{R}^n$ admits a finite representation $P = Q + C$ where Q is a polytope and C is a polyhedral cone (see, e.g., [Sch86, Corollary 7.1b]). This means that, if $P \neq \varnothing$, there are s *points* $\vec{p}_1, \ldots, \vec{p}_s \in \mathbb{R}^n$ and t *vectors* $\vec{v}_1, \ldots, \vec{v}_t \in \mathbb{R}^n$ such that $P = conv.hull(\{\vec{p}_1, \ldots, \vec{p}_s\}) + cone(\{\vec{v}_1, \ldots, \vec{v}_t\})$. There are algorithms which, given a linear constraint $A\vec{x} \geq \vec{b}$ compute $\vec{p}_1, \ldots, \vec{p}_s \in \mathbb{R}^n$ and $\vec{v}_1, \ldots, \vec{v}_t$ to obtain a finite representation of the space of solutions P.

Therefore, for all $\vec{x} \in P$, there are s non-negative numbers $\alpha_1, \ldots, \alpha_s \geq 0$ such that $\sum_{i=1}^{s} \alpha_i = 1$ and t non-negative numbers $\beta_1, \ldots, \beta_t \geq 0$ such that $\vec{x} = \sum_{i=1}^{s} \alpha_i \vec{p}_i + \sum_{j=1}^{t} \beta_j \vec{v}_j$.

Given a constraint-solving problem $\mathcal{C} = \mathcal{L} \uplus \mathcal{D}$ where $\mathcal{L}$ consists of (all) linear constraints in $\mathcal{C}$ involving variables $x_1, \ldots, x_n$, we can obtain the finite representation $P = Q \cup C$ of the convex polyhedron P which represent all solutions to $\mathcal{L}$. Now we can *replace* the variables $x_1, \ldots, x_n$ in the nonlinear part $\mathcal{D}$ of $\mathcal{C}$ by their corresponding expressions $x_j = \sum_{i=1}^{s} \alpha_i p_{i,j} + \sum_{j=1}^{s} \beta_j v_{i,j}$.

In this way, for each constraint $P(x_1, \dots, x_n, x_{n+1}, \dots, x_p) \geq 0$ we obtain a new constraint

$$P'(A_1, \dots, A_s, B_1, \dots, B_t, x_{n+1}, \dots, x_p) \geq 0$$

for new (fresh) variables $A_1, \dots, A_s, B_1, \dots, B_t$ which play the role of α_i and β_i above. We obtain a new set $\mathcal{D}'$ of constraints as follows:

$$\begin{aligned}\mathcal{D}' \quad = \quad & \{P'(A_1, \dots, A_s, B_1, \dots, B_t, x_{n+1}, \dots, x_p) \geq 0 \mid P(x_1, \dots, x_n, x_{n+1}, \dots, x_p) \geq 0 \in \mathcal{D}\} \\ & \cup \{P'(A_1, \dots, A_s, B_1, \dots, B_t, x_{n+1}, \dots, x_p) > 0 \mid P(x_1, \dots, x_n, x_{n+1}, \dots, x_p) > 0 \in \mathcal{D}\} \\ & \cup \{A_1 + \cdots + A_s = 1\}\end{aligned}$$

where we implicitly assume that $A_1, \dots, A_s, B_1, \dots, B_t \geq 0$. Note that $\mathcal{D}'$ contains $N+1$ constraints if $\mathcal{D}$ contains N constraints.

Theorem 15 *$\mathcal{C}$ has a solution if and only if $\mathcal{D}'$ has a solution.*

The main problem with the procedure sketched above is that it can lead to sets $\mathcal{D}'$ containing *more complex* constraints than those in $\mathcal{D}$.

Example 22 *Consider the set of two constraints:*

$$(1)\ m_1 - 1 \geq 0$$
$$(2)\ m_1 s_1 - m_1 \geq 0$$

Constraint (1) *is linear and its space of solutions (for the variable m_1) can be represented by $\alpha\{1\} + \beta\{1\}$ for $\alpha = 1$ and all $\beta \geq 0$. We can use the identity $m_1 = 1 + B$ to transform constraint* (2)*, thus obtaining*

$$(2') \quad (1+B)s_1 - 1 - B = s_1 + Bs_1 - B - 1 \geq 0$$

In the following section we describe an iterative algorithm which proceeds in quite a different way.

A two-phase constraint-solving procedure

We present a two-phase procedure to solve polynomial constraints over the reals. To this end, we iteratively apply an LP technique to solve the linear problem and then a CSP solver is committed to solve the nonlinear part of the problem.

Let us consider C_0 the set of constraints. This set is composed by linear constraints LC_0 and nonlinear constraints NLC_0. Thus: $C_0 = LC_0 \cup NLC_0$. Without loss of generality, we consider that $LC_0 \neq \varnothing$. We represent $VarLC_0$ as the variables involved in LC_0 and $VarNLC_0$ as the variables involved only in NLC_0. The pseudo-code of the procedure is presented in Figure 5.9.

The procedure *TwoPhaseSolver* works as follows:

```
**TwoPhaseSolver**($C_0$)
$i = 0$;
$Sol_{LC} = \varnothing$;
$C_i \leftarrow RemoveTrivial(C_i)$;
$Fail = False$;
while $LC_i \neq \varnothing$ AND NOT $Fail$ {
    $k = 0$;
    $Sol_{LC_i} \leftarrow LCSolver(LC_i)$;
    $Sol_k = \varnothing$;
    while $Sol_{LC_i} \neq \varnothing$ AND $Sol_k = \varnothing$ {
        $k++$;
        Select $Sol_k \in Sol_{LC_i}$
        $Sol_{LC_i} \leftarrow Sol_{LC_i} \setminus \{Sol_k\}$;
        $C_{i+1} \leftarrow ConstraintPropagation(Sol_k, NLC_i)$;
        if NOT $Inconsistent(C_{i+1})$ then
        { $Sol_{LC} = Sol_{LC} \cup Sol_k$;
          $C_{i+1} \leftarrow RemoveTrivial(C_{i+1})$ }
        else $Sol_k = \varnothing$; }
    if $Sol_{LC_i} = \varnothing$ AND $Sol_k = \varnothing$ then $Fail = True$;
    else $i++$; }
if $Fail$ then $Sol = \varnothing$
else if $Sol_{LC} = \varnothing$ OR $NLC_i = \varnothing$ then $Sol \leftarrow Sol_{LC}$
  else {
    $Sol_{NLC} \leftarrow CSPSolver(NLC_i)$;
    if $Sol_{NLC} \neq \varnothing$ then $Sol \leftarrow Sol_{LC} \cup Sol_{NLC}$;
    else $Sol = \varnothing$; }
return $Sol$;
```

Figure 5.9: A two-phase constraint-solving procedure

1. The function $RemoveTrivial(C_i)$ eliminates all trivial linear and nonlinear constraints.

2. The function $LCSolver(LC_i)$ solves the corresponding linear programming problem; we can thought of this function as providing a possibly infinite set of solutions to the linear problem (usually obtained from the extreme points of the hyper-polyhedron which contains the solutions to the linear problem). The solution generated by the LP technique is a partial solution to the global problem. Thus, the problem becomes easier, because many variables have been assigned and now some nonlinear constraints eventually become linear.

3. The algorithm selects the k-th solution obtained by the linear problem which assigns values to all variables involved in these linear constraints.

4. We use $ConstraintPropagation(Sol_k, NLC_i)$ to propagate the effect of the assigned value to the remainder nonlinear constraints. If no inconsistency is found, then we take Sol_k as a new component of the final solution and continue with a new subset of linear constraints which is (eventually) obtained from the recently instantiated ones in NLC_i.

5. Finally, if all constraints remain nonlinear, then a CSP solver must solve the simplified

problem composed of nonlinear constraints. If a solution is found then the procedure builds a complete solution and returns it.

In principle, an infinite number of (boundary) solutions are possible for the linearized problems. This means that our whole procedure could run forever if the original constraint has no solution.

Example 23 *Let us consider the set of polynomial constraints presented in Example 14: We first remove the trivial constraints (1), (3) and (7) using RemoveTrivial. Constraint (2) is the unique linear constraint; it can be solved using Simplex:*

$$(2)\ m_1 - 1 \geq 0 \rightarrow m_1 = 1$$

Once m_1 is given a value, the constraint (4) becomes linear, so it is solved by using Simplex again:

$$(4')\ s_1 - 1 \geq 0 \rightarrow s_1 = 1$$

Once m_1 and s_1 are assigned, the constraints (5'), (9'), and (12') become trivial, so they can be removed by the RemoveTrivial function:

$$(5')\ m_2 - m_2 \geq 0$$
$$(9')\ q_1 - q_1 \geq 0$$
$$(12')\ Q_1 - Q_1 \geq 0$$

Finally, the remaining constraints are nonlinear constraints, so that the CSP solver is committed to solve the remaining nonlinear problem.

$$\begin{aligned}(6)&\ a_0q_1 + q_2s_0 + q_0 - a_0 \geq 0\\ (8')&\ q_1s_0 + q_2s_0 - m_0q_1 - q_2s_0 - s_0 \geq 0\\ (10')&\ - m_2q_1 \geq 0\\ (11)&\ Q_1s_0 - m_0Q_1 > 0\\ (13')&\ - m_2Q_1 \geq 0\end{aligned}$$

The assignments $m_2 = 0, q_1 = 1, Q_1 = 1, q_2 = 1, s_0 = 1, m_0 = 0, q_0 = 0, a_0 = 0$ satisfy the constraints above, and a solution to the global problem is found.

