Incremental Theory Reasoning Methods for Semantic Tableaux

Bernhard Beckert & Christian Pape

Institute for Logic, Complexity and Deduction Systems
University of Karlsruhe
Am Fasanengarten 5, 76128 Karlsruhe, Germany
{beckert,pape}@ira.uka.de, http://i12.www.ira.uka.de/~beckert

Abstract. Theory reasoning is an important technique for increasing the efficiency of automated deduction systems. In this paper we present *incremental* theory reasoning, a method that improves the interaction between the foreground reasoner and the background (theory) reasoner and, thus, the efficiency of the combined system. The use of incremental theory reasoning in free variable semantic tableaux and the cost reduction that can be achieved are discussed; as an example, completion-based equality reasoning is presented, including experimental data obtained using an implementation.

1 Introduction

Theory reasoning is an important technique for increasing the efficiency of automated deduction systems. The knowledge from a given domain (or theory) is made use of by applying efficient methods for reasoning in that domain. The general purpose *foreground reasoner* calls a special purpose *background reasoner* to handle problems from a certain theory.

Following the pioneering work of Stickel [22], sound and complete theory reasoning methods have been described for various calculi; e.g., path resolution [17], the connection method [20], model elimination [2]. In addition, background reasoners have been designed for various theories, in particular for equality reasoning [4]; an overview can be found in [3].

Besides the efficiency of the foreground and the background reasoner, the interaction between them plays a critical rôle for the efficiency of the combined system: It is a difficult problem to decide whether it is useful to call the background reasoner at a certain point or not, and how much time and other resources to spend for its computations. In general, to give a perfect answer to these questions is as difficult as the theory reasoning problem itself (if the theory is undecidable, it is undecidable whether a call to the background reasoner is useful). Even with good heuristics at hand, one cannot avoid calling the background reasoner at the wrong point: either too early or too late.

This problem can (at least partially) be avoided by using incremental methods for background reasoning, i.e., algorithms that—after a futile try to solve a theory reasoning problem—allow to save the results of the background reasoner's computations and to reuse this data for a later call.[1] Then, in case of doubt the background reasoner can be called early without running the risk of doing useless computations. In addition, an incremental background reasoner can reuse

[1] This should not be confused with *partial theory reasoning*, where the background reasoner derives new formulae and hands these back to the foreground reasoner. The information derived by an incremental background reasoner cannot be used by the foreground reasoner, but only by the background reasoner during later calls.

data multiply, if different extensions of a problem have to be handled. An important example are completion-based methods for equality reasoning, that are inherently incremental.

We focus on theory reasoning in semantic tableaux [21, 11] and related methods—such as model elimination [16] and the connection method [8]—, where a background theory reasoner is used to close tableau branches resp. to compute connections or links (total theory reasoning).

The paper is organized as follows: In Section 2 we introduce notation and recall the basic definitions of theory reasoning; in Section 3 we define the particular version of free variable semantic tableaux we will be using in the following sections.[2] In Section 4 our main results are presented: incremental theory reasoning is introduced and formally defined, its use in free variable semantic tableaux is described, and the cost reduction that can be achieved is discussed. In Sections 5 and 6 we present completion-based equality reasoning as an example, and describe an actual implementation; Section 7 contains experimental data obtained using this implementation. Finally, in Section 8 we draw conclusions from our work.

2 Preliminaries

2.1 Notation

Let us fix a first-order language $\mathcal{L}$ which is built up from countable sets $\mathcal{P}$ of predicate symbols, $\mathcal{F}$ of function symbols, $\mathcal{C}$ of constant symbols and $\mathcal{V}$ of object variables in the usual manner (for each arity there are countably many function and predicate symbols). We use the logical connectives $\wedge$ (conjunction), $\vee$ (disjunction), $\supset$ (implication), $\leftrightarrow$ (equivalence), and $\neg$ (negation), and the quantifier symbols $\forall$ and $\exists$.

Since in the tableau proofs it will be necessary to introduce Skolem terms, we extend our first-order language $\mathcal{L}$ to a language $\mathcal{L}_{\mathrm{Sko}}$ by adding countably many constant symbols and function symbols for each arity which do not already appear in $\mathcal{L}$.

We use the standard notions of free and bound variable, (grounding) substitution, sentence, model, logical consequence (denoted by $\models$), valuation, satisfiability and tautology (see Definition 1).

Subst is the set of all idempotent substitutions with finite domain (without making any real restrictions we only consider substitutions of this type). A substitution σ with domain $\{x_1, \ldots, x_n\}$ can be denoted by $\{x_1/t_1, \ldots, x_n/t_n\}$, i.e. $\sigma(x_i) = t_i$ $(1 \leq i \leq n)$. The restriction of σ to a set V of variables is denoted by $\sigma_{|V}$. A substitution may be applied to quantified formulae; in that case, quantified variables are never replaced; e.g., $((\forall x)p(x, y))\{x/a, y/b\} = (\forall x)p(x, b)$.

2.2 Theory Reasoning

In general any satisfiable set of universally quantified formulae is a theory, i.e., we identify the theory with the defining set of axioms.

Definition 1. A *theory* $\mathcal{T} \subset \mathcal{L}$ is a satisfiable set of universally quantified formulae.

A $\mathcal{T}$*-interpretation* is an interpretation that satisfies $\mathcal{T}$.

A formula ϕ (a set of formulae Φ) is $\mathcal{T}$*-satisfiable* if there is a $\mathcal{T}$-interpretation satisfying ϕ (resp. Φ), else it is $\mathcal{T}$*-unsatisfiable*.

[2] We stress that the results presented in this paper can easily be adapted to other versions of semantic tableaux and similar calculi.

A sentence ϕ is a $\mathcal{T}$-*tautology* if it is satisfied by all $\mathcal{T}$-interpretations.

A formula ϕ is a $\mathcal{T}$-*consequence* of a set of formulae Ψ, denoted $\Psi \models_{\mathcal{T}} \phi$, if ϕ is satisfied by all $\mathcal{T}$-interpretations that satisfy Ψ.

The restriction to theories consisting of universally quantified formulae is necessary, because exactly for such formulae the Herbrand-type Theorem 2 holds [20], that is essential for the completeness of tableau-like calculi using theory reasoning. This restriction, however, is easy to get around, because existential quantifiers can be removed by skolemization.

