High performance Simplification-Based Automated Deduction [*]

Maria Paola Bonacina **Jieh Hsiang**
Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794-4400
{bonacina,hsiang}@sbcs.sunysb.edu

1 Introduction

Equational logic is one of the most important domains of research in computer science. Specifications of types of data structures and assertions about the behaviour of programs are naturally written in equational form. Programs made of equations are called *equational programs* and appear in functional programming, logic programming and in most combinations of high level programming paradigms [19, 23]. First order logic can be expressed equationally [20]. This formulation makes it possible to express logic programming equationally and to employ the computational model of equational languages in logic programming [7]. Set theory can also be expressed equationally [33], enabling one to reason about query languages and optimization in data bases [11].

Such a wide range of applications, not to mention the traditional applications to algebra, makes automated deduction in equational logic an important subject of research. However, the seemingly insurmountable search space caused by the symmetry and replacement properties of the equality predicate had been a serious obstacle which baffled researchers in automated deduction for several decades. It is not until very recently that methods capable of effectively reason with equations have been designed and successfully applied to an interesting range of challenging problems. These methods are based on the *term rewriting* approach to equational reasoning, which was started in [24].

The key idea in term rewriting based theorem proving is to regard a derivation as a process of *proof reduction.* Equations are oriented into rules according to a *well-founded ordering*, and equational replacement is performed only in one direction. When an expression (term, equation, clause) is *simplified* by a rule, the old expression is discarded and replaced by the new one, which is smaller in the ordering. The generation of new lemmas, the *superposition process*, is also done according to the ordering. By keeping every piece of data fully simplified at all time, the search space is *drastically* reduced.

Section 2 presents in greater detail the simplification-based theorem proving approach, according to the theoretical framework which we have proposed in [8]. Section 3 describes our theorem prover *SBR3*, which implements the simplification-based methodology. Section 4 relates some original proofs obtained automatically with *SBR*3. The last section is devoted to some discussion on our current work on *distributed theorem proving.*

[*]Research supported in part by grants CCR-8805734 and CCR-8901322, funded by the National Science Foundation. The first author is also supported by a scholarship of Universitá degli Studi di Milano, Italy.

2 Simplification-based automated deduction

A *theorem proving problem* consists in finding a proof of a given sentence φ in a given set of axioms S. The set S is a *presentation* of the theory $Th(S)$ of all the theorems of S, $Th(S) = \{\psi \mid S \models \psi\}$. For instance, in equational logic, S is a set of equations E, the axioms for an equational theory. The sentence φ to be proved is the *target* or *goal*. In equational theorem proving, the target is an equation $\forall\bar{x}s \simeq t$, where all variables are universally quantified. We write $(S;\varphi)$ to denote the problem of proving φ from S.

The first component of a *theorem proving strategy* $\mathcal{C}$ is a set I of *inference rules*. An application of an inference rule to $(S;\varphi)$ transforms it into another problem: $(S;\varphi)\vdash_I(S';\varphi')$. Clearly, the two problems must be equivalent. This is ensured by requiring that for all inference steps $(S;\varphi)\vdash_I(S';\varphi')$, the theory of S' is not larger than the theory of S, i.e. $Th(S') \subseteq Th(S)$, and $\varphi \in Th(S)$ if and only if $\varphi' \in Th(S')$. We have termed these two properties *monotonicity* and *relevance* respectively.

An inference mechanism I defines for every given input $(S_0;\varphi_0)$ the *space* of all the problems or *states* $(S;\varphi)$, which can be derived from $(S_0;\varphi_0)$ by I in zero or more steps. This space can be represented as a tree, where the nodes are labeled by pairs $(S;\varphi)$, the root is labeled by $(S_0;\varphi_0)$ and there is an arc from node $(S;\varphi)$ to node $(S';\varphi')$ if and only if $(S;\varphi)\vdash_I(S';\varphi')$. We call this tree the *I-tree rooted at* $(S_0;\varphi_0)$, because it is determined by the inference mechanism I and the input problem $(S_0;\varphi_0)$. Accordingly, a sequence of inference steps $(S;\varphi)\vdash_I(S';\varphi')$ is an *I-path*. In general, the I-tree is a directed graph, rather than a tree, since a node $(S;\varphi)$ may be reachable starting from the root by more than one I-path. However, it is always possible to transform it into a tree by allowing different nodes to have the same label.

If φ_0 is indeed a theorem of S_0, i.e. $\varphi_0 \in Th(S_0)$, the inference mechanism I should be able to prove it. This is the intuitive meaning of the *refutational completeness* of an inference system. Refutational completeness can be described on the I-tree as follows: I is refutationally complete if and only if, whenever $\varphi_0 \in Th(S_0)$, the I-tree rooted at $(S_0;\varphi_0)$ contains at least a node labeled by a *successful state* $(S;true)$, i.e. a state where the target is proved.

If our theorem proving strategy $\mathcal{C}$ has a refutationally complete inference mechanism, we know that for every true input target, we can find a proof. However, ensuring that the inference rules are sufficiently powerful to prove all theorems is just the beginning. We now face the problem of *searching* the I-tree to reach a solution. Thus, the second component of a strategy $\mathcal{C}$ is a *search plan* Σ: $\mathcal{C} =< I;\Sigma >$. Given the input state $(S_0;\varphi_0)$, Σ selects an inference rule f in I and a tuple of premises $\bar{x}$ in $S_0 \cup \{\varphi_0\}$. The first step consists then in applying f to $\bar{x}$, generating a new state $(S_1;\varphi_1)$. Choosing an inference step corresponds to choosing one of the arcs leaving node $(S_0;\varphi_0)$ in the I-tree. The process is repeated, generating a *derivation*

$$(S_0;\varphi_0)\vdash_{\mathcal{C}}(S_1;\varphi_1)\vdash_{\mathcal{C}}\ldots(S_i;\varphi_i)\vdash_{\mathcal{C}}\ldots,$$

where at each step an inference is performed according to the search plan. The derivation computed by $\mathcal{C}$ on input $(S_0;\varphi_0)$ is the unique I-path selected by Σ in the I-tree rooted at $(S_0;\varphi_0)$. A derivation is successful if it reaches a successful node $(S;true)$.