In Chapter 6, we evaluate empirically the performance of a prototype of two-phase-solver with a CSP solver. The evaluation show that the use of linear programming techniques reduces the total time for solving all problems. However, this approach returns inconsistency in some consistent problems due to the LP technique (Simplex) returns only one partial solution which does not take part in a global solution.

This situation could be improved if the LP technique could retrieve several solutions, so that the algorithm could backtrack when an inconsistency over the partial solution is found. But the improved solver would be incomplete as well, since we have to select a subset of all infinite solutions provided by the LP technique and we cannot predict a priori which partial solutions are consistent with the global solution.

5.4 Computer Algebra Techniques

Real Algebraic Geometry [BCR98] investigates the properties of sets of real numbers defined by means of polynomial constraints. The concepts, algorithms and tools coming from this field (see, e.g., [BPR03]) should be further explored and applied in termination of Term Rewriting Systems when polynomial orderings are used. So far, these techniques have been hardly considered, but we believe that they can lead to interesting results even in a theoretical perspective. Martin and Shand [MS00] already suggested that Gröbner basis and polynomial ideals [CLO97, Win96] should be considered as a uniform framework for understanding polynomial termination proofs. Rouyer's PhD Thesis has already explored the use of Sturm's Theorem to develop an algorithm to decide whether the sign of a given polynomial with integer coefficients change on a given n-dimensional rectangle [Rou91]. This property is then used in proofs of polynomial termination of TRSs. Recently, Lucas [Luc07] used Tarski-Seidenberg's principle to prove that considering polynomial interpretations with real algebraic coefficients[2] suffices for polynomial based termination proofs.

A promising technique, not considered in proof of termination, is the root classification of parametric polynomials. In [YX06], Yang and Xia proposed a method of quantifier elimination for quartics, i.e., polynomial constraints of the form: $(\forall x > 0)(x^4 + ax^3 + bx^2 + cx + d > 0)$ and $(\forall x > 0)(x^4 + ax^3 + bx^2 + cx + d \geq 0)$. Liang, Jeffrey, and Moreno [LJM08] proposed a new algorithm for the automatic computation of the complete root classification of a parametric polynomial on an interval. The idea is to collect all possible roots on the interval, together with the conditions that its coefficients must satisfy for each case. The package implementing this algorithm will be incorporated in future versions of *Maple*. These approaches do not completely fit with our setting, but they should be considered for future developments.

In previous chapters, we have studied that the satisfaction of constraints coming from poly-

[2]A real number $x \in \mathbb{R}$ is said to be *algebraic* if it satisfies an equation $x^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0 = 0$, of finite degree n where $a_i \in \mathbb{Q}$ for $0 \leq i \leq n-1$. A non-algebraic real number is said to be *transcendental*.

nomial interpretations over the naturals is, in general, undecidable [Mat70]. On the other hand, considering polynomial constraints over the reals, we obtain a constraint solving problem which is in the *first order theory of real closed fields.* Dershowitz [Der79] noticed that, according to Tarski [Tar51], the satisfaction of such polynomial constraints then becomes decidable. In other words, the termination problem can be encoded as an instance of the general decision problem for the first order theory of the reals closed fields investigated by Tarski as well. This was the main argument of Dershowitz for proposing the use of polynomials over the reals in proofs of termination. Consequently, instead of using the (efficient but incomplete) Hong and Jakuš criterion, Real Algebraic Geometry provides complete methods (quantifier elimination) for deciding about the satisfiability of polynomial constraints.

In this section, we only focus on *quantifier elimination* for sentences in the first-order logic of real closed fields.

The concepts introduced in this section have been taken from [Win96].

5.4.1 Quantifier elimination for Real Closed Fields

Many interesting problems of real algebraic geometry can be stated as systems of polynomial equations, inequations, usually with some structure of quantification. Quantifier elimination provides an approach to solving such problems. This is the case of the polynomial interpretation method.

The *elementary theory of real closed fields* (ETRCF) is the first-order theory with the constants 0 and 1, function symbols $+$, $-$, $\cdot$, predicate symbols $=$, $>$, $\geq$, $<$, $\leq$, $\neq$ (elementary algebra), and an axiom system consisting of the field axioms.

A standard atomic formula is an expression of the form $p \sim 0$, where p is a (multivariate) integral polynomial (the coefficients of these polynomials have to be integers) and $\sim$ is a predicate symbol in ETRCF. A standard formula is a formula in ETRCF in which the atomic formulas are all standard atomic formulas. A standard prenex formula is a standard formula in prenex form, i.e., a sequence of quantifiers followed by a quantifier free standard formula.

The problem of quantifier elimination for real closed fields can be expressed as: *for a given standard prenex formula φ, find a standard quantifier-free formula ψ such that ψ is equivalent to φ.*

Tarski [Tar51] proposed a quantifier elimination algorithm for transforming any formula of the theory of real closed fields into an equivalent formula that contains no quantifiers. He also showed how to decide whether a formula, which does not contain quantifiers and variables, is true. Also other approaches have been suggested [Sei54, Coh69]. Unfortunately, most of them suffer from a high complexity since the decision procedure is not elementary recursive. A real breakthrough was achieved by Collins with his CAD (Cylindrical Algebraic Decomposition) algorithm [Col75]. A recent work by Basu, Pollack and Roy [BPR96] presents a decision algorithm which is doubly exponential in the number of variables.

Our aim is to investigate the complexity of Quantifier Elimination in practice, concretely in proofs of termination of TRSs.

To this end, we use the Computer Algebra System *Mathematica* [Wol08] for computing cylindrical decomposition of a set of inequalities (using the command *CylindricalDecomposition*).

Example 24 *Let us consider the following set of constraints:*

$$\begin{aligned}
&(m_1 - 1)x + m_2 y + m_0 \geq 0\\
&(m_1 s_1 - m_1)x + (m_2 s_1 - m_2)y + (m_1 + m_2)s_0 \geq 0\\
&(M_1 s_1 - M_1)x + (m_2 s_1 - m_2)y + (M_1 + M_2)s_0 > 0
\end{aligned}$$

Note that, the variables x, y are universally quantified, but the coefficients ($m_0, m_1, m_2, s_0, s_1, M_1, M_2$) are existentially quantified (and precede the universal quantification in the formula). As said above, if all variables in a formula are quantified, then the equivalent quantifier-free formula is clearly either true or false. Consequently, we do not explicitly quantify the coefficients in the example.

Then, we compute the cylindrical decomposition in Mathematica:

```
> p1 = (m1−1)*x + m2*c0 + m0 >= 0

m0 + c0*m2+ (−1+m1)x >= 0

> p2 = (M1*s1 − M1)x + (m2*s1 − m2)y + (M1 + M2)s0 > 0

(M1 + M2)*s0 + (−M1 + M1*s1)x + (−m2 + m2*s1)y > 0

> p3 = (m1*s1 − m1)x + (m2*s1 − m2)y + (m1 + m2)s0 >= 0

(m1 + m2)s0 + (−m1 + m1*s1)x + (−m2 + m2*s1)y >= 0

> cineq = M1 >= 0 && M2 >= 0 && m1 >= 0 && m2 >= 0 && m0 >= 0 && s1 >= 0 && s0
    >= 0 && c0>= 0

M1 >= 0 && M2 >= 0 && m1 >= 0 && m2 >= 0 && m0 >= 0 && s1 >= 0 && s0 >= 0 && c0
    >= 0

> formula = ForAll[{x,y}, p1 && p2 && p3 && cineq]
```

```
forall{x,y} (m0 + x(-1 + m1) + c0*m2 >= 0 && (M1 + M2)s0 + x(-M1 + M1*s1) + y(-
    m2 + m2*s1) > 0 && (m1 + m2)s0 + x(-m1 + m1*s1) + y(-m2 + m2*s1) >= 0 && M1
    >= 0 && M2 >= 0 && m1 >= 0 && m2 >= 0 && m0 >= 0 && s1 >= 0 && s0 >= 0 && c0
    >= 0)

> vars = {x,y,M1,M2,m1,m2,m0,s1,s0,c0}

{x,y,M1,M2,m1,m2,m0,s1,s0,c0}

> TimeConstrained[CylindricalDecomposition[formula,vars,100]//Timing

(28.2978, (M1 = 0 && M2 > 0 && m1 = 1 && m2 >= 0 && m0 >= 0 && s1 = 1 && s0 >
    0 && c0 >= 0) || (M1 > 0 && M2 >= 0 && m1 = 1 && m2 >= 0 && m0 >= 0 && s1 =
    1 && s0 > 0 && c0 >= 0)

> FindInstance[(M1 = 0 && M2 > 0 && m1 = 1 && m2 >= 0 && m0 >= 0 && s1 = 1 &&
    s0 > 0 && c0 >= 0) || (M1 > 0 && M2 >= 0 && m1 = 1&& m2 >= 0 && m0 >= 0 &&
    s1 =1 && s0 > 0 && c0 >= 0), {M1,M2,m1,m2,m0,s1,s0,c0}, Reals]

{M1 -> 0,M2 -> 84,m1 -> 1,m2 -> 0,m0 -> 0,s1 -> 1,s0 -> 72,c0 -> 77}
```