Theorem 2. *A set Φ of universally quantified formulae is $\mathcal{T}$-unsatisfiable iff there is a finite set of ground instances of formulae from Φ that is $\mathcal{T}$-unsatisfiable.*

Example 1. The most important theory in practice is the equality theory $\mathcal{E}$. It consists of the following axioms:[3]

(1) $(\forall x)(x \approx x)$
(2) for all function symbols $f \in \mathcal{F}$ with arity $n \geq 0$:

$$(\forall x_1)\cdots(\forall x_n)(\forall y_1)\cdots(\forall y_n)((x_1 \approx y_1 \wedge \ldots \wedge x_n \approx y_n) \supset f(x_1,\ldots,x_n) \approx f(y_1,\ldots,y_n))$$

(3) for all predicate symbols $p \in \mathcal{P}$ with arity $n \geq 0$:

$$(\forall x_1)\cdots(\forall x_n)(\forall y_1)\cdots(\forall y_n)((x_1 \approx y_1 \wedge \ldots \wedge x_n \approx y_n) \supset (p(x_1,\ldots,x_n) \supset p(y_1,\ldots,y_n)))$$

Symmetry and transitivity of $\approx$ are implied by reflexivity (1) and monotonicity for predicate symbols (3), because $\approx \in \mathcal{P}$.

The following are the basic definitions for theory reasoning:

Definition 3. Let Φ be a set of formulae. Φ is $\mathcal{T}$-*complementary* iff every instance of Φ is $\mathcal{T}$-unsatisfiable.

Example 2. The formula $\neg(x \approx y)$ is $\mathcal{E}$-unsatisfiable; it is, however, not $\mathcal{E}$-complementary, because its instance $\neg(a \approx b)$ is $\mathcal{E}$-satisfiable. The formula $\neg(x \approx x)$ is both $\mathcal{E}$-unsatisfiable and $\mathcal{E}$-complementary.

Definition 4. Let Φ be a set of literals, called *key*. A set R of literals is a *residue* of Φ, if there is a substitution $\sigma \in \mathbf{Subst}$ such that

1. $\Phi\sigma \cup \overline{R}$ is $\mathcal{T}$-complementary ($\overline{R}$ denotes the negation $\{\neg\rho : \rho \in R\}$ of R),
2. $R = R\sigma$.

In that case, the pair $\langle\sigma, R\rangle$ is called a *refuter* for Φ. If the residue is empty, we identify the substitution σ with the refuter $\langle\sigma, \emptyset\rangle$.

It is neither really necessary to require the formulae in the key nor in the residue to be literals, but all further considerations are much simpler that way.

[3] The equality predicate is denoted by $\approx$, such that no confusion with the meta-level equality $=$ can arise.

2.3 Partial and Total Theory Reasoning

The central idea behind theory reasoning is the same for all calculi based in some way on Herband's theorem (tableau-like calculi, resolution, etc.) A key $\Phi \subset \Psi$ is chosen from the set Ψ of formulae already derived by the foreground reasoner and is passed to the background reasoner, which computes refuters $\langle \sigma, R \rangle$ for Φ.

There are two main approaches: if the background reasoner only computes refuters with an empty residue, we speak of *total* theory reasoning else of *partial* theory reasoning.

In the case of partial reasoning, where $R = \{\rho_1, \ldots, \rho_n\}$ $(n \geq 1)$, the formula $\rho_1 \vee \ldots \vee \rho_n$ is added to the derived formulae Ψ and the substitution σ is applied. If the foreground reasoner is then able to show that $(\Psi\sigma \cup \{\rho_1 \vee \ldots \vee \rho_n\})\tau$ is $\mathcal{T}$-unsatisfiable for some substitution τ, this proves that $\Psi\sigma\tau$ is $\mathcal{T}$-unsatisfiable: if $\Psi\sigma\tau$ were $\mathcal{T}$-satisfiable, then one of the sets $(\Psi\sigma \cup \{\rho_1 \vee \ldots \vee \rho_n\})\tau$ and $(\Phi\sigma \cup \overline{R})\tau$, that have been shown to be $\mathcal{T}$-unsatisfiable, had to be $\mathcal{T}$-satisfiable.

Although total theory reasoning can be seen as a special case of partial theory reasoning, the way the foreground reasoner makes use of the refuter is quite different: no further derivations have to been made by the foreground reasoner; $\Phi\sigma$ and thus $\Psi\sigma$ has been proven to be $\mathcal{T}$-complementary by the background reasoner. In the tableau framework, where (usually) the key Φ is taken from a tableau branch B, this closes B if the substitution σ is applied (see Section 3).

In the following, we restrict all our considerations to total theory reasoning; nevertheless, most of the techniques introduced in this paper are as well applicable to partial theory reasoning.

For completeness of the combination of foreground and background reasoner, the background reasoner has to compute sets of refuters that are—in a certain sense—complete. We use the following definition, which is strong enough to be sufficient for all theories and calculi:[4]

Definition 5. A set Σ of refuters is *complete* for a key Φ, if for each ground substitution $\sigma \in \mathbf{Subst}$ that is a refuter for Φ there is a $\sigma' \in \Sigma$ and a substitution τ such that $\sigma = \tau \circ \sigma'$.

3 Semantic Tableaux

3.1 Free Variable Semantic Tableaux

First, we formally define the free variable tableau calculus, using a slightly non-standard representation:[5] Tableaux are multi-sets of multi-sets of first-order formulae; as usual, the branches of a tableau are implicitly disjunctively connected and the formulae on a branch are implicitly conjunctively connected.

Definition 6. A *tableau* is a (finite) multi-set of tableau branches, where a tableau *branch* is a (finite) multi-set of first order formulae.

There are two possibilities to derive a new tableau from an old one: (1) applying a tableau expansion rule and (2) closing a branch by applying a substitution to the tableau (incremental theory reasoning provides a third possibility: calling the background reasoner, see Sec. 4.4).

