T-resolution: Refinements and Model Elimination

Andrea Formisano
Univ. of Rome *"La Sapienza"*, Dept. of Computer Science,
Via Salaria 113, 00198 ROMA (I)
formisan@dsi.uniroma1.it

Alberto Policriti
Univ. of Udine, Dept. of Mathematics and Computer Science,
Via delle Scienze 206, 33100 UDINE (I)
policrit@dimi.uniud.it

September 22, 1998

Abstract

T-resolution is a binary rule, proposed in [PS95] for theorem proving in first-order theories (T-theorem proving) that can be seen —at least at the ground level— as a variant of Stickel's *theory* resolution. In this paper we consider refinements of this rule as well as the Model Elimination variant of it.

After a general discussion concerning our viewpoint on theorem proving in first-order theories and a brief comparison with *theory* resolution, the power and generality of T-resolution are emphasized by introducing suitable linear and ordered refinements, uniformly and in strict analogy with the standard resolution approach. Then a Model Elimination variant of T-resolution is introduced and proved to be sound and complete; some experimental results are also reported.

In the last part of the paper we present two applications of T-resolution: to Constraint Logic Programming and to Modal Logic.

Key words: Automated Deduction, Theory Reasoning, Model Elimination, Resolution, Decidability.

Introduction

In order to perform automated deduction in the context of a given finitely axiomatizable first-order theory T, the deduction theorem guarantees that one can use a first-order theorem prover applied to sentences that are implications whose antecedent is the conjunction of the axioms of T. This technique has two obvious disadvantages:

1. in general, very large sentences must be manipulated to prove even simple theorems: the conjunction of the axioms of T is always part of the sentence to be proved;
2. no specific knowledge relative to T is used in the process.

In order to overcome these disadvantages, the *theory resolution* rule was proposed in [Sti85]. This rule permits one to eliminate the axioms of T from the theorem to be tested, and exploits a T-decider at each step of the inference process. Stickel's proposal has been followed by many studies seeking to cut down the search space generated by the use of theory resolution.

More recently another rule, T-resolution, was proposed in [PS95] and theoretical results were presented that make it possible to express in a natural manner, for a specified underlying theory T, conditions under which it can be mechanized.

In this paper we discuss theorem proving in the general context of a first-order theory, and, in particular, we consider linear and ordered refinements of T-resolution as well as its Model Elimination variant.

T-resolution is a binary rule that can be seen —at least at the ground level— as a variant of Stickel's theory resolution. The rule is introduced and briefly discussed in the first part of the paper and the fact that it is a *binary* rule whose behavior is very similar to that of (standard) resolution, will be extensively used in what follows. All of the proposed refinements allow us to avoid the explicit introduction of the axioms of T in proving a theorem of T. Moreover, they make systematic use of specific knowledge on T that takes the form of decision procedures for sub-languages of T.

Our main result involves a technique that corresponds to Model Elimination in the context of T-resolution, but before introducing it, we consider the standard refinement strategies: linearity, ordering, and set-of-support.

T-resolution is a powerful rule and this fact makes the problem of defining suitable (T-)analogues of the refinements considered, non-trivial. As we will see, for example, under a straightforward definition of linear T-resolution, *any* deduction can be viewed as being linear. In addition, we will show that an analogous situation arises for Stickel's rule, where trivial linearizations are also allowed. For each of the refinements considered, a suitable version for the context in which we are interested (namely when a background first-order theory is present) is discussed and defined. Then correctness and completeness results for the ground case are obtained.

As far as the proofs of completeness are concerned, the basic technique employed is the same in all the cases, but the complexity increases as we proceed. Model Elimination, the final case considered, is the most intricate; all the work preceding its treatment can be seen as a preparation for it.

As we said before, variants and refinements of Stickel's theory resolution have been studied in depth by many researchers. For example, [Bau92a] considers linear theory-resolution dealing with the case of Model Elimination, while in [Bau92b] ordered theory-resolution is introduced and shown to be complete. Another approach to theory reasoning, based on the connection method, is given in [Pet90] and [Pet93]. [BFP92] offers a unified view of these approaches.

The main (obvious) difference between these approaches and ours is the fact that we are working with a different rule. The fact that we begin with a binary rule allows us to obtain a closer similarity between our results and the corresponding results for standard resolution [CL73, Lov78]. A less obvious difference is the fact that the nature of the T-resolution rule allows a complete separation between the background level (the T-decider) of a T-theorem prover and its foreground reasoner, giving rise to further investigations of refinements, strategies, and heuristics expressly designed for T-resolution such as validity-freeness (cf. Definition 2.1).

