Simplification of Quantifier-free Formulas over Ordered Fields

ANDREAS DOLZMANN AND THOMAS STURM†

Fakultät für Mathematik und Informatik, Universität Passau, 94030 Passau, Germany

e-mail: dolzmann@alice.fmi.uni-passau.de, sturm@alice.fmi.uni-passau.de

MIP 9517

12 October 1995

Good simplification strategies are essential for implementing any kind of effective real quantifier elimination. Given a quantifier-free first-order formula over the theory of ordered fields, the aim is to find an equivalent first-order formula that is *simpler*. The notion of a formula being simpler will be specified. An overview is given over various methods, which we have implemented in REDUCE. Our methods have been developed for simplifying intermediate and final results of automatic quantifier elimination using elimination set ideas (Weispfenning, 1988, Loos and Weispfenning, 1993). They are, however, applicable to other elimination methods.

† The second author was supported by the DFG (Schwerpunktprogramm: "Algorithmische Zahlentheorie und Algebra").

1. Introduction

With quantifier elimination by elimination set methods (Weispfenning, 1988) quantifiers are eliminated one by one starting with the innermost one. The implementation works on straightforward Lisp representations of formulas. This is in contrast to CAD (Collins, 1975, Collins and Hong, 1991), where there is the cell decomposition as an intermediate representation from which the result is finally constructed. Simplification strategies with CAD have been described by Hoon Hong (1992). For us *simplification* is a procedure that maps formula representations to formula representations. We suppose that this makes our methods applicable to many other existent and future elimination methods.

For the elimination set method it is crucial to have a fast simplification for the intermediate results at hand. The size of these grows exponentially for quantifier blocks and even doubly exponentially for alternating quantifiers. Without simplifying the intermediate results the quantifier elimination would fail for lack of storage in most cases. On the other hand, frequent use of more sophisticated and hence slower methods would delay the elimination process excessively. So these are applied only to the final output in order to obtain a comprehensible result.

From the first experimental implementation (Burhenne, 1990) to the current version by the authors there has been an enormous increase in performance. While the original implementation could only eliminate toy examples, the second to last version (Kappert, 1995, Sturm, 1995) is meanwhile used commercially for detecting faulty components in networks. There are several reasons for this dramatical improvement: Better implementation techniques, improved elimination sets (Loos and Weispfenning, 1993, Weispfenning, 1995), and last not least the simplification strategies described here.

The mathematical principles underlying our simplifier are mostly well known. The aim of this article is to provide a collection of practicable methods that have been implemented and extensively tested for their relevance. We further show how to combine different ideas for simplification in such a way that a formula is obtained which cannot be further simplified with any of the described methods. In other words, our simplifiers viewed as a function are idempotent. Achieving this is by no means trivial.

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 that we had developed some time ago. These simplifiers, namely the Gröbner simplifier and the Tableau simplifiers, could even be generalized due to this new viewpoint.

1.1. DEFINITIONS

Our formulas combine atomic formulas using the Boolean connectives "$\wedge$," "$\vee$," "$\longrightarrow$," "$\longleftarrow$," "$\longleftrightarrow$," and "$\neg$." Conjunction and disjunction are not binary but allow an arbitrary number of arguments. The atomic formulas are *equations* constructed with "$=$," *inequalities* constructed with "$\neq$,", strong orderings constructed with "$<$" and "$>$," and weak orderings constructed with "$\leq$" and "$\geq$." This variety of relations enables 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. From now on, we consider all right hand sides of atomic formulas to be zero, and all left hand sides to be elements of some polynomial ring over $\mathbb{Z}$ in a suitable set of variables.

Non-atomic formulas are called *complex*. We divide the complex formulas into *flat* formulas and *deep* formulas. In flat formulas there are no complex subformulas nested. They simply combine atomic formulas to one Boolean level. Examples are conjunctions of atomic formulas or implication between two atomic formulas. It is a major feature of elimination set methods that they can operate on deep formulas instead of requiring some kind of normal form computation. As a consequence, we are particularly interested in the simplification of deep formulas. Boolean normal forms are *disjunctive normal forms* (DNF) and *conjunctive normal forms* (CNF). A DNF is a disjunction of conjunctions including degenerated cases; a CNF is its dual counterpart.

1.2. GOALS AND PARAMETERS

It is not obvious which formulas are to be considered simple. We summarize and comment on our *simplification goals*. Notice that some of them contradict each other. For these cases, the simplifier has to be parameterized such that the user can decide which goal to prefer for a particular problem. We also summarize the possible parameterizations.

Currently, our main goal is to obtain formulas containing *few atomic formulas*. In our practical applications, the final output is as a rule too large to be read and understood by a human. At the current state of our work, the results are already small enough to be used for serious applications where they are processed automatically. However, our goal is to use quantifier elimination for solving mathematical problems or at least for supporting us in doing so. For this it is essential that the output is comprehensible.

Formulas with a *comprehensible Boolean structure* are simple. Examples for comprehensible Boolean structures are relatively flat formulas or case distinctions. The reason for this claim is the same as above.

Recall that we do not only simplify the final but also intermediate results. For elimination set methods, it is convenient to have *few different atomic formulas*. In addition, this supports many simplification strategies.

Atomic formulas with *simple terms* are simple. We consider it unintuitive when information that can be encoded logically is actually encoded algebraically.

In certain contexts one can decide between different relations for an atomic formula, e.g., between $t < 0$ and $t \leq 0$ if we know $t \neq 0$ for some reason. Selecting the relation with that leads to a *small satisfaction set* for the atomic formula leads to an output that is less redundant.

Independent of satisfaction sets, there are *convenient relations*. For elimination set methods, weak orderings are more convenient than strong ones. On the other hand, equations and inequalities can be considered simpler than orderings.

There are also *convenient Boolean operators*. We consider conjunction and disjunction simpler than implication, replication, and equivalence.

Our final claim is not to the simplification result but to the procedure itself: We insist on our simplifiers to be *idempotent*.

The parameterization works by setting some global switches. These are described below together with a short discussion on which simplification goals they affect.

Prefer non-orderings to orderings and, independently, *prefer weak orderings to strong* *orderings*. These come out to prefering 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.

Translate implications to disjunctions. This can be turned off since implications can be more intuitive due to the problem that is being modeled.

1.3. THE THEORY CONCEPT

The option to pass a *theory*, say Θ, as extra parameter is a key feature of our simplifier. A theory is a set of atomic formulas considered conjunctive. The target formula φ is simplified w.r.t. Θ. For this purpose, we consider the variables in our atomic formulas as *constants* in the sense of logic. This implies that the theory $\{a^2 - a = 0\}$ does not contain a multiplicative idempotency rule but information on the constant a that also occurs in φ. Formally, we obtain a formula φ' such that

$$\Theta \models \varphi \Longleftrightarrow \varphi'.$$

For clarity we wish to point out that with our formalism all atomic formulas in Θ are closed.