The expansion rules are the classical α-, β-, γ- and δ-rules for first-order formulae. The rule patterns are summarized in Table 1.[6]

[4] Depending on the actual theory and the calculus used, weaker requirements may be

α	α_1	α_2
$\phi \wedge \psi$	ϕ	ψ
$\neg(\phi \vee \psi)$	$\neg\phi$	$\neg\psi$
$\neg(\phi \supset \psi)$	ϕ	$\neg\psi$
$\neg\neg\phi$	ϕ	ϕ

β	β_1	β_2
$\phi \vee \psi$	ϕ	ψ
$\neg(\phi \wedge \psi)$	$\neg\phi$	$\neg\psi$
$\phi \supset \psi$	$\neg\phi$	ψ
$\phi \leftrightarrow \psi$	$\phi \wedge \psi$	$\neg\phi \wedge \neg\psi$
$\neg(\phi \leftrightarrow \psi)$	$\phi \wedge \neg\psi$	$\neg\phi \wedge \psi$

γ	$\gamma_1(y)$
$(\forall x)\phi(x)$	$\phi(y)$
$\neg(\exists x)\phi(x)$	$\neg\phi(y)$

δ	$\delta_1(t)$
$\neg(\forall x)\phi(x)$	$\neg\phi(t)$
$(\exists x)\phi(x)$	$\phi(t)$

$$\frac{\alpha}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta}{\beta_1 \mid \beta_2} \qquad \frac{\gamma}{\gamma_1(y)} \qquad \frac{\delta}{\delta_1(f(x_1,\ldots,x_n))}$$

where y is a new free variable.

where f is a new (Skolem) function symbol, and $x_1, \ldots, x_n$ are the free variables occurring in δ.

Table 1. Formula types and tableau rule schemata.

To prove a formula ϕ to be a tautology, we start from the initial tableau $\{\{\neg\phi\}\}$.[7] New tableaux are derived by applying the tableau expansion rules and closing branches by applying a substitution.

Definition 7. A tableau branch B is *closed* under a substitution σ iff there are formulae $\phi, \neg\psi \in B$ such that $\phi\sigma = \psi\sigma$, i.e., $\phi\sigma$ and $\neg\psi\sigma$ are complementary.

The problem of finding a single substitution that closes all branches of a tableau simultaneously is simplified—as usual in practical applications—by closing the branches one after the other: if a substitution is found that closes a single branch, it is applied to the whole tableau to close that branch, before other branches are considered. Closed branches are removed from the tableau instead of just marking them as being closed. Thus, a proof is found, when the empty tableau has been derived.

Theorem 8 (Soundness and Completeness). *A first-order sentence ϕ is a tautology iff there is a sequence*

$$\{\{\neg\phi\}\} = T_0, T_1, \ldots, T_{n-1}, T_n = \emptyset \qquad (n \geq 0)$$

of tableaux such that for $1 \leq i \leq n$ the tableau T_i is constructed from T_{i-1} by

1. *applying one of the expansion rules from Table 1, i.e., there is a branch $B \in T_{i-1}$ and a formula $\phi \in B$ (that is not a literal) such that*

$$T_i = \begin{cases} (T_{i-1} \setminus \{B\}) \cup \{(B \setminus \{\phi\}) \cup \{\alpha_1, \alpha_2\}\} & \text{if } \phi \text{ is of type } \alpha \\ (T_{i-1} \setminus \{B\}) \cup \{(B \setminus \{\phi\}) \cup \{\beta_1\}, (B \setminus \{\phi\}) \cup \{\beta_2\}\} & \text{if } \phi \text{ is of type } \beta \\ (T_{i-1} \setminus \{B\}) \cup \{B \cup \{\gamma_1\}\} & \text{if } \phi \text{ is of type } \gamma \\ (T_{i-1} \setminus \{B\}) \cup \{(B \setminus \{\phi\}) \cup \{\delta_1\}\} & \text{if } \phi \text{ is of type } \delta \end{cases}$$

2. *or closing a branch $B \in T_{i-1}$, i.e., $T_i = (T_{i-1} \setminus \{B\})\sigma$, where the branch B is closed under σ (Def. 7).*

sufficient to preserve completeness.

[5] We stress that this calculus differs from classical free variable tableaux [11] only in notation and the way tableaux are represented.

[6] The δ-rule is more liberal than that used in [11]; it has recently been proposed and proven sound by Hähnle and Schmitt [14]. Even more liberalized δ-rules have been investigated in [7] and [1].

[7] If visualized as a binary tree, the initial tableau consists of the single node $\neg\phi$.

The construction of a closed tableau is a highly non-deterministic process, because at each step one is (in general) free (1) to choose a branch B of the tableau, (2) to expand or to close B, and to choose (3a) a formula $\phi \in B$ for expansion or (3b) a substitution that closes B.

3.2 Semantic Tableaux with Total Theory Reasoning

We make use of the fact, that if there is a $\mathcal{T}$-refuter for a key taken from a tableau branch B then B and all its instances are $\mathcal{T}$-unsatisfiable:

Definition 9. Given a theory $\mathcal{T}$, a tableau branch B is $\mathcal{T}$*-closed* under a substitution σ if σ is a refuter for a key $\Phi \subset B$.

Using the above definition, Theorem 8 can easily be adapted to theory reasoning:

Theorem 10 (Soundness and Completeness, Theory Reasoning).
Given a theory $\mathcal{T}$, a first-order sentence ϕ is a $\mathcal{T}$-tautology iff there is a sequence

$$\{\{\neg\phi\}\} = T_0, T_1, \ldots, T_{n-1}, T_n = \emptyset \qquad (n \geq 0)$$

of tableaux, such that for $1 \leq i \leq n$ the tableau T_i is constructed from T_{i-1} by

1. *applying one of the expansion rules from Table 1 (see Theorem 8 for a formal definition),*
2. *or closing a branch $B \in T_{i-1}$, i.e., $T_i = (T_{i-1} \setminus \{B\})\sigma$, where the branch B is $\mathcal{T}$-closed under σ (Def. 9).*

4 Incremental Background Reasoners

4.1 Motivation

As already mentioned in the introduction, one of the main problems in using theorem reasoning techniques in practice is the efficient combination of foreground and background reasoner and their interaction:

– A late call to the background reasoner can lead to bigger tableaux and redundancy. Although several branches may share the same subbranch and thus contain the same key for which a refuter exists, the background reasoner is called separately for these branches and the refuter has to be computed multiply.
– On the other hand, an early call to the background reasoner may not be successful and time consuming; this is of particular disadvantage if the theory is undecidable, and as a result the background reasoner might not terminate although no refuter exists.

Both these phenomena may considerably decrease the performance of a prover, and it is very difficult to decide (resp. to develop good heuristics that decide)

1. when to call the background reasoner;
2. when to stop the background reasoner if it does not find a refuter.