The paper is organized as follows: in the first section we briefly introduce the basic definitions and a general discussion concerning theorem proving within the context of a first-order theory T. The rule of T-resolution is then introduced and correctness and completeness results are stated.

In the second section we consider the problem of defining a suitable version of linear T-resolution. Next, (T-)generalizations of s-linear resolution ([Lov78]) as well as ordered resolution are introduced and shown to be complete. The Model Elimination variant of T-resolution is introduced and proved to be complete in the third section, where some experimental results are also presented.

Finally two applications of T-resolution are briefly described. In the first place, T-resolution is seen as providing a general and powerful deduction scheme framework generalizing the approach of Constraint Logic Programming. Then, the basic structure of a modal theorem prover based on T-resolution is outlined.

1 T-theorem Proving: a brief introduction

1.1 Basics

In this section we briefly discuss our viewpoint on theorem proving in first-order theories. What we consider most important is to establish criteria for classifying first-order theories with respect to their suitability for automated theorem proving. Missing proofs and details can be found in [PS95] and [PT92].

Let T be a universal theory, that is a (consistent) recursively enumerable set of sentences in purely universal form, closed under logical entailment[1]. Clearly, any such theory is trivially axiomatized by the entire set of its sentences. Those theories that are axiomatizable by a proper subset of their theorems form a significant special case. Let $\mathcal{L}(T)$ be the smallest language in which all these sentences can be written. An important feature of our approach is the fact that we do assume neither that T is finitely axiomatizable nor that $\mathcal{L}(T)$ is finite.

By an *extension* of a given language $\mathcal{L}$, we mean a language $\mathcal{L}^\star \supseteq \mathcal{L}$ such that $\mathcal{L}^\star \setminus \mathcal{L}$ contains new symbols. Unless otherwise specified, we always deal with extensions $\mathcal{L}^\star$ obtained adding to $\mathcal{L}$ only new function or constant symbols.

Given a ground formula φ, written in an extension $\mathcal{L}^\star$ of $\mathcal{L}(T)$, we will consider the T-satisfiability problem for φ, i.e., the satisfiability problem of φ with respect to the theory T.

In what follows, unless otherwise specified, letters like $A, B, C, D, \ldots$ will denote atoms, whereas $H, L, N, M, \ldots$ will usually be used for literals. If L is a literal, then L^c denotes its complement. A clause is the universal closure of a disjunction of literals; it is convenient to view a clause as a set of literals. Given a formula φ, the existential (resp. universal) closure of φ is denoted by $\vec{\exists}\varphi$ (resp. $\vec{\forall}\varphi$).

Definition 1.1 *A theory T is said to be* GROUND-DECIDABLE *if, given an unquantified formula φ of $\mathcal{L}(T)$, it is decidable whether or not there exists a model of T in which $\vec{\exists}\varphi$ is satisfied.*

Remark. Note that ground-decidability for conjunctions of literals is a sufficient condition for the ground-decidability of the entire theory.

As we will see, ground-decidability represents a minimal (albeit sufficient) requirement on T to carry out some form of theorem proving with T as underlying theory. We will always deal with ground-decidable theories. Moreover, without loss of generality, we will assume that $\mathcal{L}(T)$ contains at least one constant symbol.

Definition 1.2 *Let T be a theory and φ be a formula of an extension $\mathcal{L}^\star$ of the language of T. A* T-MODEL *is a model of the theory T; φ is* T-SATISFIABLE *if there exists an interpretation of $\mathcal{L}^\star$ which is a T-model and a model for $\vec{\exists}\varphi$, otherwise φ is* T-UNSATISFIABLE. *Two formulas φ and ψ are* T-EQUISATISFIABLE *if we have: φ is T-satisfiable if and only if ψ is T-satisfiable. φ is said to be* T-VALID *if it is true in every T-model.*

The following result is a simple consequence of the Compactness Theorem and the Herbrand Theorem [Lov78]:

Theorem 1.1 (Herbrand Theorem for T) *Given a universal theory T and a quantifier free formula φ of $\mathcal{L}^\star$, the universal closure $\vec{\forall}\varphi$ of φ is T-unsatisfiable if and only if for some integer k there exist k ground instances $\varphi\theta_1, \ldots, \varphi\theta_k$ of φ, in the language $\mathcal{L}^\star$, such that $\varphi\theta_1 \wedge \cdots \wedge \varphi\theta_k$ is T-unsatisfiable.*

