Algorithmic Strategies for Applicable Real Quantifier Elimination[1]

Andreas Dolzmann[2]

July 26, 2000

[1]Im März 2000 an der Fakultät für Mathematik und Informatik der Universität Passau eingereichte Dissertation

[2]`http://www.dolzmann.de/`, `andreas@dolzmann.de`

Contents

Chapter 1

Introduction

Consider the set $\mathbb{R}$ of real numbers. We are used to making assertions on these numbers by using variables together with arithmetic operators and relations, e.g. $x^2 - 2x + 1 + y^2 < 0$.

Let us take a more formal look at such assertions. We allow ourselves to combine variables and integer numbers by means of the arithmetic operators "$+$," "$\cdot$," and "$-$." Powers such as x^2 are abbreviations for $x \cdot x$; it follows that variable exponents are not admitted.

Besides equality, we are interested in the relations "$\neq$," "$\leq$," "$<$," "$\geq$," "$>$" with their usual meaning. Such *atomic formulas* can be combined by boolean operators. The most interesting of these operators are negation "$\neg$," conjunction "$\wedge$," and disjunction "$\vee$." We are furthermore interested in implication "$\Longrightarrow$" and equivalence "$\Longleftrightarrow$." This allows us to express things like

$$x \geq 0 \wedge y > 0 \Longrightarrow x + y + z^2 \neq 0,$$

which we refer to as *quantifier-free* formulas.

We finally add quantifiers "$\exists x$" and "$\forall x$" ranging over real numbers as they are very common in elementary calculus. As an example consider the statement that quadratic polynomials are continuous:

$$\begin{aligned}\forall x \forall \varepsilon \big(\varepsilon > 0 \Longrightarrow \exists \delta (\delta > 0 \wedge \forall x_0 (|x - x_0| < \delta \\ \Longrightarrow |c_2 x^2 + c_1 x + c_0 - (c_2 x_0^2 + c_1 x_0 + c_0)| < \varepsilon))\big).\end{aligned}$$

Formulas possibly involving such quantifiers are called *first-order formulas.* With our continuity example, there are three interesting points to be observed:

1. We have not introduced the absolute value. This can however always be expressed within our framework. For instance, $|x - x_0| < \delta$ can be

rewritten as $x - x_0 < \delta \wedge -(x - x_0) < \delta$. The absolute value can thus be viewed as an abbreviation.

2. Our formulation also covers the continuity of linear polynomials since there is not specified that $c_2 \neq 0$.

3. We cannot express this way that "all polynomials" are continuous. A polynomial representation by means of variables numbers and arithmetic operators will always have a maximal degree.

A *quantifier elimination procedure* for the reals is an algorithm computing to every first-order formula φ an equivalent quantifier-free formula φ'. For our continuity example such a quantifier-free equivalent is $0 = 0$ which encodes "true." In general we have to expect the quantifier-free equivalent φ' to contain the *parameters*, i.e. unquantified variables of φ, which in our case are c_0, c_1, c_2.

We give another example where the parameters do not disappear: Consider the formula

$$\exists x(ax + b = 0)$$

stating that a parametric affine linear function $f_{a,b}(x) = ax + b$ has a zero. It is easy to see that a quantifier-free equivalent to this formula is given by $a \neq 0 \vee b = 0$. We see that the quantifier-free equivalent yields necessary and sufficient conditions in the parameters for the original quantified formula to hold.

Surprisingly, there exist quantifier elimination procedures that compute quantifier-free equivalents for arbitrary formulas matching our above framework. This has first been proved by Tarski [66] in 1948 by actually giving one such algorithm. Unfortunately, Tarski's algorithm is not elementary recursive and thus far beyond feasibility.

Around 1975, Collins [19] considerably improved this situation by giving a double exponential algorithm for quantifier elimination: the cylindrical algebraic decomposition method (CAD). More precisely, CAD is double exponential in the number of different variables contained in the input formula making no distinction between quantified variables and parameters. The CAD method has been implemented. The first implementation by Arnon was finished in 1981. Considerable theoretical improvements by Collins and Hong [21] and more efficient implementations by Hong followed.

In 1988, Weispfenning studied linear problems in fields, ordered fields, and discretely valued fields, cf. [68]. He presented among others results a quantifier elimination procedure for linear formulas in ordered fields which is based on the computation of finite elimination sets containing test points. A linear formula is a first-order formula not containing any products of quantified variables. We refer to Weispfenning's method as quantifier elimination by virtual substitution. Using this algorithm Weispfenning proves that linear quantifier elimination for ordered fields requires at most double exponential space and time. On the other hand he also proves that in the worst case this quantifier elimination is *at least* double exponential in time and space. These results are certainly correct when measuring the complexity in terms of the word length of the input formula. Weispfenning's results, however, are actually much more precise: His algorithm is double exponential only in the number of quantifier blocks. For like quantifiers, it is single exponential in the number of quantifiers. The number and the size of the atomic formulas in the input plays a very minor role. To be precise, both time and space complexity are polynomially bounded in these parameters. Observe that in contrast to quantifiers and quantifier changes, the number of parameters, i.e., free variables does not significantly contribute to the complexity. This fact suggested that Weispfenning's algorithms provide a reasonable supplement to CAD in particular for problems involving many parameters.

In the past ten years, Weispfenning has extended the virtual substitution method in theory to arbitrary degrees. There are implementations available up to degree two. The currently most sophisticated implementations are part of the REDLOG package by the author *et al.*

REDLOG has been applied successfully for solving non-academic problems, mainly for the simulation and error diagnosis of physical networks [73]. Applications inside the scientific community include the following:

- Control theory [1, 49],
- stability analysis for PDE's [38],
- geometric reasoning [31],
- non-convex parametric linear and quadratic optimization [70], transportation problems [51],
- real implicitization of algebraic surfaces [25],
- computation of comprehensive Gröbner bases,
- implementation of guarded expressions for coping with degenerate cases in the evaluation of algebraic expressions [24, 27],
- analysis, design, and error diagnosis of physical networks [65, 73],
- applications in theoretical mechanics and mechanical engineering [40].

This thesis is concerned with algorithmic improvements of the virtual substitution method for such practical applications. Besides this we are going to analyze the suitability of REDLOG for the completely new application area of scheduling.

1.1 Overview

In Chapter 2, we are going to discuss simplification of first-order formulas as an important subalgorithm of quantifier elimination by virtual substitution.

In Chapter 3 we first introduce the virtual substitution method in the form in which it has been discussed in the literature so far. After this we analyze the method to consist in this version of four phases. This new point of view allows us to systematize all known optimization approaches and, in addition, our new contributions. The remainder of the chapter is devoted to various improvements of the method that fit well into the traditional framework introduced there.

In Chapter 4 we leave the traditional framework in that we do no longer base our elimination on the set of atomic formulas in the input formula φ. Instead we make essential use of the boolean structure of φ. We compute test points not from the entire formula but only from parts that are relevant due to the boolean structure.

In Chapter 5 we introduce and discuss the notion of condensing, which provides a generalization of virtual substitution. With condensing, substitution does not take place into the entire formula but only into relevant parts. These parts are determined during the substitution process.

In Chapter 6 we modify the specification of quantifier elimination to a similar but more efficient algorithm, which is sufficient for many practical applications. The idea is that the quantifier elimination is correct only in the area around a certain point in parameter space. Accordingly this variant is called local quantifier elimination.

In Chapter 7 we return to regular quantifier elimination and discuss scheduling problems as an application area. For the case of scheduling, we illustrate how quantifier elimination by virtual substitution can be improved based on information on the structure of the input formulas. We use the term scheduling in a very general sense. Among the traditional scheduling problems our applications include dedicated machine models and project networks. In addition we suggest an approach to delay management for railway connections. This very general problem does not fit into any of the scheduling models discussed in the literature so far.

In Chapter 8 we give an overview of the REDLOG system that constitutes the framework for all our computations. We summarize the algorithms provided by REDLOG, describe its look-and-feel, and give some application examples from various areas of science and engineering.

In Chapter 9 we finally summarize and evaluate our results.

1.2 Main results

We discuss improvements of quantifier elimination by virtual substitution for the reals. Our starting point is simplification of first-order formulas over the reals. We clarify the notion of simplicity and introduce simplification techniques for all types of redundancies, e.g. algebraic or boolean, occurring in such formulas. As one crucial tool for simplification, we introduce the idea of using an explicit and an implicit theory. The former is used for entering external information into the simplification process, and the latter is used for communicating information located on different boolean levels in deeply nested formulas. Wherever this is appropriate, our simplification algorithms are designed in such a way that they make use of an optional extra theory argument. Our simplification methods provide in a natural way a decision heuristics for simple formulas.

Turning from the important subalgorithm of simplification to the core elimination procedure we analyze it to consist of four distinct phases. This allows for the first time to systematize all optimizations of the procedure known from the literature. We newly introduce a variety of optimization strategies of a traditional kind, i.e., they fit into the traditional framework of quantifier elimination by virtual substitution.

We generalize our theory concept from simplification to quantifier elimination by virtual substitution. First restricting to an external theory we identify various places to profit from external information. We then also adapt the concept of an implicit theory to quantifier elimination. The construction of this implicit theory here slightly differs from that for simplification. The introduction of the implicit theory into quantifier elimination requires to break with the traditional approach to compute elimination sets essentially from the set of atomic formulas contained in the input formula. This new liberal view on quantifier elimination by virtual substitution suggests in turn the introduction of optimized elimination techniques for frequent special cases: This includes the deep partial Gauss elimination of which only one extreme special case was known so far. Reanalyzing this technique in terms of the implicit theory leads us to the co-Gauss technique. This completely new special case is in a highly non-trivial way complementary to our deep partial Gauss. For the quantifier elimination procedure we can in contrast to simplification identify a third independent type of theory called "invisible," which is based on the semantics of the procedure while the implicit and the explicit theory are based on the syntax of the input. Concerning the applicability the invisible theory is used like the implicit one.

The insights obtained from our new structural view suggest to replace the virtual substitution of test points by a new concept, which we refer to as condensing. With condensing, substitution takes place only within certain subformulas, which we identify to be relevant due to their position in the boolean structure of the target formula. All other subformulas are dropped thus leading to much more concise elimination results.

Having optimized quantifier elimination by virtual substitution at all stages we turn to a variant called local quantifier elimination. For a certain subset of the parameters there is a point specified. The modified elimination result will be correct for a neighborhood of this point. For this purpose the local quantifier elimination procedure is allowed to assume order and equation constraints compatible with the given point wrt. the chosen parameters. As expected, this leads theoretically and practically to smaller intermediate and final results, and to a speed-up of the elimination process. The theoretically expected gain in efficiency of local quantifier elimination in contrast to the regular quantifier elimination is even exceeded by the results of our test implementations.

Restricting our attention to first-order formulations of scheduling problems as inputs we demonstrate how quantifier elimination by virtual substitution can be tuned—in addition to all strategies already mentioned—by making use of the knowledge that the input formula has a certain form. On the scheduling side we can formulate and solve problems in the dedicated machine model as well as project networks. Our test implementation cannot compete with special purpose solvers at present. On the other hand we can make use of the extreme flexibility of our approach for considering new types of scheduling problems, which have not been discussed in the literature so far. Such a problem is delay management for railway connections.

Chapter 2

Simplification of Formulas

The notion of *simplification* plays an important role in connection with computer algebra systems. It typically refers to the simplification of algebraic expressions. One wishes to reduce terms to canonical or at least simpler forms [16]. A concrete example on which much effort has been spent is the simplification of terms involving nested radicals [18, 6, 78]. Also the whole Gröbner basis theory has developed from the question whether given polynomials are equal in the residue class ring modulo some ideal [14].

From a mathematical point of view the symbolic manipulation of terms extends fairly naturally to that of quantifier-free formulas and further to that of first-order formulas. The well-known problem of finding simpler counterparts occurs then for formulas instead of terms. Since quantifier-free formulas are certainly simpler than quantified ones, quantifier elimination procedures such as partial CAD [19, 21] or elimination set methods as discussed in this thesis can be regarded as simplification.

The simplification algorithms described here have been developed with the implementation of the quantifier elimination by virtual substitution presented in this thesis. Due to the approach of iterating the substitution of test points we obtain in contrast to the quantifier elimination by partial cylindrical algebraic decomposition, deeply nested, highly redundant formulas. The results have typically to undergo some sophisticated simplification before providing useful information to the human reader. Besides such sophisticated simplification methods, it is crucial to have a fast simplification for the intermediate results at hand. The standard simplifier presented in this chapter provides such a fast simplifier, which is used in our implementation of quantifier elimination by virtual substitution: The time spent for simplification can be neglected compared to the gain of time obtained by treating simplified formulas.

The scope of formulas within computer algebra systems is not restricted to quantifier elimination but naturally combines with features already present there. For instance, consider *guarded expressions* obtained as solutions to a parametric problem (Fitch, private communication) such as for the following integration:

$$\int \frac{x^a}{b}\,dx = \left\{ \Big(a + 1 = 0 \land b \neq 0,\ \frac{\ln x}{b}\Big),\ \Big(a + 1 \neq 0 \land b \neq 0,\ \frac{x^{a+1}}{(a+1)b}\Big) \right\}.$$

We claim that it is generally reasonable to develop both a fast *standard simplifier* and more sophisticated *advanced simplifiers*. The former can be applied implicitly to any formula input while the latter are decided for and called explicitly by the user for crunching hard problems.

The mathematical principles underlying our simplifiers are mostly well-known. In this chapter we present a collection of practicable methods that have been implemented and extensively tested for their relevance. We further show how to combine the different ideas from algebra and logic to simplifiers in such a way that they produce formulas that cannot be further simplified by iterating these simplifiers. In other words, our simplifiers viewed as a function are idempotent. Achieving this is by no means trivial.

In view of the literature on simplification of formulas in propositional calculus, cf. [7] and the references there, we wish to point out that our simplification techniques do not require a boolean normal form computation, which would possibly produce an output of exponential size.

On the algorithmic side, we introduce the concept of a background *theory* that is implicitly enlarged when entering a formula for simplification. Originally developed for detecting interactions between atomic formulas on different boolean levels, it has turned out that this concept captures also other simplifiers developed earlier. These simplifiers, namely the Gröbner simplifier and the tableau simplifiers, could be generalized due to this new viewpoint.

We have implemented our simplification methods within our REDUCE package REDLOG [30, 28].

In this chapter, we will first of all discuss simplification methods for quantifier-free formulas. Then we introduce two simplification methods for quantified formulas.

2.1 The Formal Framework

Our *quantifier-free* formulas combine atomic formulas using the boolean connectives "$\land$," "$\lor$," "$\Longrightarrow$," "$\Longleftarrow$," "$\Longleftrightarrow$," and "$\neg$." Conjunction and disjunction are not binary but allow an arbitrary number of arguments. The atomic formulas are *equations* constructed with "$=$," *disequations* constructed with "$\neq$," *strong orders* constructed with "$<$" and "$>$," and *weak orders* constructed with "$\leq$" and "$\geq$." Strong orders and disequations together are called *strong relations* and weak orders together with equations are called *weak relations.* This variety of relations allows us to eliminate any occurrence of "$\neg$" from a formula by moving it to the inside and then encoding the negation in the atomic formulas. Throughout this thesis we use "ϱ" to denote one of the above relations. A quantifier-free formula is called *positive* if it contains only the boolean operators "$\wedge$," and "$\vee$." For the terms we do as usual not use the language of fields but that of ordered rings. Abbreviating variable-free subterms by integers, every term can then be written as a multivariate polynomial over $\mathbb{Z}$ wrt. some fixed term order. Moreover, we may consider all right hand sides of atomic formulas to be zero.

Non-atomic formulas are called *complex.* We divide the complex formulas into *flat* formulas and *deep* formulas. Flat formulas combine atomic formulas to one boolean level. Examples are conjunctions of atomic formulas or implication between two atomic formulas. *Boolean normal forms* are *disjunctive normal forms* (DNF) and *conjunctive normal forms* (CNF). A DNF is a disjunction of conjunctions including degenerate cases; a CNF is its dual counterpart.

We call a quantifier-free formula ψ of degree d in a variable x if all polynomials occurring in ψ have an x-degree of at most d. The x_i-degree of $f \in \mathbb{Z}[x_1, \dots, x_n]$ is the degree of the univariate polynomial

$$f \in \mathbb{Z}[x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n][x_i].$$

Existential formulas are of the form $\exists x_1 \dots \exists x_n \psi(\underline{u}, x_1, \dots , x_n)$, where ψ is a quantifier-free formula. Similarly, *universal formulas* are of the form $\forall x_1 \dots \forall x_n \psi(\underline{u}, \underline{x})$. The variables x_i in the *matrix* ψ of an existential or an universal formula are called *quantified.* A *prenex formula* has several alternating blocks of existential and universal quantifiers in front of a quantifier-free formula. Intermixing the quantifiers and the boolean operators yields general *first-order formulas.* One can easily compute to each given first-order formula an equivalent prenex formula.

A formula φ is called *linear* in $\{x_1, \dots , x_n\}$, if each atomic formula in φ is of the form $\sum_{i=1}^{d} c_i x_i + c \ \varrho \ 0$, where the c_i are terms not containing any of the x_i.

Real quantifier elimination is the task to find to a given first-order formula φ a quantifier-free formula φ^* such that both φ and φ^* are equivalent in the reals. A procedure computing such a φ^* from φ is called a real quantifier elimination procedure.

2.2 The Notion of Simple Formulas

It is not obvious which formulas should be considered simple. We summarize some *simplification goals*:

Few atomic formulas Currently, this is our main goal. Quantifier elimination output is in general too large to be understood by a human. However, it is often small enough for applications where it is processed automatically, typically by repeatedly fixing the values of some variables and then evaluating by resimplification. Small formulas then minimize memory consumption and evaluation time.

Comprehensible boolean structure When using quantifier elimination as a tool for solving mathematical problems it is essential that the output is comprehensible. Examples for comprehensible boolean structures are comparatively flat formulas or case distinctions.

Few different atomic formulas This is convenient for quantifier elimination by elimination set methods. In addition, it supports many simplification strategies.

Simple terms We consider it unintuitive when information that can be encoded logically is actually encoded algebraically. For instance, we would prefer the disjunction $a = 0 \lor b = 0$ to the product $ab = 0$.

Small satisfaction sets of the contained atomic formulas. This leads to a formula that is less redundant. If we know e.g. that $a \neq 0$ for some reason, we can replace $a \neq 0$ by $a < 0$, which has a smaller satisfaction set.

Convenient relations For elimination set methods, weak orders are more convenient than strong ones. On the other hand, equations and disequations can be considered simpler than orders.

Convenient boolean operators We consider conjunction and disjunction to be simpler than implication, replication, and equivalence.

Some of the simplification goals given above contradict one another. For these cases, the simplifiers are parameterized in such a way that the user can decide which goal to prefer for a particular problem. This parameterization is implemented via some global switches:

We optionally *prefer non-orders to orders* and, independently, *prefer weak orders to strong orders.* These options come out to preferring the goal of convenient relations to that of small satisfaction sets.

Concerning “simple terms” vs. “few atomic formulas” we have the possibility to select *no expansion*, *expand always*, or *expand if operator matches.* The latter selection never violates the simplification goal of a comprehensible boolean structure. These switches toggle in fact only expansions. The opposite contractions, e.g. encoding conjunctions into multiplication, are never performed.

The option to pass a *theory* as extra parameter is a key feature of our simplifiers. A theory Θ is a set of atomic formulas considered as a conjunction. The target formula φ is simplified wrt. Θ. For this purpose, we consider the variables in our atomic formulas as *constants* in the sense of logic. For instance, the theory $\{a^2 - a = 0\}$ does not contain a multiplicative idempotency rule but information on the constant a that may also occur in φ. Formally, we compute a formula φ' equivalent to φ in all ordered field models of Θ:

$$\bigwedge \Theta \longrightarrow (\varphi \longleftrightarrow \varphi').$$

As an example consider $\bigwedge\{a \leq 0\} \longrightarrow (a < 0 \longleftrightarrow a \neq 0)$. The theory parameter allows to enter extra information into the simplification process without adding it conjunctively to the target formula. Note that it would be a problem to remove conjunctively added information from the simplification result since it cannot be recognized easily.

The simplifiers typically start with simplifying the input theory treating it as a conjunction. If they detect in this way that the theory is inconsistent, they raise an error. Under an inconsistent theory any two formulas are equivalent, so the simplification result would make no sense. Mind that a necessary and sufficient inconsistency test for the theory amounts to the decision problem for existential formulas in ordered fields; this is not practicable for our purposes.

The theory concept may appear like some toy feature. It will, however, play an important role in our simplification algorithm for deep formulas. There the theory—possibly empty in the beginning—is implicitly enlarged during recursion.

2.3 Atomic Formulas

A simplification procedure derived from the methods described in this section is both an algorithm for simplifying a formula in the special case that it is atomic and a subalgorithm to an algorithm that simplifies a complex formula. For applying the theory concept to an atomic formula the latter is viewed as a (trivial) conjunction, i.e., as a flat formula. Such formulas are treated in the next section.

Variable-free atomic formulas are evaluated to truth values. In other atomic formulas the left hand side polynomial is replaced by its primitive part over $\mathbb{Z}$ with positive leading coefficient wrt. the chosen term order. Throughout this section we assume all polynomials to be of such a form. Making the leading coefficient positive requires mapping "$\leq$" to "$\geq$," "$<$" to "$>$," and vice versa.

2.3.1 Squarefree Parts and Parity Decompositions

A polynomial f is *squarefree* if it has no divisor of multiplicity greater than 1. The *squarefree decomposition* of f is a list $\bigl((p_1,1),\dots,(p_n,n)\bigr)$ where the p_i are primitive over $\mathbb{Z}$ with positive leading coefficient. Moreover, they are squarefree and pairwise relatively prime, and $\prod p_i^i = f$. We call $\prod p_i$ the *squarefree part* of f. The *parity decomposition* of f is defined as the pair

$$\Bigl(\prod_{\text{odd } i} p_i, \prod_{\text{even } i} p_i\Bigr).$$

Parity decompositions can easily be computed from the respective squarefree decompositions. It is an interesting open question if there is a faster way.

Proposition 2.1. *Let $f \in \mathbb{Z}[\underline{X}]$, let F be the squarefree part of f, and let (p,q) be the parity decomposition of f. Then the following equivalences hold:*

(i). $f = 0 \longleftrightarrow F = 0 \longleftrightarrow p = 0 \lor q = 0$

(ii). $f \neq 0 \longleftrightarrow F \neq 0 \longleftrightarrow p \neq 0 \land q \neq 0$

(iii). $f > 0 \longleftrightarrow pq^2 > 0 \longleftrightarrow p > 0 \land q \neq 0$

(iv). $f \geq 0 \longleftrightarrow pq^2 \geq 0 \longleftrightarrow p \geq 0 \lor q = 0$

(v). $f < 0 \longleftrightarrow pq^2 < 0 \longleftrightarrow p < 0 \land q \neq 0$

(vi). $f \leq 0 \longleftrightarrow pq^2 \leq 0 \longleftrightarrow p \leq 0 \lor q = 0$

The decision which equivalences to use depends on the simplification goals. The latter choices meet the simplification goal of simple terms but not that of few atomic formulas. In addition, the expansions can complicate the boolean structure. To overcome this difficulty, our implementation offers here and in similar situations the option to expand only if the boolean operator coming into existence matches the operator of the current level.

2.3.2 Semidefiniteness and Definiteness Tests

A polynomial is *positive (semi)definite* if all evaluations in the ordered field considered are greater than (or equal to) zero.

Proposition 2.2. *For positive definite $f \in \mathbb{Z}[\underline{X}]$ we have*

$$f = 0 \longleftrightarrow f < 0 \longleftrightarrow f \leq 0 \longleftrightarrow \text{false},$$

$$f \neq 0 \longleftrightarrow f > 0 \longleftrightarrow f \geq 0 \longleftrightarrow \text{true}.$$

In case that f is positive semidefinite, we have

$$f < 0 \longleftrightarrow \text{false}, \quad f \geq 0 \longleftrightarrow \text{true}, \quad f > 0 \longleftrightarrow f \neq 0, \quad f \leq 0 \longleftrightarrow f = 0.$$

Note that in the last two cases f can be replaced by its squarefree part according to Proposition 2.1. The decision between $f > 0$ and $f \neq 0$ depends on whether the simplification goal of convenient relations, here no orders, or that of small satisfaction sets is preferred.

Recognizing definiteness or semidefiniteness, i.e. deciding $\underline{\forall}(f > 0)$ or $\underline{\forall}(f \geq 0)$ respectively, is too hard to become part of a simplifier. We sketch some sufficient conditions for (semi)definiteness, which we use as fast tests. Due to a famous result by Artin [2], exactly positive semidefinite polynomials can be written as sums of squares of rational functions with real coefficients. Our simplifier recognizes trivial examples for this representation. We call a polynomial a *trivial square sum* (TSQ) if in its sparse distributive representation all exponents are even and all coefficients are non-negative. A trivial square sum is *strict* (STSQ) if it has a positive constant term.

Proposition 2.3. *(i).* STSQ*'s are positive definite, and* TSQ*'s are positive semidefinite.*

(ii). A polynomial with parity decomposition (p, q) is positive semidefinite if p is a TSQ. *It is positive definite if both p and q are* STSQ*'s.*

Obviously, none of the above tests is a necessary condition. For the relevance of testing the parity decomposition consider $x^2 - 2x + 1$ with squarefree part $x - 1$ and parity decomposition $(1, x - 1)$.

The following proposition contains obvious closure properties of TSQ's and STSQ's.

Proposition 2.4. *For trivial square sums f and g the following hold:*

(i). The product fg is a TSQ, *and fg is strict if and only if both f and g are strict.*

(ii). The sum $f + g$ is a TSQ, *and $f + g$ is strict if at least one of f, g is strict.*

These assertions extend by induction to multiple products.

Part (i) has two interesting consequences. First, compared to the squarefree part F, a parity decomposition (p, q) offers no extra information on definiteness: If both p and q are STSQ's, then f is positive definite but the proposition states that in this case $F = pq$ is already an STSQ.

Second, a squarefree decomposition does not yield more information than a parity decomposition (p, q): Test (ii) of Proposition 2.3 could be extended to squarefree decompositions by testing all odd-degree squarefree factors on being TSQ's. Part (i) of Proposition 2.4 shows that whenever this test succeeds, p is already a TSQ.

2.3.3 Splitting of Tsq's

In Proposition 2.2 we have seen that an atomic formula whose term is an STSQ can be decided with any relation. In case that the term is a non-strict TSQ, an atomic formula can be decided if its relation is "$<$" or "$\geq$." In all other cases, one can additively split the trivial square sum $\sum s_i$ according to the following equivalences:

$$\sum s_i \leq 0 \longleftrightarrow \sum s_i = 0 \longleftrightarrow \bigwedge s_i = 0,$$

$$\sum s_i > 0 \longleftrightarrow \sum s_i \neq 0 \longleftrightarrow \bigvee s_i \neq 0.$$

After splitting, the new equations or inequalities have to undergo atomic formula simplification themselves.

2.3.4 Implementation and Outlook

All methods described above in this section are part of the current implementation. We use a multivariate extension of the univariate squarefree decomposition algorithm proposed by Yun, cf. [77].

The multiplicative splitting of terms in Proposition 2.1 can be extended in various ways. Computing squarefree decompositions instead of parity decompositions, one obtains more factors. With non-orders all of these can be split. With orders one would only split those with even multiplicity. For those with odd multiplicity a case distinction of exponential size would be necessary.

The next improvement would be a complete polynomial factorization treating factors of odd and even degree as described above. Although in REDUCE we have an efficient polynomial factorization at hand, squarefree decomposition is, of course, much faster. The current implementation provides factorization of equations and inequalities as an option.

2.4 Flat Formulas

Similar to the previous one, this section is not devoted to an isolated algorithm that simplifies flat formulas, but to the "flat part" of a general simplifier. In particular, the simplifications described make not use of the fact that there are no complex constituents in the formulas considered.

One can imagine to simplify flat formulas by applying the converse of the additive and multiplicative splittings discussed in the previous section. We do not so because this would increase the complexity of the terms dramatically. Later, with Gröbner basis methods and with deep simplification, we will see how one can make use of atomic formula encoding of conjunctions or disjunctions in a more sophisticated way than simply regarding it as simplification rule.

2.4.1 Boolean Simplification

We apply the simplification rules given by the following equivalences, which are of a purely boolean nature. They hold for arbitrary formulas φ.

- $\neg\mathrm{true} \longleftrightarrow \mathrm{false}, \quad \neg\mathrm{false} \longleftrightarrow \mathrm{true},$
- $\mathrm{false} \Longrightarrow \varphi \longleftrightarrow \varphi \Longrightarrow \mathrm{true} \longleftrightarrow \varphi \Longrightarrow \varphi \longleftrightarrow \mathrm{true},$
- $\mathrm{true} \Longrightarrow \varphi \longleftrightarrow \varphi, \quad \varphi \Longrightarrow \mathrm{false} \longleftrightarrow \neg\varphi,$
- $\varphi \Longleftrightarrow \mathrm{true} \longleftrightarrow \varphi, \quad \varphi \Longleftrightarrow \mathrm{false} \longleftrightarrow \neg\varphi, \quad \varphi \Longleftrightarrow \varphi \longleftrightarrow \mathrm{true},$
- $\varphi \wedge \mathrm{true} \longleftrightarrow \varphi \wedge \varphi \longleftrightarrow \varphi, \quad \varphi \wedge \mathrm{false} \longleftrightarrow \mathrm{false},$
- $\varphi \vee \mathrm{false} \longleftrightarrow \varphi \vee \varphi \longleftrightarrow \varphi, \quad \varphi \vee \mathrm{true} \longleftrightarrow \mathrm{true}.$

All replications are turned into implications. Within conjunctions, disjunctions, and equivalences the atomic formulas are being sorted. For this we use an order on the terms which we extend to atomic formulas by first sorting wrt. the left hand side term and then wrt. the relation.

Table 2.1: Order theoretical smart simplification

$\wedge$	$t = 0$	$t \leq 0$	$t \geq 0$	$t \neq 0$	$t < 0$	$t > 0$
$t = 0$	$t = 0$	$t = 0$	$t = 0$	false	false	false
$t \leq 0$		$t \leq 0$	$t = 0$	$t < 0$	$t < 0$	false
$t \geq 0$			$t \geq 0$	$t > 0$	false	$t > 0$
$t \neq 0$				$t \neq 0$	$t < 0$	$t > 0$
$t < 0$					$t < 0$	false
$t > 0$						$t > 0$

2.4.2 Smart Simplification

Smart simplification makes use of non-boolean dependencies between the atomic formulas combined on a boolean level. This includes the dependencies that become non-boolean by moving negations into the atomic formulas.

Encoding negation into the atomic formula relation is already the first smart simplification. For any given relation ϱ there is a unique $\overline{\varrho}$ among our relations considered such that $t \mathrel{\overline{\varrho}} 0$ is equivalent to $\neg t \mathrel{\varrho} 0$ for any term t. We call $\overline{\varrho}$ the *negation* of ϱ, and we extend this notion to the atomic formula involved. Our rule for simplifying flat negations is hence given by $\neg\alpha \longleftrightarrow \overline{\alpha}$.

Conjunctions and disjunctions are dual to each other. It suffices to treat the conjunction case. The first idea is to consider order theory. For instance, $x \geq 0 \wedge x \neq 0$ can be contracted to $x > 0$. Actually, every conjunction of two atomic formulas whose left hand sides are equal can be contracted to one atomic formula or "false" (cf. Table 2.1).

This idea can be extended using the theory of ordered fields. The left hand side polynomials can be additively split into their *parametric part* and their constant term. Then we can contract atomic formulas involving polynomials with identical parametric parts. Recall that our atomic formula simplification normalizes the left hand side terms such that they are primitive over $\mathbb{Z}$. In order to recognize more possible contractions, we temporarily renormalize the terms such that their parametric part is primitive over $\mathbb{Z}$ obtaining a rational constant term. Given a conjunction

$$t \mathrel{\varrho} 0 \wedge t' \mathrel{\sigma} 0,$$

with $t = p + c$, $t' = p + d$, and $c, d \in \mathbb{Q}$, we can decide $c \mathrel{\tau} d$ with τ being any of our considered relations. This makes it often though not always possible to contract or even decide the above conjunction (cf. Table 2.2). Consider

Table 2.2: Additive smart simplification assuming $c < d$

$\wedge$	$t = 0$	$t \leq 0$	$t \geq 0$	$t \neq 0$	$t < 0$	$t > 0$
$t' = 0$	false	$t' = 0$	false	$t' = 0$	$t' = 0$	false
$t' \leq 0$	false	$t' \leq 0$	false	$t' \leq 0$	$t' \leq 0$	false
$t' \geq 0$	$t = 0$	—	$t \geq 0$	—	—	$t > 0$
$t' \neq 0$	$t = 0$	—	$t \geq 0$	—	—	$t > 0$
$t' < 0$	false	$t' < 0$	false	$t' < 0$	$t' < 0$	false
$t' > 0$	$t = 0$	—	$t \geq 0$	—	—	$t > 0$

for instance

$$\begin{aligned} x > 0 \wedge 2x - 1 > 0 \wedge 3x + 5 \neq 0 &\quad\longleftrightarrow\quad 2x > 1, \\ x^2 + y + 4 \geq 0 \vee 7x^2 + 7y + 4 \leq 0 &\quad\longleftrightarrow\quad \text{true}. \end{aligned}$$

Given an n-ary conjunction, the simplification result is invariant wrt. the order in which these (binary) simplifications are performed. In other words: one cannot make a mistake when contracting the atomic formulas one by one as they occur. This can be verified via a finite though tedious case distinction.

We next clarify how to bring the theory into this process. The equivalences underlying the theory application are always the same as those underlying the corresponding local simplification. We turn the theory into a conjunction and join it with the conjunction to be simplified. Then we perform smart simplifications as long as possible. As mentioned above we arrive at a unique result, either "false"—then we are finished—or a conjunction γ. The simplified conjunction is obtained from γ by extracting all atomic formulas that are not part of the original theory. If there is no such atomic formula, the result is "true." We will see in the next section that γ plays yet another role when flat simplification is viewed as a part of the deep simplification we are going to introduce in Section 2.5.

The result obtained with the above methods meets the simplification goal of small satisfaction sets for the atomic formulas. We also have to provide the optional simplification goal of convenient relations. Therefore, for any extracted atomic formula that does not meet the currently specified simplification goals, the original theory is checked for whether an alternative is possible (cf. Table 2.3).

For disjunctions we exploit the duality to conjunctions: The target disjunction is negated obtaining a conjunction of the negated atomic formulas

Table 2.3: Convenient relations

theory	$t \leq 0$	$t \leq 0$	$t \neq 0$	$t \neq 0$	$t \geq 0$	$t \geq 0$
formula	$t < 0$	$t = 0$	$t < 0$	$t > 0$	$t = 0$	$t > 0$
alternative	$t \neq 0$	$t \geq 0$	$t \leq 0$	$t \geq 0$	$t \leq 0$	$t \neq 0$

by de Morgan's law. Then we proceed as for conjunctions. Finally we negate the simplified conjunction back. This leads to atomic formulas with large satisfaction sets. Here we have to apply the technique of checking the old theory also for obtaining small satisfaction sets.

An implication $\alpha \Longrightarrow \beta$ between two atomic formulas is resolved into the disjunction $\overline{\alpha} \vee \beta$. If the simplification result is a truth value or one atomic formula, then we are finished. Else both α and β are independently simplified as trivial conjunctions wrt. the theory.

Equivalences are resolved into deep formulas containing only "$\wedge$" and "$\vee$" as operators. To these we apply our deep simplifier with one of the following results: We either obtain a truth value, an atomic formula, a conjunction or disjunction of two atomic formulas, or a deep formula again. In the last case we simplify both the original left hand side and right hand side separately as trivial conjunctions and then sort the result. In all other cases we are finished.

2.4.3 Gröbner Basis Methods

Gröbner basis methods [14, 5] allow us to take advantage of certain algebraic interactions between the atomic formulas when equations are involved. The Gröbner basis theory requires the polynomial coefficients to be field elements. For our purpose however, it suffices to consider polynomials over the integers. By *the* Gröbner basis we mean the unique reduced Gröbner basis wrt. to a fixed term order which contains only primitive polynomials with positive leading coefficients. We naturally extend the notion of the *Gröbner basis* to sets of equations and that of *reduction* to atomic formulas. For finite families $\{a_i\}_{i \in I}$ we write $\{a_i\}$ for short. In contrast to all other simplifications described in this chapter, here both the performance of the simplifier and the simplification results depend on the chosen term order. The following proposition states the mathematical background for the method we use. By $\operatorname{rad}(M)$ we denote the radical of the ideal generated by a set M of polynomials over a field K.

Proposition 2.5. *Let $\{f_i\}$, $\{g_j\}$, $\{\tilde{f}_k\}$, and $\{\tilde{g}_j\}$ be finite subsets of $K[\underline{X}]$. Suppose further that $\operatorname{rad}(\{f_i\}) = \operatorname{rad}(\{\tilde{f}_k\})$ and that $g_j \equiv \tilde{g}_j \bmod \operatorname{rad}(\{f_i\})$ for each j. Then*

$$\bigwedge_i f_i = 0 \wedge \bigwedge_j g_j \varrho_j 0 \quad \textit{and} \quad \bigwedge_k \tilde{f}_k = 0 \wedge \bigwedge_j \tilde{g}_j \varrho_j 0$$

are equivalent. The ϱ_j are any of the relations considered.

This proposition can also be applied to disjunctions by simplifying their negation. It is then instructive to write the disjunctions as implications:

$$\Big(\bigwedge_i f_i = 0\Big) \Longrightarrow \Big(\bigvee_j g_j \varrho_j 0\Big) \quad \text{iff} \quad \Big(\bigwedge_k \tilde{f}_k = 0\Big) \Longrightarrow \Big(\bigvee_j \tilde{g}_j \varrho_j 0\Big).$$

It was actually this form that gave the idea for Gröbner simplification. As an example consider the formula

$$xy - 1 = 0 \wedge yz - 1 = 0 \Longrightarrow x - z = 0.$$

Reducing $x - z$ wrt. the Gröbner basis $\{yz - 1, x - z\}$ of $\{xy - 1, yz - 1\}$ this formula can be simplified to "true." In the sequel we restrict our attention to the simplification of conjunctions again.

For the implementation the left hand sides of *all* equations are put into $\{f_i\}$. Next, we clarify how the new left hand sides of Proposition 2.5 are determined. For $\{\tilde{f}_i\}$ we use either $\{f_i\}$ or the Gröbner basis G of $\{f_i\}$—this is a parameter of our simplifier. For obtaining a $\tilde{g}_j$, we first compute the unique normal form h of g_j modulo G. Then we check if the methods of Section 2.3 can decide $h \varrho_j 0$. If so, we may either drop the atomic formula in question or evaluate the whole conjunction to "false." Else, we perform a radical membership test, which can be done without computing a radical basis [5]. If $g_j \in \operatorname{rad}(\{f_i\})$ we may again drop the atomic formula or replace the conjunction by "false." Finally, we either keep g_j or, as an option, we set $\tilde{g}_j = h$.

Substituting the equations in the conjunction with their Gröbner basis and reducing the other atomic formulas leads to normal forms of the conjunctions in the following sense: The left hand sides of all equations are the Gröbner basis of their ideal and all other terms are in normal form wrt. this Gröbner basis. Different subformulas on a boolean level can thus become equal enabling one of the boolean simplifications above. On the other hand, these options may contradict our simplification goals: Firstly, a reduced term can be less simple than the original one. Secondly, since flat formulas are in general parts of complex formulas, reduction can increase the number of different atomic formulas. This is because like terms are reduced wrt. different Gröbner bases when occurring at different places. Thirdly, the size of the Gröbner basis can exceed the size of the given ideal basis thus increasing the number of atomic formulas.

The following proposition contains one of the indicated examples for making use of the possibility to encode certain conjunctions multiplicatively into one atomic formula.

Proposition 2.6. *Let $\varrho_j \in \{<, >\}$, let $\tilde{\varrho}_j$ denote the weak counterpart of ϱ_j, and let σ_k be any of our relations. Then the following are equivalent:*

(i). $\bigwedge_i p_i \neq 0 \wedge \bigwedge_j q_j \ \varrho_j \ 0 \wedge \bigwedge_k r_k \ \sigma_k \ 0$

(ii). $\prod_i p_i \cdot \prod_j q_j \neq 0 \wedge \bigwedge_j q_j \ \varrho_j \ 0 \wedge \bigwedge_k r_k \ \sigma_k \ 0$

(iii). $\prod_i p_i \cdot \prod_j q_j \neq 0 \wedge \bigwedge_j q_j \ \tilde{\varrho}_j \ 0 \wedge \bigwedge_k r_k \ \sigma_k \ 0$

Since $\mathrm{Id}(\{f_i\})$ is not necessary prime, this offers a chance to improve our method: The decision of an atomic formula after Gröbner reduction or the radical membership test might succeed on the constructed product but not on the single factors. If the product inequality is decided to be "true" and hence dropped, one may choose between strong or weak orders. This corresponds to an application of (ii) or (iii), respectively. Recall that obtaining weak orders can be a simplification goal.

If the decision fails, there are several possible ways to continue in view of the parameterization. The first is to forget the product and proceed as described above but saving the radical membership tests for the inequalities. Secondly, if we keep the product, we can choose between the forms in (ii) and (iii). Finally, a choice has to be made whether the product itself is taken or its normal form wrt. G. When selecting (ii) one might prefer to take $\prod_i p_i$ instead of $\prod_i p_i \cdot \prod_j q_j$.

With the techniques described above we can once more make use of our theory concept. Denote by $\{F_i\}$ and $\{G_j\}$ the sets of left hand sides of theory equations and theory non-equations respectively. The f_i are optionally reduced modulo the Gröbner basis of $\{F_i\}$ in the beginning. The g_j are reduced modulo the Gröbner basis H of $\{f_i\} \cup \{F_i\}$ instead of G. We also reduce each G_j modulo H trying to evaluate its corresponding atomic formula this way. If it becomes "false," the whole conjunction is "false," otherwise we ignore it. The left hand sides of the inequalities and strong orders among the G_j can contribute to the corresponding product in Proposition 2.6 enlarging the chance for a successful radical membership test.

2.4.4 History and Implementation

Smart simplification developed from the pure order theoretic approach over the parametric part splitting to the discussed version involving theories. Contracting atomic formulas whose terms are identical monic variables but with different absolute summands has already been indicated by Hong, cf. [35]. None of the described smart simplifications has led to any problems concerning the speed of our simplifier.

With smart simplification, one can avoid the temporary resimplification of the left hand side terms by normalizing them generally to monic polynomials over $\mathbb{Q}$ instead of primitive ones over $\mathbb{Z}$. Our decision for the primitive normalization is older than this kind of simplification. We had preferred it for readability reasons.

Gröbner bases have been introduced by Buchberger, cf. [14]. Based on ideas by Becker, Pesch, and Weispfenning, the first related Gröbner basis methods have been developed with the implementation of comprehensive Gröbner basis [69] computation. This application involved ideal and radical membership tests for conjunctions containing only equations and inequalities. The implementation was done by Pesch, cf. [54] in the CGB-package of the computer algebra system MAS by Kredel, cf. [45, 44].

Two further MAS implementations for simplifying boolean normal forms were done by the author. Both were still restricted to equations and inequalities. The latter includes polynomial factorization and recognizes interaction between different clauses. Currently, these two features are not part of the implementation described here.