The theory parameter offers the possibility to enter extra information into the simplification process without adding it conjunctively to the target formula. Notice that it would be a problem to remove conjunctively added information from the simplification result since it cannot be recognized easily.

Under an inconsistent theory any two formulas are equivalent, so the simplification result makes no sense in this case. If our simplifier happens to detect that a theory is inconsistent it raises an error. Mind that explicitely checking the theories for being consistent is the existential decision problem for ordered fields; this is not practicable for our purposes.

The theory concept may seem like some toy feature. However, it plays an important role for understanding the idea and the implementation of our simplification algorithm for deep formulas. There the theory—possibly empty in the beginning—is implicitly enlarged during recursion.

2. Atomic Formulas

Variable-free atomic formulas are evaluated to truth values. In others atomic formulas the right hand side is replaced by its primitive part over $\mathbb{Z}$ with positive head coefficient. Making the head coefficient positive implies mapping "$\leq$" to "$\geq$", "$<$" to "$>$", and vice versa.

2.1. SQUAREFREE PARTS AND DEGREE PARITY DECOMPOSITIONS

A polynomial is *squarefree* if it has no divisor of multiplicity greater than 1. The *squarefree decomposition* of a polynomial f is a list $\big((p_1, 1), \ldots, (p_n, n)\big)$ with all p_i squarefree and pairwise relatively prime such that $\prod p_i^i = f$. We call $\prod p_i$ the *squarefree part* of f. The *degree parity decomposition* of f is defined as the pair

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

It obviously contains less information than a squarefree decomposition. We introduce this notion since it is exactly what we need with orderings. Degree parity decompositions can easily be computed from the respective squarefree decompositions. It is an interesting open question if there is a faster way.

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

(i) $f = 0 \Longleftrightarrow F = 0 \Longleftrightarrow p = 0 \vee q = 0$
(ii) $f \neq 0 \Longleftrightarrow F \neq 0 \Longleftrightarrow p \neq 0 \wedge q \neq 0$
(iii) $f > 0 \Longleftrightarrow pq^2 > 0 \Longleftrightarrow p > 0 \wedge q \neq 0$
(iv) $f \geq 0 \Longleftrightarrow pq^2 \geq 0 \Longleftrightarrow p \geq 0 \vee q = 0$
(v) $f < 0 \Longleftrightarrow pq^2 < 0 \Longleftrightarrow p < 0 \wedge q \neq 0$
(vi) $f \leq 0 \Longleftrightarrow pq^2 \leq 0 \Longleftrightarrow p \leq 0 \vee q = 0$

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

2.2. SEMIDEFINITENESS AND DEFINITENESS TESTS

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

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

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

In case that f is positive semidefinite, we have

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

Notice that in the last two cases f can be replaced by its squarefree part according to Lemma 2.1. The decision between $f > 0$ and $f \neq 0$ depends on whether the simplification goal of convenient relations, here no orderings, 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 Emil Artin (1927), exactly positive semidefinite polynomials can be written as sum of squares of rational functions. 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.

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

(ii) A polynomial is positive definite if its squarefree part is an STSQ*. It is positive semidefinite if its squarefree part is a* TSQ*.*

(iii) A polynomial with degree parity decomposition (p, q) is positive semidefinite if p is a TSQ*.*

Obviously, none of the above tests is a necessary condition. We will illustrate by examples that none of the tests is redundant.

The first example shows that it is necessary to test the original left hand side polynomial for being an STSQ:

$$4x^{16} + 8x^{14} + 4x^{10} + 17x^8 + 4x^6 + 8x^2 + 4 = (2x^8 + 2x^6 - x^4 + 2x^2 + 2)^2$$

The polynomial squared on the right hand side is the squarefree part. By the way, there is no univariate non-STSQ of degree less than 8 which becomes an STSQ when being squared. We have found this and the above example using the QEPCAD package by Hoon Hong. For the necessity of testing the original polynomial for being a TSQ we have the obvious example x^2 with squarefree part x and degree parity decomposition $(1, x)$.

Our next example is $(x^2 + x + 1)^2(x^2 - x + 1) = x^6 + x^5 + 2x^4 + x^3 + 2x^2 + x + 1$, which is not a TSQ. Its squarefree part $x^4 + x^2 + 1$ shows that it is positive definite while its degree parity decomposition $(x^2 - x + 1, x^2 + x + 1)$ does not. A corresponding TSQ example can be constructed by multiplying a squarefree TSQ that does not disturb the above cancelations, e.g. $a^2 + b^2$.

For the relevance of testing the degree parity decomposition consider $x^2 - 2x + 1$ with squarefree part $x - 1$ and degree parity decomposition $(1, x - 1)$.

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

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

(i) The product fg is a TSQ*, and fg is strict iff 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. Firstly, compared to the squarefree part F, a degree parity decomposition (p, q) offers no extra information on definiteness: If both p and q are STSQ's, then f is positive definite but we have seen that in this case $F = pq$ is already an STSQ.

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

2.3. APPLYING THE THEORY

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. Generally, the theory is never applied to isolated atomic formulas but always to flat formulas. Hence the atomic formula is treated as a unary conjunction in the former case, and the discussion of theory application belongs to the next section.

Anyway, in order to give a first idea of how the theory is applied, we close this section with some examples: With $\{a \leq 0\}$ as theory, $a - 1 \neq 0$ is simplified to "true", and $a < 0$ can optionally be simplified to $a \neq 0$, this depends on the simplification goal preferred.

2.4. 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 David Y. Y. Yun (1976).

We give some further ideas, which seem worth implementing them. In Lemma 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 formulas 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:

$$\sum s_i \leq 0 \iff \sum s_i = 0 \iff \bigwedge s_i = 0 \quad \text{and} \quad \sum s_i > 0 \iff \sum s_i \neq 0 \iff \bigvee s_i \neq 0$$

This again meets the simplification goal of simple terms but not that of few atomic formulas. After splitting, the new equations or inequalities have to undergo atomic formula simplification.

In Lemma 2.1 we have already seen a multiplicative counterpart of the additive splitting. This can be extended in various ways. Computing squarefree decompositions instead of degree parity decompositions, one obtains more factors. With non-orderings all of these can be split. With orderings 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 above. In REDUCE we have a efficient polynomial factorization at hand. Nevertheless, we had decided to do without factorization since squarefree decomposition is much faster. Currently, we are considering to introduce factorization as an option.

3. Flat Formulas

To begin with, notice that 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 considered formulas.

One can imagine to apply the converse of the additive and multiplicative splittings discussed in the previous section. One might further argue that a decision for splitting atomic formulas should imply the decision to apply the converse step to flat formulas where possible. This would meet the simplification goals of few atomic formulas and possibly that of a comprehensible Boolean structure, namely when a whole Boolean level is dropped this way. Anyway, we do not so since we suppose that, in general, this would increase the complexity of the terms drastically. Later, with Gröbner methods and with theory inheritance, we will see that one can make use of atomic formula-encoding of conjunctions or disjunctions in a more sophisticated way than simply regarding it as simplification rule.