Proof. ($\Rightarrow$) Let T be expressed as a set of clauses[2] and let S be the set of clauses obtained from the conjunctive normal form of $\vec{\forall}\varphi$. $\vec{\forall}\varphi$ is T-satisfiable if and only if $T \wedge S$ is satisfiable. Thus by the Herbrand Theorem there exists a finite set T' of ground T-valid clauses and a finite set $S' = \{\alpha'_1, \ldots, \alpha'_k\}$ of ground instances of clauses of S such that $T' \wedge S'$ is unsatisfiable. Let (for all $i = 1, \ldots, k$) S'_i be a set of ground clauses such that $\alpha'_i \in S'_i = S\theta_i$ where θ_i is a ground substitution. We have that $T' \wedge S'_1 \wedge \cdots \wedge S'_k$ is unsatisfiable as well as

$$T' \wedge \{\varphi\theta_1 \wedge \cdots \wedge \varphi\theta_k\}. \tag{1}$$

It follows that $\varphi\theta_1 \wedge \cdots \wedge \varphi\theta_k$ has no T-model, since otherwise (1) would be satisfiable.

($\Leftarrow$) Let $\varphi\theta_1 \wedge \cdots \wedge \varphi\theta_k$ be T-unsatisfiable. Suppose that $\vec{\forall}\varphi$ is T-satisfiable. Then there exists a T-model M of $\vec{\forall}\varphi$. For each $i = 1, \ldots, k$, M is a T-model of $\varphi\theta_i$; this yields a contradiction. Hence $\vec{\forall}\varphi$ must be T-unsatisfiable.

[1] In this paper we treat only the case of universal theories.

[2] This is no restriction since T is a set of purely universal sentences, hence, for any of them, we can consider the corresponding clausal form.

□

The previous theorem suggests a general instantiate/check (i/c from now on) method for testing a given sentence $\vec{\forall}\varphi$ for T-satisfiability. Clearly, such a method would be computationally unsuitable for properly facing the T-satisfiability problem for the non-ground case; nevertheless it offers a good theoretical starting point. The instantiation phase is easily designed provided we have an enumeration of all terms in the Herbrand universe $\mathcal{U}_{\mathcal{L}^\star}$. The subsequent check-phase turns out to depend critically on the underlying theory T.

Supposing that T is ground-decidable, the Substitution Lemma below ensures that there exists a suitable algorithm for performing the check-phase, independently of the particular extension $\mathcal{L}^\star$ of $\mathcal{L}(T)$ used.

Following [PS95] we introduce a *substitution function* S_φ (written simply S if the given formula φ is clear from the context) which maps (substitutes) $\mathcal{L}^\star$-terms of φ into $\mathcal{L}(T)$-terms. This mapping acts like a *projection* of $\mathcal{L}^\star$-terms onto (generally open) $\mathcal{L}(T)$-terms: intuitively speaking, the syntactical structure of an $\mathcal{L}^\star$-term is recursively scanned substituting subterms with leading functor not in $\mathcal{L}(T)$ with previously unused variables of $\mathcal{L}(T)$. More formally:

$$S_\varphi(t) \quad = \quad \begin{cases} f(S_\varphi(t_1), \ldots, S_\varphi(t_n)) & \text{if } t \text{ is } f(t_1, \ldots, t_n) \text{ and } f \in \mathcal{L}(T) \\ t & \text{if } t \text{ is a constant } c \in \mathcal{L}(T) \text{ or a variable} \\ x_t & \text{otherwise, } (x_t \text{ being a newly introduced variable}). \end{cases}$$

The subscript in x_t above means that the choice of such variable is not completely independent of the term t; i.e., the mapping S_φ substitutes equal terms with the same variable (see [PS95] for technical details).

Given an unquantified formula (term) X, let $S_\varphi(X)$ be the formula (term) obtained from X by replacing each term t occurring in X by $S_\varphi(t)$.

As mentioned, the following result links the T-satisfiability problem for unquantified formulas of $\mathcal{L}^\star$ to the corresponding problem in $\mathcal{L}(T)$. Observe that Lemma 1.1 emphasizes the case of (ground-decidable) theories which include the equality.

Lemma 1.1 (Substitution Lemma) *Given an unquantified formula φ,*

1. *if the theory T does not contain the equality symbol, then φ and $S_\varphi(\varphi)$ are T-equisatisfiable;*
2. *if T is a theory including equality, then let ξ be the conjunction of the clauses*

$$S_\varphi(t_1) = S_\varphi(s_1) \wedge \cdots \wedge S_\varphi(t_k) = S_\varphi(s_k) \rightarrow S_\varphi(f(t_1, \ldots, t_k)) = S_\varphi(f(s_1, \ldots, s_k)),$$

where one of these clauses is formed for each pair $f(t_1, \ldots, t_k)$, $f(s_1, \ldots, s_k)$ of terms occurring in φ, whose leading functor f is not a symbol of the language $\mathcal{L}(T)$.

Then the formulas φ and $S_\varphi(\varphi) \wedge \xi$ are T-equisatisfiable.