The Gröbner methods are considerably slower than the smart simplification and are thus not part of our standard simplifier.

2.4.5 Outlook

We have introduced order theoretical contraction of atomic formulas and an extension of this using the theory of ordered fields. This extension was additive in nature. There is also a multiplicative extension: Given a conjunction $s \varrho 0 \land t \sigma 0$, one can check if s divides t or vice versa. Let wlog. $t = rs$, then simplification of terms, reduction of the number of atomic formulas, or evaluations to truth values are possible in many cases (cf. Table 2.4). We give some examples: $xy \geq 0 \land x < 0 \longleftrightarrow y \leq 0 \land x < 0$,

$$xy \geq 0 \land x = 0 \longleftrightarrow x = 0, \quad xy \neq 0 \land x = 0 \longleftrightarrow \text{false}.$$

With equations involved (in the conjunctive case), there are even some simplifications possible if s divides t only up to a constant residue (cf. Table 2.5);

Table 2.4: Multiplicative smart simplification

	$st \varrho 0$			
	$=$	$\leq, \geq$	$\neq$	$<, >$
$t = 0$	$t = 0$	$t = 0$	false	false
$t \leq 0$	—	—	$t < 0 \land s \neq 0$	$t < 0 \land 0 \varrho s$
$t \geq 0$	—	—	$t > 0 \land s \neq 0$	$t > 0 \land s \varrho 0$
$t \neq 0$	$t \neq 0 \land s = 0$	—	$st \neq 0$	$st \varrho 0$
$t < 0$	$t < 0 \land s = 0$	$t < 0 \land 0 \varrho s$	$t < 0 \land s \neq 0$	$t < 0 \land 0 \varrho s$
$t > 0$	$t > 0 \land s = 0$	$t > 0 \land s \varrho 0$	$t > 0 \land s \neq 0$	$t > 0 \land s \varrho 0$

Table 2.5: Multiplicative smart simplification with constant residue

$\land$	$r < 0$	$r > 0$
$st + r = 0 \land t = 0$	false	false
$st + r < 0 \land t = 0$	$t = 0$	false
$st + r > 0 \land t = 0$	false	$t = 0$
$st + r \neq 0 \land t = 0$	$t = 0$	$t = 0$
$st + r \leq 0 \land t = 0$	false	$t = 0$
$st + r \geq 0 \land t = 0$	$t = 0$	false

for instance

$$xy - 1 > 0 \land x = 0 \longleftrightarrow \text{false}, \quad xy + 1 > 0 \land x = 0 \longleftrightarrow x = 0.$$

This kind of simplification does not involve any factorization. Extreme special cases have been mentioned by Hong, Liska, and Steinberg, cf. [38]. Some problems remain to be solved with the multiplicative smart simplification: Firstly, in contrast to the additive variant, the order in which the binary simplification rules are applied becomes relevant for the final result. Secondly, additive smart simplification can both create and destroy possibilities for multiplicative smart simplification, and vice versa. Good and fast strategies for combining both concepts still have to be found.

For the Gröbner basis methods we are planning to extend the factorization ideas of the latest MAS implementation to ordered fields. The basic idea there is to factorize the polynomials in the Gröbner basis of the equations.

2.5 Deep Formulas

To begin with, recall that the boolean simplification rules of Subsection 2.4.1 hold for arbitrary formulas. They are, of course, applied during the recursion process described below. In particular, we check for identical subformulas. In contrast to the atomic formula situation, the time required for this cannot be neglected.

2.5.1 Constructing Implicit Theories

As already indicated, we are going to use our concept of a theory for relating information located on different boolean levels. Our technique is based on the following observation.

Proposition 2.7. *Let φ and ψ be quantifier-free formulas. We use dots to indicate that ψ may be deeply nested inside the formula considered. Then the following equivalences hold:*

$$\begin{aligned} \varphi \land (\dots\psi\dots) &\longleftrightarrow \varphi \land \big(\dots(\varphi \land \psi)\dots\big), \\ \varphi \lor (\dots\psi\dots) &\longleftrightarrow \varphi \lor \big(\dots(\neg\varphi \land \psi)\dots\big) \end{aligned}$$

and the corresponding dual variants

$$\begin{aligned} \varphi \land (\dots\psi\dots) &\longleftrightarrow \varphi \land \big(\dots(\neg\varphi \lor \psi)\dots\big) \\ \varphi \lor (\dots\psi\dots) &\longleftrightarrow \varphi \lor \big(\dots(\varphi \lor \psi)\dots\big). \end{aligned}$$

We have in mind to apply the implication part of the equivalences, then to simplify, and finally to step back. More precisely, we use atomic formulas located at a certain boolean level deeper inside the formula by enlarging the theory. This technique of *theory inheritance* is the content of the following proposition, which is concerned with the simplification of a formula $\bigwedge \Gamma \land \psi$ or $\bigvee \Gamma \lor \psi$, where Γ is the set of toplevel atomic formulas of the considered one. We extend the implicit negation $\overline{\alpha}$ of atomic formulas α introduced in Subsection 2.4.2 to the sets Γ of atomic formulas.

Proposition 2.8. *Let Θ be a theory, let Γ be a finite set of atomic formulas, and let ψ be a formula.*

(i). Let ψ' be such that $\bigwedge(\Theta \cup \Gamma) \longrightarrow (\psi \longleftrightarrow \psi')$. Then

$$\bigwedge \Theta \longrightarrow \big(\bigwedge \Gamma \land \psi \longleftrightarrow \bigwedge \Gamma \land \psi'\big).$$

(ii). Let ψ'' be such that $\bigwedge(\Theta \cup \overline{\Gamma}) \longrightarrow (\psi \longleftrightarrow \psi'')$. Then

$$\bigwedge \Theta \longrightarrow (\bigvee \Gamma \vee \psi \longleftrightarrow \bigvee \Gamma \vee \psi'').$$

The idea is that ψ' or ψ'' respectively are *simplified* equivalents of ψ. Within this simplification process the proposition itself can be applied recursively.

Algorithmically we proceed as follows: The target formula is traversed recursively. On every boolean level we enlarge the theory in dependence on the boolean operator of the corresponding level. In conjunctions all atomic formulas are added. In disjunctions the negations of all atomic formulas are added. In implications atomic premises are added literally, and atomic conclusions are added negated. If the theory becomes inconsistent, the whole considered subformula is "false."

Let us see how some particular boolean level φ of the formula is simplified. We have obtained a theory Θ consisting of the user input enlarged by possibly negated atomic formulas from higher levels.

1. Update Θ to Θ' wrt. the toplevel atomic formulas of φ.
2. Simplify the complex constituents wrt. Θ'.
3. If new toplevel atomic formulas come into existence in step 2, then update Θ' wrt. these and go to 2.
4. Use methods as described in Section 2.4 for simplifying the toplevel atomic formulas present now wrt. the original theory Θ.

Step 3 and hence the loop is necessary for making the simplifier idempotent. The termination follows immediately from the fact that the number of complex subformulas on our considered level decreases by at least 1 with every iteration of the loop. Without complex subformulas no new atomic formulas can come into existence. Note that new atomic formulas can come into existence by simplifying a complex formula to another one with a matching level operator. In the implementation we, of course, abort step 2 when new atomic formulas occur.

2.5.2 The Standard Simplifier vs. Advanced Simplifiers

At this point, we have described all concepts underlying our *standard simplifier*, i.e., the simplifier that is fast enough to be used within algorithms where it is called extremely often. It includes all described concepts except for the Gröbner basis methods. The various switches discussed for the included simplification methods are chosen in such a way that simple terms are preferred in contrast to both view atomic formulas and a comprehensible boolean structure: More precisely, with equations and disequations we always factorize; with orders this is certainly not reasonable. Furthermore on the level of user calls we prefer equations and disequations to orders preferring with disequations convenient relations to small satisfaction sets. For calls from inside our quantifier elimination procedures we prefer instead orderings, which meets the simplification goals of small satisfaction sets as well as that of convenient relations.

Using Gröbner basis methods within the deep simplification is the first example for an *advanced simplifier*. In the following two sections, we will describe further concepts of advanced simplifiers, which make use of the standard simplifier as a subalgorithm.

2.5.3 Illustrating Examples

As first example consider the formula $a = 0 \land \bigl(b \neq 0 \lor (c \leq 0 \land (d > 0 \lor a = 0))\bigr)$. Starting with the empty theory, we successively add $a = 0$, $b = 0$, $c \leq 0$, and $d \leq 0$. On the innermost level, it is finally possible to apply the $a = 0$ of the theory to the local $a = 0$ yielding "true." The final result is

$$a = 0 \land (b \neq 0 \lor c \leq 0).$$

If the intermediate levels were missing, this would come out to an application of a law of absorption.

Our second example illustrates the necessity of a loop for idempotency.

$$a = 0 \land \bigl(b = 0 \lor (c = 0 \land d \geq 0)\bigr) \land (d \neq 0 \lor a \neq 0).$$

The initial theory for the toplevel complex subformulas is $\{a = 0\}$. This is used for simplifying the last constituent through which $d \neq 0$ is lifted to the toplevel and thus becomes part of the theory. With this enlarged theory the second constituent can be simplified wrt. the simplification goal of small satisfaction sets. As final result we obtain $a = 0 \land \bigl(b = 0 \lor (c = 0 \land d > 0)\bigr) \land d \neq 0$.

2.5.4 Outlook and Implementation

Possible extensions of the deep simplification are cut and absorption between sibling conjunctions or disjunctions. Furthermore, atomic formulas can be put outside the brackets where possible but, in general, this complicates the boolean structure. The order between atomic formulas on the single levels should be extended to complex subformulas.

We turn once more to the possibility of encoding a conjunction or disjunction of equations or inequalities into one atomic formula. We had decided not to do so. However, the atomic formula that would come into existence in the corresponding cases should be added to the theory. Note that the theory extended by such an atomic formula must not be used for simplifying that very conjunction or disjunction.

A theory containing complex formulas would offer more possibilities. Allowing the theory to contain possibly negated flat formulas might be a reasonable first step into this direction.

We have not yet implemented the Gröbner basis methods as part of the deep simplification. Our current Gröbner simplifier works by first constructing a boolean normal form and then simplifying it as described. It already uses ideas related to the theory enlargement by the product of equations or inequalities suggested in Section 2.4. The implementation is not idempotent yet.

2.6 Tableau Methods

Although our deep simplifier already combines information located on different boolean levels, it preserves the basic boolean structure of the formula. The tableau methods, in contrast, provide a technique for changing the boolean structure of a formula by constructing case distinctions. Compared to the standard simplifier they are much slower. They provide an advanced simplifier.

2.6.1 The Basic Tableau Idea and Extensions

Given a formula φ, we systematically construct a bigger equivalent formula from it by adding a disjunctive toplevel. We obtain a formula

$$\bigvee_{\alpha\in\mathrm{A}}(\alpha\wedge\varphi)\quad\text{with}\quad\bigvee_{\alpha\in\mathrm{A}}\alpha\longleftrightarrow\mathrm{true},$$

where A is a set of atomic formulas. In other words: we form a complete case distinction. This roughly multiplies the size of the formula by the size of A. The idea is to choose a good A such that using each $\alpha\in\mathrm{A}$ as the theory for the simplification of φ inside the single branches, the final result is smaller than φ.

It can happen that several simplifications of φ in different branches are equal. Writing a simplification result of φ wrt. α as φ_α, we obtain a formula of the form

$$(\alpha_0 \land \varphi_{\alpha_0}) \lor \bigvee_{\alpha\in \mathrm{A}_1} (\alpha \land \varphi_{\alpha_0}) \lor \bigvee_{\alpha\in \mathrm{A}_2} (\alpha \land \varphi_\alpha) \quad \text{with} \quad \mathrm{A} = \{\alpha_0\} \cup \mathrm{A}_1 \cup \mathrm{A}_2.$$

Applying a law of distributivity, this can be simplified to

$$\Bigl(\Bigl(\alpha_0 \lor \bigvee_{\alpha\in \mathrm{A}_1} \alpha\Bigr) \land \varphi_{\alpha_0}\Bigr) \lor \bigvee_{\alpha\in \mathrm{A}_2} (\alpha \land \varphi_\alpha).$$

To make this more precise, consider $\alpha_0 \equiv t > 0$, $\mathrm{A}_1 = \{t < 0\}$, and φ does not contain any ordering constraint involving the term t in a way relevant for simplification.

This is a simplification that our deep simplifier does not know. We call it *contraction* of tableau branches. Note that afterwards the flat disjunction $\alpha_0 \lor \bigvee_{\alpha\in \mathrm{A}_1} \alpha$ can be simplified using the methods of Section 2.4.

Good candidates for A are case distinctions $\{t < 0, t = 0, t > 0\}$ wrt. the sign of a term t that occurs often in φ. Here, the flat disjunction coming into existence after a contraction of branches can always be simplified to one atomic formula. We call a tableau wrt. an A of such a form a *tableau step* wrt. t.

There is an *automatic* tableau, which tries tableau steps wrt. all terms in φ. In the end, if there was a tableau result smaller than the input, one of the smallest results is returned. Else, the original formula is returned. Thus the result of an automatic tableau application is at least as simple as the input taking the number of atomic formulas as measure. Our implementation provides ways to restrict the number of terms tried for the automatic tableau.

In contrast to the simple tableau, the automatic tableau is not idempotent. Iterative application can lead to a finite sequence of increasingly smaller results. There is an *iterative* automatic tableau, which automates this repetition. Optionally, the iterations can be performed on the single branches φ_α of an automatic tableau result, which leads to smaller results in most cases though not generally.

Continuing with the smallest result is a heuristic approach. Examples can be constructed where smaller final results are obtained by continuing with a tableau result that is even larger than the original input formula.

2.6.2 History and Implementation

The tableau methods described here including automatic and iterated automatic tableaux after fully implemented in REDLOG. The method is related to the analytic tableaux used for automated theorem proving [62]. A special case of the tableau method described here was originally suggested by Loos [51, Weispfenning, private communication] and firstly implemented by Burhenne, cf. [17]. This version performed tableau steps wrt. a given term t without contraction of branches. The simplifications in the branches were restricted to deciding atomic formulas with t as their left hand side.

Before our deep simplifier performed the theory inheritance described in Section 2.5, the iterative tableau method provided considerable simplifications in many cases. Meanwhile, there are only few formulas that can be simplified via the tableau method after simplification with the standard simplifier.

2.6.3 Outlook

There is the following dual variant of the tableau: Instead of performing a complete case distinction one can construct

$$\bigwedge_{\alpha \in \mathrm{A}} (\alpha \vee \varphi) \quad \text{with} \quad \bigwedge_{\alpha \in \mathrm{A}} \alpha \longleftrightarrow \text{false}$$

for a set A of atomic formulas. One would then define a tableau step wrt. a term t as taking $\mathrm{A} = \{t \geq 0, t \neq 0, t \leq 0\}$. Since these atomic formulas enter the theory negated, there are the same simplifications performed within the single branches as in the normal case. If the toplevel operator of φ is "$\vee$" one obtains one boolean level less when applying the dual tableau. There is no problem with the automatic or iterative tableau when deriving a selection strategy from this observation.

A promising variant of the tableau method is an *in-place tableau* that applies tableau steps wrt. a term t not to the whole formula but to the smallest subformula containing all occurrences of t.

Provided that the multiplicative variant of smart simplification described in the outlook of Section 2.4 is available, there is an interesting variant of the automatic tableau: One can first factorize all terms occurring in the target formula and then perform the tableau wrt. all the irreducible factors instead of all the terms. We expect the result to meet the simplification goal of simple terms better than that of few atomic formulas. One thus has to define criteria for finding the *simplest* formula obtained this way.

2.7 Boolean Normal Forms

We consider boolean normal form computation as simplification because the results meet our simplification goal of a comprehensible boolean structure.

Anyway, a computed boolean normal form can also have less atomic formulas than the input formula. We restrict our attention to DNF computations. The computation of a CNF is dual to this.

2.7.1 Computation of Boolean Normal Forms

We assume that the input formula is in *negation normal form*, i.e., it contains only "$\wedge$" and "$\vee$" as boolean operators. In order to avoid case distinctions we allow ourselves to consider atomic formulas as trivial conjunctions. Assuming that flat formulas are DNF's we recursively compute DNF's from disjunctions or conjunctions of DNF's. The former case is trivial, in the latter we have to apply a law of distributivity. The following proposition shows how this corresponds to a Cartesian product computation.

Proposition 2.9. *For $i = 1, \dots, m$ and $j = 1, \dots, n_i$ let γ_{ij} be conjunctions of atomic formulas. Set $N = \{1, \dots, n_1\} \times \dots \times \{1, \dots, n_m\}$. Then*

$$\bigwedge_{i=1}^{m} \Bigl(\bigvee_{j=1}^{n_i} \gamma_{ij}\Bigr) \quad \text{and} \quad \bigvee_{(c_1,\dots,c_m)\in N} \Bigl(\bigwedge_{i=1}^{m} \gamma_{ic_i}\Bigr)$$

are equivalent. After flattening the nested conjunction, the latter formula is a DNF.

Note that the method described does not introduce any atomic formulas different from those already present in the input.

2.7.2 Simplification of Boolean Normal Forms

In addition to the simplification methods already presented we are now going to discuss some methods particular to the simplification of boolean normal forms. Firstly, we have implemented a method corresponding to the propositional logical *cut*. We apply the equivalence

$$(\alpha_1 \wedge \dots \wedge \alpha_n \wedge t\ \varrho\ 0) \vee (\alpha_1 \wedge \dots \wedge \alpha_n \wedge t\ \sigma\ 0) \longleftrightarrow (\alpha_1 \wedge \dots \wedge \alpha_n \wedge \tau),$$

where τ is the result of the smart simplification of $t\ \varrho\ 0 \vee t\ \sigma\ 0$. As an example consider $(\varphi \wedge t = 0) \vee (\varphi \wedge t < 0) \vee (\varphi \wedge t > 0)$, which is simplified to φ.

Two further simplifications are based on the following proposition.

Proposition 2.10. *Let φ and ψ be formulas such that φ implies ψ. Then $\varphi \vee \psi$ is equivalent to ψ, and $\varphi \wedge \psi$ is equivalent to φ.*

Verifying the premise of this proposition corresponds to the decision problem for universal formulas. This is not practicable for our purposes. Therefore, we use tests for implication that are only sufficient. Formally we introduce relations "$\succeq$" such that $\varphi \succeq \psi$ implies $\varphi \longrightarrow \psi$ for conjunctions φ and ψ of atomic formulas.

We further consider two properties that are relevant for the implementation. The first one is *transitivity.* The second one is *compatibility with conjunctions*, which is defined as

$$\varphi \succeq \psi \longrightarrow \varphi \wedge \gamma \succeq \psi \wedge \gamma \quad \text{for conjunctions } \gamma \text{ of atomic formulas.}$$

A DNF is simplified by testing for each pair (φ, ψ) of conjunctions if $\varphi \succeq \psi$ holds. If so, φ is deleted from the disjunction. If "$\succeq$" is transitive, the order in which the pairs are tested is irrelevant for the result. In particular, this allows us to make use of efficient simultaneous tests for $\varphi \succeq \psi$ and $\psi \succeq \varphi$.

Compatibility ensures that one final simplification after the DNF computation yields the same result as that obtained by applying intermediate simplifications after each recursion step. This follows easily from its definition and the way we compute the DNF.

The first possible choice for such a relation is *subsumption* defining $\varphi \succeq \psi$ by $\Phi \supseteq \Psi$ where Φ and Ψ are the sets of atomic formulas contained in φ and ψ respectively. This idea treats atomic formulas like propositional logical variables. In our situation, subsumption can be extended in the following way: We define

$$\bigwedge_{i \in I} t_i \ \varrho_i \ 0 \succeq_{sub} \bigwedge_{j \in J} t_j \ \sigma_j \ 0$$

if and only if $I \supseteq J$ and for all $j \in J$ the smart simplification of Section 2.4.2 simplifies $t_j \ \overline{\varrho_j} \ 0 \vee t_j \ \sigma_j \ 0$ to "true." As an example consider

$$a > 0 \wedge b > 0 \wedge c > 0 \succeq_{sub} a > 0 \wedge b \geq 0.$$

Subsumption is transitive and compatible with conjunctions. There are efficient tests for subsumption—possibly into both directions—using the fact that atomic formulas are canonically ordered within the conjunctions.

A smarter though less efficient choice is *simplifier-recognized implication* "$\succeq_{\mathrm{rec}}$," which once more makes use of our theory concept.

We define that $\varphi \succeq_{\mathrm{rec}} \psi$ if ψ can be simplified to "true" with the atomic formulas from φ as theory. Using the standard simplifier this is both transitive and compatible with conjunctions, which obviously depends on the simplifier used. The test $\varphi \succeq_{\mathrm{rec}} \psi$ has turned out to be very time consuming. We thus apply it only at the end of each DNF computation making use of the compatibility with conjunctions.

2.7.3 History and Implementation

Since quantifier elimination by virtual substitution does not require boolean normal form computation we have, until recently, not spent much effort into this topic. Originally, we computed our boolean normal forms using the pure Cartesian product method. The next step was the application of the standard simplifier that was under development at the same time. The implementation of the ordering of the atomic formulas led to an enormous improvement in boolean normal form computation. The idea of subsumption led to further considerable improvements. We also have an ad hoc implementation of the simplifier-recognized implication. It has led to some minor improvements but it is extremely time consuming. The best boolean normal forms are currently obtained by applying the Gröbner simplifier.

2.7.4 Outlook

In the outlook of Section 2.4 we have already indicated the alternative of making the terms monic instead of primitive. Recall from Subsection 2.4.1 how the atomic formulas are canonically ordered within the conjunctions. With monic left hand sides any reasonable order "$\sqsubseteq$" on the terms extends to an order on the atomic formulas with the following property: Let p, q be left hand side terms with zero absolute summand, and let c, d be rational constant terms. Then

$$p \sqsubset q \longrightarrow p + c \mathrel{\varrho} 0 \sqsubset q + d \mathrel{\sigma} 0.$$

Both $p + c$ and $q + d$ are again valid left hand side terms. Thus the order "$\sqsubseteq$" is in some sense compatible with the addition of rational constants. This fact together with the observation that the flat simplifier described in Subsection 2.4.2 performs simplifications only between atomic formulas with the same parametric part in the monic sense gives rise to the following proposition. It provides a fast test for the failure of simplifier-recognized implication. This test can be used as a filter before the actual test.

Proposition 2.11. *Let φ denote the conjunction*

$$p_1 + c_1 \mathrel{\varrho_1} 0 \land \ldots \land p_m + c_m \mathrel{\varrho_m} 0 \quad \textit{with} \quad p_1 + c_1 \sqsubset \cdots \sqsubset p_m + c_m,$$

and let ψ denote the conjunction

$$q_1 + d_1 \mathrel{\sigma_1} 0 \land \ldots \land q_n + d_n \mathrel{\sigma_n} 0 \quad \textit{with} \quad q_1 + d_1 \sqsubset \cdots \sqsubset q_n + d_n.$$

Suppose that $q_1 \sqsubset p_1$ or $p_m \sqsubset q_n$. Then $\varphi \succeq_{\mathrm{rec}} \psi$ does not hold.

With normalization of left hand sides to primitive polynomials such a compatibility with adding rational constants cannot be achieved so easy: Replacing $p + c$ and $q + d$ by their primitive part can change the order between them wrt. many natural orders on terms.

Boolean normal form computation in propositional logic has been tackled in several papers by Quine and McCluskey, cf. [56, 57, 58, 52]. They have shown how minimal boolean normal forms can be obtained. All these methods combine a boolean variable β with its negation $\neg\beta$ in some way, where the point is that $\beta \vee \neg\beta \longleftrightarrow \text{true}$. Subsumption is used as a test for implication between clauses. In the case of propositional logic this test is even sufficient after some obvious simplifications inside the clauses.

Our adaptions of cut and subsumption provide the basic tools for implementing an analogue of these methods. More sophisticated adaptions of cut and subsumption would take parametric parts into account.

2.8 Related Work

The simplification of quantifier-free first-order formulas over the reals is a relatively new research area. In contrast to this the minimization and the efficient treatment of boolean functions are studied carefully by many authors. Boolean functions are usually defined by combining of *boolean variables* using the usual logical operations "$\neg$," "$\wedge$," and "$\vee$."

We have mentioned a first relation between the simplification of boolean functions and the simplification of formulas in the previous section on the simplification of boolean normal forms. Starting with papers by Quine and McCluskey [56, 57, 58, 52] the minimization of boolean normal forms in propositional logic was intensively studied. The notion of cut and subsumption as introduced by Quine can be easily extended to our framework by considering an atomic formula α and its negated variant $\overline{\alpha}$, in which the negation is encoded into the relation of α. Here, we were the first time faced with the observation that atomic formulas correspond somehow to the boolean variables. Considering multiple-valued logic as done by Hong [35] suggests to identify terms with logic variables.

Bryant [12, 13] has introduced ordered binary decision-diagrams (OBDD) as an efficient representation of boolean functions. Ordered binary decision-diagrams are rooted directed acyclic graphs representing boolean functions. They allow to perform many important operations such as evaluation, satisfiability tests, or equality tests on boolean functions very efficiently. Fixing an order between the variables one can transform an OBDD into a minimal normal form. The size of an OBDD can be, however, exponential in the number of involved variables. OBDD's are used as an alternative representation of boolean functions which are not longer represented as formulas in propositional logic. It is by no means clear how to translate an OBDD into a formula representation without loosing its "good" properties. In particular, the conversion of a formula into an OBDD and back to a formula may increase its size.

There are three major obstacles to using OBDD's as a tool for the simplification of first-order formulas over the reals:

1. As indicated above atomic formulas or the terms involved in atomic formulas correspond to the boolean variables. Our formulas involve many different terms and thus many different atomic formulas. The study of multi-valued decision-diagrams [42] suggests that we can encode our relations into the framework of multiple valued logic. This would allow to more elegantly identify terms instead of atomic formulas with boolean variables. This decreases the absolute number of different "boolean variables" but still leaves us with the problem of a large number of different terms.

2. The size of OBDD may be exponential in the size of the number of variables. Taking into account that we have to deal with many variables suggests that OBDD's obtained from our formulas are too large to be handled in reasonable space and time. Recall that in particular our standard simplifier is designed to be called very often as a subroutine of our quantifier elimination procedure presented in the next chapter. This forbids to incorporate such time consuming algorithms into our simplifier.

3. Any straightforward generalization of algorithms dealing with boolean functions to algorithms dealing with formulas over the language of ordered rings neglects the algebraic dependencies between different terms. Making use of these dependencies is a key feature of our simplifiers and one of the main reasons for their great success in particular in the framework of quantifier elimination.

Nevertheless finding an suitable generalization of ordered binary decision-diagrams which respects the algebraic dependencies and allows simplifications that are not lost by switching back to a formula representation is certainly an interesting research topic for its own.

2.9 Quantified Formulas

Up to now we have restricted ourselves to the simplification of quantifier-free formulas. In this section we discuss how to simplify first-order formulas that involve quantifiers. Firstly, we describe the adaption of the simplification methods introduced so far to formulas involving quantifiers. Secondly, we describe a technique for reducing the degrees of the occurrences of quantified variables. Thirdly, we introduce a method for simplifying formulas wrt. one chosen variable. This method is viewed as a simplifier for quantified formulas, because the distinguished variable is in our framework of quantifier elimination by virtual substitution a quantified variable.

2.9.1 Adaptions to Quantified Formulas

All simplification methods introduced in the previous sections can be extended to formulas involving quantifiers. The general idea is to simplify the scope of the formula by using the known methods without using the knowledge about the quantifier. The simplifier should, however, remove quantifiers binding variables that do not occur freely in their scope.

Using the theory concept for formulas involving quantifiers requires some adaptions. Recall that all variables occurring in a theory are treated as constants in the sense of logic. In the scope of a quantifier "$\exists x$," or "$\forall x$," respectively, the quantified variable x is not related with a variable occurring in some atomic formula contained in the theory. Therefore atomic formulas containing x do not contribute to the simplification of the scope of the quantifier. Technically, we delete all atomic formulas from the theory that contains the quantified variables. This deletion is, in particular, performed for the implicit theory: The simplification result of $\varphi \equiv x = 0 \lor \exists x(x = 0)$ is again φ, which is obviously equivalent to true, and not $x = 0$ obtained by simplifying the $x = 0$ occurring in the scope of "$\exists x$" wrt. the implicit theory $\{x \neq 0\}$.

2.9.2 Degree Shift

According to our simplification goal of simple terms we want to reduce the degree of the terms. The *degree shift* reduces the degree of a quantified variable which occurs only with certain powers. This is particularly useful for quantifier elimination by virtual substitution introduced in the following chapter. Since quantifier elimination by virtual substitution has certain degree restrictions the degree shift can even necessary for making possible its application.

To begin with consider, e.g. the formula $\exists x(u_1x^2+u_2<0)$. This formula is equivalent to

$$\exists y(y \geq 0 \wedge u_1y+u_2<0).$$

For proving the implication, choose $y=x^2$ and for proving the replication choose $x=\sqrt{y}$, with exists due to the condition $y \geq 0$. We have "shifted" the x to a lower degree.

We generalize this observation in the following proposition:

Proposition 2.12. *Let $\varphi \equiv \exists x\psi(\underline{u},x)$, where ψ is a quantifier-free formula with normalized atomic formulas. Let $x^{d_1}, \dots, x^{d_n}$ be the occurrences of x in the monomials of ψ. Suppose that $d=\gcd\{d_1,\dots,d_n\} \neq 0$. Let ψ' be the formula which is computed from ψ by replacing x^{d_i} by x^{c_i}, where c_i is the cofactor of d and d_i, i.e. $d_i=c_i \cdot d$. Then*

$$\exists x\psi(\underline{u},x) \Longleftrightarrow \exists y\psi'(\underline{u},y),$$

provided that d is odd, and for d even we have

$$\exists x\psi(\underline{u},x) \Longleftrightarrow \exists y\big(x>0 \wedge \psi'(\underline{u},y)\big).$$

The proof of the general case is analogous to the prove of our illustrating example using $y=x^d$ and $x=\sqrt[d]{y}$, respectively. The case of a universally quantified variable is, as usual, handled by negation.

The shift reduces the degree of x without increasing the degree of any other variable. In particular, we can sequentially apply shifts for a set of quantified variables in an arbitrary order without influencing other shifts. Applying a degree shift can, however, make other simplifications possible and can also make simplifications impossible.

Example 2.13 (Kahan's Problem). Kahan's problem, cf. [41] is one of the most well-known benchmark problems for quantifier elimination procedures:

$$\forall x\forall y\big(b^2(x-c)^2+a^2y^2-a^2b^2=0 \Longrightarrow x^2+y^2-1 \leq 0\big).$$

This can be equivalently replaced by

$$\forall x\forall y\big(y \geq 0 \Longrightarrow \big(b^2(x-c)^2+a^2y-a^2b^2=0 \Longrightarrow x^2+y-1 \leq 0\big)\big).$$

In particular with quantifier elimination by virtual substitution this leads to much faster running times and simpler results. We are going to discuss quantifier elimination for Kahan's problem in more detail in Section 6.5.3.

2.9.3 Wu–Ritt Reduction

Similar to the degree shift, the Wu–Ritt reduction [75, 76] provides a simplification method for reducing the degree of a variable. This new method is not restricted to quantified variables. It may lead, however, to an increase of the degree of other variables. In our framework of quantifier elimination we will choose the variable to be eliminated next as the main variable for the reduction. This is the reason why we consider the Wu–Ritt reduction as a simplification method for quantified formulas.

Consider as a first illustrating example the formula

$$\exists x(x^3 + c_2x^2 + c_0 = 0 \land x^3 + d_1x + d_0 > 0).$$

Rewriting the equation $x^3 + c_2x^2 + c_0 = 0$ as $x^3 = -c_2x^2 - c_0$ allows us to replace each occurrence of x^3 by $-c_2x^2 - c_0$. We then obtain the equivalent formula

$$x^3 + c_2x^2 + c_0 = 0 \land -c_2x^2 + d_1x - c_0 + d_0 > 0.$$

This formula contains an atomic formula with an x-degree of 2 while in the original formula all atomic formulas had an x-degree of 3.

We describe the general strategy for applying the Wu–Ritt simplification to an atomic formula. Suppose $c'x^{d'} + t' = 0$ is an equation contained in the theory of an atomic formula $cx^d + t \mathrel{\varrho} 0$, where t is a term with an x-degree smaller than d and t' is a term with a x-degree smaller than d' and $d' \leq d$. Suppose moreover that $c' \neq 0$ for all values of the parameters. Note that an equation in an implicit theory can actually be a disequation that occurred disjunctively.

In this situation we can deduce from the knowledge contained in the theory the equation $\frac{c'}{c}x^{(d-d')} \cdot (c'x^{d'} + t') = 0$. This equation has an x-degree of d and the x-initial is c. We can therefore replace the atomic formula $cx^d + t \mathrel{\varrho} 0$ by the atomic formula

$$cx^d + t - \frac{c'}{c}x^{(d-d')} \cdot (c'x^{d'} + t') \mathrel{\varrho} 0.$$

The original atomic formula and the new one are equivalent wrt. the implicit theory. Moreover the x-degree of the new one is less than d. Let ψ' denote the formula that results from the replacement. Then ψ and ψ' are equivalent, and the number of atomic formulas with an x-degree not less than d is decreased by 1. This process can, of course, be iterated. The degree of variables besides x may be increased by this process due to the multiplication with $\frac{c'}{c}$.

The assumption $c' \neq 0$ can be avoided by generating an explicit case distinction on the formula level. For instance, we can switch from ψ to

$$(c' \neq 0 \land \psi) \lor (c' = 0 \land \psi).$$

In this case we can apply the Wu–Ritt reduction to the first subformula of the disjunction. In the second subformula we can replace the equation $c'x^{d'} + t' = 0$ by $t' = 0$, which is also a degree reduction of one atomic formula.

The following is an application example for real quantifier elimination in the area of computational geometry [64].

Example 2.14. Consider in real 3-space the parabola $u_2 = u_1^2 \wedge u_3 = 0$. The following formula describes for parametric $r \in \mathbb{R}$ its r-offset wrt. the Euclidean metric. This is the set of all points with distance exactly r to the parabola:

$$\begin{aligned}\pi \quad \equiv \quad & \exists x_1 \exists x_2 \exists x_3 \exists s \exists t \big(x_2 = x_1^2 \wedge x_3 = 0 \wedge \\ & (x_1 - u_1)^2 + (x_2 - u_2)^2 + (x_3 - u_3)^2 = r \wedge \\ & -2sx_1 = u_1 - x_1 \wedge s = u_2 - x_2 \wedge t = u_3 - x_3\big).\end{aligned}$$

Due to certain degree restrictions, which we are going to discuss in Chapter 3, quantifier elimination by virtual substitution can eliminate all the quantifiers except for "$\exists x_1$." It thus stops with the result

$$\begin{aligned}\pi^* \quad & \equiv \quad \exists x_1(f_1 = 0 \wedge f_2 = 0) \\ f_1 \quad & = \quad r - u_1^2 + 2u_1x_1 - u_2^2 + 2u_2x_1^2 - u_3^2 - x_1^4 - x_1^2 \\ f_2 \quad & = \quad u_1 + 2u_2x_1 - 2x_1^3 - x_1.\end{aligned}$$

The reason that the procedure cannot continue here is that x_1 occurs in the two equations with degree 4 and 3, respectively, where at most degree 2 is allowed. Applying our Wu–Ritt based simplification we replace $f_1 = 0$ by another equation containing the remainder h of the pseudo division wrt. x_1 between f_1 and f_2:

$$\begin{aligned}\pi^{**} \quad & \equiv \quad \exists x_1(h = 0 \wedge f_2 = 0) \\ h \quad & = \quad 2r - 2u_1^2 + 3u_1x_1 - 2u_2^2 + 2u_2x_1^2 - 2u_3^2 - x_1^2 \\ f_2 \quad & = \quad u_1 + 2u_2x_1 - 2x_1^3 - x_1.\end{aligned}$$

The presence of the new equation, which is quadratic in x_1, allows quantifier elimination by virtual substitution to finally succeed wrt. an explicit theory $\{2u_2 - 1 \neq 0\}$. See Section 3.2.5 and Section 4.1.2 for details.

2.9.4 History and Implementation

In the current distributed version of REDLOG the standard simplifier can, of course, be applied to quantified formulas. All other variants have to be applied explicitly to the quantifier-free parts of formulas.

The degree shift is not part of any of the simplifiers but it is integrated in the quantifier elimination procedures. It is one of the heuristics included in REDLOG for treating formulas with higher degrees.

Similar to the degree shift, the Wu–Ritt reduction is currently not implemented in the simplifiers. A first test version implementing some special cases of the Wu–Reduction described above are included in the quantifier elimination code of REDLOG. This implementation is based on some code by Schweighofer [61]. Like the degree shift the Wu–Ritt reduction is of particular importance for the quantifier elimination and provides another technique for coping with higher degrees. We have obtained promising results of using the Wu–Reduction. A full version will become part of the next REDLOG release.

2.10 A Decision Heuristics

In the previous section we have seen several methods for the simplification of a formula, possibly in some normal form, wrt. a background theory. In this section we present a decision heuristics for simple formulas wrt. a background theory.

A decision heuristics is a function that maps a formula φ and a theory Θ to a value $\tau \in \{\text{true}, \text{false}, \text{dontknow}\}$ such that for $\tau = \text{true}$ we know that $\underline{\forall}\Big(\bigwedge \Theta \Longrightarrow \varphi\Big)$, and for $\tau = \text{false}$ we know $\neg\underline{\exists}\Big(\bigwedge \Theta \Longrightarrow \varphi\Big)$. If $\tau = \text{dontknow}$ we do not know anything about the formula, i.e. it may be either tautological, contradictory, or nothing of both.

Our simplifier provides such a decision heuristics. For heuristically deciding a formula we simplify it to σ by one of our simplifiers. The result τ is then defined as follows

$$\tau \equiv \begin{cases} \sigma & \text{for } \sigma \in \{\text{true}, \text{false}\} \\ \text{dontknow} & \text{otherwise} \end{cases}$$

The correctness of a decision heuristics defined this way follows immediately from the correctness of the simplifier.

Note that a function mapping an arbitrary formula to “dontknow” is also a decision heuristics. Note also, that there is a decision method for the real numbers with ordering. We will, however, have to apply the decision heuristics during the quantifier elimination very often to very simple formulas, in particular to atomic formulas. This is, however, still reasonable since is empirically turns out that the gain is enormous. The application of a decision method is not possible due to its complexity. The more time consuming Gröbner simplifier is also suitable for the application in a decision heuristics.

It should, however, not be called very often. The results are better than the results of a decision method based on the standard simplifier, in particular if the theory contains many equations.

Up to some extreme special cases a successful application of a decision heuristics based on the standard simplifier is restricted to atomic formulas and flat formulas. We finally give some examples of deciding simple formulas in the form $(\Theta, \varphi) \rightsquigarrow \tau$:

$$\begin{aligned}
(\{a > 0, b > 0\}, a + 1 > 0) &\rightsquigarrow \text{true}\\
(\{a > 0, b > 0\}, b + 1 < 0) &\rightsquigarrow \text{false}\\
(\{a > 0, b > 0\}, a + b > 0) &\rightsquigarrow \text{dontknow}\\
(\emptyset, c \neq 0 \lor c + 1 \neq 0) &\rightsquigarrow \text{true}
\end{aligned}$$

Note that in the third example we can actually prove that $a + b > 0$ is a consequence of the theory. Our simplifier, however, cannot figure this out. The last example demonstrates that the successful application of the heuristics is not restricted to a non-trivial theory.

2.11 Example Computations

All computations were done with our REDUCE package REDLOG on a SUN SPARC-4 using a heap size of $3 \cdot 10^6$ Lisp items. The timings are CPU times including garbage collection times, which make up about 3–7%.

2.11.1 A Rectangle Problem

There is a formula with 6 existential quantifiers followed by 2 universal quantifiers that asks for side lengths a, b of a rectangle such that it can be covered disjointly by two squares of different size, which is obviously impossible.

Using the standard simplifier without theory inheritance, our quantifier elimination procedure takes 171 s to compute a quantifier-free formula in a and b. This formula contains 3669 atomic formulas. It can be verified to be contradictory by applying a successive quantifier elimination to its existential closure, which takes 3.5 s.

With theory inheritance the elimination yields after only 13.3 s the result

$$2a - b < 0 \land a - b = 0 \land a > 0 \land b > 0,$$

which the Gröbner simplifier recognizes to be "false" in 0.03 s.

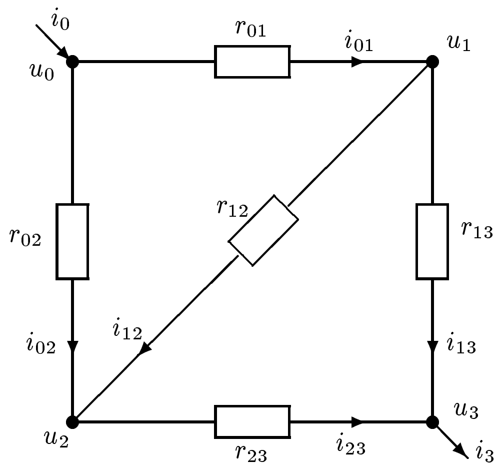


Figure 2.1: An electrical network

2.11.2 An Electrical Network

By quantifier elimination we compute for the electrical network in Figure 2.1 a quantifier-free formula which describes the currency i_{12} in terms of the resistances and the voltage $u_3 - u_0$. With the standard simplifier we obtain 31 atomic formulas in 0.5 s. If we turn off the additive smart simplification described in Subsection 2.4.2, we obtain 47 atomic formulas in 0.8 s.

2.11.3 Practical Networks

We summarize an example series obtained from quantifier eliminations in networks that describe a part of a motor. We apply our simplifiers to the final quantifier elimination results obtained with a simplifier corresponding to our standard simplifier without additive smart simplification and without theory inheritance.

The results are collected in Table 2.6, which reads from left to right as follows: subproblem number; input formula; standard simplifier; DNF with subsumption and cut; Gröbner application to this DNF; DNF with subsumption, cut, and simplifier-recognized implication; Gröbner application to this DNF; branchwise iterative tableau. Table 2.7 gives some exemplary timings.

Here the theory inheritance does not play such an important role as in Example 2.11.1. The DNF's obtained are in general much larger than the original input. After Gröbner simplification they provide in most cases the best result. Simplifier recognized implication sometimes yields smaller DNF's

Table 2.6: Motor series result sizes

no.	input	standard	DNF	Gröbner	good DNF	Gröbner	tableau
1	710	674	3000	164	3000	164	536
2	1420	966	2948	604	2948	604	829
3	94	88	57	40	30	29	35
4	292	259	439	257	273	165	203
5	157	139	162	146	102	96	97
6	994	908	3694	448	3694	448	716
7	710	199	425	107	425	107	159
8	473	410	1920	135	1920	135	376
9	235	188	389	96	389	96	158
10	478	461	1607	283	1607	283	375
11	168	156	139	133	87	87	53
12	2176	2100	2995	489	2995	489	756
13	358	342	1189	183	1189	183	251
14	710	674	3000	164	3000	164	536

Table 2.7: Motor series exemplary timings

no.	standard	DNF	Gröbner	good DNF	Gröbner	tableau
3	0.1 s	0.3 s	0.5 s	0.5 s	0.3 s	3.4 s
6	1.4 s	10.3 s	93.1 s	491.0 s	92.7 s	48.5 s
12	4.2 s	18.7 s	45.7 s	355.4 s	45.3 s	1400.0 s

but it is excessively time consuming. The tableau method is irrelevant for most cases and it is also extremely time consuming, in particular for large input formulas. There is no relation between the size of the input formula and the method of choice.