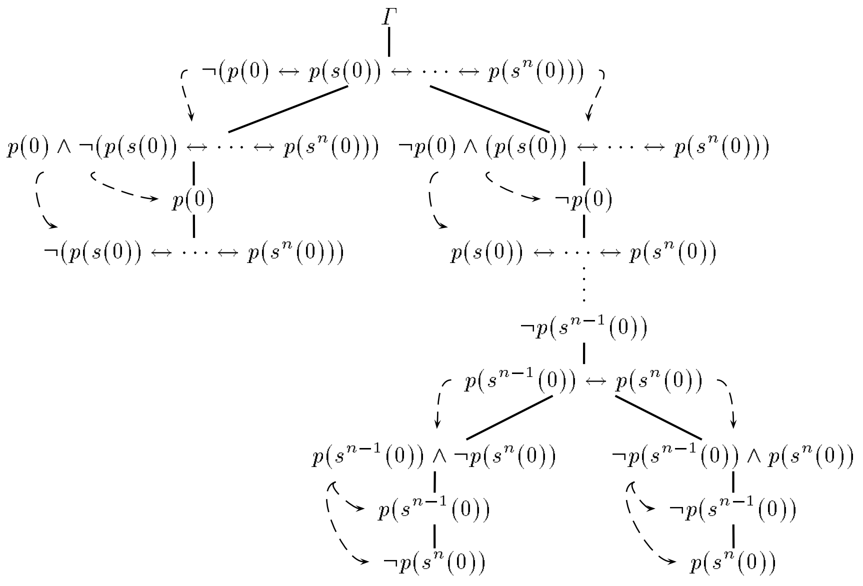
Example 3. The following example shows that earlier calls to the background reasoner can reduce the size of a tableau proof exponentially: Assume Γ to be a set of formulae such that $\Gamma \models_{\mathcal{T}} \neg p(s^n(0))$ $(n \geq 0)$ for some theory $\mathcal{T}$. Figure 1 shows a proof for

$$\Gamma \models_{\mathcal{T}} (p(0) \leftrightarrow p(s(0)) \leftrightarrow \cdots \leftrightarrow p(s^n(0))) \ ,$$

where the background reasoner is called when a literal of the form $p(s^n(0))$ appears on a branch. As a result, all the left-hand branches are closed immediately and the tableau is of linear size in n.

If the background reasoner were only called when a branch is exhausted, i.e., when no further expansion is possible, then the tableau would have 2^n branches and the background reasoner would have to be called 2^n times (instead of n times).

An *incremental* background reasoner can be of additional advantage, if the computations that are necessary to show that $\Gamma \models_{\mathcal{T}} p(s^n(0))$ are similar for all n. In that case a single call to the background reasoner in the beginning may provide information that later can be reused to close all the branches with less effort.



Γ
$\neg(p(0) \leftrightarrow p(s(0)) \leftrightarrow \cdots \leftrightarrow p(s^n(0)))$
$p(0) \land \neg(p(s(0)) \leftrightarrow \cdots \leftrightarrow p(s^n(0)))$
$\neg p(0) \land (p(s(0)) \leftrightarrow \cdots \leftrightarrow p(s^n(0)))$
$p(0)$
$\neg(p(s(0)) \leftrightarrow \cdots \leftrightarrow p(s^n(0)))$
$\neg p(0)$
$p(s(0)) \leftrightarrow \cdots \leftrightarrow p(s^n(0))$
$\neg p(s^{n-1}(0))$
$p(s^{n-1}(0)) \leftrightarrow p(s^n(0))$
$p(s^{n-1}(0)) \land \neg p(s^n(0))$
$\neg p(s^{n-1}(0)) \land p(s^n(0))$
$p(s^{n-1}(0))$
$\neg p(s^n(0))$
$\neg p(s^{n-1}(0))$
$p(s^n(0))$

Fig. 1. Short tableau proof for $\Gamma \models_{\mathcal{T}} p(0) \leftrightarrow p(s(0)) \leftrightarrow \cdots \leftrightarrow p(s^n(0))$ (Example 3).

Even the best heuristic cannot avoid calls to the background reasoner at the wrong time. However, on certain conditions it is possible to avoid the adverse consequences of early calls: If the algorithm the background reasoner uses is *incremental*, i.e., if the data produced by the background reasoner during a futile try to compute refuters can be reused for a later call.

If early calls have no negative effects, the disadvantages of late calls can easily be avoided by using heuristics that, in case of doubt, call the background reasoner at an early time. The problem of not knowing when to stop the background reasoner is solved by calling it more often with less resources (time, etc.) for each call.

An additional advantage of using incremental background reasoners in the tableau framework is that computations can be reused multiply for different extensions of a branch—even if the computation of refuters proceeds differently for these extensions.

4.2 Incremental Keys

Obviously there has to be some strong relation between the keys transferred to the background reasoner, to make it possible to reuse the information computed.

Since, between calls to the background reasoner, we want to (1) extend the tableau by new formulae and (2) apply substitutions (to the tableau), these are the two operations we want to allow for changing the keys:

Definition 11. A sequence $(\Phi_i)_{i \geq 0}$ of keys is *incremental* if for $i \geq 0$ there is a set Ψ_i of literals and a substitution σ_i such that

$$\Phi_{i+1} = \Phi_i \sigma_i \cup \Psi_i$$

where $\Psi_i = \Psi_i \sigma_i$.

In general, not all refuters of Φ_i are refuters of Φ_{i+1} (because a substitution is applied); nor are all refuters of Φ_{i+1} refuters of Φ_i (because new formulae are added).

4.3 Iterative and Incremental Algorithms

To be able to formally denote the state the computation of a background reasoner has reached and the data generated, we use the following notion of background reasoner:

Definition 12. A *background reasoner* is a triple $\langle \mathcal{A}, \mathcal{I}, \mathcal{S} \rangle$. $\mathcal{A}$ is an *algorithm* (a function) operating on a data structure $\mathbf{D}$:

$$\mathcal{A} \;:\; \mathbf{D} \longrightarrow \mathbf{D}$$

$\mathcal{I}$ is an initialization function that transforms a given key into the data structure format:

$$\mathcal{I} \;:\; 2^{\{L \in \mathcal{L}_{\mathrm{Sko}} \,:\, L \text{ is a literal}\}} \longrightarrow \mathbf{D}$$

The output function $\mathcal{S}$ extracts computed refuters from the data structure:

$$\mathcal{S} \;:\; \mathbf{D} \longrightarrow 2^{\mathbf{Subst}}$$

Of course, the input and output functions have to be reasonably easy to compute; in particular the cost of their computation has to be much smaller than that of applying the algorithm $\mathcal{A}$,[8] which is supposed to do the actual work.