The computation takes 28 seconds and returns the equivalent formula. A possible valid instantiation is provided as well:

$$\{M_1 = 0, M_2 = 84, m_1 = 1, m_2 = 0, m_0 = 0, s_1 = 1, s_0 = 72, c_0 = 77\}$$

The variable ordering given as input has an important influence in the algorithm:

```
> TimeConstrained[CylindricalDecomposition[formula,{x,y,M1,M2,m1,m0,s0,m2,c0,s1
    }],50]//Timing

{101.523,Aborted}

> TimeConstrained[CylindricalDecomposition[formula,{m1,s1,M1,c0,m2,M2,m0,s0,x,y
    }], 50]//Timing

{7.91344,m1 = 1 && s1 = 1 && ((M1 = 0 && c0 >= 0 && m2 >= 0 && M2 > 0 && m0 >= 0
     && s0 > 0) || (M1 > 0 && c0 >= 0 && m2 >= 0 && M2 >= 0 && m0 >= 0 && s0 >
    0))}
```

The previous example shows that the quantifier elimination algorithm is too costly in practice, due to its generality. Moreover, it heavily depends on the ordering of the input variables, and it is not easy to predict a suitable ordering for them. Hence, this leads to considering restricted classes of polynomial interpretations and other simplifications of this general procedure.

In [BPR03], a whole chapter is devoted to the Existencial Theory of the Reals. The decision problem for the existencial theory of the reals is to decide the truth or falsity of a sentence

$$(\exists X_1) \dots (\exists X_n) F(X_1, \dots, X_n)$$

where $F(X_1, \dots, X_n)$ is a standard atomic formula. Thus, this method is an special case of the general decision problem of the theory of the reals.

This special case could be interesting in our setting. It would be put in practice after using the Hong and Jakuš criterion. But, the main problem is that, since this is a decision problem, no equivalent constraint is returned. On the other hand, this algorithm is exponential in the number of variables, so that, it is interesting in our setting and deserves some attention.

Chapter 6

Evaluation

Over the last years, many algorithms, encodings and new variations of classical algorithms have been proposed. Often, the only way to get some evidence about their relative power or usefulness is performing some kind of empirical evaluation. In this chapter, we empirically evaluate the performance of the solvers presented in the previous chapter. In Section 6.1, we introduce a tool for solving polynomial constraints using several approaches, such that it will be used to carry out the experiments (see Section 6.2). Afterwards, in Section 6.3, the results are analyzed.

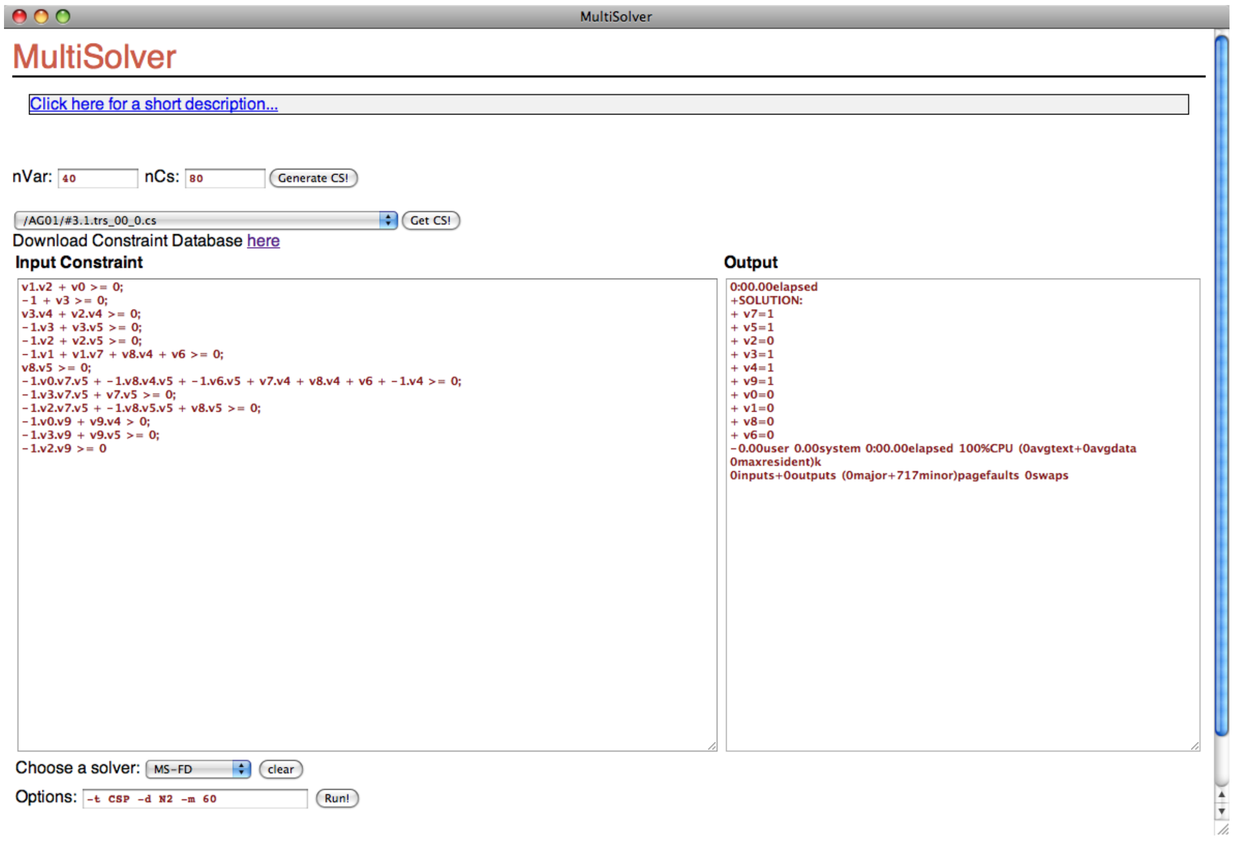
6.1 MultiSolver

MultiSolver is a prototype of symbolic constraint solving system developed from the original constraint-solving subsystem of the termination tool mu-term. MultiSolver integrates multiple solvers following several approaches, almost all of them described in Chapter 5.

We have implemented the system in the functional language *Haskell* [HPW92]. The tool is presented in two different interfaces: a command line tool and a web interface. The web interface (see Figure 6.1) has been developed in *php*, *javascript* and *html*, and performs calls to the command line version of MultiSolver (written in Haskell). MultiSolver is available here:

`http://zenon.dsic.upv.es/multisolver`

Moreover, the web interface incorporates a database of constraints from TPDB examples and a generator of aleatory constraints of a given size (number of variables in the constraints and number of constraints in the set). The constraint database has been generated by mu-term considering the dependency pair method with linear interpretations (see Chapter 3).



nVar: 40 nCs: 80

/AG01/#3.1.trs_00_0.cs

Download Constraint Database here

Input Constraint

```
v1.v2 + v0 >= 0;
-1 + v3 >= 0;
v3.v4 + v2.v4 >= 0;
-1.v3 + v3.v5 >= 0;
-1.v2 + v2.v5 >= 0;
-1.v1 + v1.v7 + v8.v4 + v6 >= 0;
v8.v5 >= 0;
-1.v0.v7.v5 + -1.v8.v4.v5 + -1.v6.v5 + v7.v4 + v8.v4 + v6 + -1.v4 >= 0;
-1.v3.v7.v5 + v7.v5 >= 0;
-1.v2.v7.v5 + -1.v8.v5.v5 + v8.v5 >= 0;
-1.v0.v9 + v9.v4 > 0;
-1.v3.v9 + v9.v5 >= 0;
-1.v2.v9 >= 0
```

Output

```
0:00.00elapsed
+SOLUTION:
+ v7=1
+ v5=1
+ v2=0
+ v3=1
+ v4=1
+ v9=1
+ v0=0
+ v1=0
+ v8=0
+ v6=0
-0.00user 0.00system 0:00.00elapsed 100%CPU (0avgtext+0avgdata
0maxresident)k
0inputs+0outputs (0major+717minor)pagefaults 0swaps
```

Choose a solver: MS-FD

Options: -t CSP -d N2 -m 60

Figure 6.1: MultiSolver web interface

Nowadays, the command line interface of MultiSolver offers the following options:

```
$ MultiSolver -h
MultiSolver [OPTIONS] <FILE>
  -v                  Print version
  -h                  Show help
  -o                  FILE Output file
  -m SECs             Set timeout (default 60 seconds)
  -d DOMAIN           Set Domain [N1,N2,...,Q1,Q2,...,[0,1,1/2,...]]
  -e                  Activate expert mode
  -t TECHNIQUES       Set kinds of solver (ALL | CSP | CSP_GEN | SAT)
```

where FILE is a mandatory option which represents the input file in an specific format described below.