Table 1. Ordering 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
$t > 0$						$t > 0$

3.1. Boolean simplification

There are some simplification rules of purely Boolean nature. All replications are turned into implications. The following equivalences, which hold for any formula φ are applied.

$$\begin{aligned}
&\neg\text{true} \Longleftrightarrow \text{false}, \quad \neg\text{false} \Longleftrightarrow \text{true},\\
&\text{false} \longrightarrow \varphi \Longleftrightarrow \varphi \longrightarrow \text{true} \Longleftrightarrow \varphi \longrightarrow \varphi \Longleftrightarrow \text{true},\\
&\text{true} \longrightarrow \varphi \Longleftrightarrow \varphi, \quad \varphi \longrightarrow \text{false} \Longleftrightarrow \neg\varphi\\
&\varphi \longleftrightarrow \text{true} \Longleftrightarrow \varphi, \quad \varphi \longleftrightarrow \text{false} \Longleftrightarrow \neg\varphi, \quad \varphi \longleftrightarrow \varphi \Longleftrightarrow \text{true},\\
&\varphi \wedge \text{true} \Longleftrightarrow \varphi \wedge \varphi \Longleftrightarrow \varphi, \quad \varphi \wedge \text{false} \Longleftrightarrow \text{false},\\
&\varphi \vee \text{false} \Longleftrightarrow \varphi \vee \varphi \Longleftrightarrow \varphi, \quad \varphi \vee \text{true} \Longleftrightarrow \text{true}
\end{aligned}$$

The atomic formulas are being sorted within conjunctions, disjunctions, and equivalences. We use an ordering on the terms which we extend to atomic formulas by first sorting w.r.t. the left hand side term and then w.r.t. the relation.

3.2. Smart simplification

Smart simplification makes use of non-Boolean dependencies between 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 $\overline{\rho}$ among our considered relations such that $t\ \overline{\rho}\ 0$ is equivalent to $\neg t\ \rho\ 0$ for any term t. We call $\overline{\rho}$ 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 ordering 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 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

$$p + c\ \rho\ 0 \wedge p + d\ \sigma\ 0,$$

Table 2. Additive smart simplification assuming $c < d$

$\wedge$	$p+c=0$	$p+c\leq 0$	$p+c\geq 0$	$p+c\neq 0$	$p+c<0$	$p+c>0$
$p+d=0$	false	$p+d=0$	false	$p+d=0$	$p+d=0$	false
$p+d\leq 0$	false	$p+d\leq 0$	false	$p+d\leq 0$	$p+d\leq 0$	false
$p+d\geq 0$	$p+c=0$	—	$p+c\geq 0$	—	—	$p+c>0$
$p+d\neq 0$	$p+c=0$	—	$p+c\geq 0$	—	—	$p+c>0$
$p+d<0$	false	$p+d<0$	false	$p+d<0$	$p+d<0$	false
$p+d>0$	$p+c=0$	—	$p+c\geq 0$	—	—	$p+c>0$

Table 3. Convenient relations

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

with c, $d \in \mathbb{Q}$, we can decide $c\ \tau\ d$ 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). Consider for instance

$$x > 0 \wedge 2x-1 > 0 \wedge 3x+5 \neq 0 \iff 2x > 1 \quad \text{or} \quad x^2+y+4 \geq 0 \vee 7x^2+7y+4 \leq 0 \iff \text{true}.$$

Given an n-ary conjunction, the simplification result is invariant w.r.t. the order in which the (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 clarify how to bring the theory into this process. The equivalences underlying the theory application are always the same as that underlying the respective 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 come to 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 another role when flat simplification is viewed as a part of deep simplification.

The result obtained with the above methods meets the simplification goal of small satisfaction sets for the atomic formulas. We also have to provide for 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 3).

For disjunctions we exploit the duality to conjunctions: The target disjunction is negated, obtaining a conjunction of the negated atomic formulas via de Morgan's law. Then we proceed as with 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. Notice that when checking one must have in mind that all relations are negated.

Due to the simplification goal of convenient Boolean operators, an implication $\alpha \longrightarrow \beta$ between two atomic formulas is resolved into the disjunction $\overline{\alpha} \vee \beta$. If the simplification result is still a disjunction of two atomic formulas, we optionally reconstruct an implication. This is done such that the atomic formulas in the implication are sorted. Note that we can obtain the contrapositive of the input this way.

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 resimplify the original left hand side and right hand side separately as unary conjunctions and then sort the result. In all other cases we are finished.

3.3. Gröbner basis methods

Gröbner basis methods allow us to take advantage of certain algebraic interactions between the atomic formulas when equations are involved. Gröbner basis theory requires the polynomial coefficients to be field elements. For our concrete application, however, it suffices to consider polynomials over the integers. By *the* Gröbner basis we mean the unique reduced Gröbner basis w.r.t. to a fixed term order which contains only primitive polynomials. 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.

The following proposition states the mathematical background for the method we use.

Proposition 3.1. *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*

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

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

The above proposition can also be applied to disjunctions by simplifying their negation. In that case, it is instructive to write the disjunctions as implications:

$$\Bigl(\bigwedge_i f_i = 0\Bigr) \longrightarrow \Bigl(\bigvee_j g_j \ \rho_j \ 0\Bigr) \quad \text{iff} \quad \Bigl(\bigwedge_k \tilde{f}_k = 0\Bigr) \longrightarrow \Bigl(\bigvee_j \tilde{g}_j \ \rho_j \ 0\Bigr).$$

Actually, it was this form that gave the idea for Gröbner simplification. 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 3.1 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 can decide $h \ \rho_j \ 0$. If so, we may either drop the respective atomic formula or evaluate the whole conjunction to "false". Else, we perform a radical membership test, which can be done without computing a radical basis (Becker and Weispfenning, 1993). 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 w.r.t. 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, viewing flat formulas as parts of complex formulas, reduction can increase the number of different atomic formulas. This is because like terms are reduced w.r.t. 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.

Our next idea is one of the indicated examples for making use of the possibility to encode certain conjunctions multiplicatively into one atomic formula.

LEMMA 3.1. *Let $\rho_j \in \{<, >\}$, let $\tilde{\rho}_j$ denote the weak counterpart of ρ_j, and let σ_k be any relation. Then the following are equivalent:*

(i) $\bigwedge_i p_i \neq 0 \wedge \bigwedge_j q_j \ \rho_j \ 0 \wedge \bigwedge_k r_k \ \sigma_k \ 0$
(ii) $\prod_i p_i \cdot \prod_j q_j \neq 0 \wedge \bigwedge_j q_j \ \rho_j \ 0 \wedge \bigwedge_k r_k \ \sigma_k \ 0$
(iii) $\prod_i p_i \cdot \prod_j q_j \neq 0 \wedge \bigwedge_j q_j \ \tilde{\rho}_j \ 0 \wedge \bigwedge_k r_k \ \sigma_k \ 0$

Since $\mathrm{Id}(\{f_i\})$ need not be prime, this offers a chance to improve our method: decision 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 orderings. This corresponds to an application of (ii) or (iii), respectively. Recall that obtaining weak orderings 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 save 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 w.r.t. 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 again make use of our theory concept. A background theory enlarges $\{f_i\}$ by the set $\{F_i\}$ of left hand sides of theory equations. Accordingly, $\{g_j\}$ is enlarged by $\{G_j\}$.

