



Simplification of Quantifier-free Formulae over Ordered Fields

ANDREAS DOLZMANN[†] AND THOMAS STURM[‡]

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

Given a quantifier-free first-order formula over the theory of ordered fields, our 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 combining elements of field theory, order theory, and logic. These methods do not require a Boolean normal form computation. They have been developed and implemented in REDUCE for simplifying intermediate and final results of automatic quantifier elimination by elimination sets. Their applicability is certainly not restricted to the area of quantifier elimination.

© 1997 Academic Press Limited

1. Introduction

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 (Buchberger and Loos, 1982). A concrete example into which much effort has been spent is the simplification of terms involving nested radicals (Caviness and Fateman, 1976; Borodin *et al.*, 1985; Zippel, 1985). Also the whole Gröbner basis theory has developed from the question whether given polynomials are equal in the residue class ring modulo some ideal (Buchberger, 1965).

From a mathematical point of view the symbolic manipulation of terms extends fairly naturally to that of quantifier-free formulae and further to that of first-order formulae. The well-known problem of finding simpler counterparts occurs then for formulae instead of terms. Since quantifier-free formulae are certainly simpler than quantified ones, quantifier elimination procedures such as partial CAD (Collins, 1975; Collins and Hong, 1991) or elimination set methods (Weispfenning, 1988, 1994, 1997b, Loos and Weispfenning, 1993) can be regarded as simplification. In this paper, we will discuss simplification methods for quantifier-free formulae.

Surprisingly, little work has been done in extending computer algebra systems by formulae and algorithms on the latter. An important exception is certainly the SAC/ALDES system (Collins, 1968, 1985), which has been used for the implementation of CAD. The

[†] E-mail: Andreas.Dolzmann@fmi.uni-passau.de

[‡] E-mail: sturm@fmi.uni-passau.de

quantifier-free formulae obtained there from a cell decomposition are already Boolean normal forms. Consequently, not much effort has been spent in the simplification of general quantifier-free formulae. Both MAPLE and REDUCE provide packages for propositional calculus. However, propositional logic does not provide the extension from algebraic terms to formulae discussed above. The formulae there are simply terms of a new kind, namely terms over a Boolean algebra.

The simplification algorithms described in this paper have been developed with the implementation of quantifier elimination by elimination set methods. There, in contrast to CAD, one typically obtains deeply nested, highly redundant formulae that have to undergo some sophisticated simplification before providing useful information. Besides such sophisticated simplification methods, it is crucial to have a fast simplification for the intermediate results at hand. Our simplification techniques have made the elimination set methods applicable to examples in industrial simulation and optimization (Weispfenning, 1997a).

The scope of formulae 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:

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

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 difficult problems.

The mathematical principles underlying our simplifiers 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 the different ideas from algebra and logic to simplifiers in such a way that they produce formulae that cannot be further simplified by themselves. 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 formulae in propositional calculus (Brayton *et al.*, 1984, 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 formulae on different Boolean levels, it has turned out that this concept also captures 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 (Dolzmann and Sturm, 1996a, b), which currently focuses on simplification and quantifier elimination.

2. Basic Definitions and Concepts

Our formulae combine atomic formulae 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 formulae are *equations* constructed with “ $=$,” *inequalities* constructed with “ \neq ,” *strong orders* constructed with “ $<$ ” and “ $>$,” and *weak orders* constructed with “ \leq ” and “ \geq ”. 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 formulae. For the terms we do as usual not use the language of fields but that of rings. Abbreviating variable-free subterms by integers, every term can then be written as a multivariate polynomial over \mathbb{Z} w.r.t. some fixed term order. Moreover, we may consider all right-hand sides of atomic formulae to be zero.

Non-atomic formulae are called *complex*. We divide the complex formulae into *flat* formulae and *deep* formulae. Flat formulae combine atomic formulae to one Boolean level. Examples are conjunctions of atomic formulae or implication between two atomic formulae. *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.

It is not obvious which formulae should be considered simple. We summarize our *simplification goals*.