Now we give the details of the options available in MultiSolver:

-v Show the current version of MultiSolver.

-h Print out the MultiSolver help showed above.

-o FILE By default, the solver output is redirected to the standard output. This option permits the redirection of the output to the given FILE.

-m SECs The time for searching for a solution (time-out) is limited to 60 seconds by default. This time-out can be changed using the option -m.

-d DOMAIN Used to specify the list of domains for search (enclosed between brackets). A single domain is possible. The available domains are the following:

- **Natural Domain** : A natural domain is represented by Ni, where i is the upper natural number of the domain. For instance, $N2$ is $\{0, 1, 2\}$.
- **Rational Domain** : The notation Qi represents domains of rational numbers based on powers of two: $Q_i = \{2^n | n \in \mathbb{Z}, -i \leq n \leq log_2 i\} \cup \{0\}$. For instance, $Q1 = \{0, 1/2, 1\}$ and $Q2 = \{0, 1/2, 1, 2\}$.
- **Explicit Domain** : User-defined domains can be specified by giving the explicit list $[a_1, a_2, \ldots, a_n]$ of its components a_i is either a natural or a positive rational number. For instance, $[0, 1/5, 1, 2/3, 2, 6]$ is a valid domain.

-e Activate the expert mode. In this mode, the system automatically chooses the order of execution according to the techniques and domains set by the user.

-t TECHNIQUES Used to specify the constraint solving techniques or solvers. The available techniques are the following:

- **CSP** Only CSP solvers for power of two domains.
- **CSP_GEN** Only CSP solvers allowing general domains (not only power of two domains).
- **SAT** Only SAT-based solvers.
- **ALL** All classes of solvers.

6.1.1 Input/Output format of constraints

The input constraint format is defined by means of the following BNF grammar:

```
<term> := <cte> | <id> | <id> "^" <cte>
<mon> := <factor> | <factor>"."<mon>
<signed_mon> := <mon> | "-"<mon>
<poly> := <signed_mon> | <signed_mon> "+" <poly>
<poly_ineq> := <poly> <op> <poly>
<op> := "=" | "<" | "<=" | ">" | ">="
<cs> := < poly_ineq> | <poly_ineq > ";" <poly_ineq>
```

This format permits the use of comments using "{−" and "−}" notation to enclose the text of the comment.

Example 25 *The following is an example of input constraint:*

```
{- Constraint from AG01/#3.1.trs -}
c0.minus2 + minus0 >= 0;
-1 + minus1 >= 0;
minus1.s0 + minus2.s0 >= 0;
-1.minus1 + minus1.s1 >= 0;
-1.minus2 + minus2.s1 >= 0;
-1.c0 + c0.quot1 + quot2.s0 + quot0 >= 0;
quot2.s1 >= 0;
-1.minus0.quot1.s1 + -1.quot2.s0.s1 + -1.quot0.s1 +
quot1.s0 + quot2.s0 + quot0 + -1.s0 >= 0;
-1.minus1.quot1.s1 + quot1.s1 >= 0;
-1.minus2.quot1.s1 + -1.quot2.s1.s1 + quot2.s1 >= 0;
-1.minus0.Fquot1 + Fquot1.s0 > 0;
-1.minus1.Fquot1 + Fquot1.s1 >= 0;
-1.minus2.Fquot1 >= 0
```

On the other hand, the output generated by MultiSolver corresponds to the following BNF grammar:

```
<result> := <noSol> | <sol>
<noSol> := ``+NO Solution:''
<sol> := ``+SOLUTION:\n''<varValue>``-''
<varValue> := ``+_'' <id>``=''<value>``\n''
<value> := <int> | <int>``/''<int>
```

where the character '_' represents an space and '\n' represents a new line as usual.

Example 26 *Consider the constraint in example 25. The solution returned by* MultiSolver *is:*

```
+SOLUTION:
+ quot0=0
+ quot1=2
+ quot2=0
+ s1=2
+ minus2=0
+ minus1=1
+ s0=2
+ Fquot=2
+ minus0=0
+ c0=0
-
```

6.1.2 Available constraint solvers

In the current version, the techniques implemented in MultiSolver are presented in the following list:

- MultiSolver-SAT uses the SAT-based constraint solver described in Section 5.1.2.
- MultiSolver-FD uses the algorithm presented in Section 5.2.2.
- MultiSolver-FD-GEN uses the algorithm presented in Section 5.2.2 together with a generalized arithmetical treatment of the numeric domains by just using the standard arithmetic operations (addition, product, power) instead of relying on binary shiftings. The algorithm can deal with general domains, e.g., the domain N_5 which cannot be expressed as a subset of powers of two.
- MultiSolver-CSP implements improved CSP described in Section 5.2.2.

Some of the solvers presented in Chapter 5 are not available yet. In the near future, we plan to integrate the following algorithms, which are

- MultiSolver-TPS implements the Two-Phase-Solver procedure described in Section 5.3.2.
- MultiSolver-ApSAT uses an implementation by R. Montagut [Mon08] in Haskell of the algorithm presented in [FGMS$^+$07].

In order to choose a solver according to the configuration, MultiSolver analyses in which domains the technique is available and, in case several techniques are available, then the most efficient is chosen. The following table summarizes the domains and techniques for each solver arranged according to their efficiency:

Solver	Techniques	Domains
MultiSolver-SAT	SAT	{N1}
MultiSolver-CSP	CSP	{Ni,Qi}*
MultiSolver-FD	CSP	{Ni,Qi}*
MultiSolver-FD-GEN	CSP_GEN	{Ni,Qi}
* Powers of two domains		

Example 27 *In order to understand how* MultiSolver *works, we give several examples of execution. Firstly, we save the example presented in example 25 to the file "AG01_3.1_00_0.cs":*

```
$ cat AG01_3.1_00_0.cs
{- Constraint from AG01/#3.1.trs -}
c0.minus2 + minus0 >= 0;
-1 + minus1 >= 0;
minus1.s0 + minus2.s0 >= 0;
-1.minus1 + minus1.s1 >= 0;
-1.minus2 + minus2.s1 >= 0;
-1.c0 + c0.quot1 + quot2.s0 + quot0 >= 0;
quot2.s1 >= 0;
-1.minus0.quot1.s1 + -1.quot2.s0.s1 + -1.quot0.s1 +
quot1.s0 + quot2.s0 + quot0 + -1.s0 >= 0;
-1.minus1.quot1.s1 + quot1.s1 >= 0;
-1.minus2.quot1.s1 + -1.quot2.s1.s1 + quot2.s1 >= 0;
-1.minus0.Fquot1 + Fquot1.s0 > 0;
-1.minus1.Fquot1 + Fquot1.s1 >= 0;
-1.minus2.Fquot1 >= 0
```

Now, we call MultiSolver *with the input constraint and the default options. Note that we have enabled a "debug mode" to understand how it works internally, so that, the messages beginning with "DEBUG" are not showed in a normal execution.*

```
$ MultiSolver AG01_3.1_00_0.cs
DEBUG Config: System T ALL Domain DS [N1,Q1,N2,Q2] Expert False Timeout 60
DEBUG SAT encoding for N1
DEBUG Timeout: 15000000
+SOLUTION:
+ Fquot=1
+ minus2=0
+ s1=1
+ minus1=1
+ minus0=0
+ s0=1
+ quot1=1
+ quot2=0
+ quot0=0
+ c0=1
-
```

Here, we see that the (current) default option uses the list of domains $[N1, Q1, N2, Q2]$*;, there is no restriction about the class of solver to be used and the timeout is set to* 60 *seconds. Note that, in this case, the first solver found a solution; otherwise, the system would have chosen the next domain (*$Q1$*).*

Now we show two example where different types of domains are used:

```
$ MultiSolver -t CSP -d N2 AG01_3.1_00_0.cs
DEBUG Config: System T CSP Domain D N2 Expert False Timeout 60
DEBUG CSP for the finite domain N2
DEBUG Timeout: 60000000
+SOLUTION:
+ quot1=1
+ s1=1
+ minus2=0
+ minus1=1
+ s0=1
+ Fquot=1
+ minus0=0
+ c0=0
+ quot2=0
+ quot0=0
-

$ MultiSolver -t CSP -d [1,1/2,2,1/4,0] AG01_3.1_00_0.cs
DEBUG Config: System T CSP Domain D [1,1/2,2,1/4,0] Expert False Timeout 60
DEBUG CSP for the finite domain [1,1/2,2,1/4,0]
DEBUG Timeout: 60000000
+SOLUTION:
+ quot1=2
+ s1=1
+ minus2=0
+ minus1=1
+ s0=2
+ Fquot=1
+ minus0=1
+ c0=0
+ quot2=0
+ quot0=0
-
```