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. In addition, one can make use of $\{G_j\}$: Each G_j is reduced modulo H, then we try to evaluate the respective atomic formula. If it is "false", the whole conjunction is "false", otherwise we ignore it. The inequalities and strong orderings among them can contribute to the respective product in Lemma 3.1 further enlarging the probability for a successful radical membership test.

3.4. HISTORY AND IMPLEMENTATION

Smart simplification developed from the pure ordering approach over the parametric part splitting to involving theories. Contracting atomic formulas whose terms are identical monic variables but with different absolute summands has already been indicated by Hoon Hong (1992). None of the described smart simplifications has led to any problems concerning the speed of our simplifier.

Based on ideas by Thomas Becker, Michael Pesch, and Volker Weispfenning, the first related Gröbner basis methods have been developed with the implementation of comprehensive Gröbner basis (Weispfenning, 1992) computation. This application involved ideal and radical membership tests for conjunctions containing only equations and inequalities. The implementation was done by Michael Pesch (1994) in the CGB-package of the computer algebra system MAS by Heinz Kredel (1993b, 1993a).

Further MAS implementations for simplifying Boolean normal forms were done by the first author (Dolzmann, 1994a, Dolzmann, 1994b). 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. Currently, they are used for the final result simplification only.

3.5. OUTLOOK

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 part is older than this kind of simplification. We had preferred it for readability reasons.

We have introduced ordering theoretical contraction of atomic formulas and an extension using the theory of ordered fields. This extension was additive in nature. There is also a multiplicative extension: Given a conjunction $s\ \rho\ 0 \wedge t\ \sigma\ 0$, one can check if s divides t or vice versa. Let w.l.o.g. $t = rs$, simplification of terms, reduction of the number of atomic formulas, or evaluations to truth values are possible in many cases (cf. Table 4). We give some examples:

$$xy \geq 0 \wedge x < 0 \Longleftrightarrow y \leq 0 \wedge x < 0, \quad xy \geq 0 \wedge x = 0 \Longleftrightarrow x = 0, \quad xy \neq 0 \wedge x = 0 \Longleftrightarrow \text{false}.$$

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 5); for instance

$$xy - 1 > 0 \wedge x = 0 \Longleftrightarrow \text{false} \quad \text{or} \quad xy + 1 > 0 \wedge x = 0 \Longleftrightarrow x = 0.$$

This kind of simplification does not involve any factorization. Some extreme special cases have been mentioned by Richard Liska and Stanly Steinberg (1995). There are some problems 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. As an example consider

$$abcd = 0 \wedge ab \neq 0 \wedge bc \neq 0 \wedge ade = 0.$$

There are two possible first steps yielding

$$cd = 0 \wedge ab \neq 0 \wedge bc \neq 0 \wedge ade = 0 \quad \text{or} \quad ad = 0 \wedge ab \neq 0 \wedge bc \neq 0 \wedge ade = 0.$$

Table 4. Multiplicative smart simplification

$\wedge$	$st = 0$	$st \leq 0$	$st \geq 0$
$t = 0$	$t = 0$	$t = 0$	$t = 0$
$t \leq 0$	—	—	—
$t \geq 0$	—	—	—
$t \neq 0$	$t \neq 0 \wedge s = 0$	—	—
$t < 0$	$t < 0 \wedge s = 0$	$t < 0 \wedge s \geq 0$	$t < 0 \wedge s \leq 0$
$t > 0$	$t > 0 \wedge s = 0$	$t > 0 \wedge s \leq 0$	$t > 0 \wedge s \geq 0$

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

Table 5. Multiplicative smart simplification with constant residue

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

In contrast to the latter case, in the former the is no further simplification possible. In other words, one can make mistakes. 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.

4. Deep Formulas

To begin with, recall that the Boolean simplification rules of Subsection 3.1 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.

4.1. CONSTRUCTING IMPLICIT THEORIES

As already mentioned, we are going to use our concept of a theory for inheriting information down the Boolean levels. Our technique is based on the following lemma.

It allows us to use atomic formulas located on a certain boolean level deeper inside the formula by enlarging the theory.

LEMMA 4.1. *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 $\Theta \cup \Gamma \models \psi \Longleftrightarrow \psi'$. Then

$$\Theta \models \bigwedge \Gamma \wedge \psi \Longleftrightarrow \bigwedge \Gamma \wedge \psi'.$$

(ii) Let ψ'' be such that $\Theta \cup \overline{\Gamma} \models \psi \Longleftrightarrow \psi''$. Then $\Theta \models \bigvee \Gamma \vee \psi \Longleftrightarrow \bigvee \Gamma \vee \psi''$.

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

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

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

1 Update Θ to Θ' w.r.t. the top level atomic formulas of φ.
2 Simplify the complex constituents w.r.t. Θ'.
3 If new top level atomic formulas come into existence in step 2, then update Θ' w.r.t. these and go to 2.
4 Use methods as described in Section 3 for simplifying the top level atomic formulas present now w.r.t. the original theory Θ.

Step 3 and hence the loop is necessary for making the simplifier idempotent. Notice 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.

4.2. THE STANDARD SIMPLIFIER

Up to this section, 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. Our implementations of elimination set methods spend about 80% of their time with simplification. The standard simplifier includes all described concepts except for the Gröbner basis methods.

Using Gröbner basis methods within the deep simplification is the first example for an *advanced simplifier*, i.e., a simplifier that is applied in the end only. In the last two sections, we will describe further concepts of advanced simplifiers, which make use of the standard simplifier as subalgorithm.

4.3. ILLUSTRATING EXAMPLES

As a first example consider the formula $a = 0 \wedge \bigl(b \neq 0 \vee (c \leq 0 \wedge (d > 0 \vee a = 0))\bigr)$. 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

$$a = 0 \wedge (b \neq 0 \vee c \leq 0).$$

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.

$$a = 0 \wedge \bigl(b = 0 \vee (c = 0 \wedge d \geq 0)\bigr) \wedge (d \neq 0 \vee a \neq 0).$$

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

4.4. HISTORY

The following lemma contains the key observation that gave the idea for our deep simplification. We had in mind to apply the implication part of the equivalences, then to simplify, and finally to step back. We give the lemma at this place because we consider it a good structural description of what the deep simplifiers do.