All input formulas are of a form better suited for DNF computation than for CNF computation. For clarity, we should point out that our methods can compute boolean normal forms for formulas of such sizes as in the example only if the latter are not too deeply nested.

There is a similar series of 10 formulas describing a stop light circuit. It confirms the results obtained from the motor series.

2.12 Conclusions

We have discussed the problem of simplifying quantifier-free formulas over ordered fields. In Section 2.2 we have specified what kind of formulas are considered simple thus making the notion of simplification more precise.

Furthermore, we have introduced the notion of a theory, which is used on one hand for entering external information into the simplification process, and on the other hand for relating information located on different boolean levels in deep formulas. The flat simplification methods (Section 2.4), namely smart simplification and the Gröbner method make use of the theory, while the deep simplification method (Section 2.5) constructs an implicit theory inheriting it to deeper boolean levels. The tableau methods (Section 2.6) systematically construct external theories, and then apply the deep simplification method.

We distinguish between a fast standard simplifier and sophisticated advanced simplifiers. The former consists of the deep simplifier with all the simplification methods for atomic formulas (Section 2.3) and the boolean methods (Subsection 2.4.1) and smart simplification (Subsection 2.4.2) for flat formulas. Adding the Gröbner method (Subsection 2.4.3) for flat formulas to the standard simplifier yields an advanced simplifier. Further examples for advanced simplifiers are the tableau methods (Section 2.6) and our simplifying boolean normal form computation (Section 2.7).

All simplifiers obey to parameterizations, which are implemented via global switches. There are three kinds of parameterization: Firstly, there are switches for resolving conflicts between different simplification goals (cf. Section 2.2). Secondly, time consuming simplification steps can be turned off, such as factorization (cf. Subsection 2.3.4), several features of the Gröbner method (cf. Subsection 2.4.3), or checking for simplifier-recognized implication with boolean normal form computations (cf. Subsection 2.7.2). Thirdly, one can turn off methods that may be disadvantageous such as branchwise iteration with the iterative tableau (cf. Subsection 2.6.1).

Our simplification methods provide in a natural way a decision heuristics for simple formulas. Such a decision heuristics has many applications in other algorithms, in particular, in our quantifier elimination.

The simplification methods are implemented in REDLOG, which is a part of REDUCE, cf. [30, 28]. In the current implementation, the Gröbner simplifier is restricted to boolean normal forms. REDLOG currently focuses on simplification and quantifier elimination. Numerous non-trivial examples illustrate the applicability and the relevance of our methods (Section 2.11).

Chapter 3

Quantifier Elimination by Virtual Substitution

In the previous chapter, we have exhaustively treated simplification as an important tool and subalgorithm for real quantifier elimination. This applies in particular to real quantifier elimination by virtual substitution, which we have made the method of our choice. Throughout this chapter we present a variety of algorithmic approaches for gaining efficiency. In contrast to the conceptual frameworks of structural elimination sets and repeated condensing discussed in the following chapters these approaches are very local in nature. At the same time we take the opportunity to give a general survey of quantifier elimination by virtual substitution, which will serve as a basis for the rest of this thesis. Our particular contributions mentioned above are the following:

- We analyze quantifier elimination by virtual substitution to consist of four distinct phases. This point of view allows us to systematically describe optimizations which otherwise appear to be quite *ad hoc* (Section 3.3).

- Boundary type selection strategies for formulas containing arbitrary quadratic constraints (Section 3.4).

- Corresponding selection strategies with elimination sets containing only terms that can be interpreted as real numbers (Section 3.5).

- Analysis of the interplay between our selection strategies and virtual substitution. This analysis is also extended to other selection strategies already present in the literature (Section 3.6).

- Efficient treatment of formulas of a special type where the quantified variable occurs only linearly except for exactly one occurrence in a quadratic *ordering* constraint. We exhibit that this approach cannot be combined with the selection strategies discussed before (Section 3.7).

3.1 History and Development

In 1988, Weispfenning studied linear problems in fields, ordered fields, and discretely valued fields, cf. [68]. Influenced by ideas of Cooper [23] and Ferrante, and Rackoff [34], he presented among others results a quantifier elimination procedure for linear formulas in ordered fields which is based on the computation of finite elimination sets containing test points. These test points are substituted into the original formula. Using this algorithm Weispfenning proves that quantifier elimination for ordered fields requires at most double exponential space and time. On the other hand he also proves that this quantifier elimination is in the worst-case *at least* double exponential in time and space. These results are certainly correct when measuring the complexity in terms of the word length of the input formula. Weispfenning's results, however, are actually much more precise: His algorithm is double exponential only in the number of quantifier blocks. For like quantifiers, it is single exponential in the number of quantifiers. The number and the size of the atomic formulas in the input plays a very minor role. To be precise, both time and space complexity are polynomially bounded in these parameters. Observe that in contrast to quantifiers and quantifier changes, the number of parameters, i.e., free variables does not significantly contribute to the complexity. This fact suggested that Weispfenning's algorithms provide in particular for problems involving many parameters a reasonable supplement to the only implemented quantifier elimination procedure at that time: Collins' cylindrical algebraic decomposition method [19].

In 1990 Burhenne [17] finished a first experimental implementation of Weispfenning's algorithm for ordered fields. At the same time Weber, who was then a student of Loos, independently worked on first implementations. Although the application range of these implementations was extremely restricted, it inspired further work in this area, which resulted in a common paper by Loos and Weispfenning [51]. Besides a more liberal notion of elimination sets, which we are going to discuss in Section 3.2.1, this joint paper contains the treatment of an extreme special case of Gauss elimination strategies, which we are going to discuss and to generalize in the next chapter.

In the following years, Weispfenning extended the idea of virtual substitution to arbitrary degrees making concrete the case of quadratic occurrences of some quantified variable [72]. Independently he has handled the special case of cubic occurrences in [71]. This great theoretical progress restimulated the interest in virtually efficient implementations of the methods. Sturm started to prepare such implementations in 1992.

Since 1995 the author together with Sturm has continuously been developing the computer logic system REDLOG containing besides the simplification methods discussed in the previous chapter and numerous general purpose algorithms on first-order formulas over various theories also highly optimized implementations of real quantifier elimination based on Weispfenning's ideas.

The current Version 2.0 of REDLOG, cf. [28, 30], is available in the commercial supported and distributed computer algebra system REDUCE 3.7. The real quantifier elimination of REDLOG 2.0 is still restricted to formulas in which the quantified variables occur with "low" degrees.

Beside the main development branch in REDUCE, the method was studied and implemented also in other systems. An alternative implementation of the quantifier elimination algorithm was done by Sturm [63] in the computer algebra system Risa/Asir. Meanwhile this implementation includes the simplification techniques described in this thesis implemented by the author together with Sturm. Sturm has parallelized the quantifier elimination using a parallel REDUCE [53] based on PVM on a Cray T3D. Another implementation was done in C using the SACLIB library [39, 15]. This version was then parallelized by the author, Gloor, and Sturm [26] using PARSAC [46, 47, 48].

The development of variants of quantifier elimination is closely connected to the development of the pure quantifier elimination algorithm. Extended quantifier elimination was firstly described by Weispfenning for solving optimization problems, [70] and firstly implemented by Kappert, cf [43]. The REDLOG implementation of the quantifier elimination provides also the possibility to compute sample points. A generic quantifier elimination based on virtual substitution was presented in a joint paper by the author together with Sturm and Weispfenning, [31]. REDLOG includes both a generic quantifier elimination and an extended generic quantifier elimination.

The most recent variant of quantifier elimination in the framework of virtual substitution is *local quantifier elimination* [32], which we are going to discuss in Chapter 6.

3.2 An Overview of the Method

3.2.1 Elimination Sets and Virtual Substitution

In an intuitive way an existential quantifier can be considered as an infinite disjunction over all real numbers. The basic idea of quantifier elimination by virtual substitution is to restrict this disjunction to a disjunction over a finite *elimination set* E of terms given parametrically in some suitable form.

It will turn out that this idea in fact always works. There are, however, a variety of obstacles, which one has to take care of. Consider, e.g. the following extremely simple situation: We wish to eliminate a single existential quantifier in front of a parametric linear equation:

$$\varphi \equiv \exists x(ax + b = 0).$$

In view of the discussion so far, it is a straightforward idea to simply substitute the formal solution of our equation:

$$E = \left\{-\frac{b}{a}\right\}, \quad \varphi' \equiv \bigvee_{t \in E} (ax + b = 0)[x /\!/ t].$$

After formally performing the substitution and formally reducing to lowest terms this uncovers the first problem:

$$(ax + b = 0)\left[x /\!/ -\frac{b}{a}\right] \longleftrightarrow -\frac{ab}{a} + b = 0 \longleftrightarrow 0 = 0 \longleftrightarrow \text{true}.$$

Our result "true" is obviously not correct since for $a = 0$ and $b \neq 0$ this equation does not have a real zero. Observe that exactly for the case $a = 0$ our formal solution $-\frac{b}{a}$ is not defined over the reals. This gives rise to the following modification of our initial idea: We redefine elimination sets to not simply contain *test terms* but additionally *guarding conditions*, which guarantee the existence of the corresponding terms in the field of the reals.

$$E = \left\{\left(a \neq 0, -\frac{b}{a}\right)\right\}, \quad \varphi' \equiv \bigvee_{(\gamma, t) \in E} \gamma \wedge (ax + b = 0)[x /\!/ t].$$

This results in

$$a \neq 0 \wedge (ax + b = 0)\left[x /\!/ -\frac{b}{a}\right] \longleftrightarrow a \neq 0,$$

which is, unfortunately, still wrong. This result does not cover the case $a = 0$ and $b = 0$ in which there exists zeroes. It is not hard to see that this second obstacle can be overcome by substituting at least one *arbitrary* test term without guard or, formally, with a guard that is equivalent to "true:"

$$E = \left\{ \left(a \neq 0, -\frac{b}{a} \right), (\text{true}, 0) \right\}, \quad \varphi' \equiv \bigvee_{(\gamma,t)\in E} \gamma \wedge (ax + b = 0)[x/\!/t].$$

This finally leads to a correct quantifier elimination result:

$$a \neq 0 \wedge (ax + b = 0)\left[x/\!/ -\frac{b}{a}\right] \vee \text{true} \wedge (ax + b = 0)[x/\!/0] \longleftrightarrow a \neq 0 \vee b = 0.$$

Note that our test term $-\frac{b}{a}$ is not a term in the language of rings, which we have fixed in Section 2.1 to be the basis for all our considerations. A *test term* is thus formally defined as a term over some suitably expanded language. In the sequel we refer to such terms as *pseudo terms*, and we denote by *test terms* both regular terms and pseudo terms. The substitution of pseudo test terms into atomic formulas is always performed in such a way that the substitution result σ has the following properties:

1. σ is a quantifier-free formula over the language of rings.
2. σ is—wrt. the expanded language—equivalent to the formal substitution result.

We refer to such substitutions as *virtual substitutions*, which explains the above notation "//" in contrast to the standard "/."

We turn the knowledge informally obtained from our example into a formal definition: Let $\psi(\underline{u}, x)$ be a quantifier-free formula. An *elimination set* for ψ wrt. x is a set of pairs (γ, t), where γ is a quantifier-free formula in the parameters $\underline{u}$ and t is a test term in the parameters $\underline{u}$, such that

$$\mathbb{R} \models \exists x(\psi(\underline{u}, x)) \longleftrightarrow \bigvee_{(\gamma,t)\in E} \gamma \wedge \psi[x/\!/t].$$

The following properties of the elimination sets are straightforward but extremely important from a computational point of view.

Proposition 3.1. *Let $\psi(\underline{u}, x)$ be a quantifier-free formula and E an elimination set for ψ wrt. x. Then the following hold:*

- *Each superset of E is an elimination set for $\psi(\underline{u}, x)$ wrt. x.*
- *Let $\psi'(\underline{u}, x)$ be a formula equivalent to $\psi(\underline{u}, x)$ then E is also an elimination set for $\psi'(\underline{u}, x)$ wrt. x.*

In his initial work [68] Weispfenning has used *Skolem sets* as elimination sets: Let $\psi(\underline{u}, x)$ be a quantifier-free positive formula. Denote by A the set of atomic formulas contained in ψ. A Skolem set S for ψ and x is a set of test terms such that the following holds: For any interpretation $\underline{a}$ of the parameters $\underline{u}$ and any choice of $x \in \mathbb{R}$, there is some test term $t(\underline{u}) \in S$ that when substituted for x simultaneously simulates the truth values of all atomic formulas in A.

In all later publications starting with [51] this framework has been considerably relaxed. All elimination problems were first reduced to the elimination of an existential quantifier in front of a quantifier-free positive formula. Then the elimination terms need not precisely simulate the truth values of all atomic formulas. Instead, it is sufficient to turn all those atomic formulas into "true," which are "true" for the chosen x. This is the framework we are going to follow in this chapter. In the following chapters on structural elimination sets and repeated condensing we will ourselves introduce a much more liberal framework, which does not at all simulate truth values of isolated atomic formulas but of entire subformulas.

Following the discussion in [51, 72] we give an outline of the algorithmic construction of elimination sets for linear and quadratic formulas starting with the linear case. Let $\psi(u_1, \dots, u_m, x)$ be a positive quantifier-free formula, and assume wlog. that all right hand sides of the contained atomic formulas are normalized to be zero. We are going to discuss how to eliminate the quantifier from $\exists x \psi(\underline{u}, x)$. This is sufficient because as mentioned in Section 2.1 any first-order formula can be turned into a positive prenex one, and then the quantifiers can be eliminated one by one starting with the innermost one; universal quantifiers are reduced to existential ones via the equivalence

$$\forall x \psi(\underline{u}, x) \longleftrightarrow \neg \exists x \neg \psi(\underline{u}, x).$$

Let $a_1, \dots, a_m \in \mathbb{R}$. We define the solution set of ψ for x wrt. $\underline{a}$ as

$$S_{\underline{a}}^{x}(\psi) = \{\, c \in \mathbb{R} \mid \psi(a_1, \dots, a_m, c) \,\}.$$

For ψ_1 and ψ_2 we have obviously that

$$S_{\underline{a}}^{x}(\psi_1 \wedge \psi_2) = S_{\underline{a}}^{x}(\psi_1) \cap S_{\underline{a}}^{x}(\psi_2), \quad S_{\underline{a}}^{x}(\psi_1 \vee \psi_2) = S_{\underline{a}}^{x}(\psi_1) \cup S_{\underline{a}}^{x}(\psi_2).$$

These identities together with the fact that the solution set of a polynomial inequality is a union of disjoint intervals show that $S_{\underline{a}}^{x}(\psi)$ itself is a union of disjoint intervals. Consider a conjunction $\psi_1 \wedge \psi_2$. It is easy to see that any endpoint of an interval in $S_{\underline{a}}^{x}(\psi_1 \wedge \psi_2)$ is also an endpoint of an interval in $S_{\underline{a}}^{x}(\psi_1)$ or it is an endpoint of an interval in $S_{\underline{a}}^{x}(\psi_2)$. This same holds for a disjunction $\psi_1 \vee \psi_2$. Iterating this argument we find that finally any endpoint of an interval in $S^x_{\underline{a}}(\psi)$ is an endpoint of an interval described by some atomic constraint

$$f \varrho 0, \quad \varrho \in \{\leq, <, =, \neq, >, \geq\},$$

which occurs literally in ψ. Note that the number, the kind, and the location of the intervals in $S^x_{\underline{a}}(\psi)$ depend on our choice of $\underline{a}$. There will, however, be suitable atomic formulas $f \varrho 0$ for any choice of $\underline{a} \in \mathbb{R}^m$.

Keep in mind that we have fixed $\underline{a} \in \mathbb{R}^m$, and let us for the moment restrict our attention to the case that our formula ψ is linear and involves only *weak* relations in the sense of Section 2.1. For such formulas all intervals in $S^x_{\underline{a}}(\psi)$ are of one of the following forms:

$$a, \quad [a, b], \quad]-\infty, a], \quad [a, \infty[, \quad \mathbb{R}.$$

If $S^x_{\underline{a}}(\psi) = \mathbb{R}$, then it obviously suffices to substitute any test term. Recall that we already have encountered this situation with the discussion of our introductory example. In all other cases the intervals contain at least one of their endpoints. It is thus sufficient to add to the elimination set all these endpoints, which obviously are the solution to the linear equations derived from the weak atomic formulas contained in ψ. As in our introductory example the guards will state that the corresponding denominators are non-zero.

We next turn to the case, where our linear ψ contains also strong relations. Then $S^x_{\underline{a}}(\psi)$ includes in addition intervals of the form

$$]a, b], \quad [a, b[, \quad]a, b[, \quad]-\infty, a[, \quad]a, \infty[.$$

The first two cases are already covered by our above elimination set construction: They contain one endpoint stemming from a weak atomic formula. For the third case $]a, b[$ we have to construct a point inside the interval. We take the exact midpoint $\frac{a+b}{2}$, where a and b are derived as usual as solutions of linear equations corresponding to the strict atomic formulas in ψ. The guard of this test term will state that the denominators of both a and b are non-zero. The last two cases are covered similarly by taking $a - 1$ and $a + 1$, respectively. This concludes the discussion of the purely linear case.

The introduction of quadratic constraints does not change anything in our case distinction wrt. the interval structure. The interval boundaries are now possibly solutions

$$\frac{-c_1 \pm \sqrt{\Delta}}{2c_2}, \quad \Delta = c_1^2 - 4c_2c_0$$

of quadratic equations $c_2x^2 + c_1x + c_0 = 0$. We refer to such pseudo terms of the form $\frac{a+b\sqrt{c}}{d}$ as root expressions. The corresponding guards are $c_2 \neq 0 \land \Delta \geq 0$. It is not hard to see that the substitution of a root expression into a polynomial yields another root expression. This observation covers in particular our computation of the arithmetic mean for intervals $]a, b[$. Weispfenning [72] has developed a virtual substitution for root expressions. As an example we give the substitution of $\frac{a+b\sqrt{c}}{d}$ for x into an atomic formula $f \leq 0$. According to our observation the substitution of our root expression into f yields another root expression $\frac{a'+b'\sqrt{c}}{d'}$, and we then have

$$\frac{a' + b'\sqrt{c}}{d'} \leq 0 \longleftrightarrow (a'd' \leq 0 \land a'^2 - b'^2c \geq 0) \lor (b'd' \leq 0 \land a'^2 - b'^2c \leq 0).$$

To conclude the discussion of the quadratic case observe that a quadratic constraint $c_2x^2 + c_1x + c_0 \; \varrho \; 0$ can be a "hidden" linear one for choices of parameters with $c_2 = 0$. For this reason such a constraint does not only generate square root expressions but also the test term $-\frac{c_0}{c_1}$ with guard $c_2 = 0 \land c_1 \neq 0$.

3.2.2 Extended Quantifier Elimination

Quantifier elimination by virtual substitution as sketched so far proceeds as follows: For the elimination of an existential quantifier there is an elimination set computed. Then the test points of this elimination set are virtually substituted into the original formula. Finally there is a finite disjunction formed over all the substitution results. We will give a more detailed analysis of the procedure into several phases in Section 3.3.

For now, consider the point at which the substitution of the elimination terms is performed. Here we actually loose some piece of information: The real numbers described by the substituted terms are actually *sample* values for the existentially quantified variable. Such sample values cannot be reconstructed from the final elimination result, which only gives necessary and sufficient conditions in the parameters for such values to exist. The idea of the *extended* quantifier elimination is now to retain this information.

For this purpose we do not construct the disjunction in the end but keep all substitution results separately and associate them with the corresponding test terms. The result for the extended elimination of one existentially quantified variable via an elimination set $E = \{(\gamma_1, t_1), \ldots , (\gamma_n, t_n)\}$ is a scheme of the following form:

$$\begin{bmatrix} \bigl(\gamma_1 \land \psi[x/\!/t_1]\bigr)(\underline{u}) & x = t_1(\underline{u}) \\ \vdots & \vdots \\ \bigl(\gamma_n \land \psi[x/\!/t_n]\bigr)(\underline{u}) & x = t_n(\underline{u}) \end{bmatrix}.$$

This scheme has to be interpreted as follows: Consider the original formula $\exists x\psi(\underline{u}, x)$. Whenever this formula holds for some choice $\underline{a} \in \mathbb{R}^m$ of the parameters $\underline{u}$, then at least one of the $\gamma_i \land \psi[x/\!/t_i]$ holds for $\underline{a}$, and the corresponding $x = t_1(\underline{a}) \in \mathbb{R}$ provides *one* sample solution for x. Conversely, whenever one of the $\gamma_i \land \psi[x/\!/t_i]$ holds, then also ψ holds for $x = t_i(\underline{a})$ and hence obviously $\exists x\psi$ holds.

Eliminating in this extended sense a block of several existential quantifiers we straightforwardly obtain an unnested scheme of the above form with sample solutions for all the involved variables obtained by backsubstitution.

For a block of universals quantifiers this semantics of the obtained scheme is dual to that of existential quantifiers: Whenever the original universally quantified formula does *not* hold then for the corresponding choice of parameters at least one of the $\gamma_i \land \psi[x/\!/t_i]$ is “false,” and the corresponding t_i provides one counterexample.

Note that we have explained extended quantifier elimination for an outmost prenex quantifier block. All inner quantifier blocks are eliminated conventionally.

In Section 2.9.2 we have described the *degree shift* as a simplification strategy reducing the degree of quantified variables. Due to the degree restrictions imposed by quantifier elimination by virtual substitution this simplification is certainly of particular importance. In connection with extended quantifier elimination there remains one point to be clarified. We have to adapt our sample solution terms obtained from a degree shifted formula in such a way that it is a sample solution for the corresponding original non-shifted formula. For this purpose, the answer $x = t$ in the extended elimination of a variable that was shifted at the degree d is replaced by $x = \sqrt[d]{t}$.

3.2.3 Pseudo Terms for Efficiency

So far we have computed elimination sets by generating test terms from solution interval boundaries. For any choice of parameters these test terms described suitable real numbers inside the interval. It has already turned out necessary for representing both fractions and roots to switch from proper terms over our formal language of ordered rings to pseudo terms over some expanded language. This in turn gave rise to the introduction of the notion of virtual substitution. Virtual substitution takes care that the substitution result is again a quantifier-free formula over the unexpanded language of ordered rings.

We are now going to introduce, for the sake of efficiency, further pseudo terms, which cannot even be interpreted as real numbers but live in some suitable real closed extension field $\mathbb{R}^*$. Again, we can, however, give a virtual substitution for these objects, which is semantically adequate and yields a quantifier-free formula in the language of ordered rings. To be more precise, our new pseudo terms will involve symbols ∞, which stands for an element larger than all standard real numbers, and ε which stands for a positive infinitesimal element.

Let us first turn to ∞. The pseudo term ∞ is used to replace all the test points of the form $a + 1$ for intervals extending to infinity. Accordingly we use $-\infty$ for replacing the test points of the form $a - 1$ for intervals extending to minus infinity.

Concerning the virtual substitution of e.g. ∞ note that there is always some fixed real number β larger than all zeroes of polynomials involved in the input formula, and for all $\beta' \geq \beta$ the signs of all polynomials and thus the truth values of all corresponding atomic formulas are fixed for given choices of parameters. It is not hard to see that for the virtual substitution ∞ should behave like β. This reduces the task of substitution to an analysis of whether the polynomial to be substituted into goes to plus or to minus infinity as x goes to infinity. This analysis in turn corresponds to a case distinction on the sign of the parameters.

For illustration we give the virtual substitution of ∞ into a constraint $\sum_{i=0}^{d} c_i x^d > 0$. The result of the substitution is the quantifier-free formula $\lambda(c_d, \dots, c_0)$, which is recursively defined as follows:

$$\lambda(c_d, \dots, c_0) \equiv c_d > 0 \vee c_d = 0 \wedge \lambda(c_{d-1}, \dots, c_0), \quad \lambda(c_0) \equiv c_0 > 0.$$

Denote by n the number of atomic formulas contained in our target formula ψ. The introduction of infinity has reduced some $O(n)$ part of the elimination set to $O(1)$. Recall that the overall size of our elimination set is $O(n^2)$. The introduction of pseudo terms involving our positive infinitesimal ε will reduce this overall size to $O(n)$ by replacing the arithmetic means causing the quadratic size.

Intuitively it should by now be clear that we wish to introduce test terms $a + \varepsilon$ or $b - \varepsilon$ for the bounded open intervals $]a, b[$ in $S^x_{\underline{a}}(\psi)$. Then we do no longer have to blindly combine all strict upper with all strict lower bounds, which caused the quadratic growth. It remains to be clarified how to virtually substitute $t \pm \varepsilon$ into atomic formulas. For non-trivial equations it is clear that any substitution of some $t \pm \varepsilon$ results in "false" because $t \pm \varepsilon$ describes a standard real number and thus cannot be a real zero of the left-hand side polynomial. Accordingly, substitutions into non-trivial disequations result in "true." For ordering constraints, there are again recursive schemes, which consider the signs of the substitution of t into the left hand side polynomials and into its derivatives.

As an example we describe the substitution of $t + \varepsilon$ into $f < 0$, where $f = \sum_{i=0}^{d} c_i x^i$. The result is $\nu(f)[x/\!/t]$, where $\nu(f)$ is recursively defined as follows:

$$\nu(f) \equiv f < 0 \vee \big(f = 0 \wedge \nu(f')\big), \quad \nu(c_0) \equiv c_0 < 0.$$

With extended quantifier elimination, unfortunately, we have to pay a price for our gain of efficiency: The "sample solutions" then do not provide precise sample values but have to be interpreted very carefully. In particular, when there occurs several symbols ∞ or several symbols ε in one sample solution for the variables of a quantifier block, then we do not know anything about the ordering between these symbols. In particular, it can be necessary for the correctness of a sample point in $\mathbb{R}^*$ that non-standard numbers introduced at different times are not identical in $\mathbb{R}^*$. It is thus reasonable to index these symbols with consecutive natural numbers such that definitely identical symbols can be identified in the end.

3.2.4 Boundary Type Selection

Recall that quantifier elimination by virtual substitution has to substitute for fixed parameter values at least one point from intervals

$$a, \quad [a,b], \quad]-\infty,a], \quad [a,\infty[, \quad \mathbb{R},$$

$$]a,b], \quad [a,b[, \quad]a,b[, \quad]-\infty,a[, \quad]a,\infty[.$$

For this purpose, we currently substitute *all* interval boundaries, shifted by $\pm\varepsilon$ where necessary. As mentioned above, it actually suffices, however, to substitute *one* point from each interval. Consider some interval boundary β coming from a linear atomic ordering constraint in which the head coefficient wrt. x of the left hand side polynomial is parameter-free. Then we know by combining the sign of this head coefficient with the ordering relation whether β imposes an upper bound or a lower bound (if it is relevant at all). Note that equations and disequations impose both upper and lower bounds. We call such boundaries β *known* boundaries. All others are called *unknown* boundaries. In Section 3.4 we will extend this idea to quadratic constraints.

Among the known boundaries we may decide for either upper or lower bounds. In either case we have to add all unknown boundaries because they match our choice in the worst-case. Of course, we will use this freedom of choice to minimize the size of the elimination set.

There is one subtle point with boundary type selection: Assume we have decided for lower bounds, but for fixed parameters $\underline{a}$ all lower bounds imposed by atomic constraints in ψ are actually irrelevant for the interval boundaries within $S^x_{\underline{a}}(\psi)$. In other words $S^x_{\underline{a}}(\psi)$ is unbounded from below. Then we are missing a relevant test point. For this case we add $(\mathrm{true}, -\infty)$ to the elimination set. This simultaneously serves as the point with trivial guard, which always has to be substituted. Analogously we add (true, ∞) when deciding for upper bounds.

3.2.5 Trivial Gauss Elimination

Let us consider an equation $ax + b = 0$ and assume that $a \in \mathbb{Q} \setminus \{0\}$. The special role of such an equation compared to all other relations is that its solution set contains only one point, namely $-\frac{b}{a}$. Let ψ be of the form

$$ax + b(\underline{u}) = 0 \wedge \psi'(\underline{u}, x),$$

and assume that we would like to eliminate $\exists x \psi$. Using the method described so far we would compute the elimination set from all atomic formulas in ψ together with the equation. We have, however, already discussed that

$$S^x_{\underline{a}}(\psi) = S^x_{\underline{a}}(ax + b = 0) \cap S^x_{\underline{a}}(\psi') \subseteq S^x_{\underline{a}}(ax + b = 0) = \left\{ -\frac{b}{a} \right\}.$$

It follows that $G = \left\{ \left(\mathrm{true}, -\frac{b}{a}\right) \right\}$ is a suitable set of test points. We call this special case of the elimination of a variable *Gauss elimination*, which has in this form been introduced in [51].

Weispfenning [72] has extended the idea to a quadratic equation instead of a linear one. In fact, this extension of Gauss elimination gave the key ideas to extend the general virtual substitution approach to the quadratic case.

3.2.6 Blockwise Elimination

For the discussion of the elimination algorithm we have restricted ourselves to the case of one existential quantifier. For several like quantifiers as in

$$\varphi \equiv \exists x_n \cdots \exists x_1 \psi(\underline{u}, x_1, \ldots, x_n),$$

we have argued to simply start with the elimination of the innermost one and then to proceed step by step to the outside.

Recall that the elimination result φ^* for the elimination of some innermost quantifier is obtained by disjunctively combining *intermediate results* φ_i^*. Each of these φ_i^* is obtained by substituting a test point into ψ. For the elimination of the second quantifier the situation thus looks as follows:

$$\exists x_n \cdots \exists x_2 \bigvee_i \varphi_i^*.$$

Suppose that we naively proceed to the elimination of "$\exists x_2$." That is, we would generate an elimination set from the atomic formulas in $\bigvee_i \varphi_i^*$ and disjunctively substitute these test points into $\bigvee_i \varphi_i^*$. This would mean that we substitute test points generated by atomic formulas of some φ_i^* also into all φ_j^* with $j \neq i$. This is not really necessary due to the obvious equivalence

$$\exists x_n \cdots \exists x_2 \bigvee_i \varphi_i^* \longleftrightarrow \bigvee_i \exists x_n \cdots \exists x_2 \varphi_i^*.$$

We refer to the application of this equivalence as *blockwise elimination*, which is clearly preferable.

For a block of like quantifiers the equivalence can be applied after the elimination of each quantifier. Due to the treatment of universal quantifiers by the rule $\forall x\psi \longleftrightarrow \neg\exists x \neg\psi$ a quantifier change from "$\exists$" to "$\forall$" corresponds to switching from "$\bigvee$" to "$\bigwedge$" and vice versa. At this point the current sequence of an applications of our blockwise elimination rule terminates. This corresponds to the fact mentioned in Section 3.1 that quantifier elimination by virtual substitution is double exponential only in the number of quantifier changes.

Let us again restrict our attention to the blockwise elimination of one single existential quantifier block.

$$\varphi \equiv \exists x_n \cdots \exists x_1 \psi(\underline{u}, x_1, \dots, x_n),$$

Our iterated application of our blockwise elimination rule can be considered as the computation of an *elimination tree* of the following form:

- Each node consists of a quantifier-free formula ψ_{kl} plus a list V_{kl} of variables. The corresponding partial elimination problem is $\exists V_{kl}\psi_{kl}$.
- The root is $(\psi_{11}, V_{11}) = (\psi, \{x_n, \dots, x_1\})$.
- For each node $(\psi_{kl}, \{x_n, \dots, x_k\})$ the children are

$$\begin{array}{rcl} (\psi_{k+1,1}, V_{k+1,1}) & = & (\gamma_1 \wedge \psi_{kl}[x_k /\!/ t_1], \{x_n, \dots, x_{k+1}\}), \\ & \vdots & \\ (\psi_{k+1,l'}, V_{k+1,l'}) & = & (\gamma_{l'} \wedge \psi_{kl}[x_k /\!/ t_{l'}], \{x_n, \dots, x_{k+1}\}), \end{array}$$

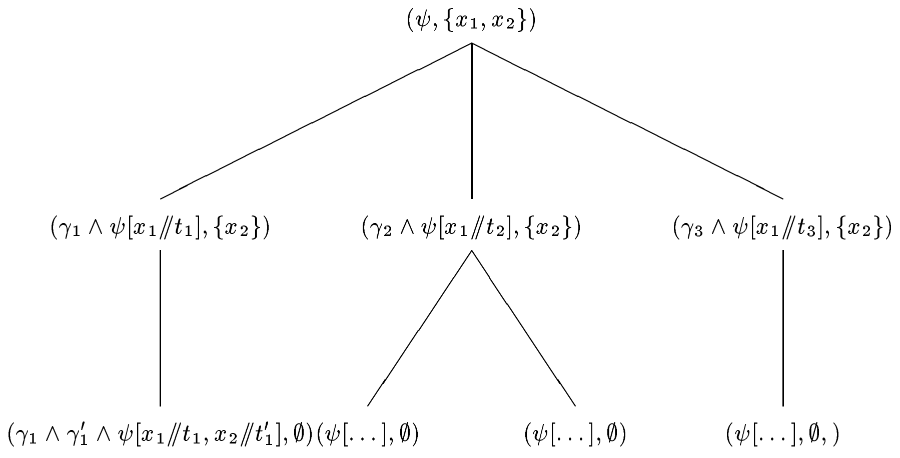

$(\psi, \{x_1, x_2\})$
$(\gamma_1 \wedge \psi[x_1/\!/t_1], \{x_2\})$
$(\gamma_2 \wedge \psi[x_1/\!/t_2], \{x_2\})$
$(\gamma_3 \wedge \psi[x_1/\!/t_3], \{x_2\})$
$(\gamma_1 \wedge \gamma_1' \wedge \psi[x_1/\!/t_1, x_2/\!/t_1'], \emptyset)$
$(\psi[\dots], \emptyset)$
$(\psi[\dots], \emptyset)$
$(\psi[\dots], \emptyset,)$

Figure 3.1: An elimination tree

where $\{(\gamma_1, t_1), \dots, (\gamma_{l'}, t_{l'})\}$ is an elimination set for ψ_{kl} wrt. x_k.

Figure 3.1 shows an example.

Elimination trees have the following properties: On each level the number of variables in all variable lists is identical. At the current state of our discussion the entire lists are actually identical but we will loosen this restriction soon. The number of variables decreases by one with each new level. All leaves of the tree contain the empty list. For each level with nodes (ψ_{k1}, V_{k1}), ... , (ψ_{kl}, V_{kl}) we have

$$\bigvee_{i=1}^{l} \exists V_{ki}\psi_{ki} \longleftrightarrow \exists x_n \cdots \exists x_1 \psi.$$

This holds in particular for the leaves of the tree for which $V_{n1} = \cdots = V_{nl} = \emptyset$. Hence the left hand side of the above equivalence is then quantifier-free.

Note that one can equivalently permute the variables within a prenex quantifier block. Consequently one can, as indicated above, choose the next variable to be eliminated independently for each node of the tree. The next variable is usually selected in such a way that the obtained elimination sets are small. It is not hard to see that this local criterion does not necessarily lead to an optimal final result.

A new insight we obtain from our elimination tree point of view is the following: Blockwise quantifier elimination as described above traverses the tree in a breadth-first search manner. Of course one can straightforwardly modify the elimination algorithm in such a way that it traverses the tree in a depth-first search manner. Both approaches have their advantages. Recall that we are actually interested only in generating the leaves of the tree. Breadth-first search keeps entire levels simultaneously in storage. This enables simplifications on the single levels including, in particular, elimination of duplicate nodes. Weispfenning [73] has shown that the majority of such duplicates comes into existence systematically. They can be detected independently on the traversion scheme by means of the so called *passive list*. Depth-first search saves storage and has the chance to discover a "true" leaf. If this happens the overall elimination result is "true," and the computation can immediately be aborted without constructing the entire tree. This fact makes depth-first search definitely preferable for decision problems.

Let us return once more to the idea of intermediate degree shift simplifications according to Section 2.9.2, which help us to avoid degree violations. It is reasonable to apply the shift not only for the next variable to be eliminated but to all variables of the current block. The reason is that simplifications after the elimination of one variable can destroy possible shifts.

3.3 The Phases of the Elimination Procedure

At first sight, quantifier elimination by virtual substitution proceeds as follows for the elimination of the prenex existential quantifier from $\exists x \psi$:

- Compute the set A of atomic formulas in ψ.
- Compute an elimination set E from A. There are numerous possible optimization strategies to be considered.
- Disjunctively substitute the test points from E into ψ using some suitable virtual substitution.

Finally we simplify the final result using the methods provided by Chapter 2. We agree to consider this simplification, in spite of its extreme importance, to be not part of the elimination procedure.

The next example shows that the simplification of the quantifier elimination results is an essential part of an efficient implementation of the quantifier elimination by virtual substitution:

Example 3.2 (Brown's Problem). The following problem has been given by Brown [10]: Let $(a_n)_{n\in\mathbb{N}}$ be a sequence of real numbers satisfying the relation

$$a_{n+2} = |a_{n+1}| - a_n.$$

Prove that $(a_n)_{n\in\mathbb{N}}$ is periodic with period 9. We translate this problem into the following first-order formula:

$$\begin{aligned}
\varphi &\equiv \forall x_1 \cdots \forall x_{11}\Big(\bigwedge_{i=1}^{9}\varphi_i \longrightarrow (x_1 = x_{10} \wedge x_2 = x_{11})\Big),\\
\varphi_i &\equiv x_{i+2} = |x_{i+1}| - x_i\\
&\equiv (x_{i+1} \geq 0 \wedge x_{i+2} = x_{i+1} - x_i) \vee\\
&\quad\ (x_{i+1} < 0 \wedge x_{i+2} = -x_{i+1} - x_i), \quad 1 \leq i \leq 9.
\end{aligned}$$

Our quantifier elimination by virtual substitution including the simplification of all intermediate results using the standard simplifier described in Section 2.5.2 computes the result "true" within 5 s. Without simplification we need 50 s to obtain the same result.

We are now at the point to state the optimized elimination set occurring computation more systematically. This is achieved by introducing another intermediate stage between the set of the atomic formulas on one side and the final elimination set on the other side. This new intermediate level is the candidate solution set.

Let A be a set of atomic formulas contained in ψ. A *candidate solution set* for ψ is a subset of the set of the formal solutions wrt. x of all equations

$$f(\underline{u}, x) = 0, \quad \text{where } f(\underline{u}, x)\ \varrho\ 0 \in \mathrm{A} \text{ for some } \varrho \in \{<, \leq, =, \neq, >, \geq\}$$

with their usual guards. This subset contains sufficiently many test points to provide for any interpretation of the parameters a superset of all real interval boundaries in $S^x_{\underline{a}}(\psi)$. The candidate solution sets used throughout this chapter are always the entire set of formal solutions as specified above. Accordingly we allow ourselves to speak of *the* candidate solution set for A. More restricted candidate solution sets will be introduced in the following chapter on structural elimination sets.

Compare the discussion of elimination set computation in Section 3.2.1 for details. The obtained pairs are expanded to triplets containing additional information on the boundary type. We know the following boundary types:

Strict lower bounds Test terms of which we definitely know that they describe the boundary a of some interval of one of the forms $]a, b]$, $]a, b[$, $]a, \infty[$.

Strict upper bounds Test terms of which we definitely know that they describe the boundary b of some interval of one of the forms $]a, b[$, $[a, b[$, $]\infty, b[$.

Strict bounds Test terms of which we know that they describe either a strict upper bound or a strict lower bound, but we cannot decide which of both. Recall from Section 3.2.4 that this happens whenever the head coefficient wrt. the current variable of some polynomial with a strict ordering is parametric.

Weak lower bounds Test terms of which we definitely know that they describe the boundary a of some interval of one of the forms $[a, b]$, $[a, b[$, $[a, \infty[$.

Weak upper bounds Test terms of which we definitely know that they describe the boundary b of some interval of one of the forms $]a, b]$, $[a, b]$, $]\infty, b]$.

Weak bounds Test terms of which we know that they describe either a weak upper bound or a weak lower bound, but we cannot decide which of both.

Isolated points Test terms which are formal solutions to equations.

Exception points Terms describing points excluded by a disequation. Note that in a certain sense such a point is quite similar to the boundary of an open interval.

Within this framework it turns out that all remaining subtasks of the elimination set computation use exactly the information encoded in our candidate solution set. We identify the following subtasks:

- The computation of the arithmetic means for coping with open intervals.
- The decision for whether to add or subtract ε.
- The decision for whether to add or subtract 1 for open intervals extending to ∞ or $-\infty$.
- All boundary selection strategies.
- The decision whether ∞, $-\infty$, or any other point has to be added.

A suitable selection of these subtasks turns our candidate solution set into an elimination set.

We finally summarize our discussion as follows: From an algorithmic point of view, quantifier elimination by virtual substitution splits into the following four phases:

1. Compute the set A of atomic formulas in ψ.
2. Compute the candidate solution set C from A.
3. Compute an elimination set E from C. This is exactly the point where the boundary selection strategies can be applied.
4. Disjunctively substitute the test points from E into ψ using some suitable virtual substitution.

3.4 Boundary Type Determination for Quadratic Constraints

We consider again the elimination of $\exists x \psi$ for positive quantifier-free ψ. Consider $\gamma \equiv c_2x^2 + c_1x - c_0 \geq 0$ with $0 \neq c_2 \in \mathbb{Z}$ and discriminant $\Delta = c_1^2 - 4c_2c_0$ contained in the atomic formulas of ψ. For $\Delta = 0$ it is easy to see that concerning the boundary type, γ yields either an isolated point or an exception point. Let us now focus on the case where $\Delta > 0$, i.e., γ delivers bounds. The test terms of the candidate solutions are then

$$s_1 = -\frac{c_1 + \sqrt{\Delta}}{2c_2}, \quad s_2 = -\frac{c_1 - \sqrt{\Delta}}{2c_2}.$$

The guards are obvious. The square root $\sqrt{\Delta}$ exists and is greater than zero. Suppose now that wlog. $c_2 > 0$. Then we have obviously $s_1 < s_2$. Taking the relation "$\geq$" of the considered atomic formula into account, it is obvious that

$$S^x_{\underline{a}}(\gamma) = \{\, x \in \mathbb{R} \mid x \leq s_1 \vee x \geq s_2 \,\}.$$

In other words s_1 is a weak upper bound and s_2 is a weak lower bound, and we have thus also determined the boundary types of our two candidate solutions. This information allows us to proceed as for the linear case in Section 3.2.4: After deciding for either substituting upper or lower bounds we are now able to also drop some quadratic test terms. The above described boundary determination can, of course, be easily extended to all other ordering constraints.

Weispfenning has sketched in [72] how to extend quantifier elimination by virtual substitution to formulas of arbitrary degree. This generalization considers, like the elimination procedure for the linear and quadratic case, the zeroes of the involved polynomials. For each parametric univariate polynomial $f(\underline{u}, x)$, we determine not only the zeroes in an appropriate way but in addition

- the number of distinct zeroes,
- the multiplicities of all zeroes,
- the ordering among all zeroes.

This information together with the sign of the head coefficient allows us to extend our ideas for boundary determination to the general case.