Few atomic formulae 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 formulae 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 formulae or case distinctions.

Few different atomic formulae 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.

Small satisfaction sets of the contained atomic formulae. This leads to a formula that is less redundant. If we know e.g. that $a \leq 0$ for some reason, we can replace $a \neq 0$ by $a < 0$, which holds for less field elements.

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

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

Our final claim is not to the simplification result but to the procedure itself: Simplifiers should be *idempotent*.

Some of the simplification goals given above contradict one another. For these cases, the simplifiers are parameterized such 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 formulae” we have the possibility of selecting *no expansion*, *expand always*, or *expand if operator matches*. The latter selection never violates the simplification goal of a comprehensible Boolean structure. These switches only toggle 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 formulae considered conjunctive. The target formula φ is simplified w.r.t. Θ . For this purpose, we consider the variables in our atomic formulae 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 :

$$\Theta \models (\varphi \iff \varphi').$$

As an example consider $\{a \leq 0\} \models (a < 0 \iff a \neq 0)$. The theory parameter allows us 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.

The simplifiers typically start with simplifying the input theory treating it as a conjunction. If they detect this way that the theory is inconsistent, they raise an error. Under an inconsistent theory any two formulae 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 formulae 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 formulae. There the theory—possibly empty in the beginning—is implicitly enlarged during recursion.

3. Atomic Formulae

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 formulae are treated in the next section.

Variable-free atomic formulae are evaluated to truth values. In other atomic formulae the left-hand side polynomial is replaced by its primitive part over \mathbb{Z} with positive leading coefficient w.r.t. 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.

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 $((p_1, 1), \dots, (p_n, n))$ where the p_i are primitive over \mathbb{Z} with positive leading coefficient. 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

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

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

PROPOSITION 3.1. *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) $f = 0 \iff F = 0 \iff p = 0 \vee q = 0$
- (ii) $f \neq 0 \iff F \neq 0 \iff p \neq 0 \wedge q \neq 0$
- (iii) $f > 0 \iff pq^2 > 0 \iff p > 0 \wedge q \neq 0$
- (iv) $f \geq 0 \iff pq^2 \geq 0 \iff p \geq 0 \vee q = 0$
- (v) $f < 0 \iff pq^2 < 0 \iff p < 0 \wedge q \neq 0$
- (vi) $f \leq 0 \iff pq^2 \leq 0 \iff p \leq 0 \vee q = 0$.

The decision of 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 formulae. In addition, the expansions can complicate the Boolean structure. To overcome this, 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.

3.2. SEMI-DEFINITENESS AND DEFINITENESS TESTS

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

PROPOSITION 3.2. *For positive definite $f \in \mathbb{Z}[\underline{X}]$ we have*

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

In case that f is positive semi-definite, we have

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

Notice that in the last two cases f can be replaced by its squarefree part according to Proposition 3.1. 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 semi-definiteness, i.e. deciding $\forall(f > 0)$ or $\forall(f \geq 0)$ 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 (1927), exactly positive semi-definite polynomials can be written as sums 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.

PROPOSITION 3.3.

- (i) STSQ's are positive definite, and TSQ's are positive semi-definite.
- (ii) A polynomial is positive definite if its squarefree part is an STSQ. It is positive semi-definite if its squarefree part is a TSQ.
- (iii) A polynomial with parity decomposition (p, q) is positive semi-definite if p is a TSQ.

Obviously, none of the above tests is a necessary condition. We give some examples for the relevance of the various tests.

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. 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 (Hong *et al.*, 1993).

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 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 cancellations, e.g. $a^2 + b^2$.

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 3.4. *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 and sums.

Part (i) has two interesting consequences. Firstly, 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.

Secondly, a squarefree decomposition does not yield more information than a parity decomposition (p, q) : Test (iii) of Proposition 3.3 could be extended to squarefree decompositions by testing all odd-degree squarefree factors on being TSQ's. Part (i) of Proposition 3.4 shows that whenever this test succeeds, p is already a TSQ.