As a consequence:

Lemma 1.2 *Given any ground-decidable theory T, we can determine algorithmically whether or not a given ground formula, written in an extension $\mathcal{L}^\star$ of $\mathcal{L}(T)$, is T-satisfiable.*

Lemma 1.2 guarantees the existence of a checking algorithm for our i/c method.

It is possible to insert, between the instantiate and the check phases, a transformation step yielding the conjunctive normal form of the ground formula to be checked.

The reader can easily check that, for example, the method presented in [HU79] can be straightforwardly adapted to our context. This transformation has polynomial time complexity but, in general, involves an extension of the language of φ (and hence of T) that may introduce new predicate symbols. To deal with such kind of extensions, [PT92] proposed a generalization of the Substitution Lemma able to treat new uninterpreted predicate symbols. On this basis, it is easy to prove the following generalization of Lemma 1.2:

Corollary 1.1 *If T is ground-decidable, then it is algorithmically decidable if any ground formula written in any extension of $\mathcal{L}(T)$ is or is not T-satisfiable.*

An alternative approach works by converting uninterpreted predicate symbols to new function symbols. For instance, assuming that the domain of discourse contains at least two elements, we can extend the language with two new constants —say $\mathbf{t}$ and $\mathbf{f}$— and a new function symbol f_P for each uninterpreted predicate symbol P. Then, the generic literal $P(t_1, \ldots, t_k)$ can be replaced by the literal $f_P(t_1, \ldots, t_k) = \mathbf{t}$. This approach allow us to employ directly Lemma 1.2, but it forces us to consider equality predicate as part of the theory; this is not the case for the approach presented in [PT92].

Given a quantifier-free formula φ, our i/c procedure could work by repeating the following steps:

- generate the ground instance φ';
- obtain the CNF-form of φ': $cnf(\varphi')$;
- check $cnf(\varphi')$ for T-satisfiability.

Methods for performing the final step of this loop will be treated in detail in the following sections, where we will see how the results just stated can be employed in theorem proving (mainly at the ground level) in ground-decidable first-order theories. This will be done in a very natural manner, showing how most of the techniques developed for the standard case (e.g., Davis-Putnam procedure, resolution procedure, refinements, etc.) can be properly "T-generalized".

1.2 Generalized Davis-Putnam procedure

At this point a generalization of the Davis-Putnam procedure suitable for treating the ground case of first-order theories can be proposed. The aim is to test the T-satisfiability of a finite set S of ground clauses written in an extension $\mathcal{L}^\star$ of $\mathcal{L}(T)$. Supposing, as usual, T to be ground-decidable, the above corollary guarantees that the procedure defined below can be completed to obtain an algorithm for testing T-satisfiability of conjunctions of ground clauses.

Let S be a set of ground clauses and $conj$ be a conjunction of literals, the Boolean function $T\text{-}Sat(\cdot)$ tests for T-satisfiability a conjunction of literals (see below).

procedure $T\text{-}DPP(S, conj)$;

if $T\text{-}Sat(conj)$ then
if $S = \emptyset$ then return *satisfiable*
elseif $\emptyset \in S$ then return *unsatisfiable*
else let L be an arbitrary literal occurring in S
$S_1 := \{\alpha \setminus L^c \mid \alpha \in S \text{ and } L \notin \alpha\}$
$conj_1 := conj \wedge L$; (L is set to TRUE)
$S_2 := \{\alpha \setminus L \mid \alpha \in S \text{ and } L^c \notin \alpha\}$
$conj_2 := conj \wedge L^c$; ($L$ is set to FALSE)
if $T\text{-}DPP(S_1, conj_1) = unsatisfiable$ and $T\text{-}DPP(S_2, conj_2) = unsatisfiable$
then return *unsatisfiable*
else return *satisfiable*
endif;
endif;
else return *unsatisfiable*
endif;

end T-DPP;

The procedure should be initially called with the starting set S and an empty conjunction as parameters.

T-DPP is obtained from the classical Davis-Putnam procedure by adding an initial call to T-$Sat(\cdot)$ and eliminating the single-literal and the pure-literal rules (as in Propositional Logic these rules can be viewed as instances of the splitting-rule). While the former rule could be actually added, the latter can be incorrect in our context as the following example shows:

Example. Let $S = \{\{A\}, \{B\}\}$ and $T = \{\{\neg A, \neg B\}\}$. S is T-unsatisfiable, but the set $S' = \{\{B\}\}$ obtained from S by applying the pure-literal rule, is T-satisfiable.

Assuming the polynomial time complexity of T-$Sat(\cdot)$, the average case complexity of T-DPP is shown to be polynomial (see [PT92]).