3.5 Boundary Type Selection for Real Elimination Sets

Recall our overview of quantifier elimination by virtual substitution in Section 3.2. Already within Section 3.2.1 we have described a completely working variant of the method. The remainder of that section focussed on the introduction of extended quantifier elimination and on optimizations. One of these optimizations was the introduction of pseudo terms containing ∞ an ε for efficiency reasons in Section 3.2.3. The new idea of this pseudo terms was that they can, in contrast to the terms used in Section 3.2.1, not be interpreted as real numbers. Accordingly, we will from now on refer to elimination sets not containing such terms as *real* elimination sets.

We have already discussed in Section 3.2.3 that non-real elimination sets, i.e., elimination sets containing ∞ or ε, are not perfectly suited for extended quantifier elimination because the sample points cannot be interpreted easily. On the other hand our discussion of boundary type selection in Section 3.2.4 was based on the availability of non-standard pseudo terms and did not discuss the compatibility of the arithmetic means required for real elimination sets with boundary type selection.

Since we will be particularly interested in extended quantifier elimination answers and thus in real elimination sets in Chapter 7, we are now going to analyze this issue.

The idea behind boundary type selection is as follows:

- For fixed parameters $\underline{u} = \underline{a}$, the solution set of our formula ψ wrt. to the variable x to be existentially eliminated is a disjoint union of possibly degenerate intervals.
- We can symbolically derive from ψ a superset of the endpoints of these intervals. Moreover, for certain endpoints we can decide whether they are lower or upper interval boundaries.

- We have to substitute at least one point from each interval. Due to the availability of ∞ and ε we can use the boundaries as test points possibly shifting them by $\pm\varepsilon$ for open intervals.

- Boundary type selection now means to either drop all upper or all lower bounds. This decreases the size of the elimination set and still yields one point in each interval.

With real elimination sets, there is no ε and we thus cannot substitute boundaries for open intervals. Instead we have computed the arithmetic means of all strict bounds, which are exactly the candidates for boundaries of open intervals. This guarantees to find a point exactly in the middle of each of the open intervals.

Let us try to save as much of the boundary selection idea as possible: We still substitute the arithmetic means of all strict bounds but concerning the weak bounds we decide for, say, upper bounds. This is not correct! Assume, e.g., that

$$S^x_{\underline{a}}(\psi) = [\alpha, \beta[\,,$$

and the largest strict lower bound γ is so small that $\frac{\gamma+\beta}{2} < \alpha$. Then our choice of test points obviously does not hit $S^x_{\underline{a}}(\psi)$.

It is easy to see that everything works fine when we additionally combine all strict bounds with all weak bounds except the upper bounds we have decided for to substitute. Unfortunately, as soon as there is any strict bound present at all, this elimination set grows at least as large as our initial one without any boundary selection. The reason is that the dropped weak lower bounds now yield by combination with strict bounds at least one test term instead of exactly one test term.

There is, however, one boundary type based optimization possible. We proceed as with the initial approach: No boundary type selection for the weak boundaries plus arithmetic means of the strict boundaries. Here we can drop all arithmetic means between like strict bounds. This yields, in general, an improvement but still keeps the size of the elimination set quadratic.

In the discussion so far we have skipped the elimination terms of the form $a \pm 1$ where a is a strict bound. Here it is obviously sufficient to use $a - 1$ for upper bounds a and $a + 1$ for lower bounds a.

3.6 Stronger Guards by Boundary Types

Let us now return to the general non-real elimination set types, where we are allowed to perform the usual boundary selection of Section 3.2.4 together with our optimizations of Section 3.4. For explaining our idea, it suffices to restrict to linear formulas.

Assume wlog. that we have decided for upper bounds, and we discover some atomic formula $\gamma \equiv c_1 x + c_0 > 0$, where $c_1(\underline{u})$ is a parametric term, which we cannot decide the sign of. This means that γ is in the worst case an upper bound. Consequently a corresponding test term for γ has to be added to our elimination set. Formally, the entry for γ in the candidate solution set is

$$\left(c_1 \neq 0,\ -\frac{c_0}{c_1},\ \text{“strict bound”}\right).$$

As indicated above, γ is only necessary for the worst case that our choice of parameters $\underline{u} = \underline{a}$ turns it into an upper bound, i.e., $c_1(\underline{a}) < 0$. This observation allows us to change the guard $c_1 \neq 0$ to the more restricted guard $c_1 < 0$.

The crucial point is that this new guard $c_1 < 0$ is much more convenient at the substitution stage. Consider the substitution of $-\frac{c_0}{c_1}$ for x into a constraint $f(\underline{u}, x)\ \varrho\ 0$. The result after the formal substitution into f has the following form:

$$\frac{T}{c_1^{\deg_x(f)}}\ \varrho\ 0.$$

In the spirit of virtual substitution this has to be turned into a quantifier-free formula over the language of ordered rings. Assume now that $\deg_x(f)$ is odd and that ϱ is an ordering relation. On the assumption that $c_1 \neq 0$ we have to multiply both sides by $c_1^{2\deg_x(f)}$ to preserve the direction of the ordering for any sign of c_1. With our improved knowledge that $c_1 < 0$ we can instead simply multiply by $c_1^{\deg_x(f)}$ and inverse the ordering relation ϱ to $\overline{\varrho}$. The result is then $T\ \overline{\varrho}\ 0$ in contrast to $Tc_1^{\deg_x(f)}\ \varrho\ 0$.

This is actually a great gain in view of the fact that c_1 is in general a parametric term containing also variables that are quantified from further outside. Avoiding a degree explosion, as we do here, can thus be crucial for quantifier elimination by virtual substitution to succeed on our current input.

All our observations are straightforwardly applicable also to quadratic bounds and also for arbitrary degrees. Again only the head coefficient wrt. x is relevant for the direction of the bound, and again this head coefficients enters the guard in a disequation.

To conclude this section we give an example demonstrating the gain of taking the chosen boundary type into account:

Example 3.3. Consider the formula

$$\exists x \psi, \quad \psi \equiv (ax + b \geq 0 \land cx + d \leq 0).$$

For eliminating the quantifier "$\exists x$"we obtain the candidate solution set

$$\Bigl\{\Bigl(a \neq 0, -\frac{b}{a}, \text{“weak bound”}\Bigr), \Bigl(c \neq 0, -\frac{d}{c}, \text{“weak bound”}\Bigr)\Bigr\}.$$

Choosing upper bounds for the computation of an elimination set allows us to change the guards $a \neq 0$ into $a < 0$ and $c \neq 0$ into $c > 0$. This results in the elimination set $\bigl\{(a < 0, -\frac{b}{a}), (c > 0, -\frac{d}{c}), (\mathrm{true}, \infty)\bigr\}$ for ψ and x. The substitution of these test points into ψ, using the knowledge introduced by the stronger guards, results finally in the following quantifier-free formula:

$$\begin{aligned}&(a < 0 \land ad - bc \geq 0) \lor (c > 0 \land ad - bc \leq 0) \lor \\ &\quad \bigl((a > 0 \lor (a = 0 \land b \geq 0)) \land (c < 0 \lor (c = 0 \land d \leq 0))\bigr).\end{aligned}$$

Without taking the boundary type into account we would obtain the elimination set $\bigl\{(a \neq 0, -\frac{b}{a}), (c \neq 0, -\frac{d}{c}), (\mathrm{true}, \infty)\bigr\}$. Substituting these test points we would obtain higher degrees in some terms:

$$\begin{aligned}&(a \neq 0 \land a^2d - abc \leq 0) \lor (c \neq 0 \land acd - bc^2 \leq 0) \lor \\ &\quad \bigl((a > 0 \lor (a = 0 \land b \geq 0)) \land (c < 0 \lor (c = 0 \land d \leq 0))\bigr).\end{aligned}$$

3.7 A Quadratic Special Case

In the previous sections we have briefly addressed the problem that substitution of elimination terms into formulas will in general increase the degree of the involved parameters. We have seen that we essentially eliminate the quantifiers in a prenex block from the inside to the outside always treating all variables quantified from further outside as parameters. We thus have the problem that increasing the degree of a "parameter" can actually mean increasing the degree of a quantified variable, which has to be eliminated later.

For a degree restricted implementation this means the following:

- The elimination procedure can fail although all occurrences of quantified variables in the input are at most quadratic.
- We cannot straightforwardly decide by inspection of the input whether this will happen or not.

Weispfenning [68] has shown that this problem does not occur when all occurrences of all quantified variables are linear. Recall that the product of two quantified variables is not a linear occurrence.

In the non-linear case it is thus crucial for the success of the elimination to at least heuristically keep the degrees low as much as possible. One such heuristics was the improved substitution of the previous section. Other heuristics are polynomial factorization, or the degree shift discussed in Section 2.9.2.

In this section we take a more systematic approach. We are going to identify situations, where quadratic occurrences of quantified variables are involved but an increase in the degrees can systematically be avoided.

The following proposition describes the relevant configuration, which we refer to as the *quadratic special case.* For certain quadratic constraints it suffices to substitute the zeroes of the formal derivatives of their left hand side wrt. the current quantified variable. These derivatives are obviously linear in this variable.

Proposition 3.4 (A Quadratic Special Case). *Let $\varphi(\underline{u}) \equiv \exists x \psi(\underline{u}, x)$, where ψ is a positive quantifier-free formula. Moreover, ψ is linear in x up to one quadratic constraint $\alpha \equiv c_2x^2 + c_1x + c_0 \ \varrho\ 0$ with $\varrho \in \{<, \leq, >, \geq\}$. Let C' be a candidate solution set obtained only from the linear constraints in ψ. Then an elimination set E for ψ and x can be computed as follows:*

1. *Compute an elimination set E' from C' without applying any boundary selection strategies.*

2. *Obtain E from E' by adding $\left(c_2 \neq 0, \frac{-c_1}{2c_2}\right)$, $\left(c_2 = 0 \land c_1 \neq 0, \frac{-c_0}{c_1}\right)$, $(\mathrm{true}, -\infty)$, and (true, ∞).*

Note that the term $-\frac{c_1}{2c_2}$ is the zero of the formal derivative of $c_2x^2 + c_1x + c_0$.

Proof. We fix the parameters $\underline{u}$ to $\underline{a} \in R^m$. Assume that $S^x_{\underline{a}}(\psi)$ is nonempty. The case of an unbounded interval is covered by the test points $(\mathrm{true}, -\infty)$ and (true, ∞). Assume now that $S^x_{\underline{a}}(\psi)$ is bounded. The endpoints of $S^x_{\underline{a}}(\psi)$ are contained in the set of zeroes of the terms in ψ. The case that at least one of these endpoints is given by a formally linear constraint in ψ, then it is covered by E' obtained from C'. The case that $c_2(\underline{a})$ degenerates to 0 is covered by the test point $\left(c_2 = 0 \land c_1 \neq 0, \frac{-c_0}{c_1}\right)$. The only remaining case is that *both* endpoints of $S^x_{\underline{a}}(\psi)$ are given by our quadratic ordering constraint α. That is, we have a parabola describing a bounded interval. Then, obviously, the extremum of the parabola lies inside the interval. This extremum is described by the formal derivative of the parabola wrt. x.

Note that from the fact that the degrees in the quantified variables are not relevantly increased we may not immediately conclude that our quantifier elimination procedure will definitely succeed for a block of quantifiers involving a quadratic special case. For this one has in addition to take care that the hypothesis of Proposition 3.4, i.e., exactly one quadratic ordering constraint is invariant under the elimination of one variable. The blockwise elimination discussed in Section 3.2.6 supports this. One can, however, still easily construct examples where the iteration fails. This can finally be avoided by suitable variable transformations.

In practice, the quadratic special case has one major disadvantage: We are not allowed to choose a boundary type for selecting an elimination set from the set of all candidates. This increases the size of the computed elimination set considerably. Our experiences with the quadratic special case suggest to use it only for a second elimination run after a degree violation.

We conclude this section with an example illustrating the incompatibility of the quadratic special case with boundary type selection. Consider the following formula, which is obviously "true:"

$$\exists x(x^2 - 1 \leq 0 \wedge 2x - 1 \geq 0)$$

In the elimination set of Proposition 3.4 only the substitution of the test point $(\mathrm{true}, 1/2)$ results in "true." This test point would be removed by selecting only upper bounds.

3.8 Conclusions

After briefly sketching the historical development of the virtual substitution method for real quantifier elimination we have given an overview over the method in general. This presentation will serve as a reference also for the remainder of this thesis.

In Section 3.3 we have analyzed quantifier elimination by virtual substitution to consist of four distinct phases. This point of view has enabled us to systematically locate the application of boundary selection strategies on one hand and the addition of certain test points not originating from atomic formulas on the other hand.

In Section 3.4 we have generalized selection strategies for test points discussed elsewhere to the quadratic case. This is a particularly exciting result since the corresponding strategies for the linear case have extended the practical application range of quantifier elimination by virtual substitution dramatically.

In Section 3.5 we have extended the idea of selection strategies based on boundary types to elimination sets not containing pseudo terms. Such elimination sets are—though obviously less efficient—of particular importance for extended quantifier elimination approaches, which we will extensively use in the context of generalized scheduling in Chapter 7.

In Section 3.6 we have demonstrated how to obtain an additional gain from all the selection strategies discussed so far: The decisions for certain boundary types, which have been originally introduced for reducing the size of the elimination set, can be reused at the substitution stage. This is a first evidence for the fact that in view of practical applicability it is not adequate to consider the various "phases," which we have identified in Section 3.3, isolated. We will rediscover this fact in Chapter 5.

Section 3.7 yields besides some relevant algorithmic "tricks" another important conceptual insight: We are now faced with two independent valuable strategy types, which are, unfortunately, incompatible. Any reasonable implementation of quantifier elimination by virtual substitution will thus not simply include a variety of optimization strategies but also some suitable heuristics on which strategy to apply.

Chapter 4

Structural Elimination Sets

For the computation of elimination sets we have up to now restricted our attention to the set of atomic formulas contained in the given formula. The boolean structure of the formulas was never considered except for the extreme special case of Gauss elimination discussed in Section 3.2.5. Structural elimination sets, in contrast, provide a concept for making use of the boolean structure of the formula. This will considerably decrease the size of the candidate solution sets introduced in Section 3.3. The main idea of structural elimination sets is to compute candidates not from atomic formulas but from suitable complex subformulas. Moreover, this idea is combined with the construction of an implicit theory similar to that used for deep simplification in Chapter 2.

4.1 Quantifier Elimination wrt. a Theory

Our simplifiers discussed in detail in Chapter 2 allow the user to specify an explicit background theory. The impressive results of simplification wrt. a theory, which is not explicitly added to the formula, gave rise to modify our quantifier elimination procedure in such a way that it also allows a background theory Θ for the elimination of quantifiers from some formula φ. In analogy to the simplification situation the output will then not be perfectly equivalent to φ, but we have

$$\bigwedge \Theta \longrightarrow (\varphi \longleftrightarrow \varphi^*).$$

Recall from the introduction of the concept of a theory in Section 2.2 that our theory Θ is a set of atomic formulas. From Section 2.9 it is clear that any atomic formula $\alpha \in \Theta$ that contains bound variables is explicitly deleted.

Let in the situation above $u_1, \dots, u_m$ be the parameter variables in φ and Θ. We define the *range* of the quantifier elimination as the following subset of the parameter space $\mathbb{R}^m$:

$$\Big\{ (a_1, \dots, a_m) \in \mathbb{R}^m \;\Big|\; \bigwedge \Theta(a_1, \dots, a_m) \Big\}.$$

Parameter interpretations in the range are called *admissible.*

The first use of the specified background theory Θ inside the quantifier elimination procedure is to simplify all intermediate results wrt. Θ. But we can do much more.

First we can make use of the theory for the computation of the candidate solution set. Second the theory will enable us to apply further Gauss eliminations, which we have introduced in Section 3.2.5.

4.1.1 Candidates wrt. a Theory

Recall from the discussion of quantifier elimination by virtual substitution in Section 3.2 and in Section 3.3 that with the computation of candidate solution sets some assumptions on coefficient terms are introduced. These assumptions state that certain coefficients of the polynomials contained in the considered formula do not vanish or are non-negative. The assumptions are encoded into the guards of the candidate solutions and are inherited to the test points of the elimination set. They finally enter the quantifier-free result formula in the substitution phase where they are conjunctively added to the virtual substitution result of the corresponding term.

In Section 3.2.4 and in Section 3.4 we have improved the step from the candidate solution set to the elimination set by introducing boundary type selection strategies. These strategies use the signs of the head coefficients of the polynomials involved in the considered formula.

With both the candidate solution set computation and the elimination set computation we are thus faced with the situation that we would like to know the signs of certain polynomials in the parameters. Such sign information would allow us on one hand to generate simpler guards and on the other hand to improve the elimination set computation by knowing the boundary type of the corresponding candidate solutions.

The sign computation for a polynomial in the parameters is trivial in the case that the polynomial is actually an integer. For polynomials containing parameters, however, the sign may also be fixed for any values of the parameters. Consider, e.g., $u^2 + 1$ which is greater zero for any choice of real numbers for the parameter u. It is easy to see that using a decision procedure for the reals we can actually compute the sign in such cases. This is of some theoretical interest but not suited for improving the practical applicability of our quantifier elimination: The gain won by knowing the sign information is not so big that it is reasonable to accept a decision procedure as a frequently called subalgorithm. Note that the complexity of the decision problems of the reals is double exponential and thus close to the quantifier elimination itself. We can and will, however, apply a decision heuristics as discussed in Section 2.10 to handle simple cases such as $u^2 + 1$ above. We have already seen that this can be done very efficiently.

We summarize where we can use knowledge of coefficient signs in our quantifier elimination by virtual substitution:

- For the computation of a candidate solution of a linear atomic formula $c_1 x + c_0 \varrho 0$ we have to assume $c_1 \neq 0$.
- In the case of a quadratic atomic formula we first have to distinguish between the proper quadratic case, given by the condition of a non-vanishing head coefficient on one hand and the degenerate linear case on the other hand. In the proper quadratic case we have to assume in addition that the discriminant of the corresponding quadratic equation is not negative.
- For both linear and quadratic constraints we can determine the boundary type if we know the sign of the head coefficient of the polynomial involved in the constraint.

Recall from Chapter 2 that all our simplifiers are designed in such a way that they accept an optional input theory. This is in particular true for the heuristics of Section 2.10, which we have suggested for checking signs. We thus specify that quantifier elimination by virtual substitution also accepts an optional input theory. This allows us to compute signs of polynomials not only when these signs are fixed over the entire parameter space but also when the signs are fixed only for the admissible parameter values. As a consequence, the elimination result is then, of course, only correct on the assumption in the input theory. The more we restrict the range of the quantifier elimination by using stronger theories the more successful sign decisions we may expect.

With respect to a theory it can happen that a non-zero polynomial containing parameters vanishes for all admissible parameter values. Consider, e.g., the quadratic constraint $x^2 + ux + 1 \geq 0$. The discriminant is $\Delta = u^2 - 4$. Specifying in the background theory that $u^2 - 5 > 0$, our decision heuristics decides $\Delta \geq 0$ to be "true" wrt. the theory, and we can drop the corresponding guard. If we add $u^2 - 4 = 0$ to our theory, then we can detect that the quadratic constraint provides only one rational candidate solution $-\frac{u}{2}$. This avoids the virtual substitution of a square root expression. In Section 3.7 we have seen that this can be crucial for the success of the overall elimination.

Our use of a background theory can be extended to determine boundary types that are only known wrt. the range of the quantifier elimination.

4.1.2 Gauss Elimination wrt. a Theory

In the previous section we have discussed how to make use of a background theory for the computation of candidate solutions. We are now going to discuss how to extend the use of the background theory from the candidate solution computation in the general case to the special case of Gauss elimination. Recall from the discussion of the trivial Gauss elimination in Section 3.2.5 that we can apply Gauss elimination only if the considered equation is non-trivial. That is for any choice of the parameters the equation has a finite solution set. A sufficient condition for this is that at least one coefficient of the considered equation does not vanish. More formally an equation $\sum_{i=0}^{d} c_i x^i = 0$ has a finite solution set, if $c_i \neq 0$ for at least one of the c_i. We try to decide this condition by again applying the decision heuristics of Section 2.10.

To be more precise we define the notion of a Gauss equation wrt. a theory Θ. An equation $\alpha \equiv \sum_{i=0}^{d} c_i x^i = 0$ is called a *Gauss equation* wrt. Θ if

$$\bigwedge \Theta \longrightarrow (c_0 \neq 0 \vee \cdots \vee c_d \neq 0).$$

Note that even with the empty theory this condition detects more Gauss cases than the informal requirement above that at least one of the c_i is non-zero. Consider e.g., the equation $u_1 x^2 + (u_1 + 1)x + u_2 = 0$ and an empty background theory. Using our new approach we can successfully apply our decision heuristics to verify that $u_1 \neq 0 \vee u_1 + 1 \neq 0 \vee u_2 \neq 0$ is "true" for any parameter values. The old approach cannot detect this because neither u_1, $u_1 + 1$, nor u_2 can be guaranteed to be different from zero. This condition is actually equivalent to the condition that α has a finite solution set for all parameter values in the range. Again we make use of our simplifier-based decision heuristics to test if a given equation is a Gauss equation. Due to the heuristics approach we can, of course, not detect all Gauss equations, but again we considerably improve the practical applicability of the procedure.

4.2 Quantifier Elimination with Implicit Theory

The concept of an implicit theory has been introduced in Section 2.5.1 for evaluating the relation between atomic formulas contained in a formula on different boolean levels. In this section we are going to describe how to make use of this concept for quantifier elimination by virtual substitution extending the ideas of the previous section for handling an explicitly given background theory.

We give a formal definition of the notion of the implicit theory Θ of a subformula ψ of a quantifier-free positive formula φ. As introduced in Section 2.4.2 we denote for an atomic formula α by $\overline{\alpha}$ its implicit negation:

1. If $\psi \equiv \varphi$, then its implicit theory is $\Theta = \{\varphi\}$, provided that φ is an atomic formula and $\Theta = \emptyset$ if φ is a complex formula.

2. Assume that ψ occurs in a conjunction $\varphi' \equiv \psi \wedge \alpha_1 \wedge \cdots \wedge \alpha_k \wedge \psi_1 \wedge \cdots \wedge \psi_l$ inside φ, where the α_i are atomic formulas and the ψ_j are complex subformulas. Assume that Θ' is the implicit theory of φ'. Then the implicit theory of ψ is
$$\Theta = \Theta' \cup \{\alpha_1, \dots, \alpha_k\}.$$

3. Assume that ψ occurs in a disjunction $\varphi' \equiv \psi \vee \alpha_1 \vee \cdots \vee \alpha_k \vee \psi_1 \vee \cdots \vee \psi_l$ inside φ, where the α_i are atomic formulas and the ψ_j are complex subformulas. Assume that Θ' is the implicit theory of φ'. Then the implicit theory of ψ is
$$\Theta = \Theta' \cup \{\overline{\alpha_1}, \dots, \overline{\alpha_k}\}.$$

Note that this definition of an implicit theory differs from that given in Section 2.5.1: In our new definition also an atomic ψ inherits a theory enriched by its atomic neighbors.

The following proposition shows how to apply the implicit theory within the elimination process.

Proposition 4.1. *Let φ be a quantifier-free positive formula, let ψ be a subformula of φ, and let Θ be the implicit theory of ψ in φ. Let φ' be the formula constructed from φ by replacing ψ with $\bigwedge \Theta \wedge \psi$. Then φ and φ' are equivalent.*

Proof. Fix an interpretation of all variables, and first assume that φ' evaluates to "true." Since both φ and φ' are positive, this evaluation cannot depend on $\bigwedge\Theta \wedge \psi$ evaluating to "false." We can thus replace $\bigwedge\Theta \wedge \psi$ by ψ.

Assume now vice versa that φ evaluates to "true," and assume furthermore that it is relevant for this that ψ evaluates to "true;" otherwise we can certainly substitute it with anything. Assume for a contradiction that substituting ψ with $\bigwedge\Theta \wedge \psi$ turns φ into "false." Then there must be some $\vartheta \in \Theta$ that evaluates to "false." There are two possible sources for this ϑ. Firstly, it can occur conjunctively with a subformula containing ψ. Then this conjunction is "false," and thus ψ is irrelevant for the truth value of φ, which contradicts our assumption. Secondly, some $\overline{\vartheta}$ equivalent to $\neg\vartheta$ can occur disjunctively with a subformula containing ψ. Since $\overline{\vartheta}$ is "true," the entire disjunction is "true," and again ψ is irrelevant, which again contradicts our assumption.

Example 4.2. Consider the formula $\varphi \equiv a > 0 \vee (b \geq 0 \wedge ax + c > 0)$ and let us focus on the subformula $\psi \equiv ax + c > 0$. Its implicit theory Θ is $\{a \leq 0, b \geq 0\}$, and we have

$$a > 0 \vee (b \geq 0 \wedge ax + c > 0) \longleftrightarrow a > 0 \vee \big(b \geq 0 \wedge (a \leq 0 \wedge b \geq 0 \wedge ax + c > 0\big)$$

as predicted by Proposition 4.1. We are going to return to our formula φ in Example 4.4.

Note that the simultaneous application of the above proposition to two subformulas and their implicit theory is not correct. We illustrate this in our next example

Example 4.3. Consider the formula $\varphi \equiv a = 0 \vee a = 0$. The implicit theory of both occurrences of $a = 0$ is $\{a \neq 0\}$. A simultaneous application of Proposition 4.1 for both occurrences of $a = 0$ would result in

$$(a \neq 0 \wedge a = 0) \wedge (a \neq 0 \wedge a = 0) \leftrightarrow \text{false} \not\leftrightarrow \varphi.$$

Fortunately, there is no risk from this observation for applying the implicit theory within a quantifier elimination step exactly like the explicit one: All decisions made for applying the theory for the candidate solution computation or for Gauss elimination combine an atomic formula containing the current variable with atomic formulas in the theory not containing it.

In Section 3.3 we have introduced the notion of a candidate solution set for a set of atomic formulas. In the previous chapter our candidate solution sets for a positive quantifier-free formula ψ and a variable x always consisted in the set of all guarded solutions of equations derived from the atomic formulas in ψ plus the corresponding boundary type. The definition was, however, more liberal. A candidate solution set contains at least the candidates for the interval boundaries of $S^x_{\underline{a}}(\psi)$ for all $\underline{a} \in \mathbb{R}^m$. In Section 4.1.1 we have already silently allowed ourselves to sort out certain candidates. The framework of an implicit theory now enables us to fully exploit the freedom provided by the definition of candidate solution sets: Our future candidate solution set computations will no longer be based on the set of atomic formulas but on ψ itself making full use of its boolean structure by means of implicit theory construction.

The theory approach actually affects all three aspects of a candidate solution, the guard, the term, and the boundary type:

1. It allows improved guards as we have already seen in Section 4.1.1.
2. A candidate solution can be completely dropped. In a certain sense this corresponds to "affecting the term." Below we will introduce another more sophisticated theory based technique that affects the term.
3. The theory yields important information concerning the boundary type. For instance, $c_1 > 0 \in \Theta$ determines $c_1x + c_0 > 0$ to be a strong *lower* bound.

As long as restricting to explicit theories theory application is a very simple deal: We receive better elimination results from the theory, and we pay with the fact that these results are equivalent only wrt. the corresponding explicit theories. With implicit theory construction in contrast it appears that we have a similar gain from the implicit theories but the final elimination result is perfectly equivalent. There is, however, a price to pay also with implicit theory optimizations: Assume that we use the implicit theory to turn some bound t coming from an ordering constraint α into an upper bound. In general, we have to expect that α occurs also at some other point in our formula, where a different implicit theory is valid. We furthermore have to expect that this other theory is not suitable for turning t into an upper bound. We then have to decide between two possibilities:

1. Generate two different candidate solutions from α.
2. Do not make use of the upper bound information for t in the first position.

In the first case we pay for better candidates by obtaining more different candidates. In the second case we do not pay anything, but we also do not receive any gain from the implicit theory. Most interestingly, in the next chapter on repeated condensing we will observe nearly the same trade-off in a completely different context.

Let us now turn to an example that illustrates that the application of the implicit theory approach is actually useful in general.

Example 4.4. As in Example 4.2 consider the formula $\varphi \equiv a > 0 \vee (b \geq 0 \wedge ax + c > 0)$ and let us perform the quantifier elimination for $\exists x \varphi$. Recall that the implicit theory of $\psi \equiv ax + c > 0$ is $\{a \leq 0, b \geq 0\}$. We thus know that the formal solution $-\frac{c}{a}$ represents an upper bound.

The implicit theory may contain not only atomic formulas in the parameters but also atomic formulas involving the current quantified variable. Such atomic formulas are actually useful. One example for using such atomic formulas is the application of the Wu–Ritt reduction as described in Section 2.9.3: For computing the candidate solutions of an atomic formulas we can reduce the involved term wrt. all non-trivial equations contained in the implicit theory. This observation is what we have announced above to be another theory based technique affecting the term of a candidate solution.

4.3 Generalized Gauss Elimination

In Section 4.1.2 we have generalized the Gauss elimination of Section 3.2.5 to Gauss elimination wrt. an implicit or explicit background theory. In this section the notion of Gauss elimination is further extended: We transfer the definition of Gauss equations to Gauss formulas, i.e. complex formulas with the same properties as Gauss equations resulting in *deep* Gauss elimination. The key observation not only to consider atomic formulas but also complex ones as *prime constituents* of a formula will lead us to the concept of *partial deep* Gauss elimination, which provides a natural way to extend the idea of Gauss elimination to subformulas of formulas and provides a clean way to incorporate all variants of Gauss elimination in our quantifier elimination procedure.

4.3.1 Deep Gauss Elimination

Recall that Gauss elimination has so far been applied to formulas of the form $f = 0 \wedge \psi$. To begin with we summarize the properties of the equation $f = 0$ that are relevant for Gauss elimination:

- The equation is a Gauss equation, i.e., it has only a finite solution set $S^x_{\underline{a}}(f = 0)$ for any admissible $\underline{a} \in \mathbb{R}^m$.

- The Gauss equation occurs conjunctively on the toplevel of the input formula.

We have also made use of the following three properties, which are generally required for the quantifier elimination by virtual substitution.

1. Each candidate solution of the equation can be expressed by a suitable pseudo term.
2. All occurring pseudo terms can be handled by an appropriate virtual substitution.
3. A superset of the candidate solutions can be computed from the equations.

Let us return to our formula $f = 0 \wedge \varphi$, and suppose that $f = 0$ is a Gauss equation. The condition that $S^x_{\underline{a}}(f = 0)$ is finite implies that

$$S^x_{\underline{a}}(\alpha \wedge \varphi) \subseteq S^x_{\underline{a}}(\alpha)$$

is also finite for all admissible $\underline{a} \in \mathbb{R}^m$. This observation gives rise to the idea of the *deep Gauss* elimination: We call a quantifier-free formula $\psi(\underline{u}, x)$ a *Gauss formula* if it has a finite solution set $S^x_{\underline{a}}(\psi)$ for all $\underline{a} \in \mathbb{R}^m$. This notion extends in a natural way to the notion of a Gauss formula wrt. a theory.

We have to clarify how to recognize a formula ψ to be a Gauss formula and how to compute the corresponding candidate solution set. Again it is not our aim to recognize all Gauss formulas, but to find a suitable method to recognize as many formulas as possible in a reasonable time. The basis here is to recognize equations as Gauss equations using our simplifier based decision heuristics.

We give a description which formulas are recognized as Gauss formulas wrt. a theory and which candidate solution sets are computed:

- Each Gauss equation α is a Gauss formula. Its candidate set consists of all its formal solutions wrt. x together with the usual guards and boundary type “isolated point.”
- Let φ_1 be a Gauss formula with candidate set C_1, then the conjunction $\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n$ is a Gauss formula also with candidate set C_1.
- Let $\varphi_1, \ldots, \varphi_n$ be Gauss formulas with candidate sets $C_1, \ldots, C_n$. Then $\varphi_1 \vee \cdots \vee \varphi_n$ is a Gauss formula with candidate set $C_1 \cup \cdots \cup C_n$.

In the case of a conjunction we need only that one of the constituents is a Gauss formula. It is, however, possible that some other constituent φ_i is also a Gauss formula. In this case one can choose between φ_1 and φ_i and thus C_1 and C_i.

Consider the formula $\varphi \equiv \exists x \psi(\underline{u}, x)$, where ψ is a Gauss formula, and let C_ψ be the corresponding candidate solution set in the sense of the recursive definition above. Then it is easy to see that this C_ψ

1. contains only terms which are formal solutions equations belonging to formulas in ψ,

2. for any interpretation of the parameters C_ψ provides a superset of the interval boundaries in $S^x_{\underline{a}}(\psi)$.

The notion of a candidate solution set used here is thus compatible with our earlier definition in Section 3.3.

We conclude this section with an example that gives a first impression of the great improvement provided by the deep Gauss elimination. Recall that trivial Gauss elimination was restricted to formulas of the form $\exists x\bigl(f(\underline{u}, x) = 0 \land \psi(\underline{u}, x)\bigr)$.

Example 4.5. Consider the formula

$$\exists x \psi, \quad \psi \equiv \bigl(x = a(\underline{u}) \lor (x = b(\underline{u}) \land \psi_1(\underline{u}, x))\bigr) \land \psi_2(\underline{u}, x).$$

Then $x = a$ is a Gauss equation. The same holds for $x = b$ with the consequence that $x = b \land \psi_1$ is a Gauss formula. Together we have that $x = a \lor (x = b \land \psi_1)$ is a Gauss formula. Finally the entire ψ is a Gauss formula. Its candidate solution set provides the terms a and b as isolated points with guard "true." It follows immediately that $\{(\mathrm{true}, a), (\mathrm{true}, b)\}$ is an elimination set for ψ and x. The subformulas ψ_1 and ψ_2 are completely irrelevant for this elimination set computation.

The next example demonstrates the dramatic improvements of our quantifier elimination by virtual substitution obtained by introducing the deep Gauss elimination.

Example 4.6 (Hydraulic Network). We consider a hydraulic network as shown in Figure 4.1. This example was originally considered by Weispfenning [73]. The aim is to compute the pressure $p3$ and the flow $f12$ in terms of $v01$, $v02$, $v12$, $v13$, and $v23$. Our quantifier elimination including the deep Gauss elimination computes in 38 s a result formula containing 900 atomic formulas. Without using the deep Gauss elimination and using only the trivial Gauss elimination as discussed in Section 3.2.5 we need 17 min to compute an output formula containing 27 771 atomic formulas.

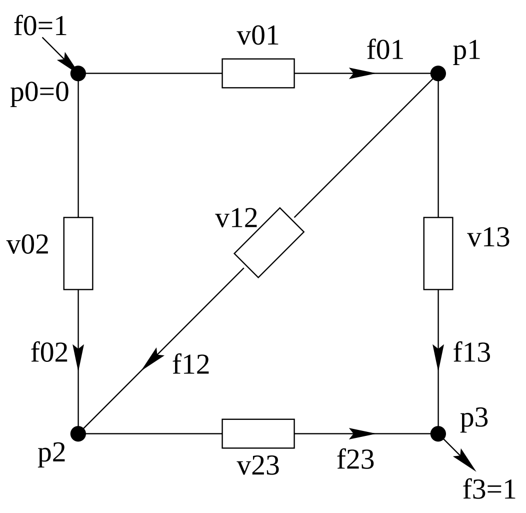


Figure 4.1: A hydraulic network

4.3.2 Partial Gauss Elimination

In the previous section we have generalized Gauss elimination to deep Gauss elimination resulting in a candidate solution set for Gauss formulas. In this section we further generalize deep Gauss elimination to *partial deep Gauss elimination.* This new generalization will lead us to a point of view where (deep) Gauss elimination is not longer a special case at all but can be incorporated into the process of regular elimination set computation.

In the following we will for simplicity speak about "initially computing the set of atomic formulas," and we will generalize this concept. The reader should keep in mind that in view of the discussion in Section 4.2 this also involves computation and application of the implicit theory concept in some computationally reasonable way.

When computing from the set of atomic formulas the set of all candidates, these candidates represent the zeroes of the equations associated with the atomic formulas. The idea is that these zeroes are candidates for the interval boundaries of the solution set. For a Gauss formula, which has by definition a finite solution set, the corresponding intervals are isolated points, and all these points are included into the Gauss elimination set. Gauss elimination thus naturally combines with general elimination set computation.

We modify the first two phases of our quantifier elimination: In the first phase we do not compute the set of all atomic formulas but a set of *prime constituents* of the subformulas. Given a quantifier-free positive formula φ the prime constituents of φ are

1. Gauss subformulas that are not proper subformulas of other Gauss subformulas,
2. the atomic subformulas that are not contained in any Gauss subformula.

Obviously, our φ can be rewritten as an $\wedge$-$\vee$-combination of its prime constituents.

Our modified candidate set computation will now proceed as follows: Instead of initially computing the set of all atomic subformulas of φ, it computes the set of all its prime constituents. Within this set atomic prime formulas can obviously be distinguished from Gauss prime formulas. For the atomic formulas we proceed as usual obtaining a partial elimination set E^{at}. Simultaneously, we compute the union E^{g} of all the candidate sets of the Gauss prime constituents. Their union $E = E^{\mathrm{at}} \cup E^{\mathrm{g}}$ yields our revised elimination set.

Recall, that Gauss candidate sets are in general much smaller than corresponding conventionally computed candidate sets. This gain in size is now lifted to our E.

Since a conventional Gauss formula consists of a single prime constituent, deep Gauss elimination as introduced in the previous section is obviously a special case of the partial deep Gauss elimination introduced here.

Again we can use our theory concept for generalizing partial Gauss elimination: The notion of a Gauss prime constituent can straightforwardly be generalized to that of a Gauss prime constituent wrt. the current theory.

In the discussion of the application of the implicit theory in Section 4.2 theory we have already mentioned that it is possible that the implicit theory contains atomic formulas involving the current quantified variable. Suppose now that such an atomic formula is a non-trivial equation η, and denote by ψ the subformula for which we have computed this particular implicit theory Θ. It is then easy to see that ψ is a Gauss formula wrt. Θ. Recall that an equation can enter the theory in two different ways:

1. There is a superformula ψ' of ψ that forms a conjunction with η.
2. There is a superformula ψ' of ψ that forms a disjunction with $\overline{\eta}$.

In the first case we need not do anything because the corresponding subformula $\psi' \wedge \eta$ forms a Gauss formula and is recognized as such. In the second case, in contrast, we can actually profit from our observation and immediately generate the formal solution of η as the only test point.

The observation that in the first case there is the deep partial Gauss as a structural counterpart to our observation within the theory gives rise to the idea that there is also such a counterpart for the second case. It is not hard to see that not all deep partial Gauss situations can be identified within the theory; Example 4.5 provides a counterexample for this assumption. It thus looks like we have got on the track of another powerful elimination technique with a certain duality to Gauss elimination.

Examples for partial deep Gauss elimination are easily obtained by deeply nesting into a boolean combination the matrix formula of some regular deep Gauss example.

Example 4.7. Consider the formula

$$\exists x\bigl((\psi \vee \psi') \wedge \psi''\bigr), \quad \psi \equiv \bigl(x = a(\underline{u}) \vee (x = b(\underline{u}) \wedge \psi_1(\underline{u}, x))\bigr) \wedge \psi_2(\underline{u}, x),$$

where ψ is chosen as in Example 4.5. According to our discussion, in this section it suffices to compute an elimination set from ψ' and ψ'', and then to unite with the elimination set $\{(\mathrm{true}, a), (\mathrm{true}, b)\}$ obtained for ψ in Example 4.5.

4.4 co-Gauss Elimination

In the previous section we have indicated that for a formula $\varphi \equiv \exists x \psi$ with $\psi \equiv x \neq 0 \vee \psi'$, we can recognize ψ' to be a Gauss formula wrt. its implicit theory $\{x = 0\}$. Doing so, we easily see that

$$\bigl\{(\mathrm{true}, 0, \text{“isolated point”}), (\mathrm{true}, 0, \text{“exception point”})\bigr\}$$

is a candidate solution set for our formula ψ and x.

The solution set $S^x_{\underline{a}}(\psi)$ is co-finite, i.e., its complement is finite. We are, in a certain sense, in a situation *complementary* to the Gauss elimination, which we shortly refer to as *co-Gauss.* More precisely we refer to ψ as a co-Gauss formula. It is easy to see that for such a formula with a co-finite solution set, the set $\{(\mathrm{true}, \infty)\}$ is an elimination set.

The co-Gauss should not be confused with the *dual* counterpart of Gauss elimination, which occurs for the elimination of $\forall x(x \neq 0 \vee \psi')$. With our approach this dual Gauss elimination is automatically translated to a regular Gauss elimination by our treatment of universal quantifiers.

The elimination of φ above is trivial: Move the existential quantifier into the disjunction; then we see that $\exists x(x \neq 0)$ and thus φ is “true,” and for the elimination of $\exists x(x \neq 0)$ it actually suffices to substitute ∞. Summarizing the co-Gauss appears to be less interesting than the regular Gauss because it appears in a situation, where the toplevel operator of the matrix is compatible with the quantifier. This changes when we extend the co-Gauss to deep partial co-Gauss as we have done for the regular Gauss.

For recognizing formulas to be co-Gauss formulas we proceed as for Gauss formulas, but we consider non-trivial disequations instead of non-trivial equations:

- A disequation $\alpha = \sum_{i=0}^{d} c_i x^i \neq 0$ is called *co-Gauss disequation* wrt. a theory Θ if
$$\bigwedge \Theta \longrightarrow (c_0 \neq 0 \vee \cdots \vee c_d \neq 0).$$
This condition is actually equivalent to the condition that α has a co-finite solution set $S^x_{\underline{a}}(\alpha)$ for all admissible $\underline{a} \in \mathbb{R}^m$. Each such co-Gauss disequation is a co-Gauss formula. Its candidate solution set consists of all formal solutions wrt. x of the corresponding equation together with the usual guards and boundary type "exception point."

- Let φ_1 be a co-Gauss formula with candidate set C_1, then the disjunction $\varphi_1 \vee \varphi_2 \vee \cdots \vee \varphi_n$ is a co-Gauss formula also with candidate set C_1.

- Let $\varphi_1, \ldots, \varphi_n$ be co-Gauss formulas with candidate sets $C_1, \ldots, C_n$. Then $\varphi_1 \wedge \cdots \wedge \varphi_n$ is a co-Gauss formula with candidate set $C_1 \cup \cdots \cup C_n$.

As with the definition of regular Gauss formulas it is not too hard to see that the notion of a candidate solution set implicitly defined here is compatible with our standard notion.

Co-Gauss formulas are particularly nice for elimination set computation. In fact, they are even nicer than regular Gauss formulas: They have a candidate solution set containing only exception points. For such candidate solution sets, $\{(\mathrm{true}, \infty)\}$ is obviously a suitable elimination set. This concludes the discussion of deep co-Gauss elimination.

It remains to discuss the *partial (deep) co-Gauss* elimination as an analog to the partial Gauss elimination. This analogy requires the notion of *co-Gauss prime constituents*, which are subformulas with a co-finite solution set. In complete analogy to regular partial Gauss elimination it turns out that co-Gauss primes can play the role of co-Gauss disequation in the recursion basis of the definition above.