3.3. SPLITTING OF TSQ'S

In Proposition 3.2 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:

$$\sum s_i \leq 0 \iff \sum s_i = 0 \iff \bigwedge s_i = 0, \quad \sum s_i > 0 \iff \sum s_i \neq 0 \iff \bigvee s_i \neq 0.$$

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

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 (1976).

The multiplicative splitting of terms in Proposition 3.1 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.

4. Flat Formulae

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

One can imagine to simplify flat formulae 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.

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 formulae φ .

$$\begin{aligned} \neg\text{true} &\iff \text{false}, & \neg\text{false} &\iff \text{true}, \\ \text{false} \implies \varphi &\iff \varphi \implies \text{true} \iff \varphi \implies \varphi \iff \text{true}, \\ \text{true} \implies \varphi &\iff \varphi, & \varphi \implies \text{false} &\iff \neg\varphi, \\ \varphi \iff \text{true} &\iff \varphi, & \varphi \iff \text{false} &\iff \neg\varphi, & \varphi \iff \varphi &\iff \text{true}, \\ \varphi \wedge \text{true} &\iff \varphi \wedge \varphi \iff \varphi, & \varphi \wedge \text{false} &\iff \text{false}, \\ \varphi \vee \text{false} &\iff \varphi \vee \varphi \iff \varphi, & \varphi \vee \text{true} &\iff \text{true}. \end{aligned}$$

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

Table 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
$t > 0$						$t > 0$

4.2. SMART SIMPLIFICATION

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

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

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 formulae 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 formulae 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

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

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

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

Given an n -ary conjunction, the simplification result is invariant w.r.t. the order in which these (binary) simplifications are performed. In other words: one cannot make a mistake when contracting the atomic formulae 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 come to a unique result, either “false”—then we are finished—or

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

a conjunction γ . The simplified conjunction is obtained from γ by extracting all atomic formulae 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 deep simplification.

The result obtained with the above methods meets the simplification goal of small satisfaction sets for the atomic formulae. 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 formulae by de Morgan’s law. Then we proceed as with conjunctions. Finally we negate the simplified conjunction back. This leads to atomic formulae 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 formulae is resolved into the disjunction $\bar{\alpha} \vee \beta$. 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 w.r.t. the theory.

Equivalences are resolved into deep formulae 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 formulae, 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.

4.3. GRÖBNER BASIS METHODS

Gröbner basis methods allow us to take advantage of certain algebraic interactions between the atomic formulae 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 w.r.t. 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 formulae. For finite families $\{a_i\}_{i \in I}$ we write $\{a_i\}$ for short. In contrast to all other simplifications described in this paper, here both the simplifier's performance and the simplification results depend on the chosen term order. The following proposition states the mathematical background for the method we use. By $\text{rad}(M)$ we denote the radical of the ideal generated by a set M of polynomials over a field K .

PROPOSITION 4.1. *Let $\{f_i\}$, $\{g_j\}$, $\{\tilde{f}_k\}$, and $\{\tilde{g}_j\}$ be finite subsets of $K[\underline{X}]$. Suppose further that $\text{rad}(\{f_i\}) = \text{rad}(\{\tilde{f}_k\})$ and that $g_j \equiv \tilde{g}_j \pmod{\text{rad}(\{f_i\})}$ for each j . Then*

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

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

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

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

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 $x - z$ w.r.t. 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 4.1 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 3 can decide $h \rho_j 0$. 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 (Becker *et al.*, 1993). If $g_j \in \text{rad}(\{f_i\})$ 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 formulae 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 subformulae 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 formulae are in general parts of complex formulae, reduction can increase the number of different atomic formulae. 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 formulae.