Next example shows that MultiSolver *warns the user when there is no solver satisfying the configuration. In the current version there is no SAT-based solver for the domain N2:*

```
$ MultiSolver -t SAT -d N2 AG01_3.1_00_0.cs
MultiSolver: No solver available for this configuration
```

Example 28 *Consider the following constraints which require the use of rational coefficients:*

```
$ cat AG01_3.42_00_0.cs
-1.v1 + v1.v2 + v0 >= 0;
-1.v1 + v1.v2.v4 + v2.v3 + v0 >= 0;
-1.v0.v4 + v2.v3.v4 + v2.v3 + v0 + -1.v3 >= 0;
-1.v2.v4 + v2.v4.v4 >= 0;
-1.v1 + v1.v6 + v5 >= 0;
v1.v6.v4 + -1.v1.v4 + v6.v3 + v5 + -1.v3 >= 0;
v6.v3.v4 + v6.v3 >= 0;
-1.v6 + v6.v4.v4 >= 0;
```

```
-1.v1.v12 + v1.v8 + -1.v10.v11 + -1.v9 + v7 >= 0;
-1.v10.v8.v2.v3 + -1.v10.v8.v0 + -1.v12.v6.v3 + -1.v10.v7 + -1.v12.v5 + -1.v9 +
    v8.v3 + v7 >= 0;
-1.v10.v8.v2.v4 + -1.v12.v6.v4 + v8.v4 >= 0;
-1.v2.v13.v3 + -1.v0.v13 + v13.v3 > 0;
-1.v2.v13.v4 + v13.v4 >= 0
```

We can use several technique and domains to search for a solution:

```
$ MultiSolver -t [SAT,CSP,CSP] -d [N1,N2,Q1] AG01_3.42_00_0.cs
DEBUG Config: System [SAT,CSP,CSP] Domain [N1,N2,Q1] Expert False Timeout 60
DEBUG SAT encoding for N1
DEBUG Timeout: 20000000
DEBUG Result: Just Nothing
DEBUG CSP for the finite domain N2
DEBUG Timeout: 40000000
DEBUG Result: Just Nothing
DEBUG CSP for the finite domain Q1
DEBUG Timeout: 60000000
+SOLUTION:
+ v9=0
+ v12=0
+ v10=0
+ v6=1
+ v5=0
+ v1=0
+ v4=1
+ v3=1/2
+ v0=0
+ v2=1/2
+ v13=1/2
+ v8=0
+ v11=0
+ v7=0
-
```

Here, a SAT solver over the domain $N1$ is executed. But, no solution is found - neither does a CSP solver over $N2$. Finally, using a CSP solver over $Q1$ a solution is reported.

Example 29 *In expert mode, the system automatically chooses the order of the techniques and domains, in case they are specified:*

```
$ MultiSolver -e -t [CSP,SAT] -d [Q2,N4,N2,[0,1]] AG01_3.42_00_0.cs
DEBUG Config: System [CSP,SAT] Domain [Q2,N4,N2,[0,1]] Expert True Timeout 60
DEBUG SAT encoding for [0,1]
DEBUG Timeout: 15000000
DEBUG Result: Just Nothing
DEBUG CSP for the finite domain N2
DEBUG Timeout: 30000000
DEBUG Result: Just Nothing
DEBUG CSP for the finite domain Q2
DEBUG Timeout: 45000000
+SOLUTION:
+ v9=0
+ v12=0
+ v10=0
+ v6=2
+ v5=0
+ v1=0
+ v4=2
+ v3=2
+ v0=0
+ v2=1/2
+ v13=2
+ v8=0
+ v11=0
+ v7=0
-
```

Here, the user chooses the domains $[Q2, N4, N2, [0, 1]]$ *and the techniques [CSP,SAT], and the systems search for a solution in this order* $[[0, 1], N2, Q2, N4]$*, using SAT techniques for the first domain and CSP techniques for the other ones.*

6.2 Benchmarks

In this section, we compare the performance of most solvers presented in Chapter 5. Algorithms and heuristics are often compared by observing their performance on benchmark problems, or on suites of random instances generated from a simple, uniform distribution. The advantage of using a benchmark problem is that if it is an interesting problem (to someone), then information about which algorithm works well on it is also interesting. The drawback is that if an algorithm beats to any other algorithm on a single benchmark problem, it is hard to extrapolate from this fact. An advantage of using random problems is that there are many of them, and researchers can design carefully controlled experiments and report averages and other statistics. A drawback of random problems is that they may not reflect real life situations.

The empirical evaluation was carried out with both different types of problems: benchmark problems (see Section 6.2.1) and random problems (see Section 6.2.2).

6.2.1 Benchmarks problems

Benchmarks problems are divided in two different sets of problems as well. The first database of problems consists on the 952 examples in the 'Standard' TRS subcategory of the 2007 Termination Competition which are part of the 2007 Termination Problem Data Base (TPDB, version 4.0). This problem database is executed in our Termination tool MU-TERM and the different engines are used to solve the polynomial constraints.

On the other hand, as we have explained in previous sections, we have collected all constraints generated by MU-TERM from the 952 termination problems using the dependency pair method with linear interpretations. This collection of 4196 polynomial constraints are considered in the seconds approach of benchmarks. The main reason for experimenting with the constraints as a special case is because the current version of MU-TERM is not completely prepared for using external constraint solvers efficiently, which could lead to an incorrect comparison. Using external solvers directly with the polynomial constraints benchmarks seem to be a better approach for comparing solvers. We assume that if a solver is "better" than another one when they are compared over the polynomial constraints, then the solver should be better when they are compared over the termination problems.

The tools were executed under OS Linux Ubuntu 8.04.1 (kernel 2.6.24), on a Intel Core 2 CPU at 2.13 GHz and 1 GByte of primary memory.

Constraint Problem Benchmarks

For the benchmarks based on constraints, the following solvers have been executed:

- FD uses the constraint solver algorithm presented in Section 5.2.2.
- CSP implements improved CSP described in Section 5.2.2.
- CiME uses an external module implementing the constraint solving algorithm described in [CMTU06] and implemented as part of CiME v2.03.
- SAT uses the translation of polynomial constraints into propositional formulas described in Section 5.1.2, and then uses MiniSat[1] to obtain a solution.
- ApSAT uses an external module implementing the SAT-based constraint solving algorithm described in [FGMS$^+$07] and implemented as part of AProVE, and which also uses MiniSat for solving the generated propositional constraints.

[1]`http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/MiniSat.html`

	FD	CSP	CiME	SAT	ApSAT
Time	538.36	206.73	568.19	1083	2676.33
# YES	1792 (42.72%)	1823 (43.46%)	1818 (43.34%)	1830 (43.62%)	1830 (43.62%)
# NO	2260 (53.87%)	2342 (55.83%)	2342 (55.83%)	2363 (56.33%)	2365 (56.38%)
# TOs	142 (3.38%)	29 (0.69%)	34 (0.81%)	0 (0%)	0 (0%)
YES Av.T.	0.05	0.05	0.14	0.31	0.67
NO Av.T.	0.2	0.05	0.13	0.22	0.61

Table 6.1: Different solvers for N_1

	FD	CSP	CiME	ApSAT
Time	2004.83	1103.46	2338.01	3711.04
# YES	2066 (49.25%)	2148 (51.2%)	2116 (50.44%)	2197 (52.37%)
# NO	1607 (38.31%)	1883 (44.89%)	1515 (36.11%)	1998 (47.63%)
# TOs	521 (12.42%)	163 (3.89%)	563 (13.42%)	0 (0%)
YES Av.T.	0.26	0.09	0.2	0.99
NO Av.T.	0.91	0.48	1.27	0.77

Table 6.2: Different solvers for N_2

- ApSAT-Q uses an external module implementing the SAT-based constraint solving algorithm for rational coefficients described in [FNOG+08] and implemented as part of AProVE, and which also uses MiniSat for solving the propositional constraints.

Tables 6.1, 6.2, and 6.3 summarize the results for N_1, N_2, and Q_1 respectively, where

- 'Time' indicates the running time of all finished proofs.
- '# YES' indicates the number of successful proofs (consistent CSP).
- '# NO' indicates the number of unsuccessful proofs (inconsistent CSP).
- '# TOs' indicates the number of unfinished proofs interrupted by the time-out of 30 seconds.
- 'YES Av. T.'/ 'NO Av. T.' indicates the average time of successful/unsuccessful proofs (in seconds).

	FD	CSP	ApSAT-Q
Time	1942.11	762.99	1898.4
# YES	1789 (42.65%)	1834 (43.72%)	1835 (43.74%)
# NO	2019 (48.13%)	2282 (54.4%)	2349 (56%)
# TOs	386 (9.2%)	78 (1.86%)	11 (0.26%)
YES Av.T.	0.14	0.1	0.68
NO Av.T.	0.84	0.26	0.28

Table 6.3: Different solvers for Q_1

	TPS	*CSP*
Time	1472.76	7749.3
#YES	1764	2155
#NO	2412	1853
#TOs	18	186
YES Av.T.	0.19	0.22
NO Av.T.	0.25	0.92

Table 6.4: Evaluation of solvers based on N_2

Finally, in [LMNS08], it is compared the performance of a prototype of two-phase-solver (called TPS) presented in Section 5.3.2 with a CSP solver. Table 6.2.1 presents the results for the particular domain $N_2 = \{0, 1, 2\}$ (which is the usual one in most termination tools).