Example 4.8 (Deep Partial co-Gauss). Consider the formula

$$\exists x \psi, \quad \psi \equiv \big(\big(x \neq a(\underline{u}) \wedge (x \neq b(\underline{u}) \vee \psi_1(\underline{u}, x))\big) \vee \psi_2(\underline{u}, x)\big) \wedge \psi_3,$$

where ψ_3 is *not* a co-Gauss formula. According to our definition the subformula $\bigl(x \neq a(\underline{u}) \wedge (x \neq b(\underline{u}) \vee \psi_1(\underline{u}, x))\bigr) \vee \psi_2(\underline{u}, x)$ is then the largest co-Gauss formula contained in ψ. Its candidate solution set is

$$\bigl\{(\mathrm{true}, a, \text{“exception point”}), (\mathrm{true}, b, \text{“exception point”})\bigr\}.$$

To obtain the candidate solution set for the entire ψ and x, we unite this with the candidate solution set of ψ_3. From this we can compute an elimination set, to which ψ_1 and ψ_2 do not contribute anything. Mind that the substitution of (true, ∞) for a co-Gauss elimination set does not work with *partial* deep co-Gauss elimination because there are candidate solutions contributed from outside, which are not exception points.

4.5 The Invisible Theory

Throughout this chapter we have seen that our theory concept originally introduced for the purpose of simplification in Chapter 2 can be used in numerous ways to improve the candidate solution sets and as a consequence the elimination sets. We have distinguished two types of theories:

1. An explicit theory provided as an extra argument to the corresponding algorithm.

2. An implicit theory constructed by the algorithm itself, when traversing recursively through the formula.

Observe that in both cases the theory information is to a certain extent syntactically represented. In this section we turn to another theory for which this is not the case. Accordingly we call it the *invisible theory.* To be more precise, the invisible theory is, as the implicit theory, constructed when recursing through the formula for candidate solution set computation. This time, however, the collected information is not taken from the formula itself but derived from the computation process.

We illustrate this by means of an example. Consider the formula

$$\exists x(\eta \wedge \varrho), \quad \eta \equiv u_1x + u_2 = 0, \quad \varrho \equiv (u_1 + 1)x + u_3 = 0 \wedge \psi_1(\underline{u}, x).$$

The matrix $\eta \wedge \varrho$ is a Gauss formula, which cannot be easily detected because both equations viewed isolated are possibly trivial. Certainly, for the purpose of candidate solution set computation, η will serve as an implicit theory for ϱ and thus for the second equation. This implicit theory is, however, only used for heuristics checks based on the simplifications in Chapter 2. For this case our particular simplifications will fail on recognizing that η and the second equation cannot be trivial simultaneously.

The invisible theory will enable us to apply Gauss elimination anyway. Let us fix in our minds the parameters $\underline{u}$ to $\underline{a} \in \mathbb{R}^3$ and focus on η. There are two possibilities:

1. One of u_1 and u_2 is non-zero for our interpretation. Then we can apply Gauss elimination, and ϱ is completely irrelevant for the candidate solution set computation.

2. Both u_1 and u_2 are zero. Then we cannot apply Gauss elimination, η vanishes, and ϱ becomes the only source for candidate solutions.

We see that ϱ is relevant if and only if $u_1 = 0$ and $u_2 = 0$. This constitutes the invisible theory $\mathrm{I} = \{u_1 = 0, u_2 = 0\}$ for ϱ. It is not hard to see that this invisible theory can be used for the candidate solution set computation for ϱ exactly as we use the implicit theory, which is $\Theta = \{\eta\}$, provided, of course, that we also add the formal solution of η to the candidate solution set. Summarizing we either have a Gauss situation or an extremely strong theory: We always win.

In our particular example, the invisible theory is even strong enough to finally discover the applicability of Gauss elimination: Our additive smart simplification introduced of Section 2.4.2 easily derives $u_1 + 1 \neq 0$ from $(u_1 = 0) \in \mathrm{I}$.

We finally wish to emphasize that the invisible theory is not a tool for detecting Gauss formulas but has exactly the same power and relevance as the implicit theory.

4.6 Conclusions

In this chapter we have generalized our theory concept for simplification introduced in Chapter 2 to quantifier elimination by virtual substitution, more precisely to the candidate solution set computation. Starting in Section 4.1 with the concept of an external theory we have identified the places where one can profit from external information there: saving guards, simplifying discriminant conditions, improving boundary type information, and detecting Gauss formulas.

These results have encouraged us to adapt in Section 4.2 also the concept of an implicit theory. The construction of this implicit theory for candidate solution set computation slightly differs from that for simplification. In analogy to the simplification case, it turns out that concerning the application there is no difference between implicit theory and explicit theory. With simplification we have observed that the construction of implicit theories is *the* tool for performing simplifications in spite of complicated boolean structures and for even profiting from these structures. Accordingly we have observed the same effect for candidate solution set computation now. We thus have finally dropped the restriction to compute the candidate solutions from the set of atomic formulas.

In the spirit of this observation we have introduced in Section 4.3 further structural concepts generalizing the Gauss elimination of Section 3.2.5 in two ways: First, the idea of a non-trivial equation generalizes to that of a subformula with finite solution set. Second, it turns out that the corresponding Gauss formula need not be on the toplevel.

Reanalyzing our generalized Gauss elimination of Section 4.3 in terms of the implicit theory of Section 4.2 has led us to the insight that there is the concept of a co-Gauss which is related to Gauss elimination exactly as co-finite sets are related to finite sets.

In Section 4.5 we have finally introduced another type of theory, which is implicit in nature, but does not collect information syntactically present in the formula. Instead the collected information is derived from the elimination process. Concerning the applicability this new theory plays the same role as the implicit theory and can be used simultaneously.

Chapter 5

Repeated Condensing

On our way to applicable quantifier elimination we have studied so far two major strategies: The first one consists in sophisticated simplification of formulas occurring as input formulas, as intermediate results, and as final results. The second strategy combines the improvement of the elimination set computation with improved substitution methods of test points into atomic formulas. In this chapter we are going to analyze the substitution in more detail, and we present the concept of condensing as a replacement for substitution. Roughly speaking condensing means substituting a term only into some parts of a formula removing all other parts. This obviously results in simpler formulas.

To begin with, we are going to discuss in Section 5.1 condensing in the case of partial deep Gauss elimination. In Section 5.2, we develop similar ideas for the general case of quantifier elimination via an arbitrary elimination set.

5.1 Condensing of Gauss Formulas

Recall from the discussion of Gauss elimination in Section 4.3.2 that in the case of a partial deep Gauss elimination on some formula ψ the elimination set is divided into a set E^{at} of candidates from atomic prime constituents plus a set $E^{\mathrm{g}} = E^{\mathrm{g}}_1 \cup \dots \cup E^{\mathrm{g}}_k$ of Gauss candidates obtained from Gauss prime constituents $\psi_1, \dots, \psi_k$ of ψ. Our idea is that we can neglect the Gauss prime constituents for the substitution of the candidates in E^{at}. Moreover, for the substitution of candidates E^{g}_i we can neglect all Gauss prime constituents ψ_j for $j \neq i$.

To make this precise, let ψ be a quantifier-free positive formula, and let ψ_1 be a Gauss prime constituent of ψ. Then we denote by $\Gamma_{\psi_1}\psi$ the formula that is obtained from ψ by replacing ψ_1 with "false." Here ψ_1 uniquely identifies one particular subformula although ψ can in general contain several copies of ψ_1. This definition naturally extends to $\Gamma_{\{\psi_1,\dots,\psi_k\}}\psi$ for several Gauss prime constituents ψ_1, … , ψ_k of ψ. We obviously have

$$\Gamma_{\{\psi_1,\dots,\psi_k\}}\psi \longrightarrow \psi.$$

This is the key observation for condensing and makes precise what we mean by "neglecting" ψ_1.

Assume that we want to eliminate from a positive quantifier-free formula $\psi(\underline{u}, x)$ the existentially quantified variable x. We have obtained corresponding elimination set parts E^{at} and $E^{\mathrm{g}} = E^{\mathrm{g}}_1 \cup \dots \cup E^{\mathrm{g}}_k \neq \emptyset$. That is, there is at least some Gauss prime constituent ψ_1 with finite solution set $S = S^x_{\underline{a}}(\psi_1)$ for any choice $\underline{a} \in \mathbb{R}^m$ for the parameters and candidate solution set $C_{\psi_1} = E^{\mathrm{g}}_1$. Let $\bigl(\gamma(\underline{u}), t(\underline{u})\bigr)$ be an arbitrary test point, where t is possibly a pseudo term. Note that we can evaluate $t(\underline{a})$ in some suitable extension field $\mathbb{R}^*$. We have either $t(\underline{a}) \in S \subseteq \mathbb{R}$ or $\mathbb{R} \models \neg\psi_1[x/\!/t](\underline{a})$. In the former case there is obviously (γ', t') in E^{g}_1 with $t(\underline{a}) = t'(\underline{a})$. We have thus proved the following equivalence:

$$\bigvee_{(\gamma,t)\in E^{\mathrm{at}}} \gamma \wedge \psi[x/\!/t] \vee \bigvee_{(\gamma,t)\in E^{\mathrm{g}}} \gamma \wedge \psi[x/\!/t] \longleftrightarrow$$

$$\bigvee_{(\gamma,t)\in E^{\mathrm{at}}} \gamma \wedge \psi[x/\!/t] \vee \bigvee_{i=1}^{k} \bigvee_{(\gamma,t)\in E^{\mathrm{g}}_i} \gamma \wedge \psi[x/\!/t] \longleftrightarrow$$

$$\bigvee_{(\gamma,t)\in E^{\mathrm{at}}} \gamma \wedge \Gamma_{\psi_1}\psi[x/\!/t] \vee \bigvee_{(\gamma,t)\in E^{\mathrm{g}}_1} \gamma \wedge \psi[x/\!/t] \vee \bigvee_{i=2}^{k} \bigvee_{(\gamma,t)\in E^{\mathrm{g}}_i} \gamma \wedge \Gamma_{\psi_1}\psi[x/\!/t],$$

and more generally

$$\bigvee_{(\gamma,t)\in E^{\mathrm{at}}} \gamma \wedge \psi[x/\!/t] \vee \bigvee_{i=1}^{k} \bigvee_{(\gamma,t)\in E^{\mathrm{g}}_i} \gamma \wedge \psi[x/\!/t] \longleftrightarrow$$

$$\bigvee_{(\gamma,t)\in E^{\mathrm{at}}} \gamma \wedge \Gamma_{\{\psi_1,\dots,\psi_k\}}\psi[x/\!/t] \vee \bigvee_{i=1}^{k} \bigvee_{(\gamma,t)\in E^{\mathrm{g}}_i} \gamma \wedge \Gamma_{\{\psi_j \mid 1\leq j\leq k, j\neq i\}}\psi[x/\!/t].$$

This describes Variant 1 of condensing substitution.

Observe that $E^{\mathrm{g}} = E^{\mathrm{g}}_1 \cup \dots \cup E^{\mathrm{g}}_k$ is in general not a disjoint union. Consequently Variant 1 of condensing above will in contrast to the naive substitution approach possibly substitute one test point several times. Moreover these substitutions lead to different substitution results since there are different condensing operators involved. The following example shows that due to the different condensing operators it would in fact not be correct to simply drop one of the substitutions:

Example 5.1. Consider the input formula

$$\varphi \equiv \exists x\Big((x = 0 \land u_1 > 0) \lor \big(u_2 > 0 \land \big(u_3 > 0 \lor (x = 0 \land u_4 > 0)\big)\big)\Big).$$

It is easy to see that this formula is equivalent to

$$\varphi^* \equiv u_1 > 0 \lor \big(u_2 > 0 \land \big(u_3 > 0 \lor u_4 > 0\big)\big),$$

which is actually generated by our quantifier elimination procedure. The two Gauss prime constituents $x = 0 \land u_1 > 0$ and $x = 0 \land u_4 > 0$ generate both the test point $(\mathrm{true}, 0)$. The corresponding Gauss condensing results are

$$u_1 > 0 \quad \text{and} \quad u_2 > 0 \land \big(u_3 > 0 \lor u_4 > 0\big),$$

respectively. None of them can be dropped from φ^* without destroying the equivalence.

We are now going to devise another variant of condensing, which correctly avoids multiple substitutions. While Variant 1 kept $E_1^{\mathrm{g}}, \dots, E_k^{\mathrm{g}}$ separated we now actually compute the union E^{g} as with the naive method but labeling each test point in E^{g} with all the ψ_j generating it, i.e.,

$$\Lambda\big((\gamma, t)\big) = \{\, \psi_i \mid 1 \leq i \leq k, (\gamma, t) \in E_i^{\mathrm{g}} \,\}.$$

Let accordingly $\overline{\Lambda}\big((\gamma, t)\big) = \{\psi_1, \dots, \psi_k\} \setminus \Lambda\big((\gamma, t)\big)$. It is not hard to see from the discussion above that the following Variant 2 of condensing is also correct:

$$\begin{aligned}&\bigvee_{(\gamma,t)\in E^{\mathrm{at}}} \gamma \land \psi[x/\!/t] \lor \bigvee_{(\gamma,t)\in E^{\mathrm{g}}} \gamma \land \psi[x/\!/t] \longleftrightarrow \\ &\qquad \bigvee_{(\gamma,t)\in E^{\mathrm{at}}} \gamma \land \Gamma_{\{\psi_1,\dots,\psi_k\}}\psi[x/\!/t] \lor \bigvee_{(\gamma,t)\in E^{\mathrm{g}}} \gamma \land \Gamma_{\overline{\Lambda}((\gamma,t))}\psi[x/\!/t].\end{aligned}$$

5.2 Positional Condensing

The results of the previous section on condensing of Gauss formulas can be summarized as follows:

> For the substitution of a particular test point, certain parts of the formula can be neglected because they are irrelevant on the premise that the test point itself is relevant.

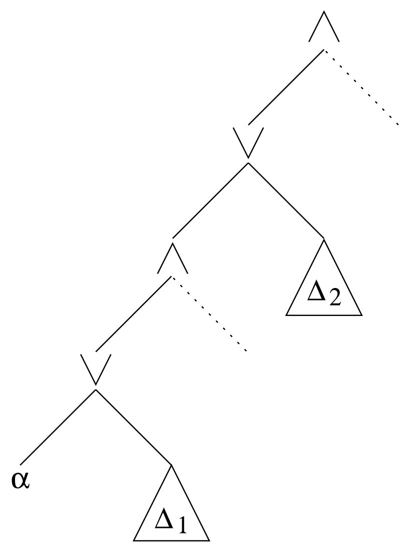


Figure 5.1: Positional Condensing

More precisely a Gauss prime constituent ψ_1 of ψ can be condensed, i.e. assumed to be "false" for the substitution of a test point stemming from an atomic prime α: Whenever, for fixed parameters, it is crucial for ψ to hold that ψ_1 is "true", then our considered substitution branch stemming from α is superfluous anyway since ψ_1 will provide all satisfying test points itself. We observe that this type of condensing is based on a particular property of the condensed formula ψ_1, namely its finite satisfaction set.

We now turn to a second type of condensing, which is not based on inherent properties of the condensed formula parts but on their *position* within in the formula. Consider e.g. the situation in Figure 5.1. We fix all parameters to real values and restrict our attention to the substitution branch, where some test point (γ, t) generated from α is substituted. Then we see immediately that Δ_1 can be condensed because the substitution of (γ, t) will obviously turn α into "true." More surprisingly, also Δ_2—and more generally each subtree occurring disjunctively on the path from α to the root of ψ—can also be condensed. This requires, however, a more sophisticated argumentation than that for Δ_1: Recall that ψ is positive, and assume that the validity of Δ_2 is crucial for ψ to hold. In such a situation we can immediately conclude two facts: First, it is crucial for ψ to hold that the disjunction immediately containing Δ_2 becomes "true," and second, the other branch of this disjunction, which contains α, becomes "false." From our second conclusion it follows that (γ, t) and thus our considered substitution branch is superfluous.

For the definition of our new condensing operator Γ^{p} we identify formulas with their operator trees and introduce a formalism for identifying positions within these trees. A first-order formula φ corresponds to a tree (V, E) with finite $V \subseteq N^* = \bigcup_{n \in \mathbb{N}} \mathbb{N}^n$ and $E \subseteq V \times V$. Each node $v \in V$ is labeled with either a boolean operator, a quantifier, or an atomic formula. We recursively define the tree representation of φ as follows:

- An atomic formula φ is represented by the tree $\bigl(\{(1)\}, \emptyset\bigr)$ where the label of (1) is φ.

- Let φ be a conjunction $\varphi_1 \land \dots \land \varphi_n$, a disjunction $\varphi_1 \lor \dots \lor \varphi_n$, or a quantified formula $\exists x(\varphi_1)$, $\forall x(\varphi_1)$. For $1 \leq i \leq n$ let (V_i, E_i) be the tree representations of φ_i. We adapt each (V_i, E_i) to (V_i', E_i') by setting

  $$\begin{aligned} V_i' &= \{\, (i, v_1, \dots, v_k) \mid (v_1, \dots, v_k) \in V_i \,\}, \\ E_i' &= \{\, \bigl((i, v_1, \dots, v_k), (i, w_1, \dots, w_k)\bigr) \mid \\ &\qquad\qquad \bigl((v_1, \dots, v_k), (w_1, \dots, w_k)\bigr) \in E_i \,\}. \end{aligned}$$

  The label of $(i, v_1, \dots, v_k) \in V_i'$ is inherited from $(v_1 \dots, v_k) \in V_i$. We obviously have that $V_i' \cap V_j' = \emptyset$, $(1) \notin V_i'$, $(i) \in V_i$, and $E_i' \subseteq V_i' \times V_i'$. Thus (V_i', E_i') is a labeled tree. The tree representation (V, E) of φ is then given by

  $$V = \bigcup_{i=1}^{n} V_i' \cup \{(1)\}, \quad E = \bigcup_{i=1}^{n} E_i' \cup \{\, ((1), (i)) \mid 1 \leq i \leq n \,\}.$$

  We define the label of (1) to be the toplevel operator of φ. As an example we show the tree representation of the formula $a = 0 \land (b = 0 \lor a = 0)$ in Figure 5.2.

Let $\underline{v} = (v_1, \dots, v_k)$ be a node in the tree representation of φ. Then $(v_1, \dots v_k)$ is not only the object representing the node, but it is also the path from the top node to $\underline{v}$. In this sense $(v_1, \dots, v_k)$ is called the *position* of v.

Each node in the tree representation of a formula identifies a subformula. Conversely we can identify a subformula with the node. This allows us to identify positions with subformulas and vice versa. For instance, we can identify a specific atomic formula of a formula with its position avoiding a conflict with other atomic formulas that are equal but occur at another position.

In analogy to the Gauss condensing operator $\Gamma_{\{\psi_1 \dots, \psi_l\}}$ our new condensing operator $\Gamma^{\mathrm{p}}_{\alpha}$ is parameterized via a subscript α. This subscript α, however,

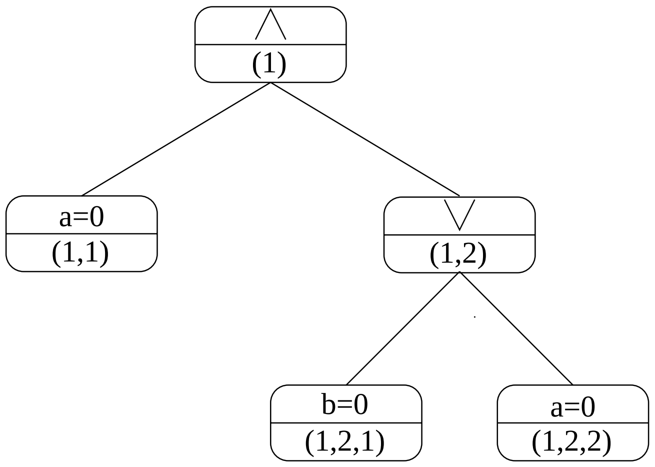


Figure 5.2: Tree representation of $a = 0 \wedge (b = 0 \vee a = 0)$

does not denote the condensed part of the formula but the atomic formula yielding the current test point and thus justifying the positional condensing. Completely analogous to Gauss condensing, α does not simply denote some atomic formula but a particular occurrence of this formula within the target formula ψ. In our newly introduced formalism α thus corresponds to a tree position, say

$$\Gamma^{\mathrm{p}}_{\alpha} = \Gamma^{\mathrm{p}}_{(v_1,\dots,v_k)}.$$

Given a positive quantifier-free formula ψ we construct $\Gamma^{\mathrm{p}}_{\alpha}\psi = \Gamma^{\mathrm{p}}_{(v_1,\dots,v_k)}\psi$ as a subtree of $\psi = (V, E)$ by specifying a subset $V' \subseteq V$ of the nodes. Simultaneously, there will be certain labels of nodes in V' modified to "false." We constructively describe V' by following the path

$$(v_1,\dots,v_k), \quad (v_1,\dots,v_{k-1}), \quad \dots, \quad (v_1) = (1)$$

from α to the root of ψ. For $1 \leq i \leq k$ we obtain nodes for V' as follows:

- If the label of $(v_1,\dots,v_i)$ is "$\vee$," then $(v_1,\dots,v_i)$ is copied to V'. Note that the node $(v_1,\dots,v_{i+1})$ is captured by another instance of the rule we are just defining. There are now possibly further descendents $(v_1,\dots,v_i,w_1)$, ..., $(v_1,\dots,v_i,w_l)$. These descendents are also copied to V' with their label changed to "false." Note that possible descendents of $(v_1,\dots,v_i,w_1)$, ..., $(v_1,\dots,v_i,w_l)$ are dropped.

- If the label of $(v_1,\dots,v_i)$ is different from "$\vee$," then $(v_1,\dots,v_i)$ is copied to V'. Again the node $(v_1,\dots,v_{i+1})$ is captured by another instance of our rule. Possible further descendents

$$(v_1, \dots, v_i, w_1), \quad \dots, \quad (v_1, \dots, v_i, w_l)$$

are recursively copied to V'.

Following the discussion so far we can use Γ^{p} for constituting the substitution rule of positional condensing. We compute an elimination set E in which each point (γ, t) is labeled with all prime constituents generating it, say

$$\Lambda\bigl((\gamma, t)\bigr) = \{\pi_1, \dots, \pi_k\},$$

where the π_i include both atomic and Gauss prime constituents. For the sake of formal cleanness, we specify that $(\mathrm{true}, \pm\infty)$ is labeled with $\{\infty\}$, and $\Gamma^{\mathrm{p}}_{\infty}\psi = \psi$. Positional condensing is then given by the equivalence

$$\bigvee_{(\gamma,t)\in E} \gamma \wedge \psi[x/\!/t] \longleftrightarrow \bigvee_{(\gamma,t)\in E} \bigvee_{\pi\in\Lambda((\gamma,t))} \gamma \wedge \Gamma^{\mathrm{p}}_{\pi}\psi[x/\!/t].$$

Observe that we possibly have multiple substitutions for test points. In this concern, our notion of positional condensing corresponds to Variant 1 of Gauss condensing in the previous section. Accordingly we call it Variant 1 of positional condensing, and turn to the question whether there is some "Variant 2" that avoids such multiple substitutions.

From Variant 2 of the Gauss case we have obtained the intuition that for the substitution of multiply labeled test point, we have to be somehow careful not to condense "too much." The domain for a combined position condensing $\Gamma^{\mathrm{p}}_{\{\pi_1,\dots,\pi_k\}}$ is determined by the intersection of the domains of all the π_i. In terms of tree representations of formulas this corresponds to the part of the tree above the smallest common ancestor of all the π_i. In our framework the position of this smallest common ancestor is described by the largest common prefix of the positions of $\pi_1, \dots, \pi_k$. Using this extended notion of Γ^{p}, Variant 2 of positional condensing is given by

$$\bigvee_{(\gamma,t)\in E} \gamma \wedge \psi[x/\!/t] \longleftrightarrow \bigvee_{(\gamma,t)\in E} \gamma \wedge \Gamma^{\mathrm{p}}_{\Lambda((\gamma,t))}\psi[x/\!/t].$$

It is not hard to see that positional condensing can be straightforwardly combined with the Gauss condensing discussed in the previous section. When applying both condensing operators Γ and Γ^{p} to ψ it is formally clean to apply Γ first, since Γ^{p} might kill the Gauss prime constituent Γ is looking for. Vice versa there is no such problem.

In the remainder of this section we consider condensing from a different point of view: Instead of the structure of the involved formulas we focus on certain properties of the disjunctive normal forms corresponding to the formulas before and after condensing, respectively.

Algorithm 5.2. Let φ be a quantifier-free positive formula and let α be a prime constituent of φ. Then the following algorithm computes quantifier-free positive formulas $\hat{\varphi}$, ψ, and $\hat{\psi}$ such that

(1) $\hat{\varphi}$ is a disjunctive normal form of φ.

(2) $\hat{\psi}$ is a disjunctive normal form of ψ.

(3) $\hat{\psi} \subseteq \hat{\varphi}$.

(4) $\mathrm{at}(\hat{\psi}) = \mathrm{at}(\psi) \subseteq \mathrm{at}(\varphi) = \mathrm{at}(\hat{\varphi})$.

(5) If $\alpha \notin \varphi$, then $\psi = \varphi$ and $\hat{\varphi} = \hat{\psi}$.

(6) If α occurs in φ, then for every conjunction $\Gamma \in \hat{\varphi}$ with $\alpha \in \Gamma$ we have that $\Gamma \in \hat{\psi}$.

For computing $\hat{\varphi}$, ψ, and $\hat{\psi}$ from φ and α we assume wlog. that all occurrences of "$\wedge$" and "$\vee$" in φ are binary.

If φ is atomic, then we set $\hat{\varphi} = \psi = \hat{\psi} = \varphi$. This obviously fulfills all required conditions.

If φ is not atomic, then it is of the form $\varphi_1 \wedge \varphi_2$ or of the form $\varphi_1 \vee \varphi_2$. In both cases we recurse on φ_1 and α as well as on φ_2 and α obtaining

$$\hat{\varphi_1} = \varphi_{11} \vee \cdots \vee \varphi_{1m}, \quad \psi_1, \quad \hat{\psi_1} = \psi_{21} \vee \cdots \vee \psi_{2p}$$

and

$$\hat{\varphi_2} = \varphi_{21} \vee \cdots \vee \varphi_{2n}, \quad \psi_2, \quad \hat{\psi_2} = \psi_{21} \vee \cdots \vee \psi_{2q},$$

respectively. We make a combined case distinction on the actual operator and on the occurrence of α in φ.

1. Let φ be a conjunction $\varphi_1 \wedge \varphi_2$ and let $\alpha \in \varphi$, wlog. $\alpha \in \varphi_1$. This implies that $\alpha \notin \varphi_2$. We set

$$\begin{aligned}\hat{\varphi} \quad = \quad & (\varphi_{11} \wedge \varphi_{21}) \vee \cdots \vee (\varphi_{11} \wedge \varphi_{2n}) \vee \\ & (\varphi_{12} \wedge \varphi_{21}) \vee \cdots \vee (\varphi_{12} \wedge \varphi_{2n}) \vee \cdots \vee \\ & (\varphi_{1m} \wedge \varphi_{21}) \vee \cdots \vee (\varphi_{1m} \wedge \varphi_{2n}),\end{aligned}$$

and $\psi = \varphi_1 \wedge \varphi_2$. Using the identity $\hat{\psi_2} = \hat{\varphi_2}$ we define

$$\begin{aligned}\hat{\psi} \quad = \quad & (\psi_{11} \wedge \varphi_{21}) \vee \cdots \vee (\psi_{11} \wedge \varphi_{2n}) \vee \\ & (\psi_{12} \wedge \varphi_{21}) \vee \cdots \vee (\psi_{12} \wedge \varphi_{2n}) \vee \cdots \vee \\ & (\psi_{1p} \wedge \varphi_{21}) \vee \cdots \vee (\psi_{1p} \wedge \varphi_{2n}).\end{aligned}$$

Since $\hat{\varphi}$ is computed by applying the distributive law for $\wedge$ over $\vee$ to the disjunctive normal forms φ_1 and φ_2 we have that $\hat{\varphi}$ is a disjunctive normal form of φ. Analogously we have that $\hat{\psi}$ is a disjunctive normal form of ψ. This proves the properties (1) and (2).

We prove the subset relation (3): Let $(\psi_{1i} \wedge \varphi_{2j})$ be a branch of $\hat{\psi}$. From the subset relation between ψ_1 and φ_1 we know that there is an i' such that $\psi_{1i} = \varphi_{1i'}$. This implies

$$(\psi_{1i} \wedge \varphi_{2j}) = (\varphi_{1i'} \wedge \varphi_{2j}) \in \hat{\varphi}.$$

Property (4) is trivially fulfilled. Since $\alpha \in \varphi$ we finally have to prove claim (6). Let $\alpha \in (\varphi_{1i} \wedge \varphi_{2j})$. Recall that we have assumed that $\alpha \in \varphi_{1i}$. According to our specifications there is an i' such that $\psi_{1i'} = \varphi_{1i}$. Thus we have

$$(\varphi_{1i} \wedge \varphi_{2j}) = (\psi_{1i'} \wedge \varphi_{2j}) \in \hat{\psi}.$$

2. Let φ be a conjunction $\varphi_1 \wedge \varphi_2$ and let $\alpha \notin \varphi$. Then we have that $\hat{\varphi}_1 = \hat{\psi}_1$ and $\hat{\varphi}_2 = \hat{\psi}_2$. To meet condition (5) we set $\psi = \varphi$, and we set

$$\begin{aligned}\hat{\psi} = \hat{\varphi} \quad = \quad & (\varphi_{11} \wedge \varphi_{21}) \vee \cdots \vee (\varphi_{11} \wedge \varphi_{2n}) \vee \\ & (\varphi_{12} \wedge \varphi_{21}) \vee \cdots \vee (\varphi_{12} \wedge \varphi_{2n}) \vee \cdots \vee \\ & (\varphi_{1m} \wedge \varphi_{21}) \vee \cdots \vee (\varphi_{1m} \wedge \varphi_{2n}).\end{aligned}$$

   Again this is obviously a disjunctive normal form of both φ and ψ. Condition (3) is trivially fulfilled, and one can easily verify condition (4).

3. Let φ be a disjunction $\varphi_1 \vee \varphi_2$ and let $\alpha \in \varphi$. We assume wlog. that $\alpha \in \varphi_1$ and thus $\alpha \notin \varphi_2$. In this case we can reduce the size of ψ compared to that of φ: We define $\psi = \psi_1$. The corresponding DNF $\hat{\psi}$ is then simply $\hat{\psi}_1$. This implies in particular (2). The DNF $\hat{\varphi}$ is computed by simply merging the disjunctive normal forms $\hat{\varphi}_1$ and $\hat{\varphi}_2$:

$$\hat{\varphi} = \varphi_{11} \vee \cdots \vee \varphi_{1m} \vee \varphi_{21} \vee \cdots \vee \varphi_{2n}.$$

   Since $\hat{\psi}_1 \subseteq \hat{\varphi}_1$ we have again $\hat{\psi} \subseteq \hat{\varphi}$ proving (3). One can easily check condition (4).

   It remains to prove (6): Assume we have a $\Gamma \in \hat{\varphi}$ with $\alpha \in \Gamma$. Then actually $\Gamma \in \hat{\varphi}_1$. This implies that $\Gamma \in \hat{\psi}_1$ and thus $\Gamma \in \hat{\psi}$.

4. Finally, we have to describe how to treat the case that φ is a disjunction $\varphi_1 \vee \varphi_2$ and $\alpha \notin \varphi$: We set $\psi = \varphi$ and
$$\hat{\varphi} = \hat{\psi} = \varphi_{11} \vee \cdots \vee \varphi_{1m} \vee \varphi_{21} \vee \cdots \vee \varphi_{2n}.$$
This is a DNF of ψ because we know that $\hat{\varphi_1} = \hat{\psi_1}$ and $\hat{\varphi_2} = \hat{\psi_2}$. It is clear that these definitions fulfill the requirements (3), (4), and (5).

Note that a restriction of the above algorithm to the computation of ψ is already sufficient for condensing. The additional formulas $\hat{\varphi}$ and $\hat{\psi}$ are used for the following more formal proof for correctness of condensing.

Given a quantifier-free formula φ and a variable x we proceed as follows. We compute by the above algorithm a disjunctive normal form $\hat{\varphi} = \varphi_1 \vee \cdots \vee \varphi_n$ of φ. Then we compute an elimination set E for φ using any of the method discussed throughout this thesis. Then we have that E can be considered as a union $\bigcup_{i=1}^n E_n$, where the E_i are elimination sets for φ_i. This property, which is not correct for an arbitrary elimination set, follows immediately from the fact that we compute elimination sets form candidate solution sets stemming from prime constituents. In this situation we have the following equivalence:

$$\begin{aligned} \exists x(\varphi) &\longleftrightarrow \exists x(\bigvee_{i=1}^n \varphi_i) \\ &\longleftrightarrow \bigvee_{i=1}^n \exists x(\varphi_i) \\ &\longleftrightarrow \bigvee_{i=1}^n \bigvee_{(\gamma,t)\in E_i} \gamma \wedge \varphi_i[x/\!/t]. \end{aligned}$$

Recall that each E_i is non-empty, and thus we have that for each $1 \leq i \leq n$ there is some $t \in \bigcup_{j=1}^n E_j$ with $i \in I_t = \{\, i \mid t \in E_i \,\}$. This allows us to group the $\varphi_i[x/\!/t]$ in a different way, and we arrive at the following equivalence:

$$\begin{aligned} \bigvee_{i=1}^n \bigvee_{(\gamma,t)\in E_i} \gamma \wedge \varphi_i[x/\!/t] &\longleftrightarrow \bigvee_{(\gamma,t)\in E} \bigvee_{i\in I_t} \gamma \wedge \varphi_i[x/\!/t] \\ &\longleftrightarrow \bigvee_{(\gamma,t)\in E} \gamma \wedge \bigvee_{i\in I_t} \varphi_i[x/\!/t] \\ &\longleftrightarrow \bigvee_{(\gamma,t)\in E} \gamma \wedge \psi(t)[x/\!/t]. \end{aligned}$$

It is easy to see that in the case of an existential quantifier in front of a disjunction, say $\exists x(\psi_1 \vee \psi_2)$, condensing will take care that there are neither test terms coming from ψ_1 substituted into ψ_2 nor vice versa. Obtaining this effect was the major motivation for introducing blockwise quantifier elimination in Section 3.2.6. At this point we wish to emphasize that blockwise elimination is still relevant also with condensing. The reason for this is the existence of selection strategies like deciding for either upper or lower bounds as described in Section 3.5. If in our example ψ_1 would contain upper bounds and ψ_2 would contain only lower bounds, then elimination set computation cannot recognize that it is sufficient to substitute $-\infty$ and ∞, respectively, since at this point ψ_1 and ψ_2 are not isolated from one another. The notion of "repeated condensing" refers, however, to the idea of eliminating several (existentially) quantified variables by repeating the application of an elimination step using the condensing operators.

To conclude this section we give an example. This example is not only an example for the idea of condensing but shows also that all our strategies for applicable quantifier elimination by virtual substitution can be combined perfectly.

Example 5.3. Consider the following quantifier elimination problem:

$$\exists x\Big(\Big(((x = a_1 \lor x = a_2) \land b_1 x \geq b_2) \lor c_1 x \geq c_2\Big) \land (d_1 x \geq d_2 \lor x \geq e_1 \lor d_1 < 0)\Big).$$

Using the strategies described in this chapter, in the previous chapter on structural elimination sets, and in Section 3.6, if applicable, we obtain the following quantifier-free result containing 15 atomic formulas.

$$\begin{aligned}
&b_1 a_1 \geq b_2 \land (d_1 a_1 \geq d_2 \lor a_1 \geq e_1 \lor d_1 < 0) \lor \\
&\quad b_1 a_2 \geq b_2 \land (d_1 a_2 \geq d_2 \lor a_2 \geq e_1 \lor d_1 < 0) \lor \\
&\quad c_1 < 0 \land (c_2 d_1 \leq c_1 d_2 \lor c_2 \leq c_1 e_1 \lor d_1 < 0 \\
&\quad c_1 > 0 \lor c_1 = 0 \land c_2 \leq 0.
\end{aligned}$$

The current version of REDLOG computes the following result formula containing 29 atomic formulas:

$$\begin{aligned}
&(b_1 a_1 \geq b_2 \lor c_1 a_1 \geq c_2) \land (d_1 a_1 \geq d_2 \lor a_1 \geq e_1 \lor d_1 < 0) \lor \\
&\quad (b_1 a_2 \geq b_2 \lor c_1 a_2 \geq c_2) \land (d_1 a_2 \geq d_2 \lor a_2 \geq e_1 \lor d_1 < 0) \lor \\
&\quad b_1 \neq 0 \land \big((b_2 = b_1 a_1 \lor b_2 = b_1 a_2 \lor b_1 b_2 c_1 \geq b_1^2 c_2) \land \\
&\qquad\qquad (b_1 b_2 d_1 \geq b_1^2 d_2 \lor b_1 b_2 \geq b_1^2 e_1 \lor d_1 < 0)\big) \lor \\
&\quad c_1 \neq 0 \land (c_1 c_2 d_1 \geq c_1^2 d_2 \lor c_1 c_2 \geq c_1^2 e_1 \lor d_1 < 0) \\
&\quad d_1 \neq 0 \land (((d_2 = d_1 a_1 \lor d_2 = d_1 a_2) \land b_1 d_1 d_2 \geq b_2 d_1^2) \lor c_1 d_1 d_2 \geq c_2 d_1^2) \lor \\
&\quad c_1 > 0 \lor c_1 = 0 \land c_2 \leq 0.
\end{aligned}$$

The lower degrees in our new result are mainly obtained by using the stronger guards introduced in Section 3.6. A reduction of 9 atomic formulas is obtained by using the structural elimination sets of Chapter 4. Using both Gauss condensing and positional condensing leads to a reduction of 5 atomic formulas. All strategies together are the reason for the simpler boolean structure of the new result formula.

We want to emphasize that the previous example demonstrates that applying our strategies leads to a simpler result formula obeying the simplification goals of simple terms, of few atomic formulas, and of comprehensible boolean structure as they have been stated in Section 2.2 though they are not really simplification algorithms.

5.3 Conclusions

In this chapter we have introduced two independent though related concepts: Gauss condensing and positional condensing. Both these concepts can be combined without problems. In addition they perfectly combine with all other ideas presented throughout this thesis. With condensing we have discovered a new place, where optimizations can take place: The disjunction of virtual substitutions as opposed to the virtual substitutions themselves.

We may expect a dramatic decrease in the size of the quantifier-free results. On the other hand we have observed that there is a trade-off between removing redundant parts of the result and multiple substitutions. Our variants of condensing were designed to cope with this trade-off.

On our way to a more liberal view of quantifier elimination by virtual substitution we have performed yet another step: Instead of operating on atomic formulas as syntactical objects we now think in terms of tree positions.

At the end of the chapter we have sketched another approach to condensing. This approach was via consideration of equivalent DNF's. We wish to emphasize once more that this point of view very often provides a valuable alternative intuition within the framework of quantifier elimination by virtual substitution.

Chapter 6

Local Quantifier Elimination

In the previous chapters we have introduced quantifier elimination by virtual substitution and we have presented algorithmic strategies for improving this quantifier elimination procedure. In this chapter we are going to present strategies for improving the quantifier elimination procedure based on a modified specification of quantifier elimination, namely local quantifier elimination. Local quantifier elimination make use of the observation that in some application areas it is not necessary to compute a quantifier-free formula which is equivalent for all parameter values: Given a first-order formula $\varphi(u_1, \dots, u_m, v_1, \dots, v_n)$ in the language of ordered rings and a point $(a_1, \dots, a_n) \in \mathbb{R}^n$ we compute a quantifier-free formula $\varphi^*(\underline{u}, \underline{v})$ and a theory $\Theta(\underline{v})$ such that

$$\bigwedge \Theta \longrightarrow (\varphi \longleftrightarrow \varphi^*) \quad \text{and} \quad \bigwedge \Theta(a_1, \dots, a_n).$$

In other words we compute from φ a quantifier-free formula φ^* and a semi-algebraic set $S \subseteq \mathbb{R}^n$ containing $\underline{a}$, such that for all $\underline{r} \in \mathbb{R}^m$, and $\underline{s} \in S$ we have that $\varphi(r, s)$ and $\varphi^*(r, s)$ are equivalent. Note that the theory Θ contains only atomic formulas in the parameters $v_1, \dots, v_n$.

We call the variables $v_1, \dots, v_n$ *local parameters*, and $\underline{a} = (a_1, \dots, a_n)$ *suggested point* for the local variables. Terms and formulas containing only variables from the set $\{v_1, \dots, v_m\}$ are also called *local.* The semi-algebraic set

$$S = \left\{ \underline{x} \in \mathbb{R}^n \;\middle|\; \bigwedge \Theta(\underline{x}) \right\}$$

is called *range* of the local quantifier elimination applied to φ.

Local quantifier elimination is designed for both decreasing the size of the output formula and for decreasing the computation time. This is achieved by restricting the parameter space to an interesting area around the suggested point.

The scope of local quantifier elimination is between regular quantifier elimination applied to $\varphi(\underline{u}, \underline{v})$ and applied to $\varphi(\underline{u}, \underline{a})$. If $m = 0$ the latter case is actually a decision problem, whereas the former one is a quantifier elimination problem. Both of these special cases can be viewed as local quantifier elimination by setting either $\Theta = \emptyset$ or setting $\Theta = \{\, v_i = a_i \mid 1 \leq i \leq n \,\}$, respectively.

We are, however, not interested in these degenerated cases. Though the range will be restricted by our method, the range will as a rule not be restricted to the pure trivial case. Instead it will in almost all cases be an infinite semi-algebraic set containing the suggested point, and most frequently a neighborhood of this point. In comparison to regular quantifier elimination the output formula will be significantly smaller and will be computed faster. The local quantifier elimination procedure, as presented here, is based on the quantifier elimination by virtual substitution. Constraints for Θ are generated, whenever they support the algorithm.

The concept of local quantifier elimination is closely related to the generic quantifier elimination. For details on generic quantifier elimination, cf. [31]. Generic quantifier elimination computes to an input formula $\varphi(u_1, \dots, u_m)$ a formula $\varphi^*(u_1, \dots, u_m)$ and a theory $\Theta(u_1, \dots, u_m)$ such that

$$\bigwedge \Theta \longrightarrow (\varphi \longleftrightarrow \varphi^*),$$

where Θ contains, in contrast to the local quantifier elimination, only disequations. As a consequence the theory Θ holds for almost all parameter values. So the range of generic quantifier elimination applied to φ is usually larger than the range of local quantifier elimination. On the other hand the corresponding output formula will be much bigger in generic quantifier elimination. The resulting trade-off between the size of the range and the size of the output formula varies with the type of input formulas.

The plan of this chapter is as follows: In Section 6.1 we describe how to adapt quantifier elimination by virtual substitution to a local quantifier elimination algorithm. In Section 6.2 we discuss the combination of local quantifier elimination together with the ideas of generic quantifier elimination. In Section 6.3 we give an explicit series of examples, for which local quantifier elimination has a better complexity than regular quantifier elimination. Section 6.4 introduces our implementation of local quantifier elimination. In Section 6.5 we give some computation examples. Finally, in Section 6.6 we summarize the results of this chapter.

6.1 Local Quantifier Elimination by Substitution

In this chapter we present a local quantifier elimination procedure based on the quantifier elimination by virtual substitution introduced in Chapter 3. We describe in detail how we have adapted the phases of the virtual substitution method to obtain an efficient local quantifier elimination. All modifications described here are compatible with the optimization strategies discussed in the Chapter 4 and in Chapter 5.