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

PROPOSITION 4.2. *Let $\rho_j \in \{<, >\}$, let $\tilde{\rho}_j$ denote the weak counterpart of ρ_j , and let σ_k be any of our relations. Then the following are equivalent:*

- (i) $\bigwedge_i p_i \neq 0 \wedge \bigwedge_j q_j \rho_j 0 \wedge \bigwedge_k r_k \sigma_k 0$
- (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 $\text{Id}(\{f_i\})$ need not be 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 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 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 4.2 enlarging the chance for a successful radical membership test.

4.4. HISTORY AND IMPLEMENTATION

Smart simplification developed from the pure order theoretic approach over the parametric part splitting to involving theories. Contracting atomic formulae whose terms are identical monic variables but with different absolute summands has already been indicated by Hong (1992). 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 (1965). Based on ideas by Becker, Pesch, and 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 con-

Table 4. Multiplicative smart simplification.

\wedge	$st = 0$	$st \rho 0, \rho \in \{\leq, \geq\}$	$st \neq 0$	$st \rho 0, \rho \in \{<, >\}$
$t = 0$	$t = 0$	$t = 0$	false	false
$t \leq 0$	—	—	$t < 0 \wedge s \neq 0$	$t < 0 \wedge 0 \rho s$
$t \geq 0$	—	—	$t > 0 \wedge s \neq 0$	$t > 0 \wedge s \rho 0$
$t \neq 0$	$t \neq 0 \wedge s = 0$	—	$st \neq 0$	$st \rho 0$
$t < 0$	$t < 0 \wedge s = 0$	$t < 0 \wedge 0 \rho s$	$t < 0 \wedge s \neq 0$	$t < 0 \wedge 0 \rho s$
$t > 0$	$t > 0 \wedge s = 0$	$t > 0 \wedge s \rho 0$	$t > 0 \wedge s \neq 0$	$t > 0 \wedge s \rho 0$

taining only equations and inequalities. The implementation was done by Pesch (1994) in the CGB-package of the computer algebra system MAS by Kredel (1993a, b).

Two further MAS implementations for simplifying Boolean normal forms were done by the first author (Dolzmann, 1994a, b). 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.

4.5. OUTLOOK

We have introduced order theoretical contraction of atomic formulae 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 $s \rho 0 \wedge t \sigma 0$, one can check if s divides t or vice versa. Let w.l.o.g. $t = rs$, then simplification of terms, reduction of the number of atomic formulae, or evaluations to truth values are possible in many cases (cf. Table 4). We give some examples: $xy \geq 0 \wedge x < 0 \iff y \leq 0 \wedge x < 0$,

$$xy \geq 0 \wedge x = 0 \iff x = 0, \quad xy \neq 0 \wedge x = 0 \iff \text{false}.$$

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

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

This kind of simplification does not involve any factorization. Extreme special cases have been mentioned by Hong *et al.* (1997). 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.

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

5. Deep Formulae

To begin with, recall that the Boolean simplification rules of Subsection 4.1 hold for arbitrary formulae. They are, of course, applied during the recursion process described below. In particular, we check for identical subformulae. In contrast to the atomic formula situation, the time required for this cannot be neglected.

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

$$\varphi \wedge (\dots \psi \dots) \iff \varphi \wedge (\dots (\varphi \wedge \psi) \dots), \quad \varphi \vee (\dots \psi \dots) \iff \varphi \vee (\dots (\neg \varphi \wedge \psi) \dots)$$

and corresponding dual variants

$$\varphi \wedge (\dots \psi \dots) \iff \varphi \wedge (\dots (\neg \varphi \vee \psi) \dots), \quad \varphi \vee (\dots \psi \dots) \iff \varphi \vee (\dots (\varphi \vee \psi) \dots).$$

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 formulae located on 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 $\bigwedge \Gamma \wedge \psi$ or $\bigvee \Gamma \vee \psi$, where Γ is the set of toplevel atomic formulae of the considered one. We extend the implicit negation $\bar{\alpha}$ of atomic formulae α introduced in Subsection 4.2 to the sets Γ of atomic formulae.