LEMMA 4.2. *Let φ and ψ be quantifier-free formulas. We use dots to indicate that ψ may be deeply nested inside the considered formula. Then the following equivalences hold:*

$$\varphi \wedge (\ldots\psi\ldots) \longleftrightarrow \varphi \wedge \bigl(\ldots(\varphi \wedge \psi)\ldots\bigr), \quad \varphi \vee (\ldots\psi\ldots) \longleftrightarrow \varphi \vee \bigl(\ldots(\neg\varphi \wedge \psi)\ldots\bigr)$$

and corresponding dual variants

$$\varphi \wedge (\ldots\psi\ldots) \longleftrightarrow \varphi \wedge \bigl(\ldots(\neg\varphi \vee \psi)\ldots\bigr), \quad \varphi \vee (\ldots\psi\ldots) \longleftrightarrow \varphi \vee \bigl(\ldots(\varphi \vee \psi)\ldots\bigr).$$

4.5. 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 ordering 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. Notice 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 above.

5. Tableau Methods

Although our deep simplifier already combines information located on different Boolean levels, it keeps the basic Boolean structure of the formula. The tableau methods in contrast provide a technique for changing the Boolean structure of formula. This is done by constructing case distinctions. Compared to the deep simplification, they are much slower. Hence, they are typically used for a final simplification step after elimination.

5.1. The basic tableau idea

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

$$\bigvee_{\alpha\in\mathbf{A}} (\alpha \land \varphi) \quad \text{with} \quad \bigvee \mathbf{A} \Longleftrightarrow \mathrm{true}$$

where $\mathbf{A}$ 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 $\mathbf{A}$. The idea is to choose a good $\mathbf{A}$ such that using each $\alpha \in \mathbf{A}$ 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 w.r.t. α as φ_α, we obtain a formula of the form

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

Applying a law of distributivity, this can be simplified to

$$\big((\alpha_0 \lor \bigvee \mathbf{A}_1) \land \varphi_{\alpha_0}\big) \lor \bigvee_{\alpha\in\mathbf{A}_2} (\alpha \land \varphi_\alpha).$$

This is a simplification that our deep simplifier does not know. We call it *contraction* of tableau branches. Notice that afterwards the flat disjunction $\alpha_0 \lor \bigvee \mathbf{A}_1$ can be simplified using the methods of Section 3.

Good candidates for $\mathbf{A}$ are case distinctions $\{t < 0, t = 0, t > 0\}$ w.r.t. 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 w.r.t. an $\mathbf{A}$ of the form above a *tableau step w.r.t. t.*

5.2. The automatic tableau

The automatic tableau tries tableau steps w.r.t. all terms in φ. In the end, if there was a tableau result simpler than the input, one of the best 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 terms tried for the automatic tableau.

5.3. THE ITERATIVE TABLEAU

In contrast to the simple tableau, the automatic tableau is not idempotent. Iterative application can lead to a finite sequence of increasingly better results.

Automizing this repetition has lead to the idea of repeating the tableau steps on the single branches instead of the whole result. This has turned out better in most cases. For instance, consider the following pattern:

$$(s \neq 0 \wedge \psi_1) \vee (s \neq 0 \wedge \psi_2) \vee (s = 0 \wedge \chi).$$

Suppose that inside ψ_1 and ψ_2, there is no successful tableau step possible, but in χ there is. Furthermore, ψ_1 and ψ_2 are taken to contain many atomic formulas compared to χ. Since the ψ_i are big, we have to start with a tableau step w.r.t. $\{s = 0, s \neq 0\}$; starting with the χ step would multiply their occurrences. We obtain

$$\bigl(s \neq 0 \wedge (\psi_1 \vee \psi_2)\bigr) \vee (s = 0 \wedge \chi).$$

Next, due to the branchwise tableau iteration, we have the possibility to apply the χ tableau more locally, i.e., without multiplying the ψ_i.

We give an example where non-branchwise iteration is better than branchwise. We write the number of atomic formulas contained in a formula φ as $|\varphi|$. Consider the following formula with $18 + \sum |\psi_i|$ atomic formulas:

$$\begin{gathered}
\bigl(t = 0 \wedge \bigl((s = 0 \wedge \psi_1) \vee (s = 0 \wedge \psi_2)\bigr)\bigr) \vee \bigl(t \neq 0 \wedge \bigl((s = 0 \wedge \psi_3) \vee (s = 0 \wedge \psi_4)\bigr)\bigr)\\
\vee \bigl(t = 0 \wedge \bigl((s < 0 \wedge \psi_5) \vee (s < 0 \wedge \psi_6)\bigr)\bigr) \vee \bigl(t \neq 0 \wedge \bigl((s < 0 \wedge \psi_7) \vee (s < 0 \wedge \psi_8)\bigr)\bigr)\\
\vee \bigl(t = 0 \wedge \bigl((s > 0 \wedge \psi_9) \vee (s > 0 \wedge \psi_{10})\bigr)\bigr) \vee \bigl(t \neq 0 \wedge \bigl((s > 0 \wedge \psi_{11}) \vee (s > 0 \wedge \psi_{12})\bigr)\bigr)
\end{gathered}$$

We suppose that both s and t do not occur as subterms in the ψ_i. A tableau step w.r.t. $\{t = 0, t \neq 0\}$ yields $14 + \sum |\psi_i|$ atomic formulas while a step w.r.t. $\{s = 0, s < 0, s > 0\}$ yields only $9 + \sum |\psi_i|$. Hence the automatic tableau chooses the latter obtaining:

$$\begin{gathered}
\bigl(s = 0 \wedge \bigl(t = 0 \wedge (\psi_1 \vee \psi_2)\bigr) \vee \bigl(t \neq 0 \wedge (\psi_3 \vee \psi_4)\bigr)\bigr)\\
\vee \bigl(s < 0 \wedge \bigl(t = 0 \wedge (\psi_5 \vee \psi_6)\bigr) \vee \bigl(t \neq 0 \wedge (\psi_7 \vee \psi_8)\bigr)\bigr)\\
\vee \bigl(s > 0 \wedge \bigl(t = 0 \wedge (\psi_9 \vee \psi_{10})\bigr) \vee \bigl(t \neq 0 \wedge (\psi_{11} \vee \psi_{12})\bigr)\bigr).
\end{gathered}$$

A subsequent branchwise tableau w.r.t. $\{t = 0, t \neq 0\}$ has no effect. Its non-branchwise variant, in contrast, yields $8 + \sum |\psi_i|$.

Continuing with the best result is a heuristic approach. Examples can be constructed where better results are obtained by continuing with a tableau result that is even larger than the input. For instance, consider the following formula containing 8 atomic formulas with branchwise iteration:

$$s \leq 0 \wedge (a = 0 \vee b = 0) \wedge (s < 0 \vee a = 0 \vee b \neq 0) \wedge (s = 0 \vee d = 0).$$

A tableau step w.r.t. s increases the number of atomic formulas up to 9. A subsequent tableau w.r.t. a in the $s = 0$ branch finally yields 6 atomic formulas.