Termination Problem Benchmarks

On the other hand, we have compared the behavior of MU-TERM when the different polynomial constraint solving engines are used to prove termination of programs. This version of MU-TERM implements the dependency pairs method with polynomial interpretations. The subterm criterion [HM04] is also considered. Each time, a single domain is considered for obtaining the coefficients of the polynomial interpretations. The following versions of MU-TERM using the solvers described above as a back-end have been considered:

- MU-TERM-FD uses the constraint solving algorithm FD.
- MU-TERM-SAT uses the SAT-based solver named SAT.
- MU-TERM-ApSAT uses an external module implementing the SAT-based solver apSAT.

	FD	CSP	CiME	SAT	ApSAT
Time	275.01	257.52	451.84	428.18	2199.08
# YES	330 (34.63%)	331 (34.73%)	332 (34.84%)	333 (34.94%)	332 (34.84%)
# ??	583 (61.18%)	595 (62.43%)	590 (61.91%)	597 (62.64%)	594 (62.33%)
# TOs	23 (2.41%)	10 (1.05%)	14 (1.47%)	6 (0.63%)	10 (1.05%)
YES Av.T.	0.05	0.05	0.11	0.13	0.99
?? Av.T.	0.44	0.41	0.7	0.64	3.15

Table 6.5: MuTerm with different solvers for N_1

- MU-TERM-ApSAT-Q uses an external module implementing the SAT-based solver for rationals ApSAT-Q.
- MU-TERM-CiME uses C*i*ME as an external module implementing the solver C*i*ME .

Tables 6.5, 6.6, and 6.7 summarize the results for N_1, N_2, and Q_1 respectively, where

- 'Time' indicates the running time of all finished proofs.
- '# YES' indicates the number of successful proofs.
- '# ??' indicates the number of unsuccessful proofs.
- '# TOs' indicates the number of unfinished proofs interrupted by the time-out of 30 seconds.
- 'YES Av. T.'/ '?? Av. T.' indicate the average time of successful/unsuccessful proofs (in seconds).

As we said before, the SAT-based solver for natural coefficients (see [FGMS$^+$07]) has been implemented in Haskell. In [Mon08], it is presented the running time for MU-TERM using this implementation (called apSAT-Haskell) and the solver 'SAT' as a back-end solver. The results are shown in Table 6.8.

	FD	CSP	CiME	ApSAT
Time	1049.09	585.91	1124.2	2480.39
# YES	336 (35.26%)	339 (35.57%)	339 (35.57%)	342 (35.89%)
# ??	529 (55.51%)	559 (58.66%)	554 (58.13%)	582 (61.07%)
# TOs	71 (7.45%)	38 (3.99%)	43 (4.51%)	12 (1.26%)
YES Av.T.	0.1	0.2	0.59	1.09
?? Av.T.	1.92	0.93	1.67	3.62

Table 6.6: MuTerm with different solvers for N_2

	FD	CSP	ApSAT-Q
Time	1152.87	501.18	519.85
# YES	340 (35.68%)	347 (36.41%)	347 (36.41%)
# ??	525 (55.09%)	561 (58.87%)	576 (60.44%)
# TOs	71 (7.45%)	28 (2.94%)	13 (1.36%)
YES Av.T.	0.15	0.09	0.25
?? Av.T.	2.1	0.83	0.75

Table 6.7: MuTerm with different solvers for Q_1

6.2.2 Random problems

As we pointed out, benchmark sets are used to test algorithms for specific problems, but in recent years, there has been a growing interest in the study of the relation among the parameters that define an instance of CSP in general (i.e., the number of variables, domain size and number of constraints). In our empirical evaluation, each set of random constraint satisfaction problems was defined by the 3-tuple $< n, c, d >$, where n was the number of variables, c the number of constraints and d the domain size. The problems were randomly generated by means of polynomial constraint constraint and by modifying these parameters.

In this section, we show two tables which set two of the parameters and varies the other one in order to evaluate the algorithm performance when this parameter increases. We evaluated 100 test cases for each type of problem and each value of the variable parameter.

In Table 6.9, we present the running time in no-consistent problems, solved with CSP, apSAT and CiME, respectively, where the number of *variables* was increased from 3000 to 10000 and the number of constraints and the domain size were set at 5000 and 3, respectively: $< n, 5000, 3 >$.

In Tables 6.10, we present the running time in problems, solved with solved with CSP, apSAT and CiME, respectively, where the number of *constraints* was increased from 500 to

	SAT	apSAT-Haskell
# YES	338	322
# ??	584	590
# TOs	14	24

Table 6.8: MuTerm with SAT solvers for N_1

	<3000,5000,3>			<7000,5000,3>			<10000,5000,3>		
	CSP	*apSAT*	*CiME*	*CSP*	*apSAT*	*CiME*	*CSP*	*apSAT*	*CiME*
#NO	100	0	65	97	0	25	95	0	0
#TOs	0	100	35	3	100	75	5	100	100
NO Av.T.	2.72	-	28.04	1.61	-	29.41	1.68	-	-

Table 6.9: Evaluation of solvers with random and no-consistent problems $< n, 5000, 3 >$

2000 and the number of variables and the domain size were set at 4000 and 3, respectively: $< 4000, c, 3 >$.

6.3 Analysis of benchmarks

In order to analyze the benchmarks presented in previous section, we need to classify the existing choices according to their suitability. On the basis of our experience in the development of tools for proving termination, we believe that the following concrete criteria are appropriate to make this selection:

1. Take the more successful technique, i.e., having the bigger '# YES'. This seems to require few justification.

	<4000,500,3>			<4000,1000,3>			<4000,1500,3>		
	CSP	*apSAT*	*CiME*	*CSP*	*apSAT*	*CiME*	*CSP*	*apSAT*	*CiME*
#YES	72	73	53	37	0	23	21	0	15
#NO	27	27	27	56	55	56	60	52	62
#TOs	1	0	20	7	45	21	19	48	23
YES Av.T.	2.92	18.94	2.15	8	-	5.08	14.1	-	8.47
NO Av.T.	0.17	2.98	0.48	0.25	8.49	1.43	0.93	18.52	3.01

Table 6.10: Evaluation of solvers with random problems $< 4000, c, 3 >$

2. Among equally successful techniques, take the ones which are *complete* regarding the implemented technique, i.e., '??' answers actually mean that the considered technique does *not* work on the considered problem[2].

3. Among the complete techniques, take the ones having the bigger '# ??'. Tools for proving termination do not use a *single* technique for proving termination. Termination provers rather proceed stepwise by following some particular sequence of several techniques which are given 'partial' time-outs which are a (small) fraction of the global time-out. This permits switching to a different technique more often.

4. Among techniques having the same '# ??', take the ones having less average time for '??' answers. This permits a fast switching to a different technique, thus saving time from the assigned time slot.

According to this and the benchmarks presented in previous section, we conclude the following:

1. The results in Tables 6.1 and 6.5 suggest that the SAT-based solver 'SAT' over N_1 is the best way to deal with such kind of constraints when N_1 is the domain of coefficients in practice.

   Recall that the size of the encoding for this solver is exponential, whereas the size of the encoding is polynomial for the 'apSAT' encoding. Therefore, it is not clear what encoding is better. Moreover, Table 6.8 suggests that, in practice, the 'SAT' encoding leads to more efficient and successful proofs.

2. Benchmarks in Tables 6.2 and 6.6 show that 'ApSAT' exhibits the best behavior over N_2.

3. Benchmarks in Tables 6.3 and 6.7 show that the performance of CSP and ApSAT-Q over rational coefficients is almost similar, although the SAT-based solver is slightly better.

4. The benchmarks show that the introduction of standard techniques for modeling and treating constraint solving problems as CSP problems leads to important improvements in the efficiency of the state-of-the-art solvers implemented as part of current termination tools.

5. Benchmarks in Tables 6.9 and 6.10 suggest that SAT-based solvers could perform in a different way when bigger problems are considered. The experiments show that the CSP-based solvers perform much better than SAT-based ones in these cases. Therefore, this is one of the most promising aspects of CSP-based algorithms regarding proofs of termination.

   Actually, as discussed in Section 3.7, the format of polynomial interpretations which are used in current termination tools is kept 'intentionally simple'. Unfortunately, the constraints which are obtained from the treatment of programs of real programming languages

[2]All tools considered here, except CiME and the Two-Phases-Solver algorithms, are complete in this sense.

tends to reach huge complexity levels even when using standard polynomial interpretations like simple-mixed ones.

6. Finally, we note that, although $N_i \subset N_j$ whenever $i < j$, the number of successful proofs with N_j is in general not much bigger than with N_i. This is due to the presence of time-outs when we consider a bigger search space. This means that first considering the smallest domains is better than a direct attempt on a bigger but 'heavier' domain of coefficients.