PROPOSITION 5.2. *Let Θ be a theory, let Γ be a finite set of atomic formulae, and let ψ be a formula.*

(i) *Let ψ' be such that $\Theta \cup \Gamma \models (\psi \iff \psi')$. Then*

$$\Theta \models (\bigwedge \Gamma \wedge \psi \iff \bigwedge \Gamma \wedge \psi').$$

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

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 run through recursively. On every Boolean level we enlarge the theory in dependence on the Boolean operator of the corresponding level. In conjunctions all atomic formulae are added. In disjunctions the negations of all atomic formulae 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 particular Boolean level φ of the formula is simplified. We have obtained a theory Θ consisting of user input enlarged by possibly negated atomic formulae from higher levels.

- (i) Update Θ to Θ' w.r.t. the toplevel atomic formulae of φ .
- (ii) Simplify the complex constituents w.r.t. Θ' .
- (iii) If new toplevel atomic formulae come into existence in step (ii), then update Θ' w.r.t. these and go to (ii).
- (iv) Use methods as described in Section 4 for simplifying the toplevel atomic formulae present now w.r.t. the original theory Θ .

Step (iii) and hence the loop is necessary for making the simplifier idempotent. Notice that new atomic formulae can come into existence by simplifying a complex formula to another one with a matching toplevel operator. In the implementation we, of course, abort step (ii) when new atomic formulae occur.

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.

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.

5.3. ILLUSTRATING EXAMPLES

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

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

The initial theory for the toplevel complex subformulae 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 w.r.t. the simplification goal of small satisfaction sets. As final result we obtain $a = 0 \wedge (b = 0 \vee (c = 0 \wedge d > 0)) \wedge d \neq 0$.

5.4. OUTLOOK AND IMPLEMENTATION

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

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 formulae would offer more possibilities. Allowing the theory to contain possibly negated flat formulae 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 4. The implementation is not idempotent yet.

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.

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

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

where A is a set of atomic formulae. 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 $\alpha \in A$ 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

$$(\alpha_0 \wedge \varphi_{\alpha_0}) \vee \bigvee_{\alpha \in A_1} (\alpha \wedge \varphi_{\alpha_0}) \vee \bigvee_{\alpha \in A_2} (\alpha \wedge \varphi_\alpha) \quad \text{with} \quad A = \{\alpha_0\} \cup A_1 \cup A_2.$$

Applying a law of distributivity, this can be simplified to

$$\left(\left(\alpha_0 \vee \bigvee_{\alpha \in A_1} \alpha \right) \wedge \varphi_{\alpha_0} \right) \vee \bigvee_{\alpha \in A_2} (\alpha \wedge \varphi_{\alpha}).$$

This is a simplification that our deep simplifier does not know. We call it *contraction* of tableau branches. Notice that afterwards the flat disjunction $\alpha_0 \vee \bigvee_{\alpha \in A_1} \alpha$ can be simplified using the methods of Section 4.

Good candidates for 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 A of such a form a *tableau step* w.r.t. t .

There is an *automatic* tableau, which tries tableau steps w.r.t. 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 formulae 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.

6.2. 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 Loos (Loos and Weispfenning, 1993, Weispfenning, private communication) and firstly implemented by Burhenne (1990). This version performed tableau steps w.r.t. a given term t without contraction of branches. The simplifications in the branches were restricted to deciding atomic formulae with t as their left-hand side.

Before our deep simplifier performed the theory inheritance described in Section 5, the iterative tableau method provided considerable simplifications in many cases. Meanwhile, there are only few formulae that can be simplified via the tableau method after simplification with the standard simplifier.

6.3. OUTLOOK

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

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

for a set A of atomic formulae. One would then define a tableau step w.r.t. a term t as taking $A = \{t \geq 0, t \neq 0, t \leq 0\}$. Since these atomic formulae 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 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 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 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 formulae. One thus has to define criteria for finding the *simplest* formula obtained this way.

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 formulae than the input formula. We restrict our attention to DNF computations. The computation of a CNF is dual to this.