In contrast to regular quantifier elimination we distinguish between non-local parameters $u_1, \dots, u_m$ and local parameters $v_1, \dots, v_n$. For the latter we specify in the input of the local quantifier elimination the suggested point $(a_1, \dots, a_n) \in \mathbb{R}^n$. Recall that local quantifier elimination computes for φ and $\underline{a}$ a quantifier-free formula φ^* and a theory Θ, such that

$$\bigwedge \Theta \longrightarrow (\varphi \longleftrightarrow \varphi^*) \quad \text{and} \quad \bigwedge \Theta(a_1, \dots, a_n).$$

Note that the condition $\Theta(\underline{a})$ guarantees, that Θ cannot become inconsistent. The constraints contained in Θ are generated according to the following scheme: Let t be a local term, $\underline{a} \in \mathbb{R}^n$, and define

$$R(s) = \begin{cases} > & \text{for } s = 1 \\ = & \text{for } s = 0 \\ < & \text{for } s = -1 \end{cases}.$$

We can automatically evaluate $t(\underline{a})$ and compute the sign s of $t(\underline{a})$. Defining

$$\theta_{\underline{a}}(t) \equiv t\ R(s)\ 0,$$

it follows obviously that $\theta_{\underline{a}}(t)(\underline{a})$ holds.

In the following subsections we discuss how to make use of a and Θ for speeding up the computation and for obtaining smaller output formulas. We consider modifications of three phases of the quantifier elimination: The computation of all candidate solutions, the virtual substitution of some candidate solutions, and the simplification of the result formula.

6.1.1 Local Computation of Candidate Solutions

Both computation time and output size of the quantifier elimination by virtual substitution depend heavily on the size of the computed elimination set. For an input formula of size $|\varphi|$ and an elimination set with size $|E|$ we compute, roughly speaking, an output formula of size $|\varphi| \cdot |E|$. Our major goal is thus to reduce the size of E. Recall from Section 3.3 that in the worst-case we compute for each atomic formula in the input formula a candidate solution including an appropriate guard. From the set of all candidate solutions we compute then an elimination set.

To begin with, consider an atomic formula $\alpha \equiv c_1x + c_0 \ \varrho \ 0$ where ϱ is one of our relations and suppose that c_1 is a local term. Adding the constraint $\theta_{\underline{a}}(c_1)$ to the theory Θ, ensures that the sign of c_1 is constant and known on the range of the local quantifier elimination. Hence we can decide whether α yields an upper bound, or a lower bound, or whether α is equivalent to $c_0 \ \varrho \ 0$. In the latter case we do not need to generate a candidate solution at all. The knowledge about the type of bound will support the elimination set computation. If c_1 is not local, i.e. it contains variables besides the parameters $\underline{v}$ we proceed as for regular quantifier elimination.

Constraints of the form $t > 0$ and $t < 0$ are valid not only for the suggested point $\underline{a}$, but also for all points in a neighborhood of $\underline{a}$. This is obviously false for equation constraints. We have therefore introduced *restricted* local quantifier elimination, which assumes only strict order relations in the theory Θ. If $\theta_{\underline{a}}(c_1)$ is an equation constraint, we do not add it to Θ and we proceed as for regular quantifier elimination.

Next we consider a quadratic atomic formula $c_2x^2 + c_1x + c_0 \ \varrho \ 0$. In this case our regular quantifier elimination procedure generates first a condition that c_2 does not vanish together with at most two candidate solutions belonging to the roots of $c_2x^2 + c_1x + c_0$. Furthermore it generates a condition, that c_2 vanishes together with the candidate solution for the atomic formula $c_1x + c_0 \ \varrho \ 0$.

If the highest coefficient c_2 is local we add $\theta_{\underline{a}}(c_2)$ to Θ. We do not consider the linear case provided that $\operatorname{sign}(c_2) \neq 0$ and we do not consider the pure quadratic case provided that $\operatorname{sign}(c_2) = 0$. If c_2 is not local we consider both the quadratic and the linear case separately. The linear case is discussed above. So we only have to clarify how to proceed in the pure quadratic case, i.e. we assume $c_2 \neq 0$. We start with the computation of the discriminant $\Delta = -4c_2c_0 + c_1^2$ and we check if it is local. If it is not local we proceed as usual. Otherwise we add the constraint $\theta_{\underline{a}}(\Delta)$ to Θ. If $\operatorname{sign}(\Delta) < 0$, which means that $c_2x^2 + c_1x + c_0$ does not have a real zero, we compute no candidate solution. If $\operatorname{sign}(\Delta) > 0$ we can drop the appropriate guard in front of the substitution result. If $\operatorname{sign}(\Delta) = 0$ we have to consider only the candidate solution $\frac{-c_1}{2c_2}$. Note that in this case, we either cannot determine the sign of c_2 or it is definitely different from 0. The restricted local quantifier elimination would, analogously to the linear case, not generate any equation constraints for Θ. In case that a local term evaluates to 0 the restricted quantifier elimination proceeds as the regular quantifier elimination. Thus, we have to code certain sign conditions into the output formula such that the output is, in general, not so short as in the unrestricted case.

The local quantifier elimination allows us to recognize more formulas as Gauss formulas or as co-Gauss formulas. The local quantifier elimination is here not only compatible with the strategies describe in Chapter 4 but contributes to the optimizations discussed there. For deciding $c_2x^2+c_1x+c_0=0$ to be non-trivial, we check, starting with c_2, successively if $\mathrm{sign}(c_i(\underline{a}))\neq 0$. If $\mathrm{sign}(c_j(\underline{a}))\neq 0$ for one j we add $\theta_{\underline{a}}(c_j)$ to Θ and apply the Gauss elimination as usual.

Finally, we summarize the number of constraints added to Θ, for the elimination of $\exists x(\psi)$. We compute to each atomic formula in ψ at most two candidate solutions. Recall, that during the computation of a candidate solution we add at most two constraints to Θ. This means, that the size of Θ is in $O(n)$, where n is the number of atomic formulas in φ. Of course, we simplify the obtained theory Θ at the very end of the quantifier elimination by applying our simplification algorithm.

6.1.2 Local Simplification

Simplification has been turned out to be crucial for a successful application of quantifier elimination. This suggests to modify the simplification algorithm in order to take advantage of the additional features of local quantifier elimination.

Only two steps of our standard simplifier are affected by the modifications for a local simplification. These steps are discussed in Chapter 2 on simplification of quantifier-free formulas. We adapt the following two steps to our framework of local quantifier elimination:

- The potential simplification of an atomic formula to a truth value. For example our simplifier recognizes $x^2+y^2\geq 0$ to be equivalent to "true."

- The potential simplification of a conjunction of two atomic formulas to a single atomic formula. For example we simplify $3x-2\geq 0\wedge 4x-3\geq 0$ to $3x-2\geq 0$. Our simplifier can combine only two atomic formulas of the form $t+q_1\ \varrho_1\ 0$ and $t+q_2\ \varrho_2\ 0$, where t is a term and q_1, q_2 are rational numbers. Note that not all conjunctions matching this condition, can be simplified.

In the next subsections we sketch how to improve these two steps in the framework of local quantifier elimination. We denote again with $\underline{a}$ the suggested point for the local parameters.

Consider a local atomic formula $t \varrho 0$, where ϱ is an arbitrary relation. Adding $\theta_{\underline{a}}(t)$ to Θ, the atomic formula is for all points in the range equivalent either to "true" or to "false." The respective truth value can be computed using the sign of $t(\underline{a})$. We can, hence, replace the atomic formula by "true" or "false," respectively. This supports the simplification in particular in the case of "true" in a disjunction or "false" in a conjunction. In this case a not necessarily local subformula can be replaced by one truth value.

Using this simplification one is faced with a larger growth of Θ than in the elimination phase. A larger Θ means in general a smaller range. Simplifying the result formula with the above sketched method can, in the worst case, add for each atomic formula in the input formula a new constraint to Θ. In our case the input formula is actually the output formula of the local quantifier elimination. This implies that in this case the size of Θ is in $O(n^2)$ instead of $O(n)$, where n is the number of atomic formula of the elimination input φ. The output formula, however, will be much shorter using this simplification.

A naive extension of our simplifier would apply this simplification for atomic formulas to each atomic formula contained in the input formula. Besides the disadvantage of the growth of Θ one has to deal with the problem of adding obviously unnecessary constraints to Θ. Consider, e.g. the subformula $v_1 > 0 \wedge v_2 > 0$ of a complex formula and suppose that $(a_1, a_2) = (1, 1)$. Then the simplifier would make the assumptions $v_1 > 0$ and $v_2 > 0$ but it is sufficient to assume either $v_1 > 0$ or $v_2 > 0$. This situation cannot be resolved uniformly, because the decision which constraint is more suitable depends on the given application. Adding further constraints to Θ means in general to further restrict the range. For a first implementation we suggest the heuristics to add successively some constraints until no more atomic formulas can be simplified this way. This avoids adding unnecessary constraints to Θ but cannot exclude that Θ is restricted much more than necessary, because this depends heavily on the added constraints and not on the number of the constraints in Θ.

Next we discuss how to combine two atomic formulas. For the general simplifier this was only possible for terms that differ only in the absolute summand. This concept can be easily extended in the framework of the local quantifier elimination. Here we can combine two atomic formulas of the form $t + p_1 \varrho_1 0$ and $t + p_2 \varrho_2 0$, where t is a term, that does not contain a local summand and both p_1 and p_2 are local terms. Adding $\theta_{\underline{a}}(p_1 - p_2)$ to Θ one can decide whether $p_1 < p_2$, $p_1 > p_2$, or $p_1 = p_2$. Using this information, we can straightforwardly generalize the techniques for simplifying conjunctions of our simplifier. As an example consider the formula $3x - v_1 - 1 >= 0 \wedge 4x - 2v_2 - 1 >= 0$ and suppose $(a_1, a_2) = (1, 1)$. Then we add the constraint $4v_1 + 4 < 6v_2 + 3$ and simplify the formula to $4x - 2v_2 - 1 >= 0$. As for the simplification of atomic formulas this simplification technique can add for each atomic formula in the input a new constraint to Θ. Again it is easy to see, how to define a restricted local simplifier that does not assume any equation constraints.

6.1.3 Local Virtual Substitution

We can take advantage of the possibility to decide the sign of a local term for obtaining better substitution results. Here we can both consider the assumptions already made and assume new constraints. The former case does not increase the size of Θ. In the latter case we may increase the size of Θ and are faced with problems similar to those discussed in the description of local simplification.

Consider a candidate solution -$\frac{c_0}{c_1}$ and an atomic formula $b_1x + b_0 > 0$. The candidate solution is necessary and valid only in the case $c_1 \neq 0$. Rewriting the substitution result without an denominator yields the formula $-b_1c_0c_1 - b_0c_1^2 > 0$ guarded with the condition $c_1 \neq 0$. If we can strengthen the condition to $c_1 > 0$ or $c_1 < 0$, respectively, as in the case of the local quantifier elimination, the result can be simplified to $-b_1c_1 - b_0 > 0$ or $-b_1c_1 - b_0 < 0$, respectively. Note that for a quotient $-\frac{c_0}{c_1}$ obtained as candidate solution of a linear constraint $c_1x + c_0 \varrho 0$ our theory Θ contains already one of the constraint $c_1 > 0$ and $c_1 < 0$, provided that c_1 is local. This observation can be easily generalized for the substitution into atomic formulas containing polynomials of an arbitrary x-degree and for the substitution of other terms containing denominators.

As discussed in the previous subsection local atomic formulas can always be evaluated to "true" or "false." Such atomic formulas are generated systematically for resolving the substitution of improper test points in atomic formulas. Consider for example the substitution of ∞ for x into an atomic formula $ax + b > 0$, where a and b are local. We obtain the result $a > 0 \lor a = 0 \land b > 0$. Both $a > 0$ and $b > 0$ are local and can by enlarging Θ appropriately be simplified.

6.1.4 Local vs. Generic Quantifier Elimination

In this subsection we discuss the relation between generic quantifier elimination, cf. [31] and local quantifier elimination. Even if the specifications of both variants of quantifier elimination are different, they are closely related to each other. Both variants automatically generate assumptions over terms in a specified subset of all parameters. These assumptions are collected in a theory Θ. The equivalence between input and output of both local and generic quantifier elimination is restricted to the semi-algebraic set represented by Θ. The main difference between the local and the generic quantifier elimination is the form of and the requirements on the theory. The generic quantifier elimination assumes only disequations. This implies on one hand that the equivalence holds for almost all parameter values. On the other hand it guarantees that Θ cannot become inconsistent. In case of the local quantifier elimination we restrict the range possibly to a zero-dimensional set. The restricted local quantifier elimination, however, does not assume equation constraints and thus the range has the full dimension of the local parameter space. The suggested point guarantees that the range contains an interesting part of the parameter space. In both variants Θ cannot become inconsistent, due to the requirement that Θ holds for the suggested point.

A second difference between the generic and the local quantifier elimination is more technical. Generic quantifier elimination may generate constraints only for the elements of the actually chosen elimination set. The local quantifier elimination, in contrast, may generate constraints for each candidate solution. Though this implies that local quantifier elimination assumes more constraints, the number of generated constraints is for both variants linear in the number of atomic formulas in the input. This changes if one allows to generate constraints for the simplification of the result or for the substitution of an elimination term into an atomic formula as described above. The idea to improve the substitution of test terms by introducing additional constraints into the theory can also be used in the framework of generic quantifier elimination.

6.2 Generic Local Quantifier Elimination

Up to now, our local quantifier elimination was only allowed to assume constraints restricting the local parameters. One can, of course, combine both the ideas of generic quantifier elimination and local quantifier elimination obtaining a "generic local quantifier elimination." For this variant of quantifier elimination we allow ourselves to assume order constraints restricting the local parameters and to assume disequations restricting all parameters. As in the case of the generic quantifier elimination, we may allow ourselves to specify a subset of all parameters on which we do not assume any constraint. The idea is in this case, that the specified parameters are actually bounded by some quantifiers outside of the considered formula.

6.3 A Remark on Complexity

In general local quantifier elimination has the same complexity as the regular quantifier elimination. In this section we give a series of examples with increasing number n of atomic formulas, such that the output formula of regular quantifier elimination as well as the generic quantifier elimination is exponential in n, whereas the output of the local quantifier elimination is only polynomial in n for an any suggested point.

We consider a parametric box in real n-space, which is given by the following formula:

$$\exists x_1 \cdots \exists x_n \big(\bigwedge_{i=1}^{n} a_i x_i \leq 1 \land b_i(x_i - 1) \leq 1\big).$$

We consider the following slightly modified input

$$\exists x_1 \cdots \exists x_n \big(\bigwedge_{i=1}^{n} a_i x_i \leq 1 \land b_i(x_i - 1) \leq 1 \land \Pi\big),$$

where Π is a quantifier-free formula in the parameters a_i and b_i.

For the elimination of x_n, regular quantifier elimination computes the following three candidate points:

$$-\infty, \quad \frac{1}{a_n}, \quad \frac{1+b_n}{b_n},$$

together with guards $a_n \neq 0$ and $b_n \neq 0$, respectively, for the latter two test points. Only all three points together form an elimination set. With other words, we cannot drop any candidate solution. The virtual substitution of the terms of the elimination set for x_i into the input formula results in the following formula:

$$\begin{aligned}
&\exists x_1 \cdots \exists x_{n-1}\Big(\\
&\quad \bigwedge_{i=1}^{n-1} a_i x_i \leq 1 \land b_i(x_i - 1) \leq 1 \land\\
&\qquad a_n \geq 0 \land b_n \geq 0 \land \Pi[x_n /\!/ \infty] \lor\\
&\quad \bigwedge_{i=1}^{n-1} a_i x_i \leq 1 \land b_i(x_i - 1) \leq 1 \land\\
&\qquad a_n \neq 0 \land b_n(a_n - a_n^2) \leq a_n^2 \land \Pi\big[x_n /\!/ \frac{1}{a_n}\big] \lor
\end{aligned}$$

$$\bigwedge_{i=1}^{n-1} a_i x_i \leq 1 \wedge b_i(x_i - 1) \leq 1 \wedge$$
$$b_n \neq 0 \wedge a_n(1 + b_n - b_n^2) \leq b_n^2 \wedge \Pi\big[x_n /\!/ \frac{1+b_n}{b_n}\big]\Big).$$

Even the generic quantifier elimination cannot drop one of the three candidate solutions. It may, however, add the conditions $a_n \neq 0$ and $b_n \neq 0$ to the theory and remove these atomic formulas from the substitution result.

For the elimination of the next quantifiers we interchange them with the toplevel disjunction. Thus we get 3 subproblems; each of them can be eliminated independently. Each of the subproblems has again the form of the considered input. Thus for the elimination of each quantifier we alway get 3 test points and we can always interchange the resulting disjunction and the remaining quantifiers. During the elimination we obtain a computation tree of depth n such that each node has exactly 3 successors. Altogether, this proves that the number of atomic formulas in the output of both regular and generic quantifier elimination is in $O(3^n)$, in other words it is exponential in n.

Next we consider the same elimination problem as an input of the local quantifier elimination procedure together with an arbitrary suggested point

$$(a_1, \dots, a_n, b_1, \dots, b_n) = (\alpha_1, \dots, \alpha_n, \beta_1, \dots, \beta_n).$$

We will discuss all possible cases: To begin with, we start with the worst case for our local quantifier elimination, namely $\operatorname{sign}(\alpha_n) = -\operatorname{sign}(\beta_n)$, where wlog. we may assume that $\operatorname{sign}(\alpha_n) = 1$. We consider, as above, the elimination of "$\exists x_n$." The local quantifier elimination then adds $\theta_{(\underline{\alpha},\underline{\beta})}(a_n)$ and $\theta_{(\underline{\alpha},\underline{\beta})}(b_n)$ to Θ. This restricts the range such that $a_n x_n \leq 1$ is an upper bound, whereas $b_i(x_i - 1) \leq 1$ is a lower bound. Due to the fact, that the constraints occur conjunctively on the toplevel and that Θ guarantees $a_n \neq 0$, one can easily see that, e.g. $\{\frac{1}{a_n}\}$ is an elimination set. The substitution result is

$$\exists x_1 \cdots \exists x_{n-1}\Big(\bigwedge_{i=1}^{n-1} a_i x_i \leq 1 \wedge b_i(x_i - 1) \leq 1 \wedge$$
$$b_n(a_n - a_n^2) \leq a_n^2 \wedge \Pi\big[x_n /\!/ \frac{1}{a_n}\big]\Big).$$

In all other cases either our elimination procedure computes either $\{\infty\}$ or $\{-\infty\}$ as an elimination set.

In all cases the substitution result matches the form of the input formula. Thus we can simply iterate the local quantifier elimination and we obtain in each iteration an elimination set containing only one element. Altogether we obtain a result formula, which is linear in n and a theory containing $2n$ constraints.

6.4 Implementation

An implementation of the local quantifier elimination in REDLOG is under development. The current version contains all discussed features except local virtual substitution and local simplification.

6.5 Application Examples

We present some example computations. All computations have been performed on a SUN ULTRA 1 computer with 140 Mhz and a heap space of 32 MByte for REDUCE.

6.5.1 Generic Quadratic Equation

In the first toy example we demonstrate the behavior of the implemented local quantifier elimination in contrast to the regular quantifier elimination and to the generic quantifier elimination. We consider the input formula $\varphi \equiv \exists x(v_2x^2 + v_1x + v_0 > 0)$ and the point $\underline{a} = (1, 1, 1)$.

Local quantifier elimination computes the result $v_0 > 0$ together with the theory $4v_0v_2 - v_1^2 > 0$. Quantifier elimination of $\exists x(v_2x^2 + v_1x + v_0 > 0)[v_0/1, v_1/1, v_2/1]$ yields the result "true," whereas regular quantifier elimination of $\exists x(v_2x^2 + v_1x + v_0 > 0)$ produces the result

$$\begin{aligned}
&v_2 > 0 \lor 2v_0v_1v_2 - v_1^3 > 0 \land v_1 \neq 0 \land v_2 = 0 \lor \\
&\quad \lor v_2 = 0 \land (v_1 > 0 \lor v_0 > 0 \land v_1 = 0) \lor \\
&\quad 4v_0v_2 - v_1^2 < 0 \land v_2 < 0.
\end{aligned}$$

Generic quantifier elimination computes the result $4v_0v_2 - v_1^2 < 0 \lor v_2 \geq 0$ together with the theory $v_2 \neq 0$. The computation time is in all cases smaller than the smallest measurable time of 10 ms.

6.5.2 Generic Polygon

We consider the input formula

$$\exists x \exists y \Big(\bigwedge_{i=1}^{n} a_i x + b_i y \leq c_i \Big),$$

which describes, whether a convex polygon is non-empty.

We fix $n = 3$ and specify the a_i and b_j as local parameters, suggesting the point

$$(a_1, a_2, a_3, b_1, b_2, b_3) = (1, -3, 5, -7, 11, -13).$$

Local quantifier elimination produces in 50 ms the output formula

$$a_1b_2^2c_3 - a_1b_2b_3c_2 - a_2b_1b_2c_3 + a_2b_2b_3c_1 + a_3b_1b_2c_2 - a_3b_2^2c_1 \le 0,$$

together with the theory

$$\begin{aligned} &a_1 > 0 \land a_2 < 0 \land a_3 > 0 \land b_1 < 0 \land b_2 > 0 \land b_3 < 0 \land \\ &\quad a_1b_2 - a_2b_1 < 0 \land a_2b_3 - a_3b_2 < 0. \end{aligned}$$

Regular quantifier elimination computes in 350 ms an output formula containing 78 atomic formula. Generic quantifier elimination computes in 450 ms the same output together with "true" as theory. Substituting the suggested point in the input the regular quantifier elimination computes in 10 ms the equivalent formula $8c_1 + 11c_2 + 5c_3 \ge 0$.

For $n = 10$ and the suggested point

$$(a_1, \dots, a_n, b_1, \dots, b_n) = (p_1, -p_2, p_3, \dots, p_{2n}),$$

where p_i is the i-th prime number we obtain the following results: The local quantifier elimination computes in 2.6 s a theory containing 55 atomic formulas and an output containing 160 atomic formulas. The regular quantifier elimination computes in 26 s a formula containing 1520 atomic formulas. This formula is also computed in 33.5 s by the generic quantifier elimination together with "true" as theory. Fixing the point allows us to compute in 190 ms a result with 160 atomic formulas.

6.5.3 Kahan's Problem

Kahan's problem, cf. [41] is one of the most well-known benchmark problems for quantifier elimination procedures:

> *The problem concerns four variables a, b, c, d to be interpreted as center (c, d) and principal semiaxes a, b of an ellipse*
>
> $$E : \Big(\frac{x-c}{a}\Big)^2 + \Big(\frac{y-d}{b}\Big)^2 - 1 = 0.$$
>
> *We wish to know when E lies inside the unit disk*
>
> $$D : x^2 + y^2 \le 1.$$

An optimal solution was computed by Lazard, cf. [50].

We consider here the special case $d = 0$, and suggest the point

$$a = 1/2, \quad b = 1/2, \quad c = 1/2.$$

Local quantifier elimination computes in 160 ms the result

$$a^2 + 2ac + c^2 - 1 \leq 0 \land a^2 - 2ac + c^2 - 1 \leq 0,$$

together with the theory

$$a^2 - b^2 = 0 \land a^2 > 0 \land b^2 c > 0 \land b^2 \neq 0.$$

Note that the theory does not imply the result formula. In other words, Θ does not restrict the problem to a trivial special case.

Regular quantifier elimination computes in 910 ms a formula containing 59 atomic formulas, and if we fix the suggested point we yield in 10 ms the result "true." Generic quantifier elimination computes a formula containing 35 atomic formulas together with the theory $a + b \neq 0 \land a - b \neq 0 \land a \neq 0$.

6.6 Conclusions

We have introduced local quantifier elimination as a variant of real quantifier elimination. For local quantifier elimination we allow ourselves to assume arbitrary order and equation constraints on local terms. As expected, this leads theoretically and practically to shorter output formulas than those produced by both regular and generic quantifier elimination. One consequence of the shorter (intermediate) results is the considerable speed-up of the elimination process. The suggested point for the local parameters guarantees that the range of the elimination is not empty and includes at least one point on which the user is interested in. Our concept of the restricted local quantifier elimination, guarantees furthermore that the range contains actually a neighborhood of the suggested point and has therefore the same dimension as the local parameter space. The theoretically expected improvements of local quantifier elimination in contrast to the regular quantifier elimination were exceeded by the results of our test implementation.

Wherever it suffices to restrict the equivalence of output and input to a neighborhood of the suggested point, the concept of local quantifier elimination is superior both to regular quantifier elimination and the generic quantifier elimination.

Local quantifier elimination is like the generic quantifier elimination one prominent example how to optimize the elimination process by a problem oriented adaption of the specification of quantifier elimination.

Chapter 7

Scheduling by Quantifier Elimination

Scheduling problems arise in many areas of science and economics. The application areas range from scheduling jobs on multi-processor machines to designing production facilities or distributing patients to physicians in a hospital. In almost all applications it is not sufficient to compute only a valid schedule but it is required to compute a schedule minimizing a given objective function. In practice one is faced with two tasks influencing each other: Selecting an appropriate model on one hand and modeling the situation on the other hand.

The general *machine model* is a very flexible approach to the formulation of scheduling problems [55, 11]. It is based on the production of some goods using some machines. It is, however, not restricted to this situation. It can, e.g., also be used to assign aircrafts to gates on an airport [8]. Many different algorithms have been developed to solve instances of the machine model efficiently. Most of the algorithms are exponential, even though some algorithms for extreme special cases are polynomial. *Project networks*, cf. [33, 59, 4] provide a more flexible model than dedicated machines. They are of particular importance for building and construction.

We examine, as a new modeling tool, the applicability of first-order formulas to describe scheduling problems. For solving scheduling problems described this way we will use extended quantifier elimination. Our approach covers both the dedicated machine model and the project networks. In addition first-order formulas allow the precise formal specification of scheduling problems that are far beyond the scope of both the dedicated machine model and project networks. For some cases they are highly relevant although currently no practical instance of the corresponding scheduling problem can be solved within reasonable time, because our first-order approach provides the only known adequate modeling tool.

For the restricted class of scheduling problems, where we have to compete with the dedicated machine model or the project networks it turns out that we can formulate the problems using only one block of existential quantifiers. The worst case complexity of our algorithm is in this situation only exponential, and we are thus in the same complexity class as the traditional algorithms. Note, however, that from a theoretical point of view the traditional approaches are better in a certain sense: They are in NP.

Our approach of solving scheduling problems by quantifier elimination is influenced by the idea to solve optimization problems with quantifier elimination [70, 73]. The formulation of scheduling problems within the dedicated machine model is related to that of solving them with constraint logic programming, cf. [67, 9].

It is not at all surprising that the first-order formulas obtained for the formulation of scheduling problems show certain structural similarities. We can use this knowledge of the structure of the formulas to dramatically improve the efficiency of quantifier elimination by virtual substitution for this special case.

This makes the entire discussion fit perfectly into our thesis. After having optimized the elimination procedure and all its subalgorithm to the most possible extent, we now show that there is still additional tuning possible by using information on class problems to be solved.

The plan of this chapter is as follows: In Section 7.1 we sketch a method for solving optimization problems with quantifier elimination. Section 7.2 summarizes the basics of the dedicated machine model. In Section 7.3 we present an algorithmic strategy for translating instances of the dedicated machine model into a first-order formula. In Section 7.4 we present various strategies how to use the knowledge about special problem classes to improve the quantifier elimination. The treatment of project networks is discussed in Section 7.5. In Section 7.6 we demonstrate the enormous scope of first-order formulation of scheduling problems by means of a case study. The example there also demonstrates how to realize multi-objective optimization.

7.1 Optimization by Quantifier Elimination

Quantifier elimination can be used for finding the optimum of certain real functions subject to real constraints. This obviously works, when the objective function can be expressed as a term. We will later extend this observation to functions that can be described by means of first-order formulas. Extended quantifier elimination can be used for finding not only the optimal value but also some optimal point.

The major part of the first-order formulation of such an optimization problem is obviously given by the constraints. Forming a conjunction of the constraints exactly corresponds to the classical optimization situations, where there is a list of constraints given, which are considered conjunctively. Within our framework we have in addition the expressive power of disjunctions and negations available.

We now have to clarify how to integrate the objective function into our first-order formulation. For the classical purely conjunctive optimization problem it is well-known from other elimination oriented approaches, such as the Fourier–Motzkin method, how to proceed [60]: The objective function enters the formulation as an additional constraint with a new variable. This new variable is the only one, which is not eliminated such that the condition in this variable obtained from elimination describes the optimal value of the objective function. The following proposition collects these well-known facts more precisely and within our framework.

Proposition 7.1. *Let ψ be a first-order formula in the variables $x_1, \dots, x_n$ and let*

$$f : D = \{ (x_1, \dots, x_n) \mid \psi(x_1, \dots, x_n) \} \to \mathbb{R}.$$

Define $\varphi(z)$ by

$$\exists x_1 \cdots \exists x_n \bigl(\psi(x_1, \dots, x_n) \land z \geq f(x_1, \dots, x_n)\bigr)$$

and let $m \in \mathbb{R}$. Then the following holds:

1. *$\varphi(z)$ is contradictory if and only if $D = \emptyset$.*
2. *$\varphi(z)$ is tautological if and only if $D \neq \emptyset$ and there is no lower bound for f.*
3. *$\varphi(z)$ is equivalent to $z \geq m$ if and only if $D \neq \emptyset$ and $\min(f) = m$.*
4. *$\varphi(z)$ is equivalent to $z > m$ if and only if $D \neq \emptyset$, $\inf(f) = m$, and there is no minimum of f.*
5. *The formula φ is equivalent to one of "true," "false," $z \geq m$, or $z > m$.*

We see that the minimum m of f is provided by some quantifier-free formula equivalent to $z \geq m$. We are, however, faced with the problem that in general we may not expect our quantifier elimination result to have such a nice syntactic form. In fact, for non-rational $m \in \mathbb{R}$ our chosen language of ordered rings imposes that there must be non-linear polynomials involved, which algebraically describes m.

Suppose, in contrast, the special case that the input is a linear formula. It is then clear that m will be rational or $-\infty$. This follows on one hand from the fact that all the input constraints describe intervals with rational or infinite boundaries. On the other hand this follows independently from the syntactic form of the elimination result, which is a linear formula over the language of ordered rings. Moreover, our simplification methods in Chapter 2 are powerful enough to guarantee that we obtain a result of the form $az+b \geq 0 \longleftrightarrow z \geq m$ with a, $b \in \mathbb{Z}$. Our whole discussion is certainly also applicable to the infimum case.

Note that we obtain the straightforward form $az + b \geq 0$ for the minimum of the objective function only with regular quantifier elimination. With extended quantifier elimination, which we want to use for simultaneously determining some point of minimal value, we obtain conditions with sample solutions

$$\eta^* \equiv \begin{bmatrix} \gamma^{(1)}(z) & \{x_1^{(1)} = t_1^{(1)}(z), \dots, x_n^{(1)} = t_n^{(1)}(z)\} \\ \vdots & \vdots \\ \gamma^{(k)}(z) & \{x_1^{(k)} = t_1^{(k)}(z), \dots, x_n^{(k)} = t_n^{(k)}(z)\} \end{bmatrix},$$

such that $\bigvee_{i=1}^{k} \gamma^{(i)}$ is a quantifier-free equivalent. According to our introduction of extended quantifier elimination by virtual substitution in Section 3.2 there is, however, no simplification between the single $\gamma^{(i)}$ performed, i.e., we do not obtain our nice description of the minimum. We thus apply the following straightforward algorithm to η^*:

1. Compute $\bigvee_{i=1}^{k} \gamma^{(i)}$ and simplify. The result will for linear input formulas be of the form $az + b \geq 0$, and generally have the same quality as the corresponding result obtained by regular quantifier elimination.

2. Substitute $-\frac{b}{a}$ for z into η^*, and simplify all the substitution results for the $\gamma^{(1)}, \dots, \gamma^{(k)}$ to either "true" or "false." This yields η^{**}. All "true" lines in η^{**} provide sample points with minimal value $-\frac{b}{a}$ of the objective function.

7.2 The Dedicated Machine Model

The dedicated machine model is a common way to formalize scheduling problems. The two main concepts of this model are *jobs* and *machines*. To each job there is exactly one machine assigned on which the job can be processed. Once started, a job can neither be interrupted nor cancelled. The time needed for processing a job is known and fixed in advance. In particular, it is independent of its starting time or of the processing of other jobs. The machines are available over the entire considered time period. Each machine can process only one job at a given time.

Scheduling means to compute a schedule, i.e. an assignment of starting times to the jobs, such that the above requirements are fulfilled. The set of *valid* schedules can furthermore be restricted by additional constraints: The period within which a job i can be processed can be restricted by a *release date* $r(i)$ and a *strict due date* $e(i)$. Between some of the jobs there can be a *precedence relation* $\prec$ given. For jobs i, j the meaning of $i \prec j$ is that the execution of job i must be finished before the execution of job j can be started.

The costs of schedules can be measured by *cost* functions defined on the set of all schedules. *Optimal scheduling* consists in finding a valid schedule, which has minimal costs among all other valid schedules. Cost functions are usually defined in terms of the *completion time* C_i and the *due date* $d(i)$ of the jobs i. The completion time C_i is the time at which the execution of a job i terminates. The due date $d(i)$ of a job i—not to be confused with the strict due date $e(i)$—specifies a *desired* end time but does not restrict the set of valid schedules. To support the satisfaction of such due dates, there are penalties introduced for each job that is finished before or after its due date. Note the difference between the functional notation of quantities that are given independently of the schedule on one hand and the index notation of quantities depending on a chosen schedule on the other hand.

Almost all cost functions considered are defined in terms of the following quantities:

Lateness $L_i = C_i - d(i)$

Earliness $E_i = \max\bigl(0, d_i - C(i)\bigr)$

Tardiness $T_i = \max\bigl(0, C_i - d(i)\bigr)$

Absolute deviation $D_i = |C_i - d(i)|$

Unit penalty $U_i = 0$ provided that $C_i \leq d(i)$; $U_i = 1$ otherwise.

In practice, one almost always restricts to cost functions $G_{\max}$ and G_Σ of the following form:

$$G_{\max}(J) = \max(\{\, \omega_i \cdot G_i \mid i \in J \,\}) \quad \text{and} \quad G_\Sigma(J) = \sum_{i \in J} \omega_i \cdot G_i,$$

Here J is a subset of all jobs, which identifies a schedule. The $\omega_i \in \mathbb{Q}$ are rational weights, and G is a syntactic placeholder for one of C, L, E, T,

Table 7.1: Processing times and machine assignments

Job i	1	2	3	4	5	6	7	8	9	10	11
Time $p(i)$	40	20	15	20	20	20	10	15	15	20	15
Machine $m(i)$	4	2	1	1	2	4	3	2	1	3	4

D, and U. One of the most frequent cost functions is $C_{\max}$. It is called *makespan.*

Since we want to minimize the costs we refer to the costs functions also as objective functions. Such an objective function is called *regular* if it is monotone in the C_i.

Some authors consider a variant of the machine model, in which the jobs are divided into tasks. The tasks are ordered linearly, and the start time of a job is the start time of the first task, the end time of the job is the end time of the last task. Each problem formulated using this formalism can easily be restated using our model. The idea for this is identifying tasks with jobs and choosing an appropriate precedence relation for the jobs.

We conclude this section with the description of an example by Breitinger and Lock, cf. [9], adapted to our formalism. Its automatic solution is discussed in the following sections. The problem consists of 11 jobs which have to be scheduled on four machines. We identify the jobs with the numbers from 1 to 11 and the machines with numbers 1 to 4. The processing times $p(i)$ and machine assignments $m(i)$ are given in Table 7.1. The precedence relation was in the original example given only by a job/task relation. We formulate it explicitly by

$$1 \prec 2 \prec 3, \qquad 4 \prec 5 \prec 6 \prec 7, \quad 8 \prec 9 \prec 10 \prec 11.$$

The objective function is $C_{\max}(1, \dots, 11)$ which is equal to $C_{\max}(3, 7, 11)$ due to the given precedence relations.

7.3 Formulating the Dedicated Machine Model

In this section we describe how to transform an instance of the dedicated machine model into a first-order formula. We consider here the scheduling problem as a special case of an optimization problem and make use of the technique introduced in Proposition 7.1. We give moreover hints how to use our flexible approach to handle generalizations of the dedicated machine model.

Let I be the set of all jobs. For a job $i \in I$ we denote by $p(i)$ its processing time and by $m(i)$ the machine assigned to the job. If for a job i one of the functions r, d, or e is not defined, we will denote this by a value of $\perp$.

The main idea for first-order formulation is to identify jobs within a schedule with their start time. We accordingly introduce for each job i a variable t_i representing its start time. This idea was originally introduced by Breitinger and Lock [9] for solving optimal scheduling problems by constraint logic programming. The end time of job i is then obviously given by $t_i + p(i)$. Using t_i and $t_i + p(i)$ we can easily code all general restrictions of dedicated machine models together with our concrete scheduling problem in a first-order formula: We define time to start at some certain point, which we denote by 0, and to extend to infinity:

$$\mathrm{T} \equiv \bigwedge_{i \in I} t_i \geq 0,$$

For our example we obtain $\mathrm{T} \equiv \bigwedge_{i=1}^{11} t_i \geq 0$. The requirement that one machine can process only one job at a given time is formalized as follows:

$$\Sigma \equiv \bigwedge_{\substack{i<j \in I \\ m(i)=m(j)}} \big((t_i + p(i) \leq t_j\big) \vee \big(t_i \leq t_j + p(j)\big),$$

for our example

$$\begin{aligned}\Sigma \equiv\ & (t_3 + 15 \leq t_4 \vee t_3 \geq t_4 + 20) \wedge (t_3 + 15 \leq t_9 \vee t_3 \geq t_9 + 15) \wedge \\ & (t_4 + 20 \leq t_9 \vee t_4 \geq t_9 + 15) \wedge (t_2 + 20 \leq t_5 \vee t_2 \geq t_5 + 20) \wedge \\ & (t_2 + 20 \leq t_8 \vee t_2 \geq t_8 + 15) \wedge (t_5 + 20 \leq t_8 \vee t_5 \geq t_8 + 15) \wedge \\ & (t_{10} + 20 \leq t_7 \vee t_{10} \geq t_7 + 10) \wedge (t_1 + 40 \leq t_6 \vee t_1 \geq t_6 + 20) \wedge \\ & (t_1 + 40 \leq t_{11} \vee t_1 \geq t_{11} + 15) \wedge (t_{11} + 15 \leq t_6 \vee t_{11} \geq t_6 + 20).\end{aligned}$$

The precedence constraints for the jobs are written down in the subformula

$$\Psi \equiv \bigwedge_{i,j \in I, i \prec j} t_i + p(i) \leq t_j.$$

This formulation of Ψ produces more constraints than necessary, because $\prec$ is transitive. Instead of $\prec$, it is possible to use a suitable relation $\prec'$ the transitive hull of which equals $\prec$. The corresponding variant of Ψ is then denoted by Ψ'. For our scheduling problem we state

$$\begin{aligned}\Psi' \ \equiv\ & t_1 + 40 \leq t_2 \wedge t_2 + 20 \leq t_3 \wedge \\ & t_4 + 20 \leq t_5 \wedge t_5 + 20 \leq t_6 \wedge t_6 + 20 \leq t_7 \wedge \\ & t_8 + 15 \leq t_9 \wedge t_9 + 15 \leq t_{10} \wedge t_{10} + 20 \leq t_{11}.\end{aligned}$$

The release date constraints are translated into the following formula:

$$\mathrm{P} \equiv \bigwedge_{i \in I, r(i) \neq \bot} t_i \geq r(i)$$

The following formula guarantees that all strict due dates are fulfilled:

$$\Delta \equiv \bigwedge_{i \in I, e(i) \neq \bot} t_i + p(i) \leq e(i).$$

For our example there are neither release dates nor strict due dates given, and we can ignore the previous two formulas.

It remains to be clarified how to specify the constraint involving the cost function, which is in general not a polynomial, as objective function: There is no need for encoding the objective function in one single constraint. Instead it suffices to give a suitable formula encoding the restriction $z \geq f(\underline{x})$. This allows a large class of objective functions including in particular all piecewise linear functions. For our example we can thus optimize the makespan $\max\{\, C_i \mid i \in I\}$ by encoding the constraint $z \geq \max\{\, C_i \mid i \in I\}$ as follows:

$$\bigwedge_{i \in I} z \geq C_i.$$

Similarly, all other common objective functions can also be coded. We generally denote the formula description of the constraint for the objective function by Ω. For our concrete example we obtain

$$\Omega \equiv z \geq t_3 + 15 \land z \geq t_7 + 10 \land z \geq t_{11} + 15.$$

So far we have collected for the formulation of our scheduling example the formula $\mathrm{T} \land \Sigma \land \Psi \land \mathrm{P} \land \Delta \land \Omega$. We denote by $\underline{\exists}_z$ the existential closure of a formula up to the variable z, and obtain

$$\underline{\exists}_z(\mathrm{T} \land \Sigma \land \Psi \land \mathrm{P} \land \Delta \land \Omega)$$

as the first-order formulation of dedicated machine scheduling.

We use our the extended quantifier elimination of REDLOG, cf. Chapter 8, to solve our scheduling problem. From the extended quantifier elimination result we obtain both the minimal costs and sample values for the t_i determining the optimal schedule. On a Sun Sparc Ultra 1 we obtain in 3.5 s the minimal value 75 together with the sample solution shown in Table 7.2. This concludes the discussion of the translation of our dedicated machine schedule. This solution can be automatically translated into a Gantt chart

Table 7.2: Optimal start times

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
0	40	60	0	20	40	65	5	25	40	60

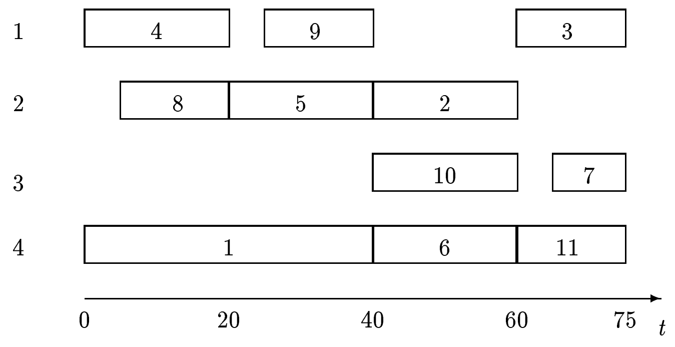


Figure 7.1: The Gantt chart of our solution

representing it graphically. It is shown in Figure 7.1.

To conclude this section let us once more turn our attention to the range of possible objective functions. Recall from above that the objective function is encoded in a formula. This approach is very flexible and admits the encoding of all usual considered objective functions. This is already an advantage to many algorithms for solving scheduling problems, which allow only a very restricted class of objective functions. In addition, we can encode more objective functions than covered by the usual solvers. In particular we can handle non-regular objective functions.

Note finally that we can even use quantifiers for expressing our objective function. Although this is due to the existence of a quantifier elimination procedure never really necessary, it sometimes offers more elegant descriptions. An example can be found in Section 7.5.2.