For the sake of simplicity, we focus on algorithms that are iterative in the following sense:[9]

Definition 13. A background reasoner is *iterative* if for every key Φ and $i \leq j$

$$\mathcal{S}(\mathcal{A}^i(\mathcal{I}(\Phi))) \subset \mathcal{S}(\mathcal{A}^j(\mathcal{I}(\Phi))) \;.^{10}$$

It is *sound* if for every key Φ and $i \geq 0$

$$\mathcal{S}(\mathcal{A}^i(\mathcal{I}(\Phi)))$$

is a set of refuters for Φ.

It is *complete* if for every key Φ

$$\bigcup_{i \geq 0} \mathcal{S}(\mathcal{A}^i(\mathcal{I}(\Phi)))$$

is a complete set of refuters for Φ (Def. 5).

[8] In practice their cost should be linear or at most polynomially in the input.

[9] This is no real restriction: If a background reasoner applies different transformations to the data at each step of its computation, this can be modeled by adding the index i and the state of the reasoner to the data structure, such that the right operation or sub-algorithm can be applied each time the background reasoner is invoked.

[10] In practice one weakens this condition to the extend that it is allowed to remove refuters that are subsumed by other refuters.

Our goal is to be able to stop the background reasoner when it has reached a certain state in its computations for a key Φ, and to proceed from that state with a new key $\Phi' = \Phi\sigma \cup \Psi$. For that purpose we need an update function, that adapts the data structure representing the state of the computation to the new literals Ψ and the substitution σ.

Definition 14. Let $\mathcal{T}$ be a theory and $\mathcal{R} = \langle \mathcal{A}, \mathcal{I}, \mathcal{S} \rangle$ a sound and complete iterative background reasoner for $\mathcal{T}$. An update function

$$\mathcal{U} : \mathbf{D} \times 2^{\{L \in \mathcal{L}_{\mathrm{Sko}} : L \text{ is a literal}\}} \times \mathbf{Subst} \longrightarrow \mathbf{D}$$

is *correct* (for $\mathcal{R}$), if for every key $\Phi' = \Phi\sigma \cup \Psi$ and

$$D_n = \mathcal{U}(\mathcal{A}^n(\mathcal{I}(\Phi)),\ \Psi,\ \sigma) \qquad (n \geq 0)$$

1. $\mathcal{S}(\mathcal{A}^i(D_n))$ is a set of refuters for Φ' for all $i \geq 0$ (*soundness*);
2. $\bigcup_{i\geq 0} \mathcal{S}(\mathcal{A}^i(D_n))$ is a *complete* set of refuters for Φ' (*completeness*).

According to the above definition a correct update function behaves as expected when used for a single incremental step. Theorem 15 shows that this behavior extends to sequences of incremental steps. In addition, the algorithm can be applied arbitrarily often between incremental steps:

Theorem 15. *Let $(\Phi_i)_{i\geq 0}$ be an incremental sequence of keys, where*

$$\Phi_{i+1} = \Phi_i\sigma_i \cup \Psi_i \qquad (i \geq 0)\ ,$$

$\mathcal{R}$ a sound and complete iterative background reasoner (Def. 13), and $\mathcal{U}$ a correct update function for $\mathcal{R}$ (Def. 14). Let $(D_i)_{i\geq 0} \subset \mathbf{D}$ be defined by

1. *$D_0 = \mathcal{I}(\Phi_0)$,*
2. *$D_{i+1} = \mathcal{U}(\mathcal{A}^{n_i}(D_i),\ \sigma_{i+1},\ \Psi_{i+1})$ for some $n_i \geq 0$.*

Then

1. *$\mathcal{S}(D_i)$ is a set of refuters for Φ_i for all $i \geq 0$ (*soundness*) and*
2. *$\bigcup_{j\geq 0} \mathcal{S}(\mathcal{A}^j(D_i))$ is a complete set of refuters for Φ_i (*completeness*).*[11]

Example 4. Let $(\Phi_i)_{i\geq 0}$ be an incremental sequence of keys, where $\Phi_{i+1} = \Phi_i\sigma_i \cup \Psi_i$ $(i \geq 0)$. Then for every sound and complete iterative background reasoner $\langle \mathcal{A}, \mathcal{I}, \mathcal{S} \rangle$ the trivial update function defined by

$$\mathcal{U}(D, \Psi_i, \sigma_i) = \mathcal{I}(\Phi_i\sigma_i \cup \Psi_i)$$

is correct.

The above example shows that it is not sufficient to use any correct update function to achieve a better performance of the calculus, because using the trivial update function means that no information is reused. A useful update function has to preserve the information contained in the computed data.

Whether there actually is a useful and reasonably easy to compute update function depends on the theory $\mathcal{T}$, the background reasoner, and its data structure. (In Section 6 we present an example for such a useful update function.)

[11] $\bigcup_{i\geq 0} \mathcal{S}(D_i)$ is (in general) not a complete set of refuters for any of the keys, since no inclusion relation holds for the sets of refuters of an incremental sequence of keys.

4.4 Semantic Tableaux and Incremental Theory Reasoning

The incremental theory reasoning method presented in the previous section is easy to use for tableau-like calculi, because the definition of incremental sequences of keys matches the construction of tableau branches. The keys of a sequence are taken from an expanding branch, and the substitutions are those applied to the whole tableau.

The keys used in calls to the background reasoner, as well as the information computed so far by the background reasoner, have to be attached to the tableau branches:[12]

Definition 16. A *tableau* is a (finite) multi-set of tableau branches, where a tableau *branch* is triple $\langle \Theta, D, \Phi \rangle$; Θ is a (finite) multi-set of first order formulae, $D \in \mathbf{D}$ (where $\mathbf{D}$ is the data structure used by the background reasoner), and Φ is a set of literals (a key).