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 formulae as trivial conjunctions. Assuming that flat formulae 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 7.1. *For $i = 1, \dots, m$ and $j = 1, \dots, n_i$ let γ_{ij} be conjunctions of atomic formulae. Set $N = \{1, \dots, n_1\} \times \dots \times \{1, \dots, n_m\}$. Then*

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

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

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

7.2. SIMPLIFICATION OF BOOLEAN NORMAL FORMS

In addition to the simplification methods already presented there are 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

$$(\alpha_1 \wedge \dots \wedge \alpha_n \wedge t \rho \ 0) \vee (\alpha_1 \wedge \dots \wedge \alpha_n \wedge t \sigma \ 0) \iff (\alpha_1 \wedge \dots \wedge \alpha_n \wedge \tau),$$

where τ is the result of the smart simplification of $t \rho 0 \vee t \sigma 0$. 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 7.2. *Let φ and ψ be formulae 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 formulae. 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 $\varphi \implies \psi$ for conjunctions φ and ψ of atomic formulae.

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

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

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 formulae contained in φ and ψ respectively. This idea treats atomic formulae like propositional logical variables. In our situation, subsumption can be extended in the following way: We define

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

iff $I \supseteq J$ and for all $j \in J$ the smart simplification of Section 4.2 simplifies $t_j \overline{\rho_j} 0 \vee t_j \sigma_j 0$ to “true”. As an example consider

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

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

A smarter though less efficient choice is *simplifier-recognized implication* “ \succeq_{rec} ,” which once more makes use of our theory concept. Formally it is based on the equivalence between

$$\Theta \models \varphi \quad \text{and} \quad \models (\bigwedge \Theta \implies \varphi).$$

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

We define that $\varphi \succeq_{\text{rec}} \psi$ if ψ can be simplified to “true” with the atomic formulae 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 $\varphi \succeq_{\text{rec}} \psi$ 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.

7.3. 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 formulae 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 it is extremely time consuming. The best Boolean normal forms are currently obtained by applying the Gröbner simplifier.

7.4. OUTLOOK

In the outlook of Section 4 we have already indicated the alternative of making the terms monic instead of primitive. Recall from Subsection 4.1 how the atomic formulae 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 formulae 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

$$p \sqsubseteq q \implies p + c \rho \ 0 \sqsubseteq q + d \sigma \ 0.$$

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 4.2 performs simplifications only between atomic formulae 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 7.3. *Let φ denote the conjunction*

$$p_1 + c_1 \rho_1 \ 0 \wedge \dots \wedge p_m + c_m \rho_m \ 0 \quad \text{with} \quad p_1 + c_1 \sqsubseteq \dots \sqsubseteq p_m + c_m,$$

and let ψ denote the conjunction

$$q_1 + d_1 \sigma_1 \ 0 \wedge \dots \wedge q_n + d_n \sigma_n \ 0 \quad \text{with} \quad q_1 + d_1 \sqsubseteq \dots \sqsubseteq q_n + d_n.$$

Suppose that $q_1 \sqsubseteq p_1$ or $p_m \sqsubseteq q_n$. Then $\varphi \succeq_{\text{rec}} \psi$ 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 w.r.t. many natural orders on terms.

Boolean normal form computation in propositional logic has been tackled in several papers by Quine (1952, 1955, 1959) and McCluskey (1956). 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 $\beta \vee \neg\beta \iff \text{true}$. 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.

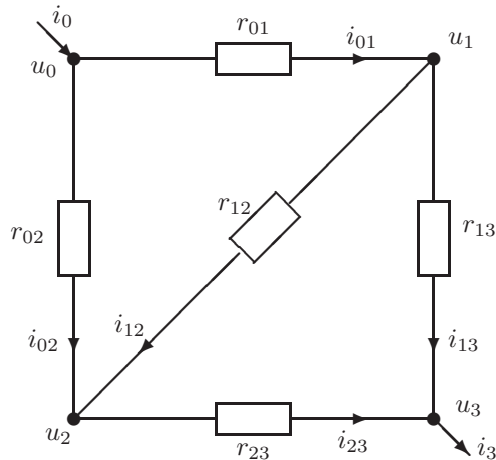


Figure 1. An electrical network.