Summarizing, we can say that, among the considered techniques, the SAT encoding for constrains over N_1 is the choice for constraint solving over N_1 in practice; otherwise the SAT-based solvers [FGMS+07, FNOG+08] should be used when domains of natural or rational numbers are considered, as long as the performance of CSP solvers does not improve even more.

An automatic 'termination expert' implementing the corresponding techniques should combine them accordingly starting with N_1, then N_2, etc.

Chapter 7

Conclusions

7.1 Summary

Termination of programs is one of the most challenging problems in software verification. We focus on termination of Term Rewriting Systems (TRSs), given that the quite mature theory of termination of TRSs provides a basic collection of abstractions, notions, and methods which have been proved useful for treating terminating problems in sophisticated programming languages. Therefore, the idea is to translate a program $\mathcal{P}$ into a TRS $\mathcal{R}$ whose termination implies termination of $\mathcal{P}$.

By using the dependency pairs method with polynomial orderings, we generate a set of polynomial constraints whose satisfaction implies the termination of the system. Consequently, a termination problem can be transformed into a constraint solving problem. The search for efficient methods for solving these constraints is essential for automating termination proofs. Polynomial interpretations over the natural is the basic technique for defining polynomial orderings, although in the last few years, polynomial interpretations over the rational numbers are receiving an increasing interest. The main reason is that polynomial interpretations over the rationals are strictly more powerful for proving termination than those over the naturals. We improve the results in [Luc06] by introducing sufficient conditions which are amenable for detecting the deficiencies of polynomial interpretations over the naturals automatically, on the basis of the structure of the rules of the TRS (see Section 4).

Up to now, several approaches have been considered for dealing with polynomial constraints in termination proofs. The main techniques are based on *SAT* (Boolean Satisfiability [FGMS$^+$07, FNOG$^+$08]), *CSP* (Constraint Satisfaction Problem [CMTU06, Luc05, Luc07]), *LP* (Linear Programming [Ste94]), and Real Algebraic Geometry [Gie95, Rou91].

SAT solvers have become very popular in the last few years due to the enormous progress in their performance. Nowadays, SAT-based encodings are the best way for dealing with polynomial constraints over both natural and rational coefficients. Since the research on specialized methods for solving typical small domains of coefficients (such as $\{0, 1\}$ or $\{0, 1/2, 1\}$) is of clear interest (see Section 5.1), we have presented a SAT-based solver for polynomial constraints over $\{0, 1\}$ (see Section 5.1), which is the basic domain for polynomial interpretations over the naturals. In practice, this encoding seems to lead to more efficient and successful proofs than the encoding presented in [FGMS$^+$07].

On the other hand, constraint satisfaction is the natural way for dealing with polynomial constraint. We have described a constraint solving algorithm for finite domains of powers of two (see Section 5.2.2). We have also shown that the introduction of standard techniques for modeling and treating constraint solving problems as CSP leads to important improvements in the efficiency of the state-of-the-art CSP solvers (see Section 5.2.2). Despite of the outstanding behavior of SAT-based solvers in practice, we have the intuition that SAT encodings are not well-suited when huge problems (w.r.t. number of variables, constraints, and domain coefficients) are considered.

The first implementation of a completely automatic system for proving termination of TRSs using polynomial orders was reported by Steinbach. Steinbach's idea was to transform the obtained constraints so that the use of standard methods from linear programming is possible. Unfortunately, this method has a number of drawbacks, which lead to an incomplete and rather unpractical algorithm. We are interested in studying possible combinations of LP and CSP techniques which can be appropriate for their implementation as part of termination tools. Following this idea, in Section 5.3.2, we have proposed a new hybrid algorithm which leads to a decrease of the total time for solving problems.

Finally, the concepts and tools from real algebraic geometry have been hardly explored since they are costly due to generality. During the last years, some interesting techniques have been developed which should be explored from a theoretical point of view (see Section 5.4).

7.2 Future work

The results presented in this thesis can be extended. In Section 4, the sufficient conditions for detecting the necessity of rational coefficients in linear polynomial interpretations could be extended to simple-mixed polynomial interpretations. Indeed it would be interesting to research on even more general conditions for detecting the necessity of simple-mixed or quadratic interpretations instead of linear interpretations.

In Section 5.2.2, we improve the efficiency considerably of our CSP algorithm by considering filtering techniques. Therefore, one could improve the performance of the CSP algorithm even more by considering search techniques (such as back-jumping or no-good recording) for reducing the search space.

On the other hand, when considering the use of constraint solving algorithms for termination tools, an essential issue is *incrementality*: constraint solvers in termination tools usually consider not only one but *several* different domains of coefficients when trying to solve a given polynomial (or even symbolic) constraint. Such domains are often related. For instance, MU-TERM first tries the domain N_1 (using the encoding presented in Section 5.1.2). If the constraint cannot be solved, then N_2 is considered, etc. But $N_1 \subseteq N_2$ and this means that, after failing with N_1, part of the search space for N_2 is known to be useless in advance. Current constraint solving algorithms implemented in termination tools like [CMTU06, FGMS$^+$07, FNOG$^+$08, Luc07] do not take benefit from this fact. A possible way of achieve an incremental algorithm could be to consider a more sophisticated SAT frameworks like SMT (SAT modulo theories [NOT06]), which seems to be a natural choice for polynomial constraints.

Beyond Diophantine constraint solving, coefficients for solving polynomial constraints could be taken from any subset of *real algebraic numbers* [Luc07]. Therefore, it would be interesting to study the explicit computation and representation of irrational coefficients for polynomial interpretations.

In Section 5.3.2, we propose an algorithm for mixing linear programming with constraint satisfaction solvers. In the CP community several methods for combining LP/CP have been proposed. Therefore, a natural way of improving our results could be to study the applicability of these techniques.

Finally, in Section 5.4, we show that the algorithms coming from real algebraic geometric are generally too costly due to generality. However, these methods deserve of some interest even if they are studied from a theoretical point of view.

Bibliography

[AG00] T. Arts and J. Giesl. Termination of Term Rewriting Using Dependency Pairs. *Theoretical Computer Science*, 236:133-178, 2000.

[BCR98] J. Bochnak, M. Coste, and M-F. Roy. Real Algebraic Geometry. Springer, 1998.

[BHZ06] L. Bordeaux, Y. Hamadi, and L. Zhang. Propositional satisfiability and constraint programming: A comparative survey. *ACM Comput. Surv.* 38(4), 2006.

[BKVH07] S. Boyd, S.J. Kim, L.Vandenberghe, and A. Hassibi. A Tutorial on Geometric Programming. *Optimization and Engineering* 8(1):67-127, 2007.

[BN98] F. Baader and T. Nipkow. Term Rewriting and All That. Cambridge University Press, 1998.

[BPR96] S. Basu, R. Pollack, and M-F. Roy. On the Combinatorial and Algebraic Complexity of Quantifier Elimination. *Journal of ACM* 43(6): 1002-1045, 1996.

[BPR03] S. Basu, R. Pollack, and M-F. Roy. Algorithms in Real Algebraic Geometry. Springer, 2003.

[CD96] P. Codognet and D. Díaz. Compiling constraints in clp(FD). *Journal of Logic Programming* 27(3):185-226, 1996.

[CDEL$^+$07] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. All About Maude – A High-Performance Logical Framework. Springer, 2007.

[CLS06] M. Codish, V. Lagoon, and P. Stuckey. Solving Partial Order Constraints for LPO Termination. In F. Pfenning, editor, *Proc of the 18th International Conference on Rewriting Techniques and Applications, RTA'06*, LNCS 3924:4-18, Springer-Verlag, 2006.

[CMMU03] E. Contejean, C. Marché, B. Monate, and X. Urbain. Proving termination of rewriting with C*i*ME. In *Proc. of WST'03*, Technical Report DSIC II/15/03, Valencia, Spain, 2003.

[CMTU06] E. Contejean, C. Marché, A.-P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning* 34(4):325-363, 2006.

[CLO97] D. Cox, J. Little, and D. O'Shea. Ideals, Varieties, and Algorithms. Springer, 1997.

[Coh69] P.J. Cohen. Decision procedures for real p-adic fields. *Comm. Pure Appl. Math.* 22:131-151, 1969.

[Col75] G.E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In H. Barkhage, H. editor, *Proc. of 2nd Conference on Automata Theory and Formal Languages*, LNCS 33:134-183, Springer, 1975.

[Dan63] G. B. Dantzig. Linear Programming and Extensions. Princeton University Press, 1963.

[Der79] N. Dershowitz. A Note on Simplification Orderings. *Inf. Process. Lett.* 9(5): 212-215, 1979.

[DLMMU08] F. Durán, S. Lucas, J. Meseguer, C. Marché, and X. Urbain. Proving Operational Termination of Membership Equational Programs. *Higher-Order and Symbolic Computation* 21(1-2): 59-88, 2008.