5.4. HISTORY AND IMPLEMENTATION

The method is related to the analytic tableaux used for automated theorem proving (Smullyan, 1968). A special case of the tableau method described here was originally suggested by Rüdiger Loos (Loos and Weispfenning, 1993, Weispfenning, private communication) and firstly implemented by Klaus-Dieter Burhenne (1990). This version performed tableau steps w.r.t. a term t without contraction of branches. The simplifications in the branches were restricted to deciding atomic formulas with t as their left hand side.

Our only complete implementation is still restricted to tableau steps w.r.t. a term but includes contraction of branches, automatic tableau, and iterative tableau with optional branchwise iteration. It is older than the implicit theory method described in Section 4. We are currently implementing the method based on the standard simplifier as described in this section. With this implementation strategy, the tableau methods will profit from all future improvements of the underlying standard or advanced deep simplifier.

Many of the simplifications that we considered typical tableau applications were actually theory applications and are thus performed by the standard simplifier now. A typical tableau simplification that occurs now is the application of the laws of distributivity such that atomic formulas are put outside the brackets.

5.5. OUTLOOK

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

$$\bigwedge_{\alpha \in \mathbf{A}} (\alpha \vee \varphi) \quad \text{with} \quad \bigwedge \mathbf{A} \Longleftrightarrow \text{false}$$

for a set $\mathbf{A}$ of atomic formulas. One would then define a tableau step w.r.t. a term t as taking $\mathbf{A} = \{t \geq 0, t \neq 0, t \leq 0\}$. Since these atomic formulas enter the theory negated, there are the same simplifications performed within the single branches as in the normal case. When the top level 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 w.r.t. 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 3 is available, there is an interesting variant of the automatic tableau: One can first factorize all terms occuring in the target formula and then perform the tableau w.r.t. all the irreducible factors instead of all the terms. We expect the result to meet the simplification goal of simple terms more than that of few atomic formulas. Hence, one has to define criterions for finding the *most simple* formula obtained this way.

6. Boolean Normal Forms

It is because of the simplification goal of a comprehensible Boolean structure that we consider Boolean normal form computation as simplification. By the way, a computed Boolean normal form can have less atomic formulas than the input formula.

6.1. COMPUTATION OF BOOLEAN NORMAL FORMS

We restrict our attention to DNF computations assuming 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.

Our algorithm works recursively with the following rules as basis:

(i) An atomic formula is a DNF.
(ii) A flat formula is a DNF.

In the recursion step, we must compute a DNF from a disjunction or conjunction of DNF's. The former case is trivial, in the latter we have to apply a law of distributivity. Following lemma shows how this corresponds to a Cartesian product computation.

LEMMA 6.1. *For $i = 1, \ldots, m$ and $j = 1, \ldots, n_i$ let γ_{ij} be conjunctions of atomic formulas. Set $N = \{1, \ldots, n_1\} \times \ldots \times \{1, \ldots, n_m\}$. Then*

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

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

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

6.2. SIMPLIFICATION OF BOOLEAN NORMAL FORMS

In addition to the simplification methods already presented we have two further ones that we use for the simplification of Boolean normal forms. Both are based on the following lemma.

LEMMA 6.2. *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.*

Again, we restrict ourselves to DNF computations applying the first equivalence of the lemma to conjunctions.

Verifying the premise of the lemma is the universal decision problem which is not practicable for our purposes. Therefore, we use tests for implication that are only sufficient. Formally we introduce relations $\preceq$ such that $\varphi \preceq \psi$ 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 is *transitivity*. The second is *compatibility with conjunctions*, which is defined as

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

A DNF is simplified by testing for each pair (φ, ψ) of conjunctions if $\varphi \preceq \psi$ holds. If so, φ is deleted from the disjunction. If $\preceq$ is transitive, the order in which the pairs are tested is irrelevant for the result. In particular, this allows us to implement simultaneous tests for $\varphi \preceq \psi$ and $\psi \preceq \varphi$, which in some cases can be implemented more efficiently.

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 $\preceq$ is subsumption. Given conjunctions φ and ψ of atomic formulas, φ *subsumes* ψ if each atomic formula in ψ is contained in φ. Subsumption is transitive and compatible with conjunctions. A test for subsumption can be implemented efficiently using the fact that atomic formulas are canonically ordered within the conjunctions. In addition, for subsumption one can implement an efficient simultaneous test whether $\varphi \preceq \psi$, $\psi \preceq \varphi$, or nothing holds.

A smarter though less efficient choice is *simplifier-recognized implication*, making use of our theory concept. Formally it is based on the equivalence between

$$\Theta \models \varphi \quad \text{and} \quad \models \bigwedge \Theta \Longrightarrow \varphi.$$

Recall that the variables in our atomic formulas are constants in the sense of logic. Thus φ and all the atomic formulas in Θ are closed.

We define that $\varphi \preceq \psi$ 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. Compatibility however is an important property here since compared to subsumption these simplifications have turned out to be much more expensive in time.

6.3. HISTORY AND IMPLEMENTATION

Since the elimination set method for quantifier elimination 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 lead to an enormous improvement in Boolean normal form computation. The idea of subsumption lead to further considerable improvements. We also have an ad hoc implementation of the simplifier-recognized implication. It has led to some minor improvements but, currently, it is extremely time expensive. Currently, the best Boolean normal forms are obtained by applying the Groebner simplifier.

6.4. OUTLOOK

In the outlook of Section 3 we have already indicated the alternative of making the terms monic instead of primitive. Here, this would allow us to implement the test on simplifier-recognized implication more efficiently. Our idea is based on the fact that the standard simplifier performs simplifications only between atomic formulas with the same parametric part. Recall that the atomic formulas are canonically ordered within the conjunctions. The ordering used is an ordering $\sqsubseteq$ on terms that is extended to atomic formulas by first ordering w.r.t. the left hand side terms and then w.r.t. some ordering on the relations. The ordering $\sqsubseteq$ is *compatible with parametric parts*, i.e., for parametric parts p, q and constant terms c, d we have $p \sqsubset q \Longrightarrow p+c \sqsubset q+d$. If the above implication holds, we even know that

$$p \sqsubset q \Longrightarrow p + c\ \rho\ 0 \sqsubset q + d\ \sigma\ 0$$

for any relations ρ and σ. This gives rise to the following lemma.

LEMMA 6.3. *Let φ denote the conjunction*

$$p_1 + c_1 \ \rho_1 \ 0 \wedge \ldots \wedge p_m + c_m \ \rho_m \ 0 \quad \textit{with} \quad p_1 + c_1 \sqsubset \cdots \sqsubset p_m + c_m,$$

and let ψ denote the conjunction

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

Suppose that $q_1 \sqsubset p_1$ or $p_m \sqsubset q_n$. Then $\varphi \preceq \psi$ does not hold.

The lemma yields a fast test that can be used as a filter before the actual test for simplifier-recognized implication. The problem with the normalization to primitive parts is that we have to renormalize the left hand sides for smart simplification destroying the ordering of the conjunction.