The refutational completeness of I guarantees that successful derivation exist. We need another property to ensure that the specific derivation computed by $\mathcal{C}$ is successful. This property is the *fairness* of the search plan: Σ is fair if and only if, whenever the I-tree rooted at $(S_0;\varphi_0)$ contains successful nodes, the derivation controlled by Σ finds one. The refutational completeness of the inference rules and the fairness of the search plan together imply the *completeness* of the strategy $\mathcal{C}$: whenever $\varphi_0 \in Th(S_0)$, the computation by $\mathcal{C}$ halts successfully. In other words, $\mathcal{C}$ is a *semidecision procedure* for theorem proving.

Meeting the completeness requirement alone is not difficult. Many refutationally complete inference systems are known and a search plan which tries exhaustively all steps is trivially fair. The more challenging question of automated deduction is to obtain a strategy which is both complete and *efficient*: not only should the strategy succeed, but it should also do it by consuming "reasonable" amounts of resources, i.e. time and memory. The notion of efficiency is clearly not an absolute one. Rather, it can be used for comparisons. Informally, given two complete strategies $\mathcal{C}_1$ and $\mathcal{C}_2$, a problem $(S_0; \varphi_0)$ and a fixed amount of memory (elapse of time), $\mathcal{C}_1$ is more efficient in time (in memory) than $\mathcal{C}_2$ on problem $(S_0; \varphi_0)$, if it reaches a solution in shorter time (using a smaller amount of memory).

The issue of efficiency can, and in fact should, be considered at both the inference level and the search level. At the inference level, the goal is to devise inference mechanisms which generate "small" search spaces, while preserving refutational completeness. It is desirable that the search space is small, since searching a small space is intuitively easier than searching a large one, but not at the expense of losing all the solution nodes! Similarly, at the search level, the goal is to design search plans which find "fast" solutions, while preserving fairness.

We attack these problems as follows. We have seen that a theorem proving derivation transforms a theorem proving problem into equivalent problems. Intuitively, it is desirable that a problem is reduced to one which is in some sense "smaller". In fact, at the end of a successful derivation we have a solved problem $(S; \mathit{true})$, where the dummy target "true" simply indicates that the original target has been proved. Thus, we need to identify what is being reduced during a theorem proving derivation. We observe that if a target φ_0 is indeed a theorem of the input set S_0, then there exist some proofs of φ_0 in S_0. On the other hand, the proof of the dummy target "true" is *empty.* At any stage $(S_i; \varphi_i)$ in between there is a (non-unique) *minimal* proof of φ_i in S_i, which represents the least amount of work which still needs to be done in order to prove φ_i from S_i. If the derivation gets closer to a solution, a minimal proof of the target gets reduced, i.e. the amount of work which is left becomes smaller. When the problem is solved, no more work needs to be done. Therefore, we regard theorem proving as *reduction of a minimal proof of the target* to the empty proof.

In order to compare proofs and to have a notion of minimal proofs, we need an *ordering* of proofs. Furthermore, this ordering needs to be *well founded*, having as bottom element the empty proof. A notion of well founded orderings on proofs, called *proof orderings*, has been introduced in [5, 6] and used to prove that Knuth-Bendix type completion procedures generate confluent systems of rewrite rules [16]. We use the same notion for a different purpose. Given a proof ordering $>_p$, at each stage $(S_i; \varphi_i)$ of a derivation, we consider the set $\Pi(S_i, \varphi_i)$ of the *minimal* proofs of φ_i in S_i, according to the ordering $>_p$. A successful derivation progressively reduces a proof in $\Pi(S_i; \varphi_i)$ to the empty proof.

This view has several advantages, both theoretical and practical. On the theoretical side, it has allowed us to give a coherent mathematical foundation to theorem proving. All concepts in theorem proving are defined and related to each other by using proof orderings and proof reduction with respect to such orderings. For instance, the above informal notions of refutational completeness and fairness can be formalized in terms of proof reduction [8, 9].

On the practical side, we require that the inference rules are *proof-reducing.* As we derive $(S_{i+1}; \varphi_{i+1})$ from $(S_i; \varphi_i)$, the set $\Pi(S_i, \varphi_i)$ is replaced by $\Pi(S_{i+1}, \varphi_{i+1})$. Clearly, we need to forbid all inference steps which would replace a proof P in $\Pi(S_i, \varphi_i)$ by a proof Q in $\Pi(S_{i+1}, \varphi_{i+1})$ such that $Q >_p P$. Such steps certainly do not help. On the other hand, we cannot impose that at every step a minimal proof of the target be reduced. This is impossible, since theorem proving is a process of search and therefore many steps generally do not contribute to the final result. We require that for every step $(S_i; \varphi_i) \vdash_{\mathcal{C}} (S_{i+1}; \varphi_{i+1})$, every proof P in $\Pi(S_i, \varphi_i)$ is either preserved, i.e. P is also in $\Pi(S_{i+1}, \varphi_{i+1})$, or reduced, i.e. P is replaced by a proof Q in $\Pi(S_{i+1}, \varphi_{i+1})$ such that $Q <_p P$. This condition is still not sufficiently general, since inference steps may not affect immediately any minimal proof of the target and still be necessary to prove it eventually. Therefore, we need to extend our attention to a larger set of theorems, which we call the *domain* $\mathcal{T}$ of the derivation. A step $(S_i; \varphi_i) \vdash_{\mathcal{C}} (S_{i+1}; \varphi_{i+1})$, such that $\Pi(S_i, \varphi_i) = \Pi(S_{i+1}, \varphi_{i+1})$, is also proof-reducing, provided that for all ψ in $\mathcal{T}$, every minimal proof is either preserved or reduced and for at least a ψ in $\mathcal{T}$ a minimal proof is reduced. Intuitively, we would like the domain $\mathcal{T}$ to be as small and as "related" to the target as possible. In practice, for the known simplification-based strategies, the domain is the set of all ground equations.