Now, the free variable tableau calculus introduced in Section 3.2 can be adapted to incremental theory reasoning: calling the background reasoner is added as a third possibility to change the tableau (besides expanding and closing branches). Soundness and completeness of the resulting calculus is a corollary of Theorems 10 and 15:

Theorem 17 (Soundness and Completeness, Incremental Version). *Given a theory $\mathcal{T}$, a sound and complete background reasoner $\mathcal{R} = \langle \mathcal{A}, \mathcal{I}, \mathcal{S} \rangle$ for $\mathcal{T}$ (Def. 13), and a correct update function $\mathcal{U}$ for $\mathcal{R}$ (Def. 14).*

A first-order sentence ϕ is a $\mathcal{T}$-tautology iff there is a sequence

$$\{\langle \{\neg\phi\}, \mathcal{I}(\emptyset), \emptyset \rangle\} = T_0, T_1, \ldots, T_{n-1}, T_n = \emptyset \qquad (n \geq 0)$$

of tableaux (Def. 16) such that for $1 \leq i \leq n$ the tableau T_i is constructed from T_{i-1} by

1. *applying one of the expansion rules from Table 1, i.e., there is a branch $B = \langle \Theta, D, \Phi \rangle \in T_{i-1}$ and a formula $\phi \in \Theta$ (that is not a literal) such that*

$$T_i = \begin{cases} (T_{i-1} \setminus \{B\}) \cup \{\langle (\Theta \setminus \{\phi\}) \cup \{\alpha_1, \alpha_2\}, D, \Phi \rangle\} & \textit{if } \phi \textit{ is of type } \alpha \\ (T_{i-1} \setminus \{B\}) \cup \{\langle (\Theta \setminus \{\phi\}) \cup \{\beta_1\}, D, \Phi \rangle, & \\ \qquad\qquad \langle (\Theta \setminus \{\phi\}) \cup \{\beta_2\}, D, \Phi \rangle\} & \textit{if } \phi \textit{ is of type } \beta \\ (T_{i-1} \setminus \{B\}) \cup \{\langle \Theta \cup \{\gamma_1\}, D, \Phi \rangle\} & \textit{if } \phi \textit{ is of type } \gamma \\ (T_{i-1} \setminus \{B\}) \cup \{\langle (\Theta \setminus \{\phi\}) \cup \{\delta_1\}, D, \Phi \rangle\} & \textit{if } \phi \textit{ is of type } \delta \end{cases}$$

2. *closing a branch $B = \langle \Theta, D, \Phi \rangle \in T_{i-1}$, i.e.,*

$$T_i = \{\langle \Theta'\sigma, D', \Phi' \rangle \ : \ \langle \Theta', D', \Phi' \rangle \in (T_{i-1} \setminus \{B\})\} ,$$

   where $\sigma \in \mathcal{S}(D)$,
3. *or calling the background reasoner, i.e., there is a branch*

$$B = \langle \Theta, D, \Phi \rangle \in T_{i-1} ,$$

   a number $c > 0$ of applications, and a key Φ' of the form[13]

$$\Phi' = \Phi\sigma \cup \Psi \subset \Theta$$

   such that

$$T_i = (T_{i-1} \setminus \{B\}) \cup \{\langle \Theta, \mathcal{A}^c(\mathcal{U}(D, \Psi, \sigma)), \Phi' \rangle\} .$$

[12] If only *maximal* keys are used (all literals on the branch), the keys do not have to be attached to the branch.

[13] There is always a key satisfying this condition (in particular the set of all literals on the branch).

4.5 Achievable Cost Reduction

The maximal cost reduction that can be achieved by using an incremental reasoner is reached if the costs are those of the non-incremental background reasoner called neither too early nor too late, i.e., if always the right key in the incremental sequence is chosen and the background reasoner is only called for that key (which is not possible in practice).

More formally: If we search for a substitution τ that is a refuter for one or more of the keys in an incremental sequence $(\Phi_i)_{i \geq 0}$ (where $\Phi_{i+1} = \Phi_i \sigma_i \cup \Psi_i$), then the index i_{min} of the "right" key and the minimal number of applications of the background reasoner n_{min} are defined by:

$$\begin{aligned} i_{min} &= \min\{k \geq 0 \ : \ \tau \text{ is a refuter for } \Phi_k\} \\ n_{min} &= \min\{n \geq 0 \ : \ \text{there is a } \tau' \in \mathcal{S}(\mathcal{A}^n(\mathcal{I}(\Phi_{i_{min}}))) \text{ more general than } \tau\} \ . \end{aligned}$$

Thus, the minimal costs of finding τ using a non-incremental approach are:[14]

$$cost(\mathcal{I}, \Phi_{i_{min}}) + cost(\mathcal{A}^{n_{min}}, \mathcal{I}(\Phi_{i_{min}})) \ .$$

However, these minimal costs cannot be reached using a deterministic non-incremental background reasoner, because the index i_{min} is not known (which is equivalent to the problem of early/late calls).

The costs of an incremental approach depend on the number c_i of applications of the algorithm during step i. The number j of incremental steps that have to be made until τ is found can be bigger than i_{min} (if the c_i have been chosen too small). The costs are:

$$cost(\mathcal{I}, \Phi_0) \ + \ \sum_{i=0}^{j-1} cost(\mathcal{U}, (D'_i, \Psi_i, \sigma_i)) \ + \ \sum_{i=0}^{j} cost(\mathcal{A}^{c_i}, D_i) \ ,$$

where $D_0 = \mathcal{I}(\Phi_0)$, $D_i = \mathcal{U}(D'_{i-1}, \Psi_{i-1}, \sigma_{i-1})$ (for $i \geq 1$), and $D'_i = \mathcal{A}^{c_i}(D_i)$ (for $i \geq 0$). The actual costs become smaller and approach their minimum, if the costs of applying the update function approach zero, if all the information computed by the background reasoner can be reused for a later call, and if the numbers c_i of applications have not been chosen too small.

If substitutions are applied, i.e., if the σ_i are not the empty substitution, usually not all information derived for a key Φ_i can be reused, because part of it becomes invalid for an instance $\Phi_i \sigma_i$ (see Section 6).

In practice, the costs of an incremental method are between the ideal value and the costs of calling a non-incremental reasoner for each of the keys in an incremental sequence (without reusing).

But even if the costs for one sequence, i.e., for closing one tableau branch, are higher than that of using a non-incremental method, the overall costs for closing the whole tableau can be small because information is reused for more than one branch.