Because of the generality of the extension $\mathcal{L}^\star$, the conjunction *conj* (which is a CNF formula) tested for T-satisfiability, may contain conjuncts not belonging to $\mathcal{L}(T)$. In this case, Corollary 1.1 ensures that ground-decidability of T is a sufficient requirement to guarantee the existence of a suitable T-$Sat(\cdot)$.

1.3 Resolution T-theorem proving

In this section we briefly review T-resolution, a generalization of standard resolution which turns out to be a refutationally sound and complete method, suitable for treating the T-satisfiability problem. The following is the ground T-resolution rule as defined in ([PS95]):

Definition 1.3 (Ground T-resolution) *Let $\alpha = \alpha_1 \cup \alpha_2$ and $\beta = \beta_1 \cup \beta_2$ be two ground clauses such that:*

$$T \models \neg(\bigvee_{l \in \alpha_1 \cup \beta_1} l) \rightarrow \neg(\bigvee_{p \in \alpha_2} p) \vee \neg(\bigvee_{q \in \beta_2} q) \tag{2}$$

*then the clause $\alpha_1 \cup \beta_1$ is said to be a T-*RESOLVENT *of α and β (written $\alpha_1 \cup \beta_1 \in Res_T(\alpha, \beta)$), and α and β are called* PARENT CLAUSES *of $\alpha_1 \cup \beta_1$.*

The above expression could be written in a compact, perhaps more readable, form, identifying clauses with disjunctions of literals:

$$T \models \alpha_2 \wedge \beta_2 \rightarrow \alpha_1 \vee \beta_1 \tag{3}$$

Remark. If α and β contain a pair of complementary literals, L and L^c respectively, then $(\alpha \setminus \{L\}) \cup (\beta \setminus \{L^c\})$ is a legal T-resolvent of α and β, since in any theory T it is always the case that $T \models L \vee L^c$. Thus T-resolution properly generalizes standard resolution.

Remark. We note that Definition 1.3 includes the case where $\alpha_1 = \alpha$ and $\beta_1 = \{L\}$. This T-resolution step, from now on referred to as *loading*, produces a subsumed clause. We will show that loading is actually needed to ensure completeness of T-resolution.

Note that our assumption of ground-decidability of T guarantees that condition (2) in Definition 1.3 is always decidable, by Corollary 1.1.

Definition 1.4 *Given a set S of ground clauses, a (ground) T-*DERIVATION *$\mathcal{D}$ of α from S is a finite sequence of clauses $\beta_1, \ldots, \beta_{r+n}$ such that:*

- *$\beta_{r+n} = \alpha$;*
- *$\beta_1, \ldots, \beta_r$ are clauses in S;*
- *for all i, $r+1 \leq i \leq r+n$, β_i is a T-resolvent of β_j and β_k for some $j, k < i$;*

$\beta_1, \ldots, \beta_r$ constitute the PREFIX *of the T-derivation $\mathcal{D}$ (denoted as prefix($\mathcal{D}$)). A T-*REFUTATION *of S is a T-derivation of a T-unsatisfiable clause from S.*

The prefix of a derivation, as introduced in [Lov78], allows us to gather all needed information (read *knowledge*) regarding the given set of clauses, in a restricted zone of the deduction, simplifying the subsequent exposition, and guarantees a uniform treatment.

Theorem 1.2 (Soundness of ground T-resolution) *If a T-unsatisfiable clause is T-derivable from a given a set S of ground clauses, then S is T-unsatisfiable.*

Proof. Let $\alpha = \alpha_1 \cup \alpha_2$ and $\beta = \beta_1 \cup \beta_2$ be two clauses in S and let $\gamma = \alpha_1 \cup \beta_1$ be a T-resolvent of α and β. From

$$T \models \alpha_2 \wedge \beta_2 \rightarrow \alpha_1 \vee \beta_1$$

it follows that if M is a T-model of S (and therefore of α and β), then it is also a T-model for $\gamma = \alpha_1 \cup \beta_1$. □

Given a set S of ground clauses, the set

$$S_e = S \cup \{\{L, L^c\} \mid L \text{ is a literal occurring in } S\}$$

is called the *tautological extension* of S. Evidently, S is T-unsatisfiable if and only if S_e is T-unsatisfiable.

The next theorem states the completeness of T-resolution [PS95]:

Theorem 1.3 *Given a set S of ground clauses, if S is T-unsatisfiable then there exists a T-refutation of S_e.*

The example below shows that it is necessary to introduce the tautological extension of the given set of clauses, and that loading is actually needed to ensure completeness of T-resolution.

The point is that loading can be exploited only on literals actually appearing in the given set of clauses, rather than on arbitrary literals of the theory. This restriction may seem unnatural but, as we shall see, it enables an advantageous separation between the theorem provers based on the T-resolution rule and the specific theories/T-deciders employed.