2.1 The simplification-based inference engine UKB

The most significant characteristic of inference rules in simplification-based strategies is that they are proof-reducing [8]. As an example, we present in the following the ones which are used in our prover *SBR*3, an automated deduction system for equational theories. Collectively, they form the **unfailing Knuth-Bendix completion procedure**, or **UKB** for short. UKB is a semi-decision procedure for the validity problem of equational theory.

The most important one is *Simplification* [28] itself. If we consider a derivation in equational logic, a presentation is a set of equations E and a target is an equational theorem $\forall \bar{x} s \simeq t$. We write the target as $\hat{s} \simeq \hat{t}$ to denote that it contains only universally quantified variables and therefore can be regarded as a ground equality. The definition of simplification involves two orderings. The first one is a *well founded* ordering on terms $\succ$ which is used to ensure that simplification replaces an equation by a smaller equation [15]. The second one is the *encompassment* ordering $\trianglerighteq$ which is defined as follows: $t \trianglerighteq s$ if $t|u = s\sigma$ for some position u and substitution σ, i.e. an instance $s\sigma$ of s occurs as a subterm in t. We write $t \triangleright s$ if $t \trianglerighteq s$ and either u is not the root position or σ is not just a renaming of variables [16].

Simplification applies to the presentation:

$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{p[r\sigma]_u \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})} \quad \begin{array}{ll} p|u = l\sigma & \qquad p \succ p[r\sigma]_u \\ p \triangleright l \vee q \succ p[r\sigma]_u & \end{array}$$

and to the target:

$$\frac{(E \cup \{l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s}[r\sigma]_u \simeq \hat{t})} \quad \begin{array}{l} \hat{s}|u = l\sigma \\ \hat{s} \succ \hat{s}[r\sigma]_u. \end{array}$$

Intuitively, a simplification step replaces an equation by a smaller equation and therefore it reduces all the proofs where the replaced equation occurred.

The second basic inference rule, a deductive inference rule called **Superposition** [21], is also proof-reducing:

$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{p \simeq q, l \simeq r, p[r]_u\sigma \simeq q\sigma\}; \hat{s} \simeq \hat{t})} \quad \begin{array}{ll} p|u \notin X & \qquad (p|u)\sigma = l\sigma \\ p\sigma \not\preceq q\sigma, p[r]_u\sigma & \end{array}$$

where X is the set of variables and σ is the most general unifier of $(p|u)$ and l. The key point is that the step is performed only if $p\sigma \not\preceq q\sigma$ and $p\sigma \not\preceq p[r]_u\sigma$. This conditions guarantee that the rule is proof-reducing.

An operator f is said to satisfy the *right cancellation law* if for every x, y, z, $f(x, z) = f(y, z)$ implies $x = y$. The *left cancellation law* is defined symmetrically. Cancellation laws can be incorporated as inference rules, which may reduce considerably the size of the equations. We present two such inference rules here. A complete list can be found in [22].

Cancellation 2:

$$\frac{(E \cup \{f(d_1,d_2) \simeq y\}; \hat{s} \simeq \hat{t})}{(E \cup \{f(d_1,d_2) \simeq y, d_1\sigma \simeq x\}; \hat{s} \simeq \hat{t})} \quad \begin{array}{ll} y \in V(d_1) & \sigma = \{y \mapsto f(x,d_2)\} \\ y \notin V(d_2) & x \textit{ is a new variable} \end{array}$$

Cancellation 4:

$$\frac{(E \cup \{f(p,u) \simeq f(q,u)\}; \hat{s} \simeq \hat{t})}{(E \cup \{p \simeq q\}; \hat{s} \simeq \hat{t})}$$

where the function f is right cancellable. In *Cancellation 2*, if the substitution $\sigma = \{y \mapsto f(x,d_2)\}$ is applied to the given equation, it becomes $f(d_1\sigma, d_2) \simeq f(x,d_2)$, since y does not occur in d_2. The cancellation law reduces this equation to $d_1\sigma \simeq x$. *Cancellation 4* is not necessary for the purpose of completeness, but it helps in improving efficiency.

Simplification-based strategies also feature rules such as **Functional subsumption**,

$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s} \simeq \hat{t})} \quad (p \simeq q) \trianglerighteq (l \simeq r)$$

which delete equations subsumed by other equations, and **Deletion**

$$\frac{(E \cup \{s \simeq s\}; \hat{s} \simeq \hat{t}\})}{(E; \hat{s} \simeq \hat{t})}$$

which delete trivial equations. These rules do not reduce any minimal proofs, but they delete equations, which are *redundant*, in the sense that they do not contribute to any minimal proofs and therefore are not needed in the derivation. Deletion also applies to the target

$$\frac{(E; \hat{s} \simeq \hat{s}\})}{(E; \mathit{true})}$$

in order to detect that the target is proved.

Another inference on the target is superposition of an un-orientable equation onto a target equality $\hat{s} \simeq \hat{t}$ to generate a new target equality. A newly generated target equality is first simplified as much as possible and then it is kept only if it is smaller than $\hat{s} \simeq \hat{t}$. This rule is called **Ordered saturation** [1]:

$$\frac{(E \cup \{l \simeq r\}; N \cup \{\hat{s} \simeq \hat{t}\})}{(E \cup \{l \simeq r\}; N \cup \{\hat{s} \simeq \hat{t}, \hat{s}' \simeq \hat{t}'\})} \quad \begin{array}{lll} \hat{s}|u = l\sigma & \hat{s}[r\sigma]_u \rightarrow^*_E \hat{s}' & \hat{t} \rightarrow^*_E \hat{t}' \\ & \{\hat{s}', \hat{t}'\} \prec_{mul} \{\hat{s}, \hat{t}\} & \end{array}$$

Ordered saturation applies if $\hat{s} \prec \hat{s}[r\sigma]_u$, since if $\hat{s} \succ \hat{s}[r\sigma]_u$ holds, simplification would apply. The target equality $\hat{s}' \simeq \hat{t}'$ might have a shorter proof than the other target equalities. Ordered saturation allows us to generate more than one target in order to broaden our chance of reaching the proof as soon as possible.