7.4 Adapting Quantifier Elimination to Scheduling

In Chapter 3 we have presented quantifier elimination by virtual substitution. In the following chapters 4 and 5 we have discussed optimization strategies for this quantifier elimination procedure. All the optimizations discussed so far have been of a general nature, i.e., they did not tune the procedure for particular types of input. One may certainly expect that it is well possible to find a variety of possible optimizations based on the assumption that the input formulas have a particular form. The idea of this chapter is to exemplify optimization of this kind for the special case of input formulas describing scheduling problems.

On one hand we can make use of certain properties of the formula occurring during our computations, and on the other hand we can explicitly specify knowledge to our quantifier elimination procedure which is encoded into the formula but cannot be recognized later by the quantifier elimination procedure.

For verifying the quality of our improvements we have implemented a special version of the general quantifier elimination procedure discussed in the thesis. This version is restricted to the linear weakly parametric case and allows only positive formulas that do not contain any strict relations. We have included in this implementation the generalized Gauss elimination discussed in Section 4.3 and the passive list approach discussed at the end of Section 3.2.6.

For the purpose of this analysis we consider two measures on an elimination run: First, the time it takes, and second, the number of nodes contained in its elimination tree, which has been defined in Section 3.2.6. This second number shows how often a particular optimization has been successfully applied. The comparison between the computation time and the number of nodes shows the trade-off between the extra time needed for applying this optimization and the gain in reducing the number of nodes.

Using our implementation in the basic form described above, we obtain the optimal schedule for our dedicated machine model example of the previous section in 3.5 s computing 894 nodes. In a variant without the passive list we compute the same result in 3.2 s but computing 1235 nodes. This illustrates how the combination of both our measures of complexity provide interesting insights into the effect of optimizations: The passive list idea causes an impressive decrease in the number of nodes to be computed, but more research has to be spent into its efficient realization.

For tuning our general quantifier elimination to become a specialized scheduling problem solver we analyze the effects of three independent optimizations ideas, which can easily be combined:

1. Result inheritance.
2. Estimating the objective function.
3. Evaluating the partial order.

The following subsections are devoted to these ideas.

7.4.1 Result Inheritance

We first discuss the *result inheritance*, a technique which is related to the well-known “branch-and-bound” approach in optimization algorithms. The main idea for result inheritance is to make use of the partial results obtained earlier.

For this, we consider the elimination of all quantified variables. Recall moreover from Section 3.2.6 that we can process the elimination tree either in a depth-first search manner or in a breadth-first search manner. We now consider the case that we process the elimination tree in a depth-first search manner. The leaves of the tree are the partial results φ_i^* which are combined disjunctively to the result of the elimination. We are thus in the situation

$$\varphi^* \equiv \varphi_1^* \vee \cdots \vee \varphi_k^*.$$

By transforming the disjunction into a disjoint disjunction we obtain the equivalent formula

$$\varphi_1^* \vee (\neg\varphi_1^* \wedge \varphi_2^*) \vee \cdots \vee (\neg\varphi_1^* \wedge \cdots \wedge \neg\varphi_{k-1}^* \wedge \varphi_k^*).$$

Traversing the elimination tree during the quantifier elimination in the depth-first manner we sequentially obtain $\varphi_1^*, \ldots, \varphi_k^*$. It is easy to see that we can assume $\theta \equiv \neg\varphi_1^* \wedge \cdots \wedge \neg\varphi_{i-1}^*$ already during the computation of φ_i^*. This is, of course, correct also for the general quantifier elimination and is not restricted to the situation of scheduling or optimization. There are now two things to be clarified: First, we certainly have to explain how to use this knowledge. Second, we have to explain how to technically communicate these information, since we certainly do not want to explicitly modify the formula as we have done for didactic reasons above. It is a straightforward idea in this concern to pass down the information as an implicit theory $\{\neg\varphi_1^*, \ldots, \neg\varphi_{i-1}^*\}$. Unfortunately, the φ_i^* are in general complex subformulas, which we have not allowed to enter a theory. It would not help to weaken this restriction, because Chapter 2 does not offer suitable heuristics for the application of non-atomic theories. In other words, there would be no way to efficiently make use of the knowledge.

Luckily, it turns out that in the special case of scheduling problems the majority of the information contained in θ can be extracted in the form of atomic formulas and can thus be added to the implicit theory Θ. We explain how to extend an already constructed implicit theory Θ after having computed the partial result φ_i^*. Recall from Proposition 7.1 that the output formula φ^* describes an interval $[m, \infty[$ bounded from below, where m is the minimum we are looking for. Thus each of the solution sets of the φ_i^* are also bounded from below. In almost all cases, we have heuristically observed that φ_i^* is actually an atomic formula describing an interval $[m', \infty[$ where $m \leq m'$. If φ_i^* is a disjunction of atomic formulas, and thus the negation $\neg\varphi_i^*$ is equivalent to a conjunction $\overline{\varphi_i^*}$ of atomic formulas, we can extend Θ by the atomic formulas contained in $\overline{\varphi_i^*}$. In all other cases we proceed as follows: We turn $\neg\varphi_i^*$ into an equivalent positive conjunctive normal form

$$\alpha_1 \wedge \cdots \wedge \alpha_k \wedge \gamma_1 \wedge \cdots \wedge \gamma_q,$$

where the $\alpha_1, \ldots, \alpha_k$ are atomic formulas and the $\gamma_1, \ldots, \gamma_q$ are complex formulas. We then extend Θ by $\alpha_1, \ldots, \alpha_k$.

For almost all applications of quantifier elimination by virtual substitution to scheduling problems it turns out that this approach considerably decreases both the computation time and the number of nodes. In our example we compute 819 nodes in 2.6 s without using the passive list and 652 nodes in 2.9 s using the passive list.

The simplifications that take place here can be described in terms of scheduling. The simplifier can by means of the implicit theory discussed here simplify an intermediate result to "false," which belongs to a branch encoding a lower interval bound m' such that $m \leq m'$, where m is the actual minimum. In other words, we prune a branch that encodes a lower bound not less than the minimal value m.

7.4.2 Estimating the Objective Function

In the previous section we have seen that we can use intermediate results to prune superfluous branches. A branch is superfluous if it provides an optimal value worse than an already known optimal value. In the approach of the previous section this was recognized by our standard simplifier. In this section we extend this approach by providing information about the scheduling problem explicitly. This external information is accessed and used during the quantifier elimination run.

We restrict our attention to the case of a minimizing makespan. Our results can then be easily adapted to other objective functions. Our idea is to maintain during the quantifier elimination process an estimate for the lower bound of the objective function. We prune the current branch if our estimate tells us that its local optimal value will definitely be worse than that inherited from some other branch already processed.

One ingredient for the estimate is the set of *offsets* $\omega(i)$ of all jobs i. These offsets are in an initial step computed from the specification of the scheduling problem. They remain invariant during the entire elimination process. The

Table 7.3: Computed offsets

i	1	2	3	4	5	6	7	8	9	10	11
$\omega(i)$	75	35	15	70	50	40	10	65	50	35	15

offset of a job i to the end is the time which is at least necessary to complete all jobs j with $i \prec j$, formally

$$\omega(i) = \sum_{i \prec j} p(j).$$

Example 7.2 (Offset). Consider the Breitinger–Lock example introduced at the end of Section 7.2. We obtain for obeying $1 \prec 2 \prec 3$ for job 1 the sum $\omega(1) = p(1) + p(2) + p(3) = 75$. In Table 7.3 we summarize all other offsets that can be obtained for our example.

We may now assume that the elimination knows all the offsets. We describe how the elimination step for a particular node of the elimination tree is performed. Let ψ be the formula contained in our node. We can assume that the toplevel operator of ψ is "$\wedge$" since all disjunctions are split into separate nodes. We extract from the conjunctive toplevel of ψ all linear weak lower bounds of the form $qt_i - p \varrho 0$ with p, $q \in \mathbb{Z}$. In view of the additive smart simplification of Section 2.4.2 we may assume that there is at most one such bound present for each job i. From the present bounds we derive the lower interval boundary $\frac{p}{q}$ and denote it by β_i where i is the job number. For jobs j for which there is no bound present we set $\beta_j = 0$; recall that the time axis starts at time 0. The global offsets ω_i together with the local boundaries β_i enable us to compute the following lower estimate for the makespan:

$$\eta = \max_{i \in I}(\beta_i + \omega_i).$$

Summarizing, we have just started the elimination step for one node of the elimination tree. As our first action we have computed a lower estimate η for the objective function. Recall now from the previous section that we inherit from neighboring branches an upper bound m' on the minimum of the objective function. It follows that for $m' \leq \eta$ our current branch cannot improve the already found temporary minimum m'. In this case we can immediately abort the treatment of our node and backtrack to the next branch. Otherwise our heuristics has failed, and we continue as usual.

Our lower estimate of the objective function leads to dramatic improvements both on the number of processed nodes and on the computation time. In our example we now compute 199 nodes in 1.0 s.

7.4.3 Evaluating the Partial Order

The success of the approach of the previous section results from a lower estimate of the objective function. For this estimate we have made use of the offsets computed for each job. Using the offsets we have actually made use of the partial order $\prec$ given with the specification of the scheduling problem. This partial order $\prec$ is successively extended during the computation of a schedule. This gives rise to the idea not only to consider $\prec$ but the current extension $\prec'$ of $\prec$ for the estimate.

Even though the transitivity of $\prec'$ is encoded implicitly into the formula it is neither recognized by our simplifier nor does it enter the estimate for the objective function discussed in the previous section. Consider, e.g., the formula

$$10 \leq t_1 \wedge t_1 + 10 \leq t_2 \wedge \psi,$$

where ψ is an arbitrary formula. Then it follows that $20 \leq t_2$ and thus $20 + p(2)$ is a lower bound for makespan. Without the information that $t_1 \prec t_2$ we cannot recognize this by means of the technique described in the previous section.

This gives rise to the idea to explicitly store and update the ordering information on the jobs within the nodes of the elimination tree. Recall from Section 7.3 that the typical atomic formula of our formulation has the form $c_1 t_1 + c_2 t_2 + c_0 \mathrel{\varrho} 0$, where c_0, c_1, and $c_2 \in \mathbb{Z}$ and $\varrho \in \{\leq, \geq\}$. Let us for the moment refer to such formulas as *job connections*. Suppose we substitute a corresponding test point

$$\left(\text{true}, -\frac{c_2 t_2 + c_0}{c_1}\right)$$

for the elimination of t_1 into another job connection. In terms of scheduling the substitution of this test point means fixing the ordering between t_1 and t_2. This information is stored in the elimination tree node that origins from the substitution. Note that the substitution result is again a job connection, which means that we can iterate the process on our child nodes.

From above, we already know one possible application for our new knowledge: It enables us to detect lower bounds. There is, however, another application of the same importance: The ordering information successively stored within the nodes can become inconsistent. In such a case we may abort the corresponding branch, and backtrack to the next one.

Technically, it is convenient to use a graph representation for the ordering information within the elimination tree nodes.

Using this approach our example computes 189 nodes in 4.1 s. This is a slight improvement in the number of nodes even compared to the surpris-

Table 7.4: Strategies and their combination

Strategy		R	R,E	R,O	R,E,O
Nodes	1235	819	228	219	219
Time (s)	3.1	2.6	0.7	4.9	1.9
Strategy	P	R,P	R,P,E	R,P,O	R,P,E,O
Nodes	894	652	199	189	189
Time (s)	3.5	2.8	1.2	4.1	1.49

ingly good result of the optimization techniques discussed in the previous section. The computation time, however, indicates a considerable overhead introduced by the implementation of the strategy discussed here. We have experimentally verified that this is caused mainly by adding and extracting ordering information.

7.4.4 Comparison of the Strategies

In Table 7.4 we summarize computation times and node numbers of our scheduling problem solver using the different strategies and combinations of these strategies. The strategies are abbreviated as follows:

R Result inheritance

P Applying the passive list

E Estimating the objective function

O Evaluating the partial order

Note that application of “E” or “O” requires “R”. The data for the computation without result inheritance are given only for comparison purposes. The use of “R” has no disadvantages in our framework because it does not introduce any relevant overhead.

The table shows that the strategy of estimating objective function should definitely be applied. It reduces both the number of nodes and the computation time. The use of the passive list reduces the number of computed node but increases the computation times slightly. The evaluation of the partial order is time consuming and provides only a slight improvement.

It is, however, a promising idea to look for heuristics that lie somehow between the estimation of the objective function described in Section 7.4.2 and the evaluation of the partial ordering described in Section 7.4.3. An natural approach would be using strategy “O” for a simpler homomorphic image of the partial order.

7.5 Multiple Resource Scheduling

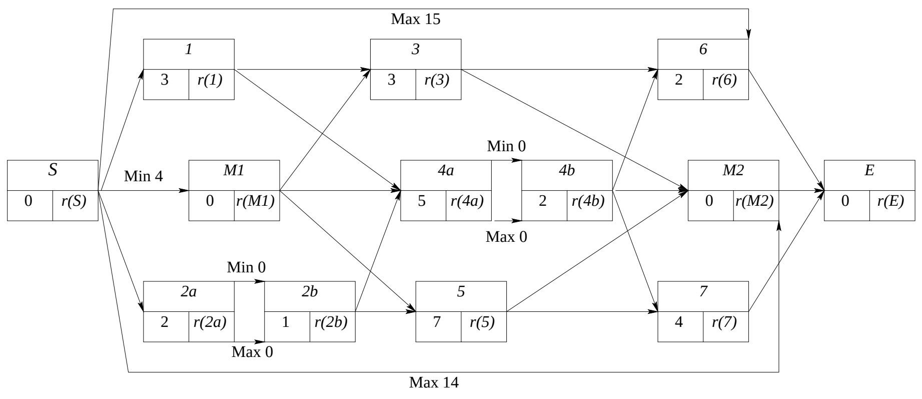
With the increasing relevance of solving scheduling problems there were many generalizations of the machine model discussed. *Project networks*, cf. [33, 4], provide an important alternative to the dedicated machine model discussed so far. In this alternative framework we consider *jobs* exactly as in the dedicated machine model. In addition, we introduce the notion of *resources*, which are a generalization of the machines of the dedicated machine model. There is an essential difference between resources and machines: We can have more than one instance of each resource type. The notion of “multiple resource scheduling” refers exactly to the possibility of such multiple instances of a resource type.

7.5.1 Project Networks

For each job we specify how many instances of which resource types are required. Moreover, within a project network one can specify not only the precedence between two jobs but it is also allowed to specify in addition both minimal and maximal times between these two jobs. This can be done in various ways, which we are going to discuss below. For now note that these intermediate time constraints are concerned with time periods passing between jobs, in contrast to periods within which resource instances have to remain unused.

Project networks are given as directed graphs with labeled vertices and edges. The labeled vertices of such a graph represent jobs. The labeled edges give the temporal dependencies. Figure 7.2 shows a sample project network. Figure 7.3 explains the layout of the vertices, which contain the job identifier i, the processing time $p(i)$, and the resource requirements $r(i)$ explaining which number of units of which resource type is required. The directed edges are labeled with the minimal time between the jobs they connect and with the maximal time between these jobs. Both these labels are optional. Omitting them formally corresponds to a zero minimal time and an infinite maximal time.

There are several locations at the vertices where edges are allowed to start or to end. The various choices correspond to the information which times, start times or end times, of the two connected jobs are related to one another. Since we have two time information for each of the two jobs there are four



Job	1	2a	2b	3	4a	4b	5	6	7	S	M1	M2	E
Resource Type 1	1	3	2	0	0	0	0	0	0	0	0	0	0
Resource Type 2	0	0	0	1	1	0	1	0	0	0	0	0	0
Resource Type 3	0	0	0	0	0	0	1	1	0	0	0	0	0

Figure 7.2: Bartusch's Example

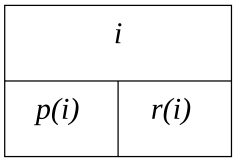


Figure 7.3: Job representation in project networks

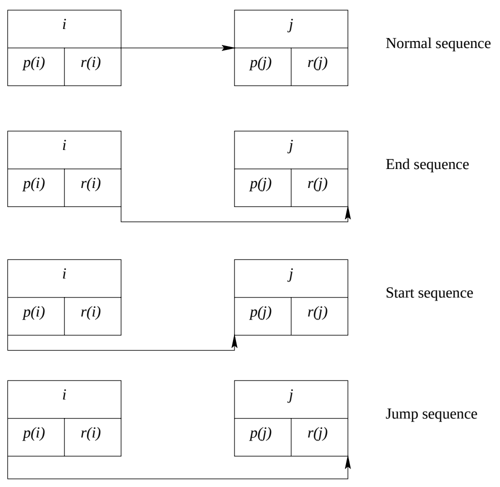


Figure 7.4: Types of sequences

possible configurations, which are pictured in Figure 7.4: A *normal sequence* relates the end time of one job with the start time of the next job, an *end sequence* relates the end times of two jobs, a *start sequence* relates the start times of two jobs, and finally a *jump sequence* relates the start time of a job with the end time of the next job. It is clear that all these sequences can be formally described by appropriate additional labels for the edges. Since we know the processing times of the jobs, we can translate all these sequences to standard sequences. They thus simply provide a nice modeling tool but not a conceptual extension.

Our task is now to compute a schedule, i.e., a set of suitable start times for the jobs. In the literature, there are certain special cases sometimes considered separately:

- Computation of the *technological project time* together with a corresponding schedule. This is the time within which the project can be finished on the assumption that there is an arbitrary number of instances of each resource type available.

- Computation of the *project time by resource shortness* together with a corresponding schedule. This is the project time on the assumption that there is exactly one instance of each resource type available. This is only feasible if no job requires more than one instance of a certain resource type.

7.5.2 First-order Formulation of Project Networks

For the description of how a given project network can be translated into a first-order formula, we start with the special cases discussed above, and then proceed to the general case.

For the computation of the technology project time we identify, as with the dedicated machine model, each job i with a variable t_i representing its starting time. Due to our special case we can ignore the information about the resource requirements. It remains to clarify how to translate the edges into constraints: Consider two jobs i and j and the normal sequence between these two jobs labeled with $\min = \delta$ and $\max = \Delta$. Then we translate the edge into the constraints

$$t_i + p(i) + \delta \leq t_j, \quad t_j \leq t_i + p(i) + \Delta.$$

The second constraint is dropped in the case $\Delta = \infty$. The objective function is usually the end time of the last job. We can, however, also use more sophisticated objective functions as discussed with the dedicated machine model.

In the case of resource shortness we have to add to our first-order formulation above further constraints for managing the resources. Assume first, that there is only one resource type required for each job. Then we can use the same technique as for the dedicated machine model: We state for each pair of jobs requiring the same resource type that they are processed during disjoint time intervals. The situation that a job may require a set of resources can be handled similarly: Instead of considering pairs of jobs that require the same resource type we then consider pairs of jobs with non-disjoint resource requirements.

Finally, we have to discuss how to handle the general case, which includes the existence of multiple instances of certain resource types. To begin with, we restrict our attention to the case that there is only one resource type. In analogy to the discussion in [4] we will use the concept of minimal forbidden sets. A *forbidden set* is a set of jobs that cannot be processed in parallel due to their resource requirements. A forbidden set is *minimal* if there is no proper subset that is forbidden. Bartusch [3] has given algorithms for the efficient computation of the system $\mathcal{M}$ of all minimal forbidden sets for a given project network. From [4] we know that we only have to guarantee that no two jobs appearing together in some $M \in \mathcal{M}$ run in parallel. We thus only have to sequence all minimal sets. After that we can proceed as for the technological project time case. Sequencing of the minimal sets can be described by a first-order formula as follows:

$$\bigwedge_{M \in \mathcal{M}} \neg \exists t \bigwedge_{i \in M} t_i < t < t_i + p(i).$$

Note that here we have for the first time used our option to code constraints by quantified formulas. There is a surprisingly concise quantifier-free equivalent for our constraint, which we are going to use for our formulation. We certainly cannot expect our quantifier elimination to find such elegant solutions. The following lemma provides the key idea for getting rid of the quantifier.

Lemma 7.3. *Let S be a finite set of non-empty open bounded real intervals. Then we have that $I \cap J \neq \emptyset$ for all I, $J \in S$ with $I \neq J$ if and only if $\bigcap_{I \in S} I \neq \emptyset$.*

Proof. Assume that $I \cap J \neq \emptyset$ for all I, $J \in S$ with $I \neq J$. We identify the largest lower boundary $\alpha = \max_{I \in S} \inf I$ and the smallest lower boundary $\beta = \min_{I \in S} \sup I$ among the intervals in S. There are intervals $I_\alpha =]\alpha, \beta^*[\in S$ and $I_\beta =]\alpha^*, \beta[\in S$ with $I_\alpha \cap I_\beta \neq \emptyset$ and thus we have $\alpha < \beta$. Assume for a contradiction that $\frac{\alpha+\beta}{2} \notin \bigcap_{I \in S} I$. Then there is some interval $]\alpha', \beta'[\in S$ such that

$$\frac{\alpha+\beta}{2} \leq \alpha' \quad \text{or} \quad \beta' \leq \frac{\alpha+\beta}{2}.$$

The former case implies

$$\frac{\alpha'+\beta}{2} \leq \frac{\alpha+\beta}{2} \leq \alpha',$$

which implies $\alpha' + \beta \leq 2\alpha'$ and thus we have $\alpha < \beta \leq \alpha'$ a contradiction to our choice of α as the largest lower bound. For the case $\beta' \leq \frac{\alpha+\beta}{2}$ we can analogously derive a contradiction.

The quantified subformula $\exists t \bigwedge_{i \in M} t_i < t < t_i + p(i)$ actually states that

$$\bigcap_{i \in M}]t_i, t_i + p(i)[\neq \emptyset.$$

By our Lemma, we only have to say that all pairs of intervals $]t_i, t_i + p(i)[$ and $]t_j, t_j + p(j)[$ for i, $j \in M$ are non-disjoint. Note that two intervals $]a, b[$ and $]a', b'[$ are non-disjoint if and only if $a < b' \wedge a' < b$. We thus obtain

$$\bigwedge_{M \in \mathcal{M}} \neg \bigwedge_{i,j \in M} t_i < t_j + p(j) \wedge t_j < t_i + p(i).$$

In the case of several distinct resource types we apply our approach separately for each resource type and conjunctively add all obtained conjunctions to our formulation.

We are now going to model and to solve Bartusch's project network example in Figure 7.2, which is taken from [4]. There are 3 instances of resource type 1, 2 instances of type 2, and 1 instance of type 1. The cost function is specified as follows:

$$\begin{gathered}10 + 2C_E + \max(0, C_E - 15) + \frac{5}{C_E - 18} \cdot \max(0, C_E - 18) + \\ 3\max(0, C_{M2} - 12) + 2\max(0, C_6 - 12) + 3\min(0, C_{M2} - 14),\end{gathered}$$

where C_i denotes the end time of a job i.

In a first step we compute the forbidden sets for all three resource types: For resource type 1 one easily sees that $\{2a, 1\}$ and $\{2a, 2b\}$ are the minimal forbidden sets. We can restrict ourselves to the minimal set $\{2a, 1\}$ due to the fact that a parallel processing of the jobs $2a$ and 1 are excluded by the specified temporal dependencies. This minimal set is translated into the subformula

$$t_1 + 3 \leq t_{2a} \vee t_{2a} + 2 \leq t_1.$$

It is easy to see that $\{3, 4a, 5\}$ and $\{5, 6\}$ are the only minimal forbidden sets for resource type 2 and 3, respectively.

Altogether we obtain as the first-order formulation of Bartusch's example the formula

$$\exists t_1 \exists t_{2a} \exists t_{2b} \exists t_3 \exists t_{4a} \exists t_{4b} \exists t_5 \exists t_6 \exists t_7 \exists t_e \exists t_{m1} \exists t_{m2} (\pi \wedge \varrho \wedge \omega),$$

where the precedence constraints π, the resource constraints ϱ, and the description ω of the objective function are given as follows:

$$\begin{aligned}\pi \quad \equiv \quad & 0 \leq t_1 \wedge t_1 + 3 \leq t_3 \wedge t_{m2} \leq t_e \wedge t_6 + 2 \leq 15 \wedge 4 \leq t_{m1} \wedge t_{m2} \leq 14 \wedge \\ & t_1 + 3 \leq t_{4a} \wedge 0 \leq t_{2a} \wedge t_{2a} + 2 = t_{2b} \wedge t_{2b} + 1 \leq t_{4a} \wedge t_{2b} + 1 \leq t_5 \wedge \\ & t_{m1} \leq t_3 \wedge t_3 + 3 \leq t_6 \wedge t_3 + 3 \leq t_{m2} \wedge t_{m1} \leq t_{4a} \wedge t_{4a} + 5 = t_{4b} \wedge \\ & t_{4b} + 2 \leq t_6 \wedge t_{4b} + 2 \leq t_{m2} \wedge t_{4b} + 2 \leq t_7 \wedge t_{m1} \leq t_5 \wedge t_5 + 7 \leq t_{m2} \wedge \\ & t_5 + 7 \leq t_7 \wedge t_6 + 2 \leq t_e \wedge t_7 + 4 \leq t_e \\ \varrho \quad \equiv \quad & (t_1 + 3 \leq t_{2a} \vee t_{2a} + 2 \leq t_1) \wedge (t_5 + 7 \leq t_6 \vee t_6 + 2 \leq t_5) \wedge \\ & \neg(t_{4a} < t_3 + 3 \wedge t_3 < t_{4a} + 5 \wedge t_5 < t_3 + 3 \wedge t_3 < t_5 + 7 \wedge \\ & t_5 < t_{4a} + 5 \wedge t_{4a} < t_5 + 7)\end{aligned}$$

$$\begin{aligned}\omega \equiv\ & \exists m_{e1}\exists m_{e2}\exists m_{m2}\exists n_{m2}\exists m_6 \bigl(\\ & \bigl((m_{e1} = t_e - 15 \land t_e - 15 \geq 0) \lor (m_{e1} = 0 \land t_e - 15 \leq 0)\bigr) \land \\ & \bigl((m_{e2} = 1 \land t_e - 18 \geq 1) \lor (m_{e2} = 0 \land t_e - 18 \leq 0)\bigr) \land \\ & \bigl((m_{m2} = t_{m2} - 12 \land t_{m2} - 12 \geq 0) \lor (m_{m2} = 0 \land t_{m2} - 12 \leq 0)\bigr) \land \\ & \bigl((n_{m2} = t_{m2} - 14 \land t_{m2} - 14 \leq 0) \lor (n_{m2} = 0 \land t_{m2} - 14 \geq 0)\bigr) \land \\ & \bigl((m_6 = t_6 - 12 + 2 \land t_6 - 12 + 2 \geq 0) \lor \\ & (m_6 = 0 \land t_6 - 12 + 2 \leq 0)\bigr) \land \\ & k = 10 + 2t_e + m_{e1} + 5m_{e2} + 3m_{m2} + 2m_6 + 3n_{m2}\bigr).\end{aligned}$$

This is input into our scheduling problem solver which computes an optimal schedule within 43 s. Our schedule differs only slightly from that given by Bartusch [4].

7.6 Railway Delay Management

In this section we give an example of how to solve a very complex scheduling problem by an approach analogous to that for the dedicated machine model and the project networks:

1. Generation of a first-order formula describing the problem,
2. extended quantifier elimination,
3. interpretation of the extended quantifier elimination result in terms of scheduling.

Our new example does not fit into any class of scheduling problems described by common machine models. The previous sections have shown that there is a certain correspondence between the syntactic form of our first-order description on one hand and scheduling models on the other hand. In view of these facts already a successful first-order formulation has the status of a theoretical result by giving a precise formal description of the problem.

We consider a railway system. Our general assumptions about such a system are the following:

- Trains connect towns via fixed tracks.
- Trains run according to a fixed schedule.
- The set of all station pairs that are connected directly, i.e., offer connections without changing trains, is relatively small compared to the set of all station pairs.

Passengers travel from one station to another station using a connection given by a sequence of trains with the obvious restrictions. The problem we are going to solve is the following: Trains may arrive at a station later than the scheduled arrival time. We call such trains delayed. Besides the problem that some passengers in these trains arrive a little late at their final destination there is a much more annoying problem: Other passengers miss their connections. They have to choose alternative connections that in general arrive at the destination much later than the original ones. This usually causes a serious delay for this group of passengers.

The railway company can support our second group of passengers by delaying their connection trains. This is perfect for those who reach their final destination with the connection train. It is also good for those who have to change trains once more but do not run into trouble because this intentional delay is usually not too long. For the remaining group of our passengers it simply shifts the problem to the next station. On the other hand there are passengers in the intentionally delayed train that now run into trouble. Their trouble is not caused by some unexpected accidental delay but a decision of the railway company. This usually makes them angry. From an objective point of view any such decision is critical because it causes two delayed trains running in the system.

However we proceed, some passengers will have to choose alternative connections. They can be supported by the railway company by providing them as early as possible with detailed information on their complete new connection scheme.

It is clear that not all problems of delayed trains can be solved without increasing the travel time of some passengers. We are looking for a solution that minimizes the sum of the relative delay of all passengers. Note, however, that simply using this sum as an objective function is not sufficient. This would cause, e.g., some trains without any passenger aboard to be delayed without any reason. Another objective function to be minimized is thus given by the sum of all additional waiting times introduced for trains. We see that we are faced with a *multi-objective* scheduling problem. Clearly, the sum of relative delays of passengers is our main objective and the sum of additional waiting times of trains plays a subsidiary role.

The plan is as follows: In Section 7.6.1 we define the interface of our scheduler. That is, we summarize the available input information, and we specify the required output information. In Section 7.6.2 we discuss how to generate the first-order formula input from the input information. In Section 7.6.3 we finally discuss how to translate the elimination result obtained from the input formula into the required output information.

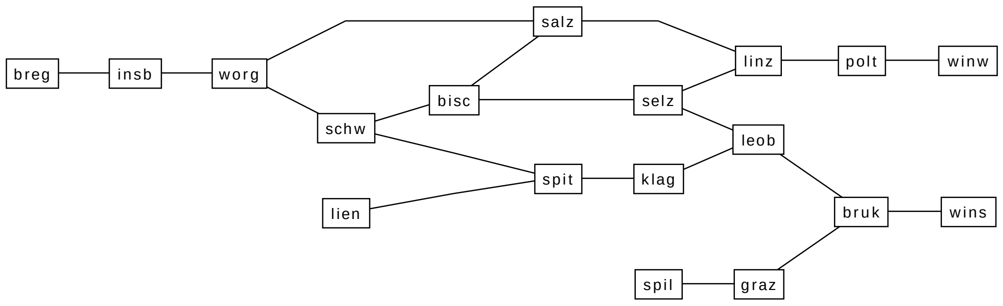


Figure 7.5: The Austrian intercity net

7.6.1 The Interface

We first summarize all data available for the generation of our input formula. We give some illustrating examples for such data taken from our experimentation environment, which models the Austrian intercity (IC) net. This intercity net consists of qualified trains connecting major cities. Figure 7.5 shows all stations together with the existing direct connections. This network graph provides an interesting background information but is in this isolated form not part of our input. Instead it is implicitly contained in the schedule of all trains.

By *train* we denote a train starting its journey at some station and reaching after various intermediate stops finally some destination station. When it then returns to its origin we consider it a different train. The *schedule of all trains* is a list containing for one day the relevant information for each train. This relevant information is the list of stations served by this train together with the arrival time and the departure time for each station. Figure 7.6 displays a clipping of the Austrian IC schedule. The entire schedule contains 161 trains.

For each train t and each station s there is a minimal waiting time $\mu(t,s)$ specified. That is, whenever t reaches s it must stop there for at least time $\mu(t,s)$. Our schedule, of course, fulfills this requirement. The idea for this constraint is to prevent delayed trains from leaving stations to soon.

Our schedule says what *should* happen. Of course we need in addition the information what *has* actually happened: For each train we have available the actual arrival and departure times in all stations that have been already served. We think of this information as functions mapping trains and stations to times. In practice these functions can be realized in various different

```
{

 {ic190,{
  {wins,start,1255},
  {bruc,1457,1501},
  {leob,1512,1514},
  {klag,1720,1722},
  {spit,1824,1827},
  {schw,1944,1946},
  {bisc,2000,2003},
  {salz,2046,end}
 }},

 {ic505,{
  {linz,anfang,0957},
  {selz,1135,ende}
 }},

 {ic640,{
  {winw,anfang,0728},
  {linz,0926,0929},
  {salz,1050,1105},
  {bisc,1150,ende}
 }},

 ...
}
```

Figure 7.6: Part of the Austrian IC schedule

```
{1,
  {winw,linz,0926,0728},
  {linz,selz,0957,1135}
}
```

Figure 7.7: A connection from Wien (West) to Selzthal

ways ranging from requests to databases that are updated automatically to telephone calls from single stations.

Besides the actual arrival and departure times we have a *list of expected delays*. A *delay* consists of a train, a pair of adjacent stations, and the number of minutes of an expected additional delay between these stations. One reason for such delays can be that the train has to pass this track segment slowly for technical reasons. Another possible reason is that the train has to run around the segment because some delayed train with a higher priority passes it. Of course, only delays after stations for which we know the actual arrival time are of interest.

Another obvious piece of information, which should, however, be explicitly mentioned is the current time.

Finally we turn to the passengers. Since there is no obligatory reservation system in the Austrian IC net, we cannot expect to know in detail how many passengers with which destination are using which train. Note that when considering cargo transportation, with pieces of cargo corresponding to single passengers, we usually have this full information at hand. For our situation we thus approximate this information by defining the notion of a traveling stream. A *traveling stream* is a considerable number of people starting at time t_0 in station s_0 and arriving at time t_1 in station s_1. For this they usually have to change trains. The entire stream uses the same trains. Formally a stream uses a connection.

A *connection* is a sequence of *steps* subject to certain constraints. A step consists of an origin station with the scheduled departure time and a destination station with the scheduled arrival time. Figure 7.7 shows a sample connection based on our schedule clipping in Figure 7.6. There we also see that each connection is uniquely labeled by some integer. This label is used for algorithmic purposes. The constraints mentioned for a connection require that destination station and origin station of neighboring steps match. There is, however, *no* constraint on the corresponding arrival and departure times. For scheduled connections one would obviously require that the train of step n arrives in station s before the departure time of the train in step $n+1$ there. We will, however, not only consider scheduled connections but also connections involving delayed trains. Such trains can possibly be used as connection trains where this is impossible by schedule.

Assume now that there is a traveling stream with a delayed train involved such that a connection train cannot be reached. We then have to consider alternative connections for this stream. It is not hard to see that there are essentially only finitely many alternative connections and that all these alternative connections can be computed from the information we have introduced so far. This produces, of course, an enormous overhead. We instead introduce one final input information. We assume to have available a set of reasonable alternative connections in such a situation. Note that as soon as our stream has left one of its initial or intermediate stations, we are interested only in alternatives for the remaining parts of the journey. An alternative connection for a given stream is a connection starting at any of the stations of the stream and ending at its destination station.

Let us summarize all the input information we have specified above:

1. The schedule of all trains,
2. the minimal waiting time for each train in each relevant station,
3. actual arrival time, actual departure time for all trains and stations,
4. the expected delays for all trains,
5. the current time,
6. all traveling streams,
7. a set of reasonable alternative connections for each stream.

Our algorithm will be started on this input whenever there is the chance that the situation in our railway system has changed in such a way that we have to delay some trains or that we have to communicate a new schedule to some traveling stream. Note that these decisions are independent: it is possible that some stream has to switch to an alternative connection train t_1 in some station although the scheduled train t_0 can be reached and starts in time. One reason might be that there is a considerable expected delay for t_0.

Let us now turn to the output of our delay management system. The output splits into two parts. First, we obtain for each of the input streams one possibly alternative connection. Using the list of output connections guarantees that the sum of the relative delays of the streams is minimized, and, moreover, among all these minimal delays for the streams we have minimized the sum of additional waiting times of all trains.

Second, we obtain additional waiting times for trains and stations wherever this is necessary to make the alternative connections work.

7.6.2 First-order Formulation of Delay Management

As already discussed in the introduction to this section applying our algorithm means generating a first-order formula on the basis of the input data specified above. To this first-order formula we then apply quantifier elimination and interpret the results as the solution to our problem. For constructing the formula we introduce the following variables:

1. For each stream S we introduce a variable z_S. It is intended to be the estimated remaining travel time of the stream. This period of time starts with the current time specified in the input and ends with the arrival time of the stream at their final destination station.

2. For each train t and each station s served by t we introduce the variable $x_{t,s}$. It is intended to be the *total waiting time* of t in s. Total waiting times are obtained as sums of scheduled waiting times and additional waiting times. Recall from the previous section that the scheduled waiting time can be obtained from the schedule of all trains, and that the additional waiting time is one of the values we want to compute.

3. For each stream S we introduce a variable c_S. In the elimination result c_S will describe the alternative connection for S by being assigned its unique label.

4. The variable z is as usual the tag variable for introducing the main objective function into the first-order formulation. Recall that this objective function is the sum of all relative delays of the streams possibly weighted by the size or the importance of the single streams.

5. The variable z_Σ is the tag variable for minimizing our subsidiary objective, i.e., the sum of all additional waiting times of all trains. To be precise we actually use the sum $\sum_{t,s} \max(0, x_{t,s})$ of all total waiting times, which is equivalent.

Before constructing the first-order input formula, we have to introduce various concepts and notations: T is the current time contained in the input. We denote by $\mathcal{S}$ the set of all streams, which is contained in the input. For each stream $S \in \mathcal{S}$ we denote by $\operatorname{acon}(S)$ the set of alternative connections in the input.

By $\mathcal{T}$ we denote the set of all trains, which is itself not part of the input but can immediately be derived from the schedule of all trains. The same holds for the set $\operatorname{st}(t)$ of all stations served by train $t \in \mathcal{T}$, and for the scheduled time of departure $\operatorname{std}(t, s)$ of train $t \in \mathcal{T}$ in station $s \in \operatorname{st}(t)$. Recall that we denote by $\mu(t,s)$ the minimal waiting time of train t in station s, which is explicitly given in the input.

We now turn to some more sophisticated notions. These notions can also be derived from the input data but much less straightforwardly than those introduced so far. The first new notion is the estimated time of arrival $\mathrm{eta}(t,s)$ of train t in station s. Syntactically $\mathrm{eta}(t,s)$ is a term. If t has already reached s, then $\mathrm{eta}(t,s)$ equals the actual time of arrival for t in s. This is part of the input. Otherwise there are three pieces of information participating in its generation:

1. The actual departure time of train t in the last station s_0 it is known to have left. This is part of the input.
2. The expected delays for train t behind station s_0. This is also part of the input.
3. Additional waiting times in stations s' between s_0 and s. These waiting times can be derived from the variables $x_{t,s'}$, which are the corresponding total waiting times.

There is a very similar notion of the estimated time of departure $\mathrm{etd}(t,s)$ of train t in station s. We use this for defining $\mathrm{etd}(c) = \mathrm{etd}(t_0, s_0)$, where t_0 is the first train of the connection c starting in s_0. In the same way we extend $\mathrm{std}(c) = \mathrm{std}(t_0, s_0)$.

The next notion is the *compatibility relation* $\Gamma(S,c)$ for $S \in \mathcal{S}$ and $c \in \mathrm{acon}(S)$. Recall that one essential part of a stream is a connection. We can thus view streams as connections. In this sense s is compatible with c if c is feasible and the travelers have the opportunity to change from s to c. A connection is feasible if all future changes of trains will be possible. Note that again the $x_{t,s}$ will contribute to the relation, which is consequently given as a formula.

Our next notion is the term $\tau(c)$ for a connection c describing the overall travel time for this connection. This again depends on the $x_{t,s}$. Finally, $\mathrm{id}(c)$ extracts the unique integer label from connection c.

The notions introduced so far allow us to specify the constraints, in contrast to the encodings of the objective functions, of our first-order formulas:

$$\begin{aligned}\gamma \;\equiv\; \Big(\bigwedge_{S\in\mathcal{S}} \bigvee_{c\in\mathrm{acon}(S)} & \Gamma(S,c) \wedge c_S = \mathrm{id}(c) \wedge \big((T \le \mathrm{etd}(S) \wedge z_S = \tau(c)) \vee \\ & (\mathrm{etd}(S) \le T \wedge z_S = \tau(c) + \mathrm{etd}(c) - T)\big)\Big) \wedge \\ \Big(\bigwedge_{t\in\mathcal{T}} \bigwedge_{s\in\mathrm{st}(t)} & x_{t,s} \ge \max\big(\mathrm{std}(t,s) - \mathrm{eta}(t,s), \mu(t,s)\big)\Big).\end{aligned}$$

Note that $\max(a,b) \leq c$ can be written in our language of ordered rings as $a \leq c \wedge b \leq c$. Combinations of maximums terms with the other relations can be expanded similarly.

For the description of the objective functions we have to introduce one more concept: The relative delay $\delta(S)$ of a stream $S \in \mathcal{S}$ given by

$$\delta(S) = \max\left(1, \frac{\mathrm{ett}(S)}{\mathrm{stt}(S)}\right).$$

The estimated travel time $\mathrm{ett}(S)$ of the stream S is a term containing z_S. The scheduled travel time $\mathrm{stt}(S)$ is, in contrast, a constant number and may thus appear as a denominator. We finally explain why the maximum is introduced here: There will in general be streams arriving earlier than scheduled. Either their last train arrives a little early, or some delay allows the passengers to reach a faster connection than possible by schedule. We filter out these cases for our objective function, because we want to avoid delaying streams for such accelerations of other streams.

The part of our formula describing the objective functions reads as follows:

$$\omega \equiv \sum_{S \in \mathcal{S}} \delta(S) \leq z \wedge \sum_{t \in \mathcal{T}} \sum_{s \in \mathrm{st}(t)} \max(0, x_{t,s}) \leq z_\Sigma.$$

In the formulation of the input for extended quantifier elimination both tag variables z and z_Σ remain unquantified:

$$\varrho \equiv \cdots \exists z_S \cdots \exists x_{t,s} \cdots \exists c_S \cdots (\gamma \wedge \omega).$$

We finally wish to mention that ϱ is a linear formula.

7.6.3 Computing the Solution

Recall from our previous scheduling models and from optimization the role of the tag variable z. We apply extended quantifier elimination to the first-order description of our problem obtaining a set of conditions on z each with the corresponding sample solution. In the linear case these conditions will always have the nice form $z \geq \min$, where min is the minimum of the objective function.

Our multi-objective formulation ϱ above unfortunately contains two tag variables z and z_Σ, where z is the main one. They both have remained unquantified. Consequently, the result of extended quantifier elimination will not provide the optimal solution for z in such an explicit form.

We proceed as follows: From the extended quantifier elimination result we compute a quantifier-free equivalent ϱ' for ϱ by disjunctively combining all conditions in the parameters z and z_Σ. Then we apply regular quantifier elimination to $\exists z_\Sigma \varrho'$. The quantifier-free equivalent for this will explicitly provide the rational optimal value for z. This optimal value is substituted for z into ϱ' explicitly yielding the rational optimal value for z_Σ. Finally, both rational optimal values are substituted into the extended quantifier elimination result for ϱ.

The sample solutions for the $x_{t,s}$ now provide the total waiting times for all trains $t \in \mathcal{T}$ and stations $s \in \mathrm{st}(t)$. From this we can compute the additional waiting times. From the sample solutions c_S we obtain the unique identifier for a connection c that can serve as an optimal alternative connection for the passengers traveling with stream S.

7.7 Implementation

In the previous section we have discussed the construction scheme of the first-order formulation of our delay management system. Though the formula structure and the ideas of the single constraints can be easily described in terms of some function, the actual generation of the formula is much more difficult. For the purpose of the generation we have implemented a module for the generation of the first-order formula. This implementation contains about 1000 lines of code. It includes code for functions like std extracting information from the input, code for functions like eta computing terms by combining several input data, and procedures for automatically resolving the maximum terms that appear during the formula generation.