Example. Let $T = \{\{\neg A_1, \neg A_2, \neg A_3\}\}$ and $S = \{\{A_1\}, \{A_2\}, \{A_3\}\}$. S is T-unsatisfiable but, following Definition 1.3, no useful clause is derivable from it by T-resolution, since we cannot use clauses of T. In fact, the only new clauses obtainable from $\{A_1\}$ and $\{A_2\}$ would be $\{A_1, A_2\}$ and $\square$ but the former is obtained by loading whereas the latter is not a T-resolvent of $\{A_1\}$ and $\{A_2\}$ since $T \not\models A_1 \wedge A_2 \rightarrow \square$.

Nevertheless the following is a T-refutation of S_e. Notice that any T-refutation of S_e must employ loading:

1)	$\{A_1, \neg A_1\}$	
2)	$\{A_2, \neg A_2\}$	
3)	$\{A_1\}$	
4)	$\{A_2\}$	
5)	$\{A_3\}$	
6)	$\{A_3, \neg A_2\}$	loading of $\neg A_2$ (from 2) in 5
7)	$\{A_3, \neg A_2, \neg A_1\}$	loading of $\neg A_1$ (from 1) in 6
8)	$\{\neg A_2, \neg A_1\}$	T-resolvent of 7 and any other clause
9)	$\{\neg A_1\}$	resolvent of 8 and 4
10)	$\square$	resolvent of 9 and 3

where deriving clause at line 8 is justified as follows:
From

$$T \models \neg A_1 \vee \neg A_2 \vee \neg A_3$$

Choosing an arbitrary clause α, we have:

$$T \models \alpha \wedge A_3 \rightarrow \neg A_1 \vee \neg A_2$$

hence $\{\neg A_2, \neg A_1\} \in Res_T(\{A_3, \neg A_2, \neg A_1\}, \alpha)$.

To avoid dealing with tautological extensions of sets of clauses we reformulate the definition of T-resolution rule, giving more emphasis to the necessity of loading.

Definition 1.5 *Let $\alpha = \alpha_1 \cup \alpha_2$ and $\beta = \beta_1 \cup \beta_2$ be ground clauses. γ is a T-*RESOLVENT *of the two* PARENT CLAUSES *α and β (written $\gamma \in Res_T(\alpha, \beta)$) if and only if*

- *$\gamma = \alpha_1 \cup \beta_1$ and $T \models \alpha_2 \wedge \beta_2 \rightarrow \alpha_1 \vee \beta_1$; or*
- *$\gamma = \alpha \cup \{L\}$ where either $L \in \beta$ or $L^c \in \beta$ (loading).*

Remark. Instead of defining loading as a binary rule it could have been defined as a unary rule that derives $\alpha \cup \{L\}$ from α and any literal L such that L (or L^c) occurs in a given set of clauses.

Remark. Several restrictions could be applied to loading, for example we can avoid considering T-resolvents of two parent clauses both obtained by loading steps. Other such restrictions and some open problems relative to this issue are discussed in Section 2.4.

From now on we adopt Definition 1.5 as the actual definition of ground T-resolvent. We can then easily reformulate the definitions of T-derivation and T-refutation, as well as our soundness and completeness results in a simpler manner:

Theorem 1.4 *A given set S of ground clauses is T-unsatisfiable if and only if there is a T-refutation of S.*

Proof. The soundness of the rule is essentially proved as was done in Theorem 1.2. The completeness follows from the completeness of a linear refinement of the T-resolution rule given by Corollary 2.1 in Section 2.2.2. □

Although our exposition is focused on the ground case, we will give just a brief description of the treatment of the general case. Definition 1.6 extends T-resolution to the non-ground case, following the approach outlined above (see also [PS95] where T-resolution was first introduced).

Definition 1.6 (Non-ground T-resolution) *Given two clauses $\alpha = \alpha_1 \cup \alpha_2$ and $\beta = \beta_1 \cup \beta_2$, γ is said to be a T-*RESOLVENT *of α and β (written $\gamma \in Res_T(\alpha, \beta)$) if there exists a substitution θ such that one of the following conditions holds:*

- *γ has the form $(\alpha_1 \cup \beta_1)\theta$ and $T \models \vec{\forall}(\alpha_2 \wedge \beta_2 \rightarrow \alpha_1 \vee \beta_1)\theta$; or*
- *$\gamma = (\alpha \cup \{L\})$ where either $L \in \beta$ or $L^c \in \beta$.*

It is easily shown that non-ground T-resolution properly generalizes both standard non-ground resolution and factoring.