5 Equality Handling in Semantic Tableaux

If total theory reasoning methods are employed for handling equality in free variable tableaux, the background reasoner has to solve rigid E-unification problems [12] to compute refuters:

[14] $cost(f, x)$ denotes the costs of computing the application of the function f to the argument x.

Definition 18. A *rigid E-unification problem* $\langle E, s, t\rangle$ consists of a finite set E of equalities $(l \approx r) \in \mathcal{L}_{\mathrm{Sko}}$ and terms s and t.

A substitution σ is a *solution* to the problem iff $E\sigma \models_{\mathcal{E}} (s\sigma \approx t\sigma)$ where the free variables in $E\sigma$ are "held rigid", i.e. treated as constants.

A complete set of refuters for a key K can be computed by extracting the set $P(K)$ of rigid E-unification problems from K according to the following definition and solving the problems in $P(K)$:

Definition 19. Let K be a key. Then $E(K) = \{l \approx r \ : \ (l \approx r) \in K\}$ is the *set of equalities in* K, and $P(K) =$

$$\begin{array}{ll} \{\langle E(K), \langle s_1, \ldots, s_n\rangle, \langle t_1, \ldots, t_n\rangle\rangle : & p(s_1, \ldots, s_n), \neg p(t_1, \ldots, t_n) \in K, \, p \not\approx\} \ \cup \\ \{\langle E(K), s, t\rangle & : \ \neg(s \approx t) \in K\} \end{array}$$

is the *set of rigid E-unification problems in K.*

Theorem 20. *For any key K the set of solutions to the rigid E-unification problems in $P(K)$ is a complete set of refuters for K (w.r.t. the equality theory $\mathcal{E}$).*

Various methods for computing rigid E-unifiers have been described [12, 9, 5], the most efficient of which are completion-based.[15] Fortunately, completion-based methods for rigid E-unification can easily be used for *incremental* background reasoning: Let $(\Phi_i)_{i \geq 0}$ be a sequence of incremental keys, then the following equations hold for the sequence $(P(\Phi_i))_{i \geq 0}$ of corresponding rigid E-unification problems:

$$\begin{aligned} E(\Phi_{i+1}) &= E(\Phi_i \sigma_i) \cup E(\Psi_i) \\ P(\Phi_{i+1}) &= P(\Phi_i \sigma_i) \cup P(\Psi_i) \cup P' \end{aligned}$$

(where P' contains additional E-unification problems extracted from literals $p(s_1, \ldots, s_n)$, $\neg p(t_1, \ldots, t_n)$ one of which is in $\Phi_i \sigma_i$ and one of which is in Ψ_i). Therefore, a correct update function only has to

1. apply the substitution σ_i to the old set of E-unification problems and rewrite rules,
2. add the new rewrite rules and E-unification problems to the old ones, and
3. remove the rewrite rules that are not valid for the substitution σ_i (these rules constitute information that cannot be reused).

6 Implementation

A completion-based method for solving *mixed* E-unification problems [5], which is an extension of rigid E-unification,[16] has been implemented as part of the

[15] We use the version of total theory reasoning in semantic tableaux where branches are closed one after the other. To close all branches simultaneously, a simultaneous rigid E-unification problem has to be solved. This is much more difficult than the non-simultaneous version: simultaneous rigid E-unification is undecidable [10] whereas the non-simultaneous problem is NP-complete [12].

[16] Mixed E-unification is a combination of the classical universal E-unification and rigid E-unification. The performance of provers using E-unification for handling equality can be increased considerably, if mixed E-unification is used instead of the purely rigid version: An equality has often to be applied more than once in a proof, each time with different substitutions for the variables occurring in it. In tableau-like calculi the mechanism to do so is to generate several instances of the equality. It is, however, often possible to recognize equalities that are "universal" w.r.t. variables they contain (e.g. equalities that occur on only one branch of a tableau). If *mixed* E-unification is used, this knowledge can be used to avoid generating additional copies of equalities.

tableau-based theorem prover $_3T^AP$ [6, 13]. The E-unification problems extracted from a branch (resp. key) are transformed into (sets of) constrained terms and rewrite rules; the constraints describe the sets of substitutions for which, if the substitution is applied to the tableau, a derived term or rewrite rule remains valid. An algorithm that can be seen as an extension of the Unfailing Knuth-Bendix-Algorithm [15] with narrowing [18] is employed to search for refuters. In $_3T^AP$ only maximal keys are used, i.e., all literals from a tableau branch. The indeterminism of free variable tableaux is resolved by closing branches from left to right, using a fixed order in which formulae are expanded, and backtracking w.r.t. the substitutions that are applied to the tableau: if a branch cannot be closed, the last application of a substitution σ is undone and other closing substitutions are searched for that close the same branches as σ.

In the old version of $_3T^AP$ information computed by the background reasoner could not be reused, and the background reasoner was either

- only called for exhausted tableau branches, i.e., if no expansion rule was applicable (observing a limit on the number of γ-rule applications), which usually led to late calls; or
- called each time before a β-rule was applied; which usually led to early calls.

Fortunately, due to the inherently incremental nature of $_3T^AP$'s algorithm for solving rigid E-unification problems, it has been easy to design and implement a correct and reasonably simple update function $\mathcal{U}(D, \Psi, \sigma)$: rewrite rules and unification problems are extracted from the new literals in Ψ; they can be added to the data structure D without any further changes. The substitution σ is applied to the constrained rules and terms in D. Which rewrite rules and terms are not valid for σ and have to be removed can be checked using the constraints attached to rules and terms (experiments show that in practice only few rules and terms have to be removed).

The new incremental version of the background reasoner is always called before a β-rule is applied. The number of iterative steps during a call is determined by a heuristic, that the user can affect by changing certain parameters.

7 Experiments and Results

In the following we present some experimental data obtained using the implementation described in the previous section. Three different theory reasoning methods are compared:

1. Calling the background reasoner each time before a β-rule is applied
   (a) reusing the computed information for later calls (reuse),
   (b) without reusing information (no reuse);
2. calling the background reasoner for exhausted branches only (late call).[17]

The generated tableaux are in general the same for Cases 1a and 1b,[18] they are different (larger) if the background reasoner is only called for exhausted branches (Case 2). In the statistics TR is the number of tableau rule applications and EQ denotes the number of calls to the equality background reasoner. Proof times are given in seconds, running on a SUN SPARC 10 ("∞" means that no proof could be found in reasonable time).