Rules such as simplification, subsumption and deletion are called *contraction* inference rules, because they delete equations or replace them by smaller equations. Rules like superposition and ordered staturation instead are *expansion* inference rule, because they generates new equations and add them to E or to the target. Roughly speaking, a step which deletes a sentence also deletes the portion of the search space which depends on that sentence, i.e. all the inferences which could be applied to that sentence. On the contrary, an expansion step expands the data base and therefore the search space. It follows that in order to keep the size of the search space manageable, it is desirable to apply as much as possible the contraction rules and to restrict as much as possible the application of the expansion rules. This is in fact the philosophy of the simplification-based strategies. First of all, these strategies adopt *simplification-first* search plans [21], i.e. search plans which give priority to contraction inference rules. Under such search plans, expansion rules are applied only if no contraction rule applies. Consequently, the current set of equations and therefore the current search space is always kept as reduced as possible. Secondly, simplification-based strategies impose strong ordering based restrictions on the expansion rules, such as those embedded in the definitions of superposition and ordered saturation. Such restrictions make the inference rules proof-reducing and limit their applicability, thereby reducing their capability of expanding the search space. These choices have turned out to be very successful in practice, up to the point of bringing within reach unsolved challenge problems, as described in the following section.

3 Putting theory into practice

We have developed a family of theorem provers for equational theories whose design strictly adheres to the aforementioned methodology. The latest versions is $SBR3$, written in CLU and runs on Sun3. A new version in C++, $SBR4$, with the same functionalities is being developed. $SBR4$ runs on any machines that supports C++, and is much faster than $SBR3$. On the problems which we have tested on both $SBR3$ and $SBR4$ (the latter on a Sparcstation), the latter is usually at least ten times faster.

$SBR3$ takes as inputs an equational theory E and an equation $s \simeq t$ and tries to prove that $s \simeq t$ is a theorem of E. It proves a theorem the refutational way. That is, it replaces all variables in $s \simeq t$ by new Skolem constants and tries to find a contradiction to $E \cup \{\hat{s} \neq \hat{t}\})$ where $\hat{s}$ and $\hat{t}$ are the skolemization of s and t. Then the prover will try to deduce an instance of $x \neq x$ which yields the contradiction.

In addition to the theory and the equation, the user should also provide an ordering for comparing the terms. Usually the ordering should be a *complete simplification ordering* (a simplification ordering which is total on ground terms). In $SBR3$ the user has the choice of assigning a precedence among the operators in the theory and choose an ordering from a list implemented in the system. However, $SBR3$ will not check the totality for the user. The lack of totality on ground terms may actually be turned into a powerful search strategy similar to *Ordered Saturation* described in the previous section.

The backbone of $SBR3$ is a variation of unfailing Knuth-Bendix completion, mentioned in the previous section, which also incorporates the commutative and associative (AC) axioms of an operator into the unification algorithm. We term this procedure AC-UKB. Although the AC axioms can be handled simply as equations, it is advantageous to treat them implicitly in the unification process to the number of unnecessary *Superposition* inferences.

What differ $SBR3$ from the other provers, in addition to the simplification-based inference system, are its simple yet extremely powerful search plans. Search plans are usually treated in theorem proving in an *ad hoc* and incomplete way – anything that produces proofs is allowed. Fairness (thus completeness of the proof strategy) is usually compromised by the concern for greater efficiency. Using the notion of proof reduction, we have demonstrated that it is possible to achieve both completeness (fairness of the search plan) as well as efficiency. In $SBR3$, only fair search plans are implemented. Our experiments showed that they, if done properly, can indeed be both complete and very efficient.

The most important design choice common to all the search plans in $SBR3$ is that they are *simplification-*

first plans. That is, no superposition step is ever performed if there are still simplification steps and functional subsumption steps to be done. This search plan, coupled with cancellation, controls the growth of the number and size of equations sufficiently enough to obtain proofs for simple to moderately difficult problems. For more difficult problems, however, the search space quickly grows to an unmanageable size.

The first question we tackle is one of finding a shorter path to a solution. UKB, being complete, guarantees the existence of a proof through simplification and superposition should there be one. It does not, however, guarantee to provide a *short* proof. Suppose the prover can look at several different inequalities and tries to find a contradiction simultaneously[1], then conceivably one can find a proof faster. On the other hand, one should also keep in mind not to inundate the search space with irrelevant inequalities.

$SBR3$ provides a facility for maintaining a reasonable number of inequalities, to check for shorter proofs, by modifying the *ordered saturation* rule. When an un-orientable equation is generated, we superimpose it into an existing inequality (say A) to create a new inequality if possible. Then the new inequality is simplified using the rest of the equations and rules into B. The inequality B is kept, without deleting A, if $A \not\lesssim B$ according to the ordering. We term this method the *inequality ordered-saturation strategy.* This strategy is indispensable for proving some of the more difficult problems which we experimented [1].

Another challenge is to eliminate redundant critical pairs. This problem is especially serious in AC-rewriting due to the potentially astronomical number of AC-unifiers. In the term rewriting literature there are a handful of critical pair criteria, whose purpose is to eliminate unnecessary critical pairs. However, all of them are designed not to destroy the confluence property of any given two terms. In refutational theorem proving, on the other hand, we are only interested in the confluence of the two terms of the targeted theorem. Therefore a critical pair can be deleted or suspended as long as it does not destroy the confluence of the intended terms.