Remark. Notice that in Definition 1.6 above, there is no restriction on the choice of the substitution θ. Considering the mechanizability issue for non-ground T-resolution, it will be necessary to restrict such choice to a suitable set of substitutions, depending on the properties of T. In [PS95] an approach involving the concept of $\mathcal{L}^\star$-covering (strictly related to the notion of complete set of T-refuters, as introduced, for example, in [BFP92]) is proposed.

Definition 1.7 *Given a set S of clauses, a (non-ground) T-*DERIVATION *$\mathcal{D}$ of α from S is a finite sequence of clauses $\beta_1, \ldots, \beta_{r+n}$ such that:*

- *$\beta_{r+n} = \alpha$;*

- $\beta_1, \ldots, \beta_r$ *are clauses in* S*;*

- *for all* i, $r+1 \leq i \leq r+n$, β_i *is a (non-ground)* T*-resolvent of* β_j *and* β_k *with* $j, k < i$.

As before, $\beta_1, \ldots, \beta_r$ *constitute the* PREFIX *(prefix*$(\mathcal{D})$*) of* $\mathcal{D}$*. A* T-REFUTATION *of* S *is a* T*-derivation of a* T*-unsatisfiable clause from* S.

Soundness in the general case follows from:

Theorem 1.5 (Soundness of non-ground T-resolution) *If a clause* γ *whose universal closure* $\vec{\forall}\gamma$ *is* T*-unsatisfiable is derivable by* T*-resolution from* S*, then* S *is* T*-unsatisfiable.*

As happens with standard resolution, a completeness result is obtained from ground completeness using a lifting technique. We need a generalization of the standard Lifting Lemma; the following holds:

Lemma 1.3 (Lifting Lemma for T) *Let* α' *and* β' *be ground instances of* α *and* β *respectively, and let* γ' *be a ground clause derivable by* T*-resolution from* α' *and* β'*. Then* γ' *is derivable by* T*-resolution from* α *and* β.

On this basis, using the Herbrand Theorem for T, we can prove the following:

Theorem 1.6 (Completeness of non-ground T-resolution) *If* S *is a* T*-unsatisfiable set of clauses, then a clause admitting a* T*-unsatisfiable ground instance is derivable from* S *by* T*-resolution.*

The treatment outlined above of the non-ground case is very natural but somehow unsatisfactory because the condition to automatize non-ground T-resolution places very strong conditions on T. Moreover, the above mentioned condition allowing the mechanizability of non-ground T-resolution, is the existence of an algorithm to determine the $\mathcal{L}^\star$-covering for a given $\mathcal{L}^\star$-formula φ. In fact, such a condition may depend on the particular extended language $\mathcal{L}^\star$ and hence cannot be considered as a (uniform) condition on the underlying theory T.

1.4 T-resolution and Theory Resolution

We will briefly recall Stickel's theory resolution rule ([Sti85]) and outline a comparison of the two approaches.

Definition 1.8 *Let* $\alpha_1, \ldots, \alpha_m$ $(m > 1)$ *be* m *non-empty clauses such that, for each* i, $\alpha_i = \alpha_{i,1} \cup \alpha_{i,2}$ *with* $\alpha_{i,2} \neq \emptyset$*. Let* $\{M_1\}, \ldots, \{M_n\}$ *be* $n \geq 0$ *unit clauses such that* $\{\alpha_{1,2}, \ldots, \alpha_{m,2}, \{M_1\}, \ldots, \{M_n\}\}$ *is a* T*-unsatisfiable set. Then the clause* $\alpha_{1,1} \cup \cdots \cup \alpha_{m,1} \cup \{M_1^c, \ldots, M_n^c\}$ *is called a* THEORY RESOLVENT *of* $\alpha_1, \ldots, \alpha_m$.

It is a TOTAL THEORY RESOLVENT *if and only if* $n = 0$ *and a* PARTIAL THEORY RESOLVENT *otherwise. The set* $\{\alpha_{1,2}, \ldots, \alpha_{m,2}\}$ *is the key-set of the theory resolution operation. The disjunction* $M_1^c \vee \cdots \vee M_n^c$ *is the* RESIDUE *of the theory resolution operation. The theory resolvent is a* NARROW THEORY RESOLVENT *if each* $\alpha_{i,2}$ *is a unit, or a* WIDE THEORY RESOLVENT *otherwise.*

At the ground level, the fact that Stickel's rule is n-ary can be seen as an advantage, being more general than T-resolution, but it has an undesired side-effect: it forces dealing with minimality, requiring stronger properties of the decision procedure. Moreover, the generality of the rule has to be controlled introducing suitable tools, like criteria for key selections ([Sti85]). In our approach these phenomena can be considered to have a counterpart in the loading operation, and restrictions directly controlling loading are discussed in Section 2.4.