[17] A branch is exhausted if no expansion rule can be applied (observing a pre-defined limit on the number of γ-rule applications).

[18] They can differ, if without reusing information (Case 1b) a limit (e.g. on the number of equality applications) is reached before a branch is closed, and that limit is not reached if information is reused (Case 1a).

			branches closed		time [sec]		
Problem	TR	EQ	background	foreground	reuse	no reuse	late call
pel48	4	7	4	0	0.75	0.95	0.76
pel49	27	21	14	0	25.88	29.42	28.79
pel51	29	20	8	4	4.32	4.38	3.96
pel52	26	18	8	2	5.15	5.19	4.59
pel55	102	30	4	20	8.95	5.74	32.73
hash3	334	151	76	0	25.25	61.33	∞
hash9	929	545	273	0	84.77	∞	∞
hash11	250	63	32	0	27.23	43.72	∞
hash12	173	19	10	0	14.45	31.96	∞
hash13	260	63	32	0	34.40	39.06	∞
hash25	530	251	126	0	50.62	∞	∞

Table 2. Statistics for some of Pelletier's problems (pel) and problems from program verification (hash).

Table 2 shows results for some of Pelletier's problems [19] (pel) and problems taken from an application in program verification where lemmata on a specification of hash tables are to be proven (hash).

The tableaux for Pelletier's problems are quite small. Here, reusing information does not lead to an improvement, neither does it have any negative effects. The proof for problem pel55 is shortened considerably by making early calls to the background reasoner.

The more difficult examples from program verification show that the improvement gained by reusing information corresponds roughly to the size of the tableau proof: the more branches there are, the more re-computations of the same information can be avoided.

8 Conclusion

Incremental theory reasoning is a technique that improves the interaction between foreground and total background reasoner. The adverse effects of early or late calls to the background reasoner may—if only partially—be avoided; in addition, information computed by the background reasoner can be reused multiply to compute refuters for different extensions of a key.

The experimental evidence presented in Section 7 shows that—although in practice not all information can be reused—using incremental methods may indeed increase the overall performance of a deduction system.

Up to now most of the work in theory reasoning has been directed towards designing more efficient foreground and background reasoners. However, our work shows, that they should not be completely separated; their interaction is equally important. Besides using incremental methods, it is essential to develop good heuristics (depending on the theory or domain) in order to decide when, with which key, and for how long to call the background reasoner.

References

1. M. Baaz and C. G. Fermüller. Non-elementary speedups between different versions of tableaux. In *Proceedings, 4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods, St. Goar*, LNCS 918, pages 217–230. Springer, 1995.

2. P. Baumgartner. A model elimination calculus with built-in theories. In H.-J. Ohlbach, editor, *Proceedings, German Workshop on Artificial Intelligence (GWAI)*, LNCS 671, pages 30–42. Springer, 1992.
3. P. Baumgartner, U. Furbach, and U. Petermann. A unified approach to theory reasoning. Forschungsbericht 15/92, University of Koblenz, 1992.
4. B. Beckert. Adding equality to semantic tableaux. In K. Broda, M. D'Agostino, R. Goré, R. Johnson, and S. Reeves, editors, *Proceedings, 3rd Workshop on Theorem Proving with Analytic Tableaux and Related Methods, Abingdon*, pages 29–41, Imperial College, London, TR-94/5, 1994.
5. B. Beckert. A completion-based method for mixed universal and rigid E-unification. In A. Bundy, editor, *Proceedings, 12th International Conference on Automated Deduction (CADE), Nancy, France*, LNCS 814, pages 678–692. Springer, 1994.
6. B. Beckert, S. Gerberding, R. Hähnle, and W. Kernig. The tableau-based theorem prover ${}_3T^AP$ for multiple-valued logics. In *Proceedings, 11th International Conference on Automated Deduction (CADE), Saratoga Springs, NY*, LNCS 607, pages 758–760. Springer, 1992.
7. B. Beckert, R. Hähnle, and P. H. Schmitt. The even more liberalized δ-rule in free variable semantic tableaux. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Proceedings, 3rd Kurt Gödel Colloquium (KGC), Brno, Czech Republic*, LNCS 713, pages 108–119. Springer, 1993.
8. W. Bibel. *Automated Theorem Proving.* Vieweg, Braunschweig, second revised edition, 1987.
9. E. de Kogel. Rigid E-unification simplified. In *Proceedings, 4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods, St. Goar*, LNCS 918, pages 17–30. Springer, 1995.
10. A. Degtyarev and A. Voronkov. Simultaneous rigid E-unification is undecidable. UPMAIL Technical Report 105, Uppsala University, May 1995. Presented at: *Annual Conference of the European Association for Computer Science Logic (CSL'95), Paderborn.*
11. M. C. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer, 1990.
12. J. H. Gallier, P. Narendran, S. Raatz, and W. Snyder. Theorem proving using equational matings and rigid E-unification. *Journal of the ACM*, 39(2):377–429, Apr. 1992.
13. R. Hähnle, B. Beckert, and S. Gerberding. The many-valued tableau-based theorem prover ${}_3T^AP$. Tr 30/94, Universität Karlsruhe, Fakultät für Informatik, Nov. 1994.
14. R. Hähnle and P. H. Schmitt. The liberalized δ-rule in free variable semantic tableaux. *Journal of Automated Reasoning,*, 13(2):211–222, Oct. 1994.
15. D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebras*, pages 263–297. Pergamon Press, Oxford, 1970.
16. D. W. Loveland. A simplified format for the model elimination procedure. *Journal of the ACM*, 16(3):233–248, July 1969.
17. N. V. Murray and E. Rosenthal. Theory links: Applications to automated theorem proving. *Journal of Symbolic Computation*, 4:173–190, 1987.
18. W. Nutt, P. Réty, and G. Smolka. Basic narrowing revisited. *Journal of Symbolic Computation*, 7(3/4):295–318, 1989.
19. F. J. Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2:191–216, 1986.
20. U. Petermann. How to build-in an open theory into connection calculi. *Journal on Computer and Artificial Intelligence*, 11(2):105–142, 1992.
21. R. Smullyan. *First-Order Logic.* Springer, 1968.
22. M. E. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1:333–355, 1985.