Taking advantage of this property, we employed a notion of *measure* in $SBR3$. A measure is defined syntactically on the structure of terms: for example, the number of occurrences of a specific operator may be a measure. The measure estimates the likelihood of whether a critical pair may contribute to an eventual proof of the intended theorem. Critical pairs are ordered according to the measure which decides the next equation to be chosen to perform superposition. Certain measures even allow us to delete critical pairs if they are deemed irrelevant for producing a proof. This search strategy is called *filtration-sorted strategy* and its details can be found in [2]. Three different types of measure are implemented in $SBR3$.

4 Experimental results: automatic proofs by SBR3

We have conducted extensive experiments on $SBR3$. We tested the prover on all the examples in equational theorem proving which we could find, as well as some new ones. The experiments we performed showed a dramatically small search space, just as expected. As a simple example, for the well-known *Salt and Mustard* puzzle of Lewis Carroll, first suggested by the Argonne Theorem Proving Group as a challenge problem for theorem provers, the Argonne prover Otter [25] generated more than 32,000 clauses before finding the solution while ours succeeded after generating less than 2000 rewrite rules.

The performance of $SBR3$ on serious mathematical problems is even more impressive. The celebrated Jacobson's Theorem of ring theory for $n = 3$ [31], the independence of ternary algebra axioms [27], etc., have all been proved in a few minutes. In the following we describe some of the problems for which $SBR3$ provided

[1]The basic UKB only looks at one.

the *first* computer proofs.

Classical Regular Languages

In [14], there is an equational formulation of classical regular languages by Yanov (page 108 of [14]) which completely axiomatize regular languages containing the empty string. The axioms are:

$x + x == x$

$z.(x + y) == (z.x) + (z.y)$

$(x + y).z == (x.z) + (y.z)$

$(x^*)^* == x^*$

$x^*.x^* == x^*$

$x + (x + y)^* == (x + y)^*$

$(x + y)^* == (x^* + y)^*$

$(x + y)^* == (x.y)^*$

$(x.y)^*.x == (x.y)^*$

$x.(x.y)^* == (x.y)^*$

$y + (x.y) == x.y$

$x + (x.y) == x.y$

$x + y == y + x$

$(x.y).z == x.(y.z)$

$(x + y) + z == x + (y + z)$

where "." is concatenation. $SBR3$ proved that

$$(\sum_{i=1}^{n} A_i)^* = (\sum_{i=1}^{n} A_i.\overline{A_i}^*.A_i)^*(1 + \sum_{i=1}^{n} A_i.\overline{A_i}^*)$$

where $\overline{A_i} = A_1 + \cdots + A_{i-1} + A_{i+1} + \cdots + A_n$, for $n = 3$ and $n = 4$, and the languages contain the empty strings[2]. In [14], Conway used an entire chapter to introduce a new technique to prove these two problems and remarked (page 119) that *"... even for $n = 3$ it is difficult to produce a proof without using the general ideas of this chapter, and for $n = 4$ I doubt if a completely written out proof could be fitted into 10 pages"*. The *direct* proof, produced by $SBR3$, needs no more than five new critical pairs, in addition to the simplication steps! The cpu time needed for $n = 3$ is about 4 minutes and 42 minutes for $n = 4$.

Dependency of Lukasiewicz's fifth axiom

Lukasiewicz's many-valued logic is defined using the following four axioms:

$true \Rightarrow x == x$

$(x \Rightarrow y) \Rightarrow ((y \Rightarrow z) \Rightarrow (x \Rightarrow z)) == true$

[2]The equations are not true in classical regular algebra when $n \geq 5$.

$(x \Rightarrow y) \Rightarrow y == (y \Rightarrow x) \Rightarrow x$

$(not(x) \Rightarrow not(y)) \Rightarrow (y \Rightarrow x) == true.$

The problem is whether the fifth axiom $x \Rightarrow y \vee y \Rightarrow x == true$ is necessary [17]. The conjecture of its dependency was given by Lukasiewicz in the 20's, as reported in [32], and proved many years later [13, 26].

The proof by $SBR3$ is done by first deriving a few lemmas from the axioms, one of which leads to the definition of an additional operator *or*. Then $SBR3$ proves that *or* is AC. Finally, the conjecture is proved in about 2 minutes. For the final session, the inputs are

$true \Rightarrow x == x$

$x \Rightarrow x == true$

$x \Rightarrow true == true$

$(x \Rightarrow y) \Rightarrow ((y \Rightarrow z) \Rightarrow (x \Rightarrow z)) == true$

$not(not(x)) == x$

$(x \Rightarrow y) \Rightarrow y == (y \Rightarrow x) \Rightarrow x$

$or(not(x), y) == x \Rightarrow y$

$x \vee y == (x \Rightarrow y) \Rightarrow y$

Declared AC-operator: *or*.

Theorem proved: $x \Rightarrow y \vee y \Rightarrow x == true.$

A detailed description of the experiments in Lukasiewicz logic can be found in [3, 4, 10].

Moufang identities in alternative rings

Alternative rings are rings with the associativity of $*$ replaced by two *alternative* axioms. The Moufang identities are a set of equational theorems of alternative rings. The Moufang identities as a challenge to theorem provers was first suggested in [30], although no automated proof was given. They were later proved automatically using a special-purpose theorem prover designed for ring theory [35]. $SBR3$ is the first syntactic theorem prover which proved them automatically.

Alternative rings are defined by

$0 + x == x$

$0 * x == 0$

$x * 0 == 0$

$g(x) + x == 0$

$g(x + y) == g(x) + g(y)$

$g(g(x)) == x$

$x * (y + z) == (x * y) + (x * z)$

$(x + y) * z == (x * z) + (y * z)$

$(x * y) * y == x * (y * y)$

$$(x * x) * y == x * (x * y)$$
$$g(x) * y == g(x * y)$$
$$x * g(y) == g(x * y)$$
$$a(x, y, z) == ((x * y) * z) + g(x * (y * z)$$

where a is an auxiliary operator.