In addition we have implemented a front end for our system. This front end provides functions for adding and removing expected delays, a function for setting the "current" time, and a function to compute the solution. This function generates the formula, performs all necessary quantifier elimination steps, extracts the used data from the sample points obtained by the extended quantifier elimination, and converts the raw output data into precise information for the railway system. Implementing this front end we have shown that the underlying mathematical ideas can be completely hidden from the user.

7.8 Conclusions

In this chapter, we have examined the application of our extended quantifier elimination strategies to the area of scheduling in a very general sense. We have naturally started in Section 7.2 with the description of the dedicated machine model, which plays a very prominent role in this research area. In Section 7.3, it has turned out that we can straightforwardly describe scheduling problems of this model by first-order formulas, and then solve them by extended quantifier elimination. The timings for a computation example suggest, however, that we cannot compete with dedicated special-purpose algorithms.

This has motivated the new approach in Section 7.4 to tune the quantifier elimination on the basis of the special type of input formulas obtained for scheduling problems. This constitutes an entirely new class of tuning ideas, and is thus the link to the remainder of this thesis. We have discussed and analyzed the effects of various kinds of such optimizations. The results are very convincing, but still we cannot compete with established methods at present.

This gives rise to the idea to make use of the extreme flexibility of our approach considering much more general scheduling problems. In particular, we attack new types of problems, which have not been discussed in the literature so far.

As a first step on this way, we have considered in Section 7.5 another scheduling model common in the literature: project networks, which are more general then the dedicated machine model. The formulation of project networks has given a good impression of various techniques for first-order formulation. Again, we can solve non-trivial problems within reasonable time and space, but again there are dedicated algorithms superior to our approach available.

In Section 7.6, we finally have turned to a very general problem that, to our knowledge, does not fit into any known scheduling model: We have performed delay management for railway connections. Here, the first-order formulation is so complex that we have to provide dedicated algorithms for its computation. Similarly the automatic interpretation of the results requires some minor algorithmic ideas. We have furthermore used this example to demonstrate how to realize multi-objective optimization. At the current state of the research it is not yet possible to solve non-academic problems within a reasonable time, and this is certainly a research project on its own. Note, however, that already the formal specification of the problem by means of our first-order input formula is a theoretical result in its own right. The most important result of this section for our purposes, however, is the successful formulation of this very general problem.

Chapter 8

The REDLOG Programming Environment

In Section 2.4.4 we have mentioned that much effort has been spent by the author and others to provide highly optimized and reliable well-designed and well-documented implementations of quantifier elimination by virtual substitution and related algorithms. The latest product resulting from this effort is the Version 2.0 of the computer logic system REDLOG, which is part of the computer algebra system REDUCE. In this chapter we conclude our thesis with an outline of REDLOG with which we have performed all practical computations mentioned throughout our thesis. Our description follows a publication on REDLOG [28] by the author of this thesis *at al.*

8.1 Introduction

REDLOG stands for REDuce LOGic system. It provides an extension of the computer algebra system (CAS) REDUCE to a *computer logic system* (CLS) implementing symbolic algorithms on first-order formulas wrt. temporarily fixed first-order languages and theories. As underlying theories currently available there are *ordered fields*, *discretely valued fields*, and *algebraic closed fields.* The system is designed to be easily extensible by further theories. The REDLOG design allows to implement algorithms generically for different theories. The present algorithms have already been implemented generically wherever this appeared reasonable. In fact, they share the largest part of their code.

The focus of the system is on simplification as described in Chapter 2, effective quantifier elimination by virtual substitution, and corresponding applications. The implemented algorithms include the following:

- The majority of the simplification algorithms presented in Chapter 2.
- Quantifier elimination by virtual substitution. Implementations of the ideas of Chapter 4 through 6 currently have experimental status. They are scheduled to be integrated into REDLOG 3.0.
- Generic quantifier elimination: An efficient variant of the quantifier elimination procedure developed for geometric theorem proving: Non-degeneracy conditions are detected automatically [31]. This approach has also turned out valuable for physical network analysis.
- Extended quantifier elimination as described in 3.2.2.
- Linear *optimization* using quantifier elimination techniques [70].
- Numerous valuable tools for constructing, decomposing, and analyzing formulas.

REDLOG is designed as a *general-purpose* computer logic system. As such, its scope goes beyond that of SAC/ALDES [20], which has been very successfully used for the implementation of quantifier elimination by (partial) cylindrical algebraic decomposition [21, 36]. In contrast to *constraint logic programming* systems [22], the algebraic component is not only used for supporting the logical engine but the largest part of the logical algorithms is defined and implemented in terms of algebraic algorithms.

REDLOG has been applied successfully for solving non-academic problems, mainly for the simulation and error-diagnosis of physical networks [73].

Applications inside the scientific community include the following:

- Control theory [1].
- Stability analysis for PDE's [37].
- Geometric reasoning [31].
- Disjunctive parametric scheduling.
- Non-convex parametric linear and quadratic optimization [70], transportation problems [51].
- Real implicitization of algebraic surfaces.
- Computation of comprehensive Gröbner bases.
- Implementation of *guarded expressions* for coping with degenerate cases in the evaluation of algebraic expressions [24, 27].

8.2 Application Examples

8.2.1 Simplification

We start with an example using the REDLOG implementation of the Gröbner basis simplification of Section 2.4.3. [29].

```
REDUCE 3.6, 15-Jul-95, patched to 22 Dec 96 ...
%
1: load rl;

2: phi := x*y+1<>0 or x=z or y*z+1<>0;

phi := x*y + 1 <> 0 or x - z = 0 or y*z + 1 <> 0

3: rlgsn phi;

true
```

This computation requires 17 ms of CPU time on a SUN SPARC 4. Let us once more discuss by example how this simplification works: The formula `phi` can be rewritten as

$$xy + 1 = 0 \wedge yz + 1 = 0 \longrightarrow x - z = 0.$$

It is not hard to see that it can be equivalently transformed by reducing $x - z$ modulo the Gröbner basis $\{yz + 1, x - z\}$ of $\{xy + 1, yz + 1\}$.

8.2.2 Geometry Proving

For the situation in Figure 8.1, we wish to prove that $\angle ACB = \angle AMB/2$. Our algebraic translation of this problem is described in [31].

```
REDUCE 3.6, 15-Jul-95, patched to 22 Dec 96 ...

1: load rl;

2: geo := all({b,c,tan1,tan2,tan0},c**2=a**2+b**2 and
2: c**2=x0**2+(y0-b)**2 and tan1*y0=(a+x0) and
2: tan2*y0=(a-x0) and tan0*(1-tan1*tan2)=tan1+tan2
2: impl tan0*b=a)$

3: rlgqe geo;

{{y0 <> 0},true}
```

This generic quantifier elimination requires 119 ms of CPU time on a SUN SPARC 4. Besides the elimination result "true" we obtain a non-degeneracy condition $y_0 \neq 0$ stating that AMB is a proper triangle.

8.2.3 Periodicity

Consider the infinite sequence of real numbers defined by $x_{i+2} = |x_{i+1}| - x_i$ where x_1 and x_2 are arbitrary numbers. Our aim is to show that this sequence is always periodic and that the period is 9. Colmerauer has proved this automatically using Prolog III [22].

Our approach requires much less human intelligence than writing a logic program. The assertion can be encoded straightforwardly into a first-order formula:

$$\forall \underline{x}\Big(\big(\bigwedge_{i=3}^{11} x_i = |x_{i-1}| - x_{i-2}\big) \longrightarrow (x_1 = x_{10} \wedge x_2 = x_{11})\Big).$$

In practice, we have to encode the absolute value into a case distinction. The equation $x_i = |x_{i-1}| - x_{i-2}$ amounts to

$$(x_{i-1} \geq 0 \wedge x_i = x_{i-1} - x_{i-2}) \vee (x_{i-1} < 0 \wedge x_i = -x_{i-1} - x_{i-2}).$$

```
REDUCE 3.6, 15-Jul-95, patched to 22 Dec 96 ...

1: load rl;

2: p9 := rlall(
2: (for i:=3:11 mkand
2:    (mkid(x,i-1)>=0 and
2:       mkid(x,i)=mkid(x,i-1)-mkid(x,i-2) or
2:    mkid(x,i-1)<0 and
2:       mkid(x,i)=-mkid(x,i-1)-mkid(x,i-2)))
2: impl x1=x10 and x2=x11)$

3: rlqe p9;

true
```

REDLOG eliminates the 11 universal quantifiers in 9.2 s of CPU time on a SUN SPARC 4.

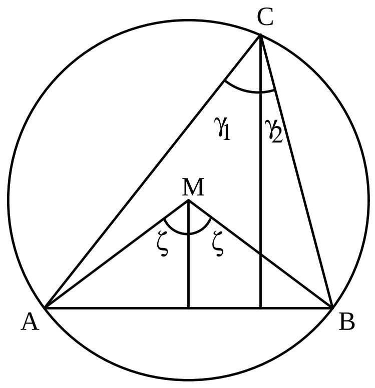


Figure 8.1: Geometry proving: The angle at circumference is half the angle at center.

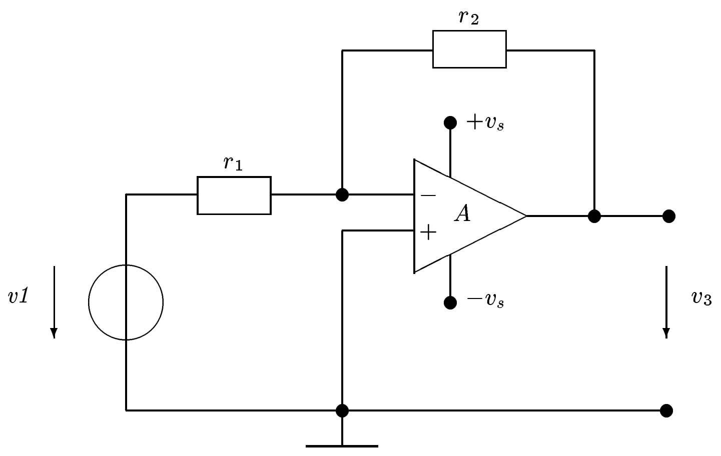


Figure 8.2: An inverting operation amplifier circuit

8.2.4 Network Analysis

This example is taken from [65]. For the operation amplifier circuit shown in Figure 8.2, we want to determine the output voltage $\mathrm{V_{OUT}} = \mathrm{v1}$ as a function of the input voltage $\mathrm{V_{IN}} = \mathrm{v\_3}$. The algebraic formulation ω of the circuit is the conjunction over the following equations:

$$\begin{aligned}
\mathrm{v\_1} &= \mathrm{v1}\\
\mathrm{v\_2} &= -\mathrm{v\_pm\_op1}\\
\mathrm{v\_3} &= \mathrm{v\_og\_op1}\\
\mathrm{v\_1} + \mathrm{i\_v0}\cdot\mathrm{r1} &= \mathrm{v\_2}\\
\mathrm{v\_2}\cdot\mathrm{r1} + \mathrm{v\_2}\cdot\mathrm{r2} - \mathrm{v\_3}\cdot\mathrm{r1} - \mathrm{v\_1}\cdot\mathrm{r2} &= \mathrm{i\_pm\_op1}\cdot\mathrm{r1}\cdot\mathrm{r2}\\
\mathrm{v\_3} + \mathrm{i\_og\_op1}\cdot\mathrm{r2} &= \mathrm{v\_2}\\
\mathrm{v\_og\_op1} - \mathrm{v\_pm\_op1}\cdot\mathrm{x\_op1}^2 &= 0\\
\mathrm{vs}^2\cdot\mathrm{x\_op1}^2 + \mathrm{a}\cdot\mathrm{v\_og\_op1}^2 &= \mathrm{a}\cdot\mathrm{vs}^2\\
\mathrm{i\_pm\_op1} &= 0.
\end{aligned}$$

The variables to be (existentially) eliminated are

$$V := \{\mathrm{i\_og\_op1}, \mathrm{v\_2}, \mathrm{i\_pm\_op1}, \mathrm{v\_1}, \mathrm{i\_v0}, \mathrm{v\_pm\_op1}, \mathrm{v\_og\_op1}, \mathrm{x\_op1}\}$$

We apply generic quantifier elimination. The result obtained for $\exists V(\omega)$ after 323 ms is:

$$\mathrm{a}\cdot\mathrm{r1}\cdot\mathrm{v\_3}^3 - \mathrm{a}\cdot\mathrm{r1}\cdot\mathrm{v\_3}\cdot\mathrm{vs}^2 + \mathrm{a}\cdot\mathrm{r2}\cdot\mathrm{v1}\cdot\mathrm{v\_3}^2 - \mathrm{a}\cdot\mathrm{r2}\cdot\mathrm{v1}\cdot\mathrm{vs}^2 - \mathrm{r1}\cdot \mathrm{v\_3}\cdot\mathrm{vs}^2 - \mathrm{r2}\cdot\mathrm{v\_3}\cdot\mathrm{vs}^2 = 0 \land \mathrm{a}\cdot\mathrm{v\_3}^2 - \mathrm{a}\cdot\mathrm{vs}^2 \leq 0 \land \mathrm{vs} \neq 0$$

valid under the following conditions:

$$\mathrm{a} \neq 0, \quad \mathrm{r1} \neq 0, \quad \mathrm{r2} \neq 0, \mathrm{v}_3 + \mathrm{vs} \neq 0, \quad \mathrm{v}_3 - \mathrm{vs} \neq 0, \quad \mathrm{v}_3 \neq 0.$$

None of the conditions is a problem: a is the amplification factor, r1 and r2 are resistors, the absolute value of the output voltage v_3 can certainly never get equal to the supply power vs.

8.2.5 Parallelization

The quantifier elimination based linear optimization code of REDLOG has been parallelized on a CRAY YMP4/T3D using the PVM version of PSL based REDUCE. There is a switch `rlparallel` for actually using the parallel code on such a machine.

Table 8.1 summarizes some timings obtained with small standard optimization benchmarks. We obtain a speedup factor of about 3 with 8 processors.

Table 8.1: Sequential vs. parallel (8 processors) timings for linear optimization benchmarks from the ZIB NETLIB library.

problem	time seq.	time par.	factor
afiro	333 s	95 s	3.5
sc50a	>900 s	272 s	>3.3
sc50b	70 s	25 s	2.8

8.3 From Computer Algebra to Computer Logic

When we originally started the implementation, our aim was to make the quantifier elimination procedures available to others for solving practical problems in physics and engineering [37]. The decision for taking an existing computer algebra system like REDUCE as basis has the following advantages compared to a completely new implementation:

- CLS user interfaces require no concepts that go beyond that of CAS interfaces. Hence there is no reason for spending time designing and implementing yet another interface. In addition, we expect the large community of REDUCE users to be quickly familiar with REDLOG.

- The underlying system provides a reliable well-tested implementation of polynomials, which can serve as first-order terms in many languages. In addition, there is a large library of up-to-date algebraic algorithms available.

- There is no need for portability considerations. The system will simply run with all architectures and operation systems to which REDUCE is ported.

On the other hand, the underlying CAS itself benefits from the implementation first-order formulas and corresponding algorithms. Consider, e.g., the implementation of *guarded expressions*. The need for guarded expressions arises naturally with the implementation of parametric algorithms, such as symbolic integration, computation of Gröbner systems [69], or parametric optimization. For details on guarded expressions cf. [24, 27].

Another possible application is with the widespread *solve* operator for solving systems of equations and, possibly, inequalities. The solutions to such systems can be conjunctions as for $x^2 < 25$, disjunctions as for $x^2 > 25$, or more complicated formulas. The solutions are typically given as nested lists encoding DNF's. Here, the use of formulas would be more natural. At this point, one should mention that MATHEMATICA actually offers the option for printing the result as formula. Except for distributive expansion, there are, however, no algorithms operating on these formulas [74].

8.4 A User's View on the System

This section focuses on the usage of REDLOG, mainly from the algebraic mode (AM) of REDUCE. The last subsection sketches the symbolic mode interface. After loading REDLOG into REDUCE as a package, a *context* has to be selected.

8.4.1 Contexts

A context determines a language and a theory in the sense of first-order logic. These selections are not independent from each other. The language selection is weak in the following sense: A context does not specify which predicate symbols are allowed or prohibited. The algorithms associated with the context, however, know certain predicates. We hence speak of *known* and *unknown* predicates. Some algorithms can handle unknown predicates straightforwardly. Simplification, for instance, simply leaves unknown predicates unchanged. This behavior is quite similar to that of algebraic REDUCE operators for which no rules are known. Quantifier elimination, in contrast, would exit with an error. Schemes allowing the user to determine for unknown predicates how to behave within certain REDLOG algorithms are under consideration.

Each context is encoded into a *context identifier*, for example `ofsf`, which stands for *ordered fields* (with standard form term representation). The name `ofsf` is a relic from early versions which restricted the quantifier elimination to the linear case. In its current state the context actually implements real closed fields. Certain contexts are parameterized. When selecting, e.g., `dvfsf` ("discretely valued field standard form") one has to pass the characteristic of the residue class field wrt. the valuation. All following computations are performed wrt. the selected context until a different decision is made.

When the context is changed, formulas produced in the old context can become invalid, but they need not. Certain formulas of the old context may still be meaningful in the new context. Consider for instance a formula produced in an ordered field context: If it happens to contain only variables as terms, it can be reused in an ordered set context. The same applies to formulas that can be straightforwardly rewritten in such a form as, e.g.,

$a - b < 0$, which would automatically be converted into $a < b$ with every access.

After fixing a context, the REDUCE functionality is extended in two ways:

1. In addition to the built-in data types such as rational functions or matrices, one can input first-order formulas.

2. There are new procedures available that apply to first-order formulas.

In the sequel, we assume that the context `ofsf` is selected, which knows the binary predicates $=$, $\neq$, $\leq$, $\geq$, $<$, and $>$.

8.4.2 Expressions

We have extended the look-and-feel of REDUCE to first-order formulas. Invalid expressions are detected, and appropriate error messages are given.

Expression Format and Input

The format for the truth values, quantifiers, and propositional connectives is specified uniformly for all contexts. Besides the reserved identifiers `true` and `false`, there are the following operators: a unary `not`, binary infix `impl`, `repl`, `equiv`, and n-ary infix `and`, `or`. Binary prefix operators `ex` and `all` serve as quantifiers. Their first argument is a variable, and their second argument is a formula.

In general, all atomic formulas are constructed with operators that are considered as predicates. Here again infix operators are possible. What is left to the context is determining which predicates are known and what the terms are. Furthermore, a context can impose some extra restrictions on the form of the atomic formulas. Consider for instance `ofsf`: The known predicates given at the end of the last subsection can all be written infix. Terms are polynomials over the integers. As an additional restriction, all right hand sides of the predicates must be zero.

The handling of the input is much more liberal than the specification of valid expressions. For the easy input of large systematic conjunctions and disjunctions the `for`-loops have been extended by *actions* `mkand` and `mkor`, in analogy to `sum` or `product`. With the quantifiers `ex` and `all`, the first argument may be a list of variables. In `ofsf` the input may contain rational coefficients and non-zero right hand sides. In all these cases the input is converted to the right expression format immediately.

The `ofsf` context further allows the input of *chains* such as `a<>b<c>d=f`, which is turned into `a-b<>0 and b-c<0 and c-d>0 and d-f=0`. Here, only adjacent terms are considered to be related.

Simplification vs. Evaluation

Polynomials entered into REDUCE are automatically converted into some canonical form, say into distributive polynomials wrt. some term ordering. Canonical means that expressions that are mathematically *equal* are converted into syntactically equal forms. We refer to this conversion as *evaluation*.

The natural extension of evaluation to first-order formulas would be converting *equivalent* formulas into syntactically equal forms. Generally, this is impossible since in non-recursive structures there is no algorithm converting sentences that hold into `true`. Even in most recursive structures it is by no means obvious, how to obtain suitable canonical forms for open formulas. Note that such normal forms must be user-friendly and fast to compute.

Instead of evaluation, we use the weaker concept of *simplification* [29]. This means, we replace formulas by equivalents that are more user-friendly though not canonical. Automatic simplification can be toggled by a REDUCE switch.

Interface Problems

Similar to other computer algebra systems, in REDUCE, interpreter variables are identified with the transcendental elements occurring in rational functions. When introducing first-order formulas, such an identification leads to problems.

Firstly, for many contexts the first-order terms will be implemented as rational functions or some suitable subset. One would certainly like to identify the kernels occurring inside these terms with the interface variables. If such a kernel is quantified, any non-kernel assignment to it violates the well-formedness of the respective formula. The problem of expressions becoming ill-formed due to subsequent assignments is actually not new. This also happens with the rational function $1/x$ when assigning $x := 0$. To avoid confusion, we invalidate a formula in case of *any* assignment to a quantified variable including kernel assignments.

Another problem arises from the fact that interpreter variables interpreted as kernels are valid rational functions. They are, in contrast, not valid formulas. Hence the user is not allowed to enter things like

```
f := ex(x,g); g := x>0;
```

such that afterwards `f` be evaluated to `ex(x,x>0)`. It is, however, possible to input the above statements in reverse order. Note: The expansion of interpreter variables should not be mixed up with *substitution*, which is correctly implemented for quantified formulas.

8.4.3 Procedures

In order to avoid name conflicts, all REDLOG procedures and switches available in the AM are prefixed by `rl`. The procedure names are to be understood declaratively, e.g., `rlqe` stands for "apply the default procedure for quantifier elimination." Which algorithm is actually applied depends on the selected context. This pattern makes REDLOG easy to learn. Moreover, it allows to combine procedure calls to new (AM) procedures that do not depend on the context.

As usual in REDUCE we have a fairly liberal syntax including optional arguments with default values, procedures expecting either an expression of a certain type or a list of such expressions, and procedures for which the format of the return value depends on a switch.

Most of the REDLOG algorithms offer numerous options. Options are selected by setting corresponding switches.

Some procedures provide the option to protocol the progress of the computation onto the screen. We refer to this as *verbosity* output. In future versions there might be different levels of such output. It is specified, however, that there is *one* switch, namely `rlverbose`, by which all verbosity output can be turned off.

Concerning the return values, our procedures are designed to cooperate with the standard REDUCE. For example, in the ordered field context, the quantifier elimination can optionally compute sample points for existentially quantified formulas. The coordinates of such a point are returned as a list of equations because there are many built-in algorithms that operate on such lists.

8.4.4 Context Dependent Switches

There is a mechanism for passing the control over certain switches, say s_i, to a context c. This means when c is turned on, the current setting of the s_i is saved and then replaced by context specific values. Anyway, the user is allowed to change the setting. When the context is changed again, the current values of the s_i become the new context specific values for c, and the original values are restored. The new context can in turn take control over some switches.

This may appear to be bad style since the system modifies global settings which the user expects to be completely under his control. We need this option, however, for situations where options are not available or extremely undesirable in a certain context.

8.5 Documentation

The REDLOG user manual [30] is written in the GNU Texinfo format from which an online hypertext manual and a TEX document are created. There are also tools available for creating an HTML version of the document.

8.6 Conclusions

REDLOG is an algebraic computer logic system, which is freely available to the scientific community. Several research groups have found applications of REDLOG in their area. There is currently no other published system comparable to it. We have discussed typical applications of REDLOG and its look-and-feel.

Chapter 9

Conclusions

In this thesis, we have concerned ourselves with algorithmic improvements of quantifier elimination by virtual substitution for the reals.

We have started in Chapter 2 by optimizing a crucial tool, which is quite isolated from the actual quantifier elimination procedure: simplification of formulas. After having specified what kind of formulas are considered "simple" thus making the notion of simplification more precise, we have thoroughly investigated all aspects of both quantifier-free and quantified formulas where simplification can take place. The focus was on quantifier-free formulas. This is adequate not only for simplifying the final result of quantifier elimination but also for application during the quantifier elimination procedure, where mainly quantifier-free intermediate results occur, which have to be simplified.

As one crucial tool for simplification, we have introduced here for the first time our ubiquitous idea of using an explicit and an implicit theory. The former is used for entering external information into the simplification process, and the latter is used for communicating information located on different boolean levels in deeply nested formulas. Wherever this is appropriate, our simplification algorithms are designed in such a way that they make use of an optional extra theory argument. In later chapters, we have reencountered the theory concept for quantifier elimination itself, where it participates in a variety of optimization strategies.

On the conceptual side we have introduced the distinction between a fast standard simplifier and sophisticated advanced simplifiers. All simplifiers admit parameterizations, which are implemented via global switches. Our simplification methods provide in a natural way a decision heuristics for simple formulas.

In Chapter 3 we have turned to the core elimination procedure by virtual substitution. After some historical information, we have given an overview over the method as described in the literature so far. This description has served as a reference for the remainder of the thesis.

Our new contributions for this chapter start with Section 3.3. Here we have analyzed the procedure to consist of four distinct phases. This allows for the first time a systematization for all already present optimizations. The remainder of the chapter was devoted to a variety of newly developed optimization strategies of a traditional kind, i.e., they perfectly fit into the traditional framework mentioned above.

In Chapter 4 we have generalized our theory concept from simplification to quantifier elimination by virtual substitution. Starting with the concept of an external theory we have identified the places where one can profit from external information there. These results have encouraged us to adapt also the concept of an implicit theory. The construction of this implicit theory for candidate solution set computation slightly differs from that for simplification. With simplification we have observed that the construction of implicit theories is *the* tool for performing simplifications in spite of complicated boolean structures and for even profiting from these structures. Here we have observed the same effect for the candidate solution set computation phase of our quantifier elimination procedure. We have thus dropped the restriction of the traditional approach to compute elimination sets essentially from the set of atomic formulas.

In the spirit of this observation we have introduced further structural concepts generalizing the Gauss elimination special case of the traditional approach in various ways. Moreover, reanalyzing our generalized Gauss elimination in terms of the implicit theory has led us to the co-Gauss technique. This is a new special case unknown so far, which is in a highly non-trivial way complementary to our generalized Gauss.

Finally, we have introduced another type of theory, which is implicit in nature, but does *not* collect information syntactically present in the formula. Instead the collected information is derived from the elimination process. Concerning the applicability this new theory plays the same role as the implicit theory and can be used simultaneously with it.

In Chapter 5 we have introduced two independent though related concepts: Gauss condensing and positional condensing. The concepts can be combined without problems. In addition they perfectly combine with all other ideas presented throughout this thesis. With condensing we have discovered a new place, where optimizations can take place: The disjunction of virtual substitutions as opposed to the virtual substitutions themselves.

On our way to a more liberal view of quantifier elimination by virtual substitution we have performed yet another step: Instead of operating on atomic formulas as syntactical objects we now think in terms of tree positions.

In Chapter 6, we have introduced local quantifier elimination as a variant of real quantifier elimination. For a certain subset of the parameters there is a point specified. The modified elimination result will be correct for a neighborhood of this point. For this purpose the local quantifier elimination procedure is allowed to assume order and equation constraints compatible with the given point wrt. these variables. As expected, this leads theoretically and practically to smaller intermediate and final results, and to a speed-up of the elimination process. Wherever it suffices to restrict the equivalence of output and input to a neighborhood of the suggested point the concept of local quantifier elimination is superior both to regular quantifier elimination and generic quantifier elimination. The theoretically expected improvements of local quantifier elimination in contrast to the regular quantifier elimination are even exceeded by the results of a test implementation.

Local quantifier elimination provides an example for gaining efficiency and thus applicability by modifying the specification of quantifier elimination. In other words: We do not perform quantifier elimination but instead something similar, which we know to be sufficient for certain problems.

In Chapter 7, we have examined the application of our extended quantifier elimination strategies to the area of scheduling in a very general sense. We have started by demonstrating that we can formulate by first-order formulas and solve by extended quantifier elimination scheduling problems of both the *dedicated machine model* and *project networks*. These models have extensively been discussed within the scheduling community. Our computation times for some small non-trivial examples are reasonable but cannot compete with existent dedicated special-purpose algorithms.

This gives rise to the idea to make use of the extreme flexibility of our approach considering much more general scheduling problems. In particular, we attack new types of problems, which have not been discussed in the literature so far. Such a problem that, to our knowledge, does not fit into any known scheduling model is delay management for railway connections. Here, the first-order formulation is so complex that we had to provide dedicated algorithms for its computation. Our current test implementation cannot compute practical examples of reasonable size yet. Anyway, already the formal specification of the problem by means of our first-order input formula is a theoretical result in its own right. Furthermore this example demonstrates that we can easily formulate multi-objective optimization.

The final Chapter 8 does not contain new contributions but gives an overview of REDLOG, which is an algebraic computer logic system. REDLOG has been realized by the author and others. It provides the programming environment for all our implementations mentioned throughout this thesis.

Bibliography

[1] Chaouki T. Abdallah, Peter Dorato, Wei Yang, Richard Liska, and Stanly Steinberg. Applications of quantifier elimination theory to control system design. In *Proceedings of the 4th IEEE Mediterranean Symposium on Control and Automation*, pages 340–345. IEEE, 1996.

[2] Emil Artin. Über die Zerlegung definiter Funktionen in Quadrate. *Hamburger Abhandlungen*, 5:100–115, 1927.

[3] Martin Bartusch. An algorithm for generating all maximal independent subsets of posets. *Computing*, 26(4):343–354, 1981.

[4] Martin Bartusch. Optimierung von Netzplänen mit Anordnungsbeziehungen bei knappen Betriebsmitteln. Technical Report MIP-8618, FMI, Universität Passau, D-94030 Passau, Germany, 1986. Nachdruck der Dissertation, RWTH Aachen, 1983.

[5] Thomas Becker, Volker Weispfenning, and Heinz Kredel. *Gröbner Bases, a Computational Approach to Commutative Algebra*, volume 141 of *Graduate Texts in Mathematics*. Springer, New York, 1993.

[6] Allan Borodin, Ronald Fagin, John E. Hopcroft, and Martin Tompa. Decreasing the nesting depth of expressions involving square roots. *Journal of Symbolic Computation*, 1(2):169–188, June 1985.

[7] Robert King Brayton, Gary D. Hachtel, Curtis T. McMullen, and Alberto L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, The Hague, Dordrecht, Lancaster, 1984.

[8] Robert P. Brazile and Kathleen M. Swigger. GATES: An airline gate assignment and tracking expert system. *IEEE Expert*, 3(2):33–39, Summer 1988.

[9] Silvia Breitinger and Hendrik C.R. Lock. Using constraint logic programming for industrial scheduling problems. In Christoph Beierle and Lutz Plümer, editors, *Logic Programming: Formal Methods and Practical Applications*, volume 11 of *Studies in Computer Science and Artifical Intelligence*, chapter 9, pages 273–299. Elsevier, Amsterdam, 1995.

[10] Morton Brown. Problem proposal in the *Problems and Solutions* section. *American Mathematical Monthly*, 90(8):569, October 1983.

[11] Peter Brucker. *Scheduling Algorithms.* Springer, Berlin, Heidelberg, 1995.

[12] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[13] Randal E. Bryant. Ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[14] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* Doctoral dissertation, Mathematical Institute, University of Innsbruck, Innsbruck, Austria, 1965.

[15] Bruno Buchberger, George E. Collins, Mark J. Encarnacion, Hoon Hong, Jeremy R. Johnson, Werner Krandick, Rüdiger Loos, Ana M. Mandache, Andreas Neubacher, and Herbert Vielhaber. Saclib 1.1 user's guide. RISC-Linz Series Technical Report 93-19, Research Institute for Symbolic Computation, Johannes Kepler University, A-4040 Linz, Austria, 1993.

[16] Bruno Buchberger and Rüdiger Loos. Algebraic simplification. In Bruno Buchberger, George E. Collins, Rüdiger Loos, and Rudolf Albrecht, editors, *Computer Algebra: Symbolic and Algebraic Manipulation*, pages 11–44. Springer-Verlag, Wien, New York, second edition, 1982.

[17] Klaus-Dieter Burhenne. Implementierung eines Algorithmus zur Quantorenelimination für lineare reelle Probleme. Diploma thesis, Universität Passau, D-94030 Passau, Germany, December 1990.

[18] Bob F. Caviness and Richard J. Fateman. Simplification of radical expressions. In R. D. Jenks, editor, *Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation*, pages 329–338, New York, 1976.

[19] George E. Collins. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In H. Brakhage, editor, *Automata Theory and Formal Languages. 2nd GI Conference*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Gesellschaft für Informatik, Springer-Verlag, Berlin, Heidelberg, New York, 1975.

[20] George E. Collins. The SAC-2 computer algebra system. In Bob F. Caviness, editor, *EUROCAL '85; Proceedings of the European Conference on Computer Algebra*, volume 104 of *Lecture Notes in Computer Science*, pages 34–35. Springer, Berlin, Heidelberg, New York, Tokyo, April 1985.

[21] George E. Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, September 1991.

[22] Alain Colmerauer. Prolog III. *Communications of the ACM*, 33(7):70–90, July 1990.

[23] D. C. Cooper. Theorem proving in arithmetic without multiplication. In *Machine Intelligence*, volume 7, pages 91–99, New York, 1972. American Elsevier.

[24] Robert M. Corless and David J. Jeffrey. Well ... it isn't quite that simple. *ACM SIGSAM Bulletin*, 26(3):2–6, August 1992.

[25] Andreas Dolzmann. Solving geometric problems with real quantifier elimination. In Xiao-Shan Gao, Dongming Wang, and Lu Yang, editors, *Automated Deduction in Geometry*, volume 1669 of *Lecture Notes in Artificial Intelligence (Subseries of LNCS)*, pages 14–29. Springer-Verlag, Berlin Heidelberg, 1999.

[26] Andreas Dolzmann, Oliver Gloor, and Thomas Sturm. Approaches to parallel quantifier elimination. In Oliver Gloor, editor, *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation (ISSAC 98)*, pages 88–95, Rostock, Germany, August 1998. ACM, ACM Press, New York, 1998.

[27] Andreas Dolzmann and Thomas Sturm. Guarded expressions in practice. In Wolfgang W. Küchlin, editor, *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation (ISSAC 97)*, pages 376–383, Maui, HI, July 1997. ACM, ACM Press, New York, 1997.

[28] Andreas Dolzmann and Thomas Sturm. Redlog: Computer algebra meets computer logic. *ACM SIGSAM Bulletin*, 31(2):2–9, June 1997.

[29] Andreas Dolzmann and Thomas Sturm. Simplification of quantifier-free formulae over ordered fields. *Journal of Symbolic Computation*, 24(2):209–231, August 1997.

[30] Andreas Dolzmann and Thomas Sturm. Redlog user manual. Technical Report MIP-9905, FMI, Universität Passau, D-94030 Passau, Germany, April 1999. Edition 2.0 for Version 2.0.

[31] Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning. A new approach for automatic theorem proving in real geometry. *Journal of Automated Reasoning*, 21(3):357–380, 1998.

[32] Andreas Dolzmann and Volker Weispfenning. Local quantifier elimination. Technical Report MIP-0003, FMI, Universität Passau, D-94030 Passau, Germany, February 2000.

[33] Salah E. Elmaghraby. *Activity Networks: Project Planning and Control by Network Models.* Wiley Interscience, New York, London, Sydney, Toronto, 1977.

[34] Jeanne Ferrante and Charles W. Rackoff. *The Computational Complexity of Logical Theories.* Number 718 in Lecture Notes in Mathematics. Springer-Verlag, Berlin, 1979.

[35] Hoon Hong. Simple solution formula construction in cylindrical algebraic decomposition based quantifier elimination. In Paul S. Wang, editor, *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC 92)*, pages 177–188, Berkeley, CA, July 1992. ACM, ACM Press, New York, 1992.

[36] Hoon Hong, George E. Collins, Jeremy R. Johnson, and Mark J. Encarnacion. QEPCAD interactive version 12. Kindly provided to us by Hoon Hong, September 1993.

[37] Hoon Hong, Richard Liska, and Stanly Steinberg. Testing stability by quantifier elimination. To appear in the Journal of Symbolic Computation, 1996.

[38] Hoon Hong, Richard Liska, and Stanly Steinberg. Testing stability by quantifier elimination. *Journal of Symbolic Computation*, 24(2):161–187, August 1997. Special issue on applications of quantifier elimination.

[39] Hoon Hong, Andreas Neubacher, and Wolfgang Schreiner. The design of the SACLIB/PACLIB kernels. *Journal of Symbolic Computation*, 19(1–3):111–132, January–March 1995.

[40] Nikolaos I. Ioakimidis. REDLOG-aided derivation of feasibility conditions in applied mechanics and engineering problems under simple inequality constraints. *Journal of Mechanical Engineering (Strojnícky Časopis)*, 50(1):58–69, 1999.

[41] W. Kahan. An ellipse problem. *ACM SIGSAM Bullitin*, 9(3):11, August 1975. SIGSAM Probem #9.

[42] Timothy Kam, Tiziano Villa, Robert King Brayton, and Alberto Sangiovanni-Vincentelli. Multi-valued decision diagrams: Theory and applications. *Multiple-Valued Logic*, 4(1–2):9–62, 1998.

[43] Michael Kappert. Eliminationsverfahren zur linearen und quadratischen Optimierung. Diploma thesis, Universität Passau, D-94030 Passau, Germany, December 1995.

[44] Heinz Kredel. MAS modula-2 algebra system, interactive usage. Technical report, Universität Passau, Passau, February 1993. Available for anonymous ftp from alice.fmi.uni-passau.de.

[45] Heinz Kredel. MAS modula-2 algebra system, specifications, definition modules, indexes. Technical report, Universität Passau, Passau, February 1993. Available for anonymous ftp from alice.fmi.uni-passau.de.

[46] Wolfgang W. Küchlin. PARSAC-2: A parallel SAC-2 based on threads. In Shojiro Sakata, editor, *Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes: 8th International Conference, AAECC-8*, volume 508 of *Lecture Notes in Computer Science*, pages 341–353, Tokyo, Japan, August 1990, 1991. Springer-Verlag, Berlin, Heidelberg, New York.

[47] Wolfgang W. Küchlin. The S-threads environment for parallel symbolic computation. In Richard E. Zippel, editor, *Computer Algebra and Parallelism, Second International Workshop*, volume 584 of *Lecture Notes in Computer Science*, pages 1–18, Ithaca, USA, May 1990, 1992. Springer-Verlag, Berlin, Heidelberg, New York.

[48] Wolfgang W. Küchlin. PARSAC-2: Parallel computer algebra on the desk-top. In J. Fleischer, J. Grabmeier, F. Hehl, and W. Küchlin, editors, *Computer Algebra in Science and Engineering*, pages 24–43, Bielefeld, Germany, August 1994, 1995. World Scientific, Singapore.

[49] Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. A new class of decidable hybrid systems. In Frits W. Vaandrager and Jan H. van Schuppen, editors, *Hybrid Systems and Control. Proceedings of the Second International Workshop, HSCC'99, Berg en Dal, The Netherlands, March 1999*, volume 1569 of *Lecture Notes in Computer Science*, pages 137–151. Springer, Berlin, Germany, 1999.

[50] Daniel Lazard. Quantifier elimination: Optimal solution for two classical examples. *Journal of Symbolic Computation*, 5(1&2):261–266, February 1988.

[51] Rüdiger Loos and Volker Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462, 1993. Special issue on computational quantifier elimination.

[52] E. J. McCluskey. Minimization of Boolean functions. *Bell Systems Technical Journal*, 35:1417–1444, April 1956.

[53] Herbert Melenk and Winfried Neun. RR: Parallel symbolic algorithm support for PSL based REDUCE. Preliminary Draft, 1995.

[54] Michael Pesch. Die MAS-Implementation des Algorithmus zur Berechnung umfassender Gröbnerbasen. Enclosure of a DFG application, April 1994.

[55] Michael Pinedo. *Scheduling: Theory, algorithms, and systems.* Prentice Hall international series in industrial and system engineering. Prentice-Hall, Englewood Cliffs, New Jersey, 1995.

[56] W. V. Quine. The problem of simplifying truth functions. *American Mathematical Monthly*, 59:521–531, November 1952.

[57] W. V. Quine. A way to simplify truth functions. *American Mathematical Monthly*, 62:627–631, November 1955.

[58] W. V. Quine. On cores and prime implicants of truth functions. *American Mathematical Monthly*, 66:755–760, November 1959.

[59] F. J. Radermacher. Scheduling of resource constraint project networks. Technical Report MIP-8405, FMI, Universität Passau, D-94030 Passau, Germany, 1984.

[60] Alexander Schrijver. *Theory of Linear and Integer Programming.* Wiley-Interscience Series in Discrete-Mathematics. John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore, October 1987.

[61] Markus Schweighofer. Algorithmische Beweise für Nichtneagtiv- und Positivstellensätze. Diploma thesis, Universität Passau, D-94030 Passau, Germany, March 1999.

[62] Raymund M. Smullyan. *First-order Logic.* Springer-Verlag, Berlin, Heidelberg, New York, 1968.

[63] Thomas Sturm. Real quadratic quantifier elimination in Risa/Asir. Research Memorandum ISIS-RM-5E, ISIS, Fujitsu Laboratories Limited, 1-9-3, Nakase, Mihama-ku, Chiba-shi, Chiba 261, Japan, September 1996.

[64] Thomas Sturm. *Real Quantifier Elimination in Geometry.* Doctoral dissertation, Department of Mathematics and Computer Science. University of Passau, Germany, D-94030 Passau, Germany, December 1999.

[65] Thomas Sturm. Reasoning over networks by symbolic methods. *Applicable Algebra in Engineering, Communication and Computing*, 10(1):79–96, September 1999.

[66] Alfred Tarski. A decision method for elementary algebra and geometry. Technical report, RAND, Santa Monica, CA, 1948.

[67] Pascal van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge, Massachusetts, 1st edition, 1989.

[68] Volker Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5(1–2):3–27, February–April 1988.

[69] Volker Weispfenning. Comprehensive Gröbner bases. *Journal of Symbolic Computation*, 14:1–29, July 1992.

[70] Volker Weispfenning. Parametric linear and quadratic optimization by elimination. Technical Report MIP-9404, FMI, Universität Passau, D-94030 Passau, Germany, April 1994.

[71] Volker Weispfenning. Quantifier elimination for real algebra—the cubic case. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC 94)*, pages 258–263, Oxford, England, July 1994. ACM Press, New York, 1994.

[72] Volker Weispfenning. Quantifier elimination for real algebra—the quadratic case and beyond. *Applicable Algebra in Engineering Communication and Computing*, 8(2):85–101, February 1997.

[73] Volker Weispfenning. Simulation and optimization by quantifier elimination. *Journal of Symbolic Computation*, 24(2):189–208, August 1997. Special issue on applications of quantifier elimination.

[74] Steven Wolfram. *The Mathematica Book*. Wolfram Media and Cambridge University Press, 3rd edition, 1996.

[75] Wen-Tsun Wu. Basic principles of mechanical theorem proving in elementary geometries. *Journal of Systems Sciences and Mathematical Sciences*, 4:207–235, 1984.

[76] Wen-Tsun Wu. Basic principles of mechanical theorem proving in elementary geometry. *Journal of Automated Reasoning*, 2:219–252, 1986.

[77] David Y. Y. Yun. On square-free decomposition algorithms. In R. D. Jenks, editor, *Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation*, pages 26–35, New York, NY 10036, August 1976. The Association for Computing Machinery.

[78] Richard Zippel. Simplification of expressions involving radicals. *Journal of Symbolic Computation*, 1(2):189–210, June 1985.