[FGMS$^+$07] C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT Solving for Termination Analysis with Polynomial Interpretations. In J. Marques-Silva and K.A. Sakallah, editors, *Proc. of the 10th International Conference on Theory and Applications of Satisfiability Testing, SAT'07*, LNCS 4501:340-354, Springer-Verlag, 2007.

[FNOG$^+$08] C. Fuhs, R. Navarro-Marset, C. Otto, J. Giesl, S. Lucas, and P. Schneider-Kamp. Search Techniques for Rational Polynomial Orders. *Proc. of the 9th International Conference on Artificial Intelligence and Symbolic Computation, AISC'08*, LNAI 5144:109-124, Springer-Verlag, 2008.

[GB65] S. Golomb and L. Baumert. Backtrack programming. In *Journal of ACM*, 12:516-524, 1965.

[Gie95] J. Giesl. Generating Polynomial Orderings for Termination Proofs. In J. Hsiang, editor, *Proc. of 6th International Conference on Rewriting Techniques and Applications, RTA'95*, LNCS 914:426-431, Springer-Verlag, Berlin, 1995.

[GSST06] J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated Termination Analysis for Haskell: From Term Rewriting to Programming Languages. In F. Pfenning, editor, *Proc of the 18th International Conference on Rewriting Techniques and Applications, RTA'06*, LNCS 4098:297-312, Springer-Verlag, 2006.

[GST06] J. Giesl, P. Schneider-Kamp and R. Thiemann. AProVE1.2: Automatic Termination Proofs in the Dependency Pair Framework. In *Proc. of the International Joint Conference on Automated Reasoning, IJCAR'06*, LNAI 4130:281-286, Springer-Verlag, 2006.

[HJ98] H. Hong and D. Jakuš. Testing Positiveness of Polynomials. *Journal of Automated Reasoning* 21:23-38, 1998.

[HM04] N. Hirokawa and A. Middeldorp. Dependency Pairs Revisited. *I*n Proc. of the 18th International Conference on Rewriting Techniques and Applications, RTA'04, LNCS 3091:249-268, Springer-Verlag, 2004.

[HM07] N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation* 205:474-511, 2007.

[HL89] D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In N. Dershowitz, editor, *Proc of the International Conference on Rewriting Techniques and Applications, RTA'89*, LNCS 355: 167-177, Springer-Verlag, 1989.

[HPW92] P. Hudak, S. Peyton-Jones, and P. Wadler. Report on the Functional Programming Language Haskell: a non–strict, purely functional language. *SIGPLAN Notices* 27:1-164, 1992.

[Kar84] N. Karmarkar. A new polynomial algorithm for linear programming. *Combinatorica* 4:373-395, 1984.

[KNT99] K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. In *Proc. of 1st International Conference Principles and Practice od Declarative Programming, PPDP'99*, LNCS 1702:47-61, Springer-Verlag, 1999.

[Lan75] D.S. Lankford. Canonical algebraic simplification in computer logic. Technical Report, University of Texas, Austin, 1975.

[Lan79] D.S. Lankford. On proving term rewriting systems are noetherian. Technical Report, Louisiana Technological University, Ruston, LA, 1979.

[LJM08] S. Liang, D.J. Jeffrey, and M. Moreno. The complete root classification of a parametric polynomial on an interval. In R. Sendra and L. Gonzalez-Vega, editors, *Proc. of the 21st International Symposium on Symbolic and algebraic computation, ISSAC'08*, 189-196, ACM Press, 2008.

[LMNS08] S. Lucas, R. Montagut, R. Navarro-Marset, and M.A. Salido. A hybrid approach to polynomial constraint-solving for termination tools. In *Proc. of the 8th Spanish Conference on Programming and Computer Languages, PROLE'08*, to appear, 2008.

[LN08] S. Lucas and R. Navarro-Marset. Comparing CSP and SAT solvers for polynomial constraints in termination provers. *Electronic Notes in Theoretical Computer Science*, 206:75-90, 2008.

[LNS09] S. Lucas, R. Navarro-Marset, and M.A. Salido. Paper in preparation.

[Luc04] S. Lucas. MU-TERM: A Tool for Proving Termination of Context-Sensitive Rewriting In V. van Oostrom, editor, *Proc. of 15h International Conference on Rewriting Techniques and Applications, RTA'04*, LNCS 3091:200-209, Springer-Verlag, 2004. Available: `http://zenon.dsic.upv.es/muterm`.

[Luc05] S. Lucas. Polynomials over the reals in proofs of termination: from theory to practice. *RAIRO Theoretical Informatics and Applications*, 39(3):547-586, 2005.

[Luc06] S. Lucas. On the relative power of polynomials with real, rational, and integer coefficients in proofs of termination of rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 17(1):49-73, 2006.

[Luc07] S. Lucas. Practical use of polynomials over the reals in proofs of termination. In *Proc. of 9th International Symposium on Principles and Practice of Declarative Programming, PPDP'07*, pp 39-50, ACM Press, 2007.

[Mat70] Y. Matjasevich. Enumerable sets are Diophantine. *Soviet Mathematics (Dokladi)*, 11(2):354-357, 1970.

[MN70] Z. Manna and S. Ness. On the termination of Markov algorithms. In *Proc. of 3rd Hawaii International Conference on System Science*, pp 789-792, 1970.

[Mon08] R. Montagut. Resolución eficiente de restricciones polinómicas para verificar la terminación de programas. Proyecto Final de Carrera, Universidad Politéctica de Valencia, 2008.

[Mor88] R. Mohr and G. Masini. Good old discrete relaxation. In *Proc. of the European Conference on Artificial Intelligence, ECAI'88)*, pp 651-656, 1988.

[MS00] U. Martin and D. Shand. Invariants, Patters and Weights for Ordering Terms. *Journal of Symbolic Computation* 29:921-957, 2000.

[NM95] U. Nilsson and J. Maluszynski. Logic, Programming and Prolog. John Wiley and Sons, 1995.

[NOT06] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Longemann-Loveland Procedure to DPLL(T). *Journal of ACM*, 53(6):937-977, 2006.

[Ohl02] E. Ohlebusch. Advanced Topics in Term Rewriting. Springer-Verlag, Berlin, 2002.

[PBG05] M.R. Prasad, A. Biere, and A. Gupta. A survey of recent advances in SAT-based formal verification. *International Journal on Software Tools for Technology Transfer* 7(2):156-173, 2005.

[Rou91] J. Rouyer. Calcul formel en géometrie alebrique reelle applique al terminaison des sistemes de reecriture. PhD thesis, Universite de Nancy, 1991.

[RVW06] F. Rossi, P. Var Beek, and T. Walsh. Handbook of Constraint Programming. Foundations of Artificial Intelligence, Elsevier, 2006.

[Sal08] M.A. Salido. A non-binary Constraint Ordering Heuristic for Constraint Satisfaction Problems. *Applied Mathematics and Computation* 198: 280-295, 2008.

[Sch86] A. Schrijver. Theory of linear and integer programming. John Wiley and Sons, 1986.

[Sei54] A. Seidenberg. A new decision method for elementary algebra and geometry. *Ann. Math.* 60: 365-374, 1954.

[SF06] N. Sadeh and M.S. Fox. Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence*, 1:1-41, 2006.

[SGST07] P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated Termination Analysis for Logic Programs by Term Rewriting. In R. K. Shyamasundar, editor, *Proc. of the 16th International Symposium on Logic-Based Program Synthesis and Transformation, LOPSTR '06*, LNCS 4407:177-193, Springer-Verlag, 2007.

[Ste92] J. Steinbach. Proving Polynomials Positive. In R. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, LNCS 652:191-202, India, 1992.

[Ste94] J. Steinbach. Generating Polynomial Orderings. *Information Processing Letters* 49:85-93, 1994.

[Tar51] A. Tarski. A Decision Method for Elementary Algebra and Geometry. University of California Press, Berkeley, 1951.

[Ter03] TeReSe, editor, Term Rewriting Systems, Cambridge University Press, 2003.

[TS07] M. Thang-Nguyen, and D. De Schreye. Polytool: Proving termination automatically based on polynomial interpretations. In R. K. Shyamasundar, editor, *Proc. of the 16th International Symposium on Logic-Based Program Synthesis and Transformation, LOPSTR '06,*LNCS 4407:210-218, 2007.

[Tsa95] E. Tsang. Foundations of Constraint Satisfaction. Academic Press, London, 1995.

[Tse68] G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic*, pp 115-125, 1968.

[Win96] F. Winkler. Polynomial Algorithms in Computer Algebra. Springer, 1996.

[Wol08] Wolfram Research, Inc. Mathematica Version 6.0. Champaing, IL, 2008.

[YX06] L. Yang and B. Xia. Quantifier Elimination for quartics. In *Proc. of the 8th International Conference on Artificial Intelligence and Symbolic Computation, AISC'06*, LNAI 4120:131-145, 2006.

[ZM07] H. Zankl and A. Middeldorp. In F. Baader, editor, *Proc. of the 18th International Conference on Rewriting Techniques and Applications, RTA'07*, LNCS 4533:389-403, Springer-Verlag, 2007.