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

8. Example Computations

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

8.1. 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 formulae. 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

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

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

8.2. AN ELECTRICAL NETWORK

By quantifier elimination we compute for the electrical network in Figure 1 a quantifier-free formula which describes the current i_{12} in terms of the resistances and the voltage $u_3 - u_0$. With the standard simplifier we obtain 31 atomic formulae in 0.5 s. If we turn off the additive smart simplification described in Subsection 4.2, we obtain 47 atomic formulae in 0.8 s.

Table 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 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

8.3. 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 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 7 gives some exemplary timings.

Here the theory inheritance does not play such an important role as in Example 8.1. 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 but it is excessively time consuming. The tableau method is irrelevant for most cases and is also extremely time consuming, in particular for large input formulae. There is no relation between the size of the input formula and the method of choice.

All input formulae 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 formulae of such sizes as in the example only if the latter are not too deeply nested.

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

9. Conclusions

We have discussed the problem of simplifying quantifier-free formulae over ordered fields. In Section 2 we have, besides some common definitions, specified what kind of formulae 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 formulae. The flat simplification methods (Section 4), namely smart simplification and the Gröbner method make use of the theory, while the deep simplification method (Section 5) constructs an implicit theory inheriting it to deeper Boolean levels. The tableau methods (Section 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 formulae (Section 3) and the Boolean methods (Subsection 4.1) and smart simplification (Subsection 4.2) for flat formulae. Adding the Gröbner method (Subsection 4.3) for flat formulae to the standard simplifier yields an advanced simplifier. Further examples for advanced simplifiers are the tableau methods (Section 6) and our simplifying Boolean normal form computation (Section 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). Secondly, time consuming simplification steps can be turned off, such as factorization (cf. Subsection 3.4), several features of the Gröbner method (cf. Subsection 4.3), or checking for simplifier-recognized implication with Boolean normal form computations (cf. Subsection 7.2). Thirdly, one can turn off methods that may be disadvantageous such as branchwise iteration with the iterative tableau (cf. Subsection 6.1).

We have implemented our simplification methods within our REDUCE package REDLOG Dolzmann and Sturm (1996a, b). 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 8).

Note

The REDLOG package is freely available for non-commercial use. The current Version 1.0 can be obtained on <http://www.fmi.uni-passau.de/~redlog/>.

References

- Artin, E. (1927). Über die Zerlegung definiter Funktionen in Quadrate. *Hamburger Abhandlungen*, **5**:100–115.
- Becker, T., Weispfenning, V., Kredel, H. (1993). *Gröbner Bases, a Computational Approach to Commutative Algebra*, volume 141 of *Graduate Texts in Mathematics*. New York, Springer.
- Borodin, A., Fagin, R., Hopcroft, J. E., Tompa, M. (1985). Decreasing the nesting depth of expressions involving square roots. *J. Symbolic Computation*, **1**(2):169–188.
- Brayton, R. K., Hachtel, G. D., McMullen, C. T., Sangiovanni-Vincentelli, A.L. (1984). *Logic Minimization Algorithms for VLSI Synthesis*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, The Hague, Dordrecht, Lancaster.
- Buchberger, B. (1965). *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. Doctoral dissertation, Mathematical Institute, University of Innsbruck, Innsbruck, Austria.
- Buchberger, B., Loos, R. (1982). Algebraic simplification. In Buchberger, B., Collins, G. E., Loos, R., Albrecht, R., editors, *Computer Algebra: Symbolic and Algebraic Manipulation*, pages 11–44. Wien, New York, Springer-Verlag, second edition.