$SBR3$ proved the following properties (the middle alternative law and two skew-symmetries of a) within 20 seconds:

$$(x * y) * x == x * (y * x)$$
$$a(y, x, z) == g(a(x, y, z))$$
$$a(z, y, x) == g(a(x, y, z))$$

The Moufang identities are defined as:

$$(((x * y) * x) * z) = (x * (y * (x * z))) \text{ (left Moufang)}$$
$$(((z * x) * y) * x) = (z * (x * (y * x))) \text{ (right Moufang)}$$
$$((x * y) * (z * x)) = ((x * (y * z)) * x) \text{ (middle Moufang)}$$

and they are proved in 49, 55, and 41 minutes respectively.

By adding the left and right Moufang into the input set, we are able to give a direct proof of

$$a(x * x, y, z) == ((a(x, y, z) * x) + (x * a(x, y, z)))$$

in 13 minutes. A full account of our experiments in alternative rings is given in [1].

Another series of problems which we are working on now is to verify the theorems of the book *A Formalization of Set Theory without Variables* by Tarski and Givant. As pointed out in [11], this will have direct implication on the design and optimization of query languages in relational data bases and program synthesis.

Our experiments are encouraging. They show us that high performance automated deduction is feasible even with our current knowledge and technology. We believe that the philosophy of simplification underlying our prover is the most significant reason for the dramatic reduction of search space, which made all our automatic proofs possible.

5 Distributed theorem proving

We are currently working on the design of a simplification-based strategy for *parallel automated deduction* in a *distributed multi-processing* environment. We feel that simplification-based theorem proving is an ideal candidate for application of parallel computation, because the rewriting approach couples a strong and elegant theoretical foundation with an extremely encouraging experimental record. A deep understanding of the problem at hand is necessary to design an architecture that exploits successfully the increased computing power of a parallel environment. It would also open a new perspective of application for parallel computation which has not been investigated before.

Relatively little work has been done in this area so far. Parallelizing a simplification-based strategy is significantly different from parallelizing a conventional, space consuming theorem proving strategy. The latter uses mostly expansion inferences and it is relatively easy to perform expansion steps in parallel, because expansion steps are more or less independent from each other. More precisely, any two inference steps which do not have premises in common are trivially independent and can proceed concurrently, at least in principle. For expansion inferences, two steps which share one or more premises are also independent, because expansion steps do not modify their premises. Expansion steps simply need to be granted *read-access* to their premises. Since concurrent read can be safely admitted, the parallelization of expansion inferences does not raise basic conceptual problems. In a simplification-based strategy, however, inference rules are intertwined. The reason is that contraction inferences do modify their premises. A contraction step needs not just read-access, but also *write-access* to its premises. Therefore, two contraction steps which share premises may cause a *write-write* conflict if they attempt to modify concurrently the same data. Also, contraction steps may have *read-write* conflicts with concurrent expansion steps.

Even this very basic analysis of the problem shows that the presence of contraction rules makes the design of a parallel strategy harder. However, we think that the gain is well worth the additional effort. Firstly, there is ample empirical evidence that sequential strategies with contraction rules are much more powerful than those without contraction. This behaviour is also justified theoretically by our proof reduction view. Based on this, it is reasonable to foresee that the same pattern of behaviour will appear when comparing parallel strategies. In fact, we expect an even much better improvement. By grossly simplifying the problem, let $\mathcal{C}_0$ be a sequential strategy without contraction rules and let t be the time spent by $\mathcal{C}_0$ to prove a given input $(S;\varphi)$. Let $\mathcal{C}_1$ be the sequential strategy obtained by adding contraction to $\mathcal{C}_0$ and let t/s, for some $s>1$, be the time required by $\mathcal{C}_1$ on $(S;\varphi)$. Furthermore, let $\mathcal{C}_2$ and $\mathcal{C}_3$ be respectively a parallel version of $\mathcal{C}_0$ and a parallel version of $\mathcal{C}_1$. We expect that if $\mathcal{C}_2$ takes time t/n, $n>1$, to prove φ, $\mathcal{C}_3$ will take time t/p, where $p>n\cdot s$. In other words, we expect the speedup of a parallel simplification-based strategy to be much higher than the mere combination of the speedup induced by simplification and the speedup induced by parallelism. This may not be true for all inputs, but we expect it to hold for most targets. The intuitive reason for our expectation is the following. Roughly speaking, if we execute in parallel an expansion-only strategy, we will be able to perform expansion steps by batches rather than one by one. The equations will be generated faster and the derivation will succeed at an earlier stage than the sequential one. However, the solution obtained is in some sense the same, as the same equations are generated. On the other hand, if we execute in parallel a simplification-based strategy, powerful simplifiers may be generated much sooner than in the sequential derivation. In a simplification-first strategy, the early application of such simplifiers may trigger the early generation of other simplifiers and an eventual radical modifications of the data base, leading the prover to find a different and much faster successful path than the one found by the sequential execution.