As indicated, the main difference between our approach and Stickel's, is in the background decidability property assumed. Ground-decidability permits the realization of an inference system where the deduction activity is focused on the given theorem to be proved, never explicitly dealing with entities extraneous to it. With theory resolution this may not be the case, because the background reasoner may have to explicitly introduce language entities not strictly related to the specific theorem being tested. This happens essentially because the theory decider employed in theory resolution plays an "active" role in the deduction process:

it provides suitable conditions (in the form of a residue to be added to the resolvent) enabling the theory resolution step. What the decider knows about the problem to be solved is just the supplied *key* of the theory resolution operation; hence it does not know which part of the theory is strictly related to that problem.

In T-resolution there is no mechanism producing the entire residue in a single step; the residue has to be constructed using loading operations (which are always restricted to literals taken from the given set of clauses).

Since in theorem provers based on T-resolution, the task of controlling loading operations (i.e., residue generation) is performed at the foreground level, the theory reasoner can be designed independently of the particular inferences, search strategies or heuristics employed by the theorem prover, only considering the characteristics of the theory. The following example shows how, even at the ground level, the introduction of symbols extraneous to the given set of input clauses currently processed could generate undesired effects.

Example. Suppose we are looking for linear refutation by binary theory resolution of the set

$$S = \{\{q(a)\}, \{q(b)\}, \{q(c)\}\}$$

with respect to the theory:

$$T = \{\{\neg q(a), \neg q(b), \neg q(c)\}, \{\neg q(a), \neg q(b), \neg p(d)\}, \{p(x), \neg p(f(x)), \neg q(c)\}\}.$$

To resolve $q(a)$ and $q(b)$ there are (at least) two residues that the background T-decider could return to the foreground reasoner: $\neg q(c)$ and $\neg p(d)$ (while with T-resolution only $\neg q(c)$ can be built using loading operations). If the former is used then S is refuted with a further step (using $\{q(c)\} \in S$). Otherwise, using $\neg p(d)$ (which does not occur in S) as residue could lead the prover into an infinite branch of the search space consisting of the clauses: $\{\neg p(d)\}, \{\neg p(f(d))\}, \{\neg p(f(f(d)))\}, \ldots$

The decider used in theory resolution may not be compatible with all derivation strategies; this because of its "active" nature. In other words, it is not the case that for each restriction imposed on the kind of inferences allowed at the foreground level, the decider is able to produce suitable residues. As we will see, T-resolution seems to be better suited to refinements using standard or ad-hoc techniques. This is essentially because the rule is binary and the decision algorithm it employs remains the same in any of the cases considered.

2 Refinements of T-resolution

In this section we generalize to T-resolution the linear and ordered refinements employed with resolution in pure predicate calculus (see, for example, [CL73, Lov78]).

2.1 Basics

Let T be a theory and φ be a ground formula of $\mathcal{L}^\star$. From Definition 1.2 we have that φ is said to be T-VALID (or simply valid) if it is true in every T-model. T-validity generalizes, to the theory case, the concept of tautological formula. The interesting case is when $T \neq \emptyset$; otherwise tautologies are the only T-valid formulas.

Some other simple concepts:

Definition 2.1 *A set of clauses is* TAUTOLOGY-FREE *(resp.* VALIDITY-FREE*) if it does not contain any tautological (resp. T-valid) clause. A derivation $\mathcal{D}$ is* TAUTOLOGY-FREE *(resp.* VALIDITY-FREE*) if any tautological (resp. T-valid) clause occurs only in the prefix.*

Since we will treat only the ground case of T-resolution, we will adopt the following conventions: T, as usual, will be a ground-decidable first-order theory; T' will denote a set of non-tautological ground T-valid clauses such that each predicate symbol occurring in T' is in $\mathcal{L}(T)$; S will be a set of ground clauses in $\mathcal{L}^\star$ and $\mathcal{D}$ (possibly subscripted or primed) will denote a derivation (unless otherwise specified) by T-resolution. From now on the prefix T- (of T-satisfiable, T-derivation, ...) will often be omitted; ambiguities will be

resolvable from the context or explicitly clarified. Given a derivation $\mathcal{D}$ and a clause α, $\mathcal{D} \circ \alpha$ will denote the derivation obtained adding α at the end of $\mathcal{D}$, assuming that $\mathcal{D} \circ \alpha$ is a legal derivation, i.e., that α can be derived from clauses of $\mathcal{D}$.

Two immediate results frequently employed in what follows:

Proposition 2.1 *Let $\alpha = \alpha_1 \cup \alpha_2$ and $\beta = \beta_1 \cup \beta_2$ be