- Burhenne, K.-D. (1990). Implementierung eines Algorithmus zur Quantorenelimination für lineare reelle Probleme. Diploma thesis, Lehrstuhl Prof. Dr. V. Weispfenning, Universität Passau.
- Caviness, B. F., Fateman, R. J. (1976). Simplification of radical expressions. In Jenks, R. D., editor, *Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation*, pages 329–338, New York.
- Collins, G. E. (1968). The SAC-1 polynomial system. Technical Report 115, Computer Science Department, University of Wisconsin, Madison, Wisconsin.
- Collins, G. E. (1975). Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In Brakhage, H., editor, *Automata Theory and Formal Languages. 2nd GI Conference*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183, Berlin, Heidelberg, New York. Gesellschaft für Informatik, Springer-Verlag.
- Collins, G. E. (1985). The SAC-2 computer algebra system. In Caviness, B. F., editor, *EUROCAL '85; Proceedings of the European Conference on Computer Algebra*, number 104 in *Lecture Notes in Computer Science*, pages 34–35, Berlin, Heidelberg, New York, Tokyo. Springer.
- Collins, G. E., Hong, H. (1991). Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symbolic Computation*, **12**(3):299–328.
- Dolzmann, A. (1994a). Simplification of Boolean combinations of polynomial equations. Talk during the workshop “Gröbner bases and related topics” in Dagstuhl, Germany.
- Dolzmann, A. (1994b). Vereinfachung boolescher Kombinationen polynomialer Gleichungen. Enclosure of a DFG application.
- Dolzmann, A., Sturm, T. (1996a). Redlog—computer algebra meets computer logic. Technical Report MIP-9603, FMI, Universität Passau, D-94030 Passau, Germany.
- Dolzmann, A., Sturm, T. (1996b). Redlog user manual. Technical Report MIP-9616, FMI, Universität Passau, D-94030 Passau, Germany. Edition 1.0 for Version 1.0.
- Hong, H. (1992). Simple solution formula construction in cylindrical algebraic decomposition based quantifier elimination. In Wang, P. S., editor, *Proceedings of the ISSAC 92, Berkeley*, pages 177–188, Baltimore, MD. ACM Press.
- Hong, H., Collins, G. E., Johnson, J. R., Encarnacion, M. J. (1993). QEPCAD interactive version 12. Kindly communicated to us by Hoon Hong.
- Hong, H., Liska, R., Steinberg, S. (1997). Testing stability by quantifier elimination. *J. Symbolic Computation*, **24**:161–187.
- 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`.
- Loos, R., Weispfenning, V. (1993). Applying linear quantifier elimination. *The Computer Journal*, **36**(5):450–462. Special issue on computational quantifier elimination.
- McCluskey, E. J. (1956). Minimization of Boolean functions. *Bell Systems Technical Journal*, **35**:1417–1444.
- Pesch, M. (1994). Die MAS-Implementation des Algorithmus zur Berechnung umfassender Gröbnerbasen. Enclosure of a DFG application.
- 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*. Springer-Verlag, Berlin, Heidelberg, New York.
- Weispfenning, V. (1988). The complexity of linear problems in fields. *J. Symbolic Computation*, **5**(1):3–27.
- Weispfenning, V. (1992). Comprehensive Gröbner bases. *J. Symbolic Computation*, **14**:1–29.
- Weispfenning, V. (1994). Quantifier elimination for real algebra—the cubic case. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation in Oxford*, pages 258–263, New York. ACM Press.
- Weispfenning, V. (1997a). Simulation and optimization by quantifier elimination. Technical Report MIP-9607, FMI, Universität Passau, D-94030 Passau, Germany. *J. Symbolic Computation*, **24**:189–208.
- Weispfenning, V. (1997b). Quantifier elimination for real algebra—the quadratic case and beyond. *AAECC* **8**(2):85–101.
- Yun, D. Y. Y. (1976). On square-free decomposition algorithms. In Jenks, R. D., editor, *Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation*, pages 26–35, New York, NY 10036. The Association for Computing Machinery.
- Zippel, R. (1985). Simplification of expressions involving radicals. *J. Symbolic Computation*, **1**(2):189–210.

Originally received 10 October 1995
Accepted 4 December 1996