Problems related to those of parallel deduction have been addressed by the study of parallel and distributed implementations of the *Buchberger algorithm* [34, 29, 18]. The Buchberger algorithm works on polynomials, equated to 0 and treated as oriented equations. It takes as input a set of polynomials and gives as output a *basis* for the ideal generated by the input polynomials. The basis has the property that it reduces to 0 all and only the polynomials belonging to the ideal [12]. The Buchberger algorithm is related to the simplification-based strategies because it features an expansion inference rule which is similar to superposition and a contraction rule which is similar to simplification. There are also substantial differences, because the Buchberger algorithm has a much less general purpose than a theorem proving strategy. The Buchberger algorithm is an algorithm, whereas the theorem proving strategies are semidecision procedures. Its inferences do not use unification, since there are no variables, as the "variables" in the polynomials are constants logically. It follows that expansion steps are much less expensive than in theorem proving. Also, the equations are all trivially oriented into rewrite rules, because they are obtained by equating polynomials to 0. Nonetheless, parallel implementations of the Buchberger algorithm need to deal with the problem of the coexistence of expansion and contraction inferences. The three approaches presented in [34, 29, 18] address the problem within three different models of parallel computation: a shared memory multi-processor in [34], a data-flow machine in [29] and a distributed memory multi-processor in [18]. All three algorithms have interesting features. However, none of them implements a simplification-first methodology. In fact, the data base of polynomials is not maintained fully simplified by any of these three implementations. In particular, very little *backward contraction*, i.e. simplification of formerly existing equations by newly generated ones, is performed. As a consequence, expansion rules are applied to equations which are not fully reduced, unnecessary equations are generated and the search space swells. It seems that this phenomenon has prevented these three implementations from achieving better speedups. The trouble is that requiring equations to be fully simplified, before they are allowed to expand, introduces some sequentiality. An expansion process cannot be granted read-access to an equation until all simplification processes have had write-access to it. We have then two at least partially conflicting desiderata: on one hand, we would like to simplify as much as possible before expanding, while in the meantime we would like to perform as many steps in parallel as possible. The problem is to find a satisfactory trade-off between these two.

We have kept this issue in mind since the early stages of our project. So far, we have settled on a few basic choices. The first one is *coarse grain* versus *fine grain* parallelism or, equivalently, *coarse granularity* of protection versus *fine granularity* of protection. For the purpose of this discussion, we regard as fine granularity the *term level* and as coarse granularity the *equation (or clause) level*. Thus, fine granularity means that every term is a *grain* of memory with its own access rights. Fine granularity allows parallel processes to access different subterms of the same term. Parallel matching, parallel rewriting and parallel unification are examples of fine grain parallelism. On the other hand, coarse granularity means that if a process is granted access to an equation, no other process can access any part of it. Fine grain parallelism is well suited for equational programs, where just one term needs to be reduced by a static set of equations. In theorem proving we have a dynamic set of equations where every single term is subject to simplification. It seems to us that under these conditions the overhead of handling fine granularity would be unreasonably high. Therefore, we choose to concentrate ourselves on coarse grain parallelism, although some fine grain parallelism might be considered at a later stage.

The second basic choice is *shared memory* versus *distributed memory*. This choice is related to the previous one. Fine grain parallelism leads in general to adopt a shared memory, since it does not seem realistic to scatter the terms of an equation over a distributed memory. Coarse grain parallelism can be implemented in principle in both a shared memory and a distributed memory. However, we are oriented toward distributed memory, for the following reasons. Theorem proving is basically search for solutions in a generally huge search space. We expect parallelism to help in two ways: by keeping the search space small by eager, parallel simplification and by searching it in parallel along different paths. In order to realize this intuitive idea of *parallel search*, we need the parallel processes to be rather independent. Thus, the processors should be rather *loosely coupled*, with no shared memory. We envision a situation where each processor has in its own memory a set of equations S^i and the union of all the S^i's form the current data base S. The S^i's are initially disjoint, but in general they do not remain disjoint during the derivation. Also, each processor is originally given a copy of the input target φ_0. Since different processors perform different steps, φ_0 may be reduced to different, yet equivalent targets, one per processor. Each processor performs its own inference steps searching for a proof. However, the processors do communicate by broadcasting their equations to all the other processors. When receiving equations from the outside, a processor uses them to perform inferences with its own equations. The simplification-first methodology is strictly enforced at the local level. Each processor maintains its own data base fully reduced, including the equations received as messages. No expansion step is performed if the equations involved are not fully reduced, at least locally. Clearly, they are not guaranteed to be reduced with respect to the global data base. However, our strategy is *fair* in the sense that it guarantees that any two equations generated at remote sites will be able to interact through messages, if they are not simplified locally beforehand. The cost of handling such messages is the price to pay for the high degree of independence of the processors. In addition, this scheme induces a certain amount of redundancy, as the data bases at different sites are not guaranteed to be disjoint and therefore it may happen that a same step is executed by more than one processor.

This is just a very brief sketch of a few basic ideas in our work. We are currently studying the details, trying to minimize redundancy and the cost of message passing. Based on the investigations conducted so far and on the observation that the implementations in [34, 29, 18] obtained significant speedups even in the absence of full simplification, we expect that this on going research will ultimately increase the speed of a theorem prover like *SbReve* by at least a hundred times.

References

[1] S.Anantharaman and J.Hsiang, Automated Proofs of the Moufang Identities in Alternative Rings, *Journal of Automated Reasoning*, Vol. 6, No. 1, 76–109, 1990.

[2] S.Anantharaman and A.Andrianarivelo, Heuristical Critical Pair Criteria in Automated Theorem Proving, in A.Miola (ed.), *Proceedings of the International Symposium on the Design and Implementation of Symbolic Computation Systems*, Capri, Italy, April 1990, Springer Verlag, Lecture Notes in Computer Science 429, 184–193, 1990.

[3] S.Anantharaman and M.P.Bonacina, Automated Proofs in Lukasiewicz Logic, Technical Report, Department of Computer Science, SUNY at Stony Brook, November 1989.

[4] S.Anantharaman and M.P.Bonacina, An Application of the Theorem Prover SBR3 to Many-valued Logic, in M.Okada and S.Kaplan (eds.), *Proceedings of the Second International Workshop on Conditional and Typed Term Rewriting Systems*, Montréal, Canada, June 1990, Springer Verlag, Lecture Notes in Computer Science, to appear.

[5] L.Bachmair, N.Dershowitz and J.Hsiang, Orderings for Equational Proofs, in *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science*, 346–357, Cambridge, Massachussets, June 1986.

[6] L.Bachmair and N.Dershowitz, Equational inference, canonical proofs and proof orderings, *Journal of the ACM*, to appear.

[7] M.P.Bonacina and J.Hsiang, On Rewrite Programs: Semantics and Relationship with Prolog, *Journal of Logic Programming*, to appear.