Boolean normal form computation in propositional logic has been tackled in several papers by Quine (1952, 1955, 1959). He has shown how minimal Boolean normal forms can be obtained. All methods described by Quine have to combine a Boolean variable β with its negation $\neg\beta$ in some way, where the point is that $\beta \vee \neg\beta \Longleftrightarrow$ true. Subsumption is used as test for implication between clauses. In the case of propositional logic this test is even sufficient after some obvious simplifications inside the clauses.

In our context, one has to combine atomic formulas α and their ordering theoretical negations $\overline{\alpha}$ instead, since there are no explicit negations. One can even do more: Pairs of atomic formulas that do not combine to "true" but to one atomic formula are also of interest. In particular, when considering these contractions, via iteration, one also recognizes triplets of atomic formulas that combine to "true."

Currently, we are thinking about contractions involving atomic formulas with different terms combined with simplifier-recognized implication instead of subsumption.

A. Example computations

All computations were done with our REDUCEimplementation on a SUN SPARC-10 using a heap size of $3 \cdot 10^6$ Lisp items.

A.1. A RECTANGLE PROBLEM

The following formula 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.

$$\exists x_1 \exists x_2 \exists y_1 \exists y_2 \exists d_1 \exists d_2 \forall z_1 \forall z_2 \big(a > 0 \wedge b > 0 \wedge 0 \leq x_1 \wedge x_1 < a \wedge 0 \leq x_2 \wedge x_2 < a \wedge 0 \leq y_1 \wedge y_1 < b \wedge 0 \leq y_2 \wedge y_2 < b \wedge 0 < d_1 \wedge d_1 < d_2 \wedge x_1 + d_1 \leq a \wedge y_1 + d_1 \leq b \wedge x_2 + d_2 \leq a \wedge y_2 + d_2 \leq b \wedge (0 \leq z_1 \wedge z_1 < a \wedge 0 \leq z_2 \wedge z_2 < b \longrightarrow (x_1 \leq z_1 \wedge z_1 < x_1 + d_1 \wedge y_1 \leq z_2 \wedge z_2 < y_1 + d_1 \wedge \neg(x_2 \leq z_1 \wedge z_1 < x_2 + d_2 \wedge y_2 \leq z_2 \wedge z_2 < y_2 + d_2) \vee x_2 \leq z_1 \wedge z_1 < x_2 + d_2 \wedge y_2 \leq z_2 \wedge z_2 < y_2 + d_2 \wedge \neg(x_1 \leq z_1 \wedge z_1 < x_1 + d_1 \wedge y_1 \leq z_2 \wedge z_2 < y_1 + d_1)))\big)$$

Using the standard simplifier without theory inheritance, our quantifier elimination procedure takes 5 529 046 ms plus 13 226 ms GC time to compute a quantifier-free formula in a and b. This formula contains 13780 atomic formulas. It can be verified to be contradictive by applying a successive quantifier elimination to its universal closure, which takes 2 074 ms.

With theory inheritance we obtain the result "false" already from the original elimination after only 21 624 ms.

A.2. The x-axis ellipse problem

Volker Weispfenning has shown how elimination set methods can also be applied to non-linear problems (Weispfenning, 1993, Weispfenning, 1994). The following quadratic problem is originally due to Daniel Lazard (1988). For the special case

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

we have computed an elimination result using an implementation by Eva Nolden (1994), which contains the standard simplifier without theory inheritance. The elimination result has 134 atomic formulas. A standard simplifier application yields 85 atomic formulas after 204 ms. We obtain a CNF with 134 atomic formulas after 459 ms. The original elimination result was not in CNF. The Gröbner simplifier reduces the CNF to 107 atomic formulas in 2 227 ms. Applying simplifier-recognized implication does not lead to a better CNF. A branchwise iterative tableau to the elimination result yields 59 atomic formulas in 17 714 ms plus 323 ms GC time.

A.3. Cut without undercutting

This example is taken from (Loos and Weispfenning, 1993). The quantified formula is

$$\exists x \exists y (x > 0 \wedge y < 0 \wedge xr - xt + t = qx - sx + x \wedge xb - xd + d = ay - cy + c).$$

We apply quantifier elimination with different standard simplifier features. Without theory inheritance and smart simplification, the elimination result has 82 atomic formulas (187 ms). Adding smart simplification yields 67 atomic formulas (119 ms). Then adding theory inheritance, i.e. with the complete standard simplifier, one obtains 29 atomic formulas (136 ms).

A.4. The motor series

We summarize an example series that comes from our applications in faulty component detection. We simplify quantifier elimination results of formulas describing a part of motor. The eliminations have been performed using the standard simplifier without theory inheritance. The results are collected in Table 6.

A.5. The stop light circuit series

Our final example is another practical application. This series of formulas describes a stop light circuit. The results are given in Table 7.

Table 6. Motor series

From left to right: subproblem number, input formula, standard simplifier, DNF with subsumption, Gröbner application to this DNF, DNF with simplifier-recognized implication, Gröbner application to this DNF, branchwise iterative tableau. For each problem, the first line gives the number of atomic formulas, and the second gives the computation times. The times do not include GC times, which are about 3–7% with $3 \cdot 10^6$ Lisp items.

no.	input	standard	DNF	Gröbner	good DNF	Gröbner	tableau
1	710	674	3000	164	3000	164	536
		510 ms	14 739 ms	54 366 ms	677 263 ms	53 941 ms	566 372 ms
2	1420	966	2948	604	2948	604	829
		918 ms	17 170 ms	63 121 ms	661 810 ms	61 914 ms	627 572 ms
3	94	88	150	112	88	86	35
		34 ms	238 ms	1 139 ms	1 462 ms	765 ms	3 111 ms
4	292	259	439	257	273	165	203
		153 ms	765 ms	4 454 ms	8 755 ms	2 669 ms	27 693 ms
5	157	139	162	146	102	96	97
		68 ms	289 ms	1 445 ms	1 343 ms	748 ms	8 398 ms
6	994	908	3694	448	3694	448	716
		595 ms	10 931 ms	71 247 ms	644 623 ms	76 296 ms	486 676 ms
7	710	199	425	107	425	107	159
		102 ms	884 ms	5 508 ms	13 719 ms	5 389 ms	48 382 ms
8	473	410	1920	135	1920	135	376
		289 ms	8 024 ms	27 965 ms	303 314 ms	26 996 ms	255 612 ms
9	235	188	389	96	389	96	158
		119 ms	867 ms	4 726 ms	12 393 ms	4 896 ms	38 369 ms
10	478	461	1607	283	1607	283	375
		306 ms	6 681 ms	31 756 ms	190 655 ms	29 478 ms	261 885 ms
11	168	156	308	245	192	159	53
		68 ms	442 ms	3 502 ms	4 624 ms	2 091 ms	8 160 ms
12	2176	2100	6458	1050	6458	1050	756
		1 377 ms	30 464 ms	127 551 ms	2702 473 ms	127 092 ms	1308 932 ms
13	358	342	1189	183	1189	183	251
		221 ms	4 080 ms	17 952 ms	100 623 ms	18 343 ms	201 348 ms
14	710	674	3000	164	3000	164	536
		425 ms	14 603 ms	53 108 ms	672 367 ms	52 819 ms	565 896 ms

Table 7. Stop light circuit

From left to right: subproblem number, input formula, standard simplifier, DNF with subsumption, Gröbner application to this DNF, DNF with simplifier-recognized implication, Gröbner application to this DNF, branchwise iterative tableau, non-brachwise iterative tableau. For each problem, the first line gives the number of atomic formulas, and the second gives the computation times. There was no GC with $3 \cdot 10^6$ Lisp items.

no.	input	standard	DNF	Gröbner	good DNF	Gröbner	bw tableau	non-bw
1	60	43	35	18	35	18	39	39
		17 ms	51 ms	153 ms	136 ms	153 ms	612 ms	731 ms
2	133	40	18	18	18	18	11	11
		17 ms	51 ms	51 ms	102 ms	34 ms	544 ms	544 ms
3	62	46	52	25	47	25	38	39
		17 ms	102 ms	221 ms	255 ms	170 ms	1 037 ms	1 156 ms
4	51	43	26	16	26	16	22	23
		0 ms	68 ms	85 ms	153 ms	68 ms	425 ms	476 ms
5	133	40	26	18	26	18	15	17
		34 ms	68 ms	102 ms	136 ms	85 ms	850 ms	799 ms
6	81	53	41	23	35	19	46	46
		17 ms	85 ms	187 ms	119 ms	153 ms	850 ms	918 ms
7	60	49	45	23	30	19	44	44
		34 ms	102 ms	136 ms	153 ms	119 ms	1 105 ms	1 275 ms
8	54	38	36	18	30	14	33	33
		17 ms	51 ms	153 ms	102 ms	119 ms	578 ms	714 ms
9	63	29	28	18	28	18	27	27
		0 ms	34 ms	102 ms	85 ms	119 ms	527 ms	612 ms
10	34	22	18	14	10	8	17	16
		17 ms	17 ms	68 ms	68 ms	51 ms	238 ms	340 ms
11	92	54	57	28	46	28	47	47
		34 ms	85 ms	204 ms	221 ms	136 ms	833 ms	901 ms
12	33	28	37	25	31	25	24	24
		0 ms	34 ms	187 ms	102 ms	170 ms	357 ms	459 ms

References

Artin, E. (1927). Über die Zerlegung definiter Funktionen in Quadrate. *Hamburger Abhandlungen*, **5**, 100–115.

Becker, T. and Weispfenning, V. (1993). *Gröbner Bases*. New York: Springer-Verlag.

Burhenne, K.-D. (1990). Implementierung eines Algorithmus zur Quantorenelimination für lineare reelle Probleme. Diplomarbeit, Universität Passau, Passau.

Collins, G. E. (1975). Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraical decomposition. In Brakhage, H. (Ed.), *Automata Theory and Formal Languages. 2nd GI Conference*, volume 33 of *LNCS*, (pp. 134–183), Berlin, Heidelberg, New York. Gesellschaft für Informatik, Springer-Verlag.

Collins, G. E. and Hong, H. (1991). Partial Cylindrical Algebraic Decomposition for Quantifier Elimination. *Journal of Symbolic Computation*, **12**(3), 299–328.

Dolzmann, A. (1994a). Simplification of Boolean Combinations of Polynomial Equations. Talk during the workshop Gröbner and related topics in Dagstuhl, Germany.

Dolzmann, A. (1994b). Vereinfachung boolscher Kombinationen polynomialer Gleichungen. Enclosure of a DFG-application.

Hong, H. (1992). Simple solution formula construction in cylindrical algebraic decomposition based quantifier elimination. In Wang, P. S. (Ed.), *Proceedings of the ISSAC 92, Berkeley*, (pp. 177–188), Baltimore, MD. ACM Press.

Kappert, M. (1995). Eliminationsverfahren zur linearen und quadratischen Optimierung. Diplomarbeit, Universität Passau, Passau.

Kredel, H. (1993a). MAS modula-2 algebra system, interactive usage. Technical report, Universität Passau, Passau. Available for anonymous ftp from alice.fmi.uni-passau.de.

Kredel, H. (1993b). MAS modula-2 algebra system, specifications, definition modules, indexes. Technical report, Universität Passau, Passau. Available for anonymous ftp from alice.fmi.uni-passau.de.

Lazard, D. (1988). Quantifier Elimination: Optimal Solution for Two Classical Examples. *Journal of Symbolic Computation*, **5**(1&2), 261–266.

Liska, R. and Steinberg, S. (1995). Using quantifier elimination to test stability. Submitted.

Loos, R. and Weispfenning, V. (1993). Applying linear quantifier elimination. *The Computer Journal*, **36**(5), 450–462. Special issue on computational quantifier elimination.

Nolden, E. (1994). Implementierung eines Quantoreneliminationsverfahrens für quadratische Ungleichungen. Diplomarbeit, Universität Passau, Passau.

Pesch, M. (1994). Die MAS-Implementation des Algorithmus zur Berechnung umfassender Gröbnerbasen. Anlage zu einem DFG-Antrag.

Quine, W. V. (1952). The problem of simplifying truth functions. *American Mathematical Monthly*, **59**, 521–531.

Quine, W. V. (1955). A way to simplify truth functions. *American Mathematical Monthly*, **62**, 627–631.

Quine, W. V. (1959). On cores and prime implicants of truth functions. *American Mathematical Monthly*, **66**, 755–760.

Smullyan, R. M. (1968). *First-order Logic*. Berlin, Heidelberg, New York: Springer-Verlag.

Sturm, T. (1995). *Redlog, a Reduce Library of Algorithms for the Manipulation of First-Order Formulas*. Universität Passau. Unpublished.

Weispfenning, V. (1988). The complexity of linear problems in fields. *Journal of Symbolic Computation*, **5**(1), 3–27.

Weispfenning, V. (1992). Comprehensive Gröbner Bases. *Journal of Symbolic Computation*, **14**, 1–29.

Weispfenning, V. (1993). Quantifier elimination for real algebra—the quadratic case and beyond. Preprint.

Weispfenning, V. (1994). Quantifier elimination for real algebra—the cubic case. In *Proceedings of the Internation Symposium on Symbolic and Algebraic Computation in Oxford*, (pp. 258–263), New York. ACM Press.

Weispfenning, V. (1995). Applying quantifier elimination to problems in simulation and optimization. Submitted.

Yun, D. Y. Y. (1976). On square-free decomposition algorithms. In Jenks, R. D. (Ed.), *Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation*, (pp. 26–35), New York, NY 10036. The Association for Computing Machinery.