[8] M.P.Bonacina and J.Hsiang, Completion procedures as Semidecision procedures, in M.Okada and S.Kaplan (eds.), *Proceedings of the Second International Workshop on Conditional and Typed Term Rewriting Systems*, Montréal, Canada, June 1990, Springer Verlag, Lecture Notes in Computer Science, to appear.

[9] M.P.Bonacina and J.Hsiang, On fairness of completion-based theorem proving strategies, in R.V.Book (ed.), *Proceedings of the Fourth International Conference on Rewriting Techniques and Applications*, Como, Italy, April 1991, Springer Verlag, Lecture Notes in Computer Science 488, 348–360, 1991.

[10] M.P.Bonacina, Problems in Lukasiewicz logic, in *Newsletter of the Association for Automated Reasoning*, No. 18, June 1991.

[11] P.Broome, Applications of Algebraic Logic to Recursive Query Optimization, in *Proceedings of the Eighth Army Conference on Applied Mathematics and Computing*, 1990, to appear.

[12] B.Buchberger, An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-dimensional Polynomial Ideal, (in German), PhD thesis, Department of Mathematics, University of Innsbruck, Austria, 1965.

[13] C.C.Chang, in *Transactions American Mathematical Society*, No. 87, 55–56, 1958.

[14] J.H.Conway, *Regular Algebra and Finite Machines*, Chapman and Hall, 1971.

[15] N.Dershowitz, Termination of Rewriting, *Journal of Symbolic Computation*, Vol. 3, No. 1 & 2, 69–116, February/April 1987.

[16] N.Dershowitz and J.-P.Jouannaud, Rewrite Systems, Chapter 15, Volume B, *Handbook of Theoretical Computer Science*, North-Holland, 1989.

[17] J.M.Font, A.J.Rodriguez and A.Torrens, Wajsberg algebras, *Stochastica*, Vol. 8, No. 1, 5–31, 1984.

[18] D.J.Hawley, A Buchberger Algorithm for Distributed Memory Multi-Processors, in *Proceedings of the International Conference of the Austrian Center for Parallel Computation*, Linz, Austria, October 1991, Springer Verlag, Lecture Notes in Computer Science, to appear.

[19] C.M.Hoffmann and M.J.O'Donnell, Programming with Equations, *ACM Transactions on Programming Languages and Systems*, Vol. 4. No. 1, 83–112, January 1982.

[20] J.Hsiang, Refutational Theorem Proving Using Term Rewriting Systems, *Artificial Intelligence*, Vol. 25, 255–300, 1985.

[21] J.Hsiang and M.Rusinowitch, On word problems in equational theories, in Th.Ottman (ed.), *Proceedings of the Fourteenth International Conference on Automata, Languages and Programming*, Karlsruhe, Germany, July 1987, Springer Verlag, Lecture Notes in Computer Science 267, 54–71, 1987.

[22] J.Hsiang, M.Rusinowitch and K.Sakai, Complete set of inference rules for the cancellation laws, in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milano, Italy, August 1987, 990–992.

[23] C.Kirchner, H.Kirchner and J.Meseguer, Operational semantics of OBJ3, in *Proceedings of the 9th International Conference on Automata, Languages and Programming*, LNCS 241, Springer Verlag, 1988.

[24] D.E.Knuth and P.B.Bendix, Simple Word Problems in Universal Algebras, in J.Leech (ed.), *Proceedings of the Conference on Computational Problems in Abstract Algebras*, Oxford, England, 1967, Pergamon Press, Oxford, 263–298, 1970.

[25] W.W.McCune, OTTER 2.0 Users Guide, Technical Report ANL-90/9, Argonne National Laboratory, Argonne, Illinois 1990.

[26] C.A.Meredith, in *Transactions American Mathematical Society*, No. 87, 54, 1958.

[27] A.J.Nevins, A human-oriented logic for automatic theorem proving, *Journal of the ACM*, Vol. 4, 606–621, 1974.

[28] M.Rusinowitch, Theorem-proving with Resolution and Superposition, *Journal of Symbolic Computation*, Vol. 11, No. 1 & 2, 21–50, January/February 1991.

[29] K.Siegl, Gröbner Bases Computation in STRAND: A Case Study for Concurrent Symbolic Computation in Logic Programming Languages, Master thesis and technical Report No. 90-54.0, RISC-LINZ, November 1990.

[30] R.L.Stevens, Challenge Problems from Nonassociative Rings for Theorem Provers, in E.Lusk and R.Overbeek (eds.), *Proceedings of the Ninth Conference on Automated Deduction*, Argonne, Illinois, May 1988, Springer Verlag, Lecture Notes in Computer Science 310, 730-734, 1988.

[31] M.E.Stickel, A case study of theorem proving by the Knuth-Bendix method: Discovering that $x^3 = x$ implies ring commutativity, in *Proceedings of the Seventh Conference on Automated Deduction*, Springer Verlag, Lecture Notes in Computer Science 170, 248–258, 1984.

[32] A.Tarski and J.Lukasiewicz, Investigations into the sentential calculus, Chapter IV in A.Tarski, *Logic, Semantics and Metamathematics*, 38–56, Clarendon Press, Oxford, 1956.

[33] A.Tarski and S.Givant, *A Formalization of Set Theory Without Variables*, American Mathematical Society, Colloquium Publications, Vol. 41, 1987.

[34] J.-P.Vidal, The Computation of Gröbner Bases on A Shared Memory Multiprocessor, in A.Miola (ed.), *Proceedings of the International Symposium on the Design and Implementation of Symbolic Computation Systems*, Capri, Italy, April 1990, Springer Verlag, Lecture Notes in Computer Science 429, 81–90, 1990.

[35] T.C.Wang, Case Studies of Z-module Reasoning: Proving Benchmark Theorems from Ring Theory, *Journal of Automated Reasoning*, Vol. 3, No. 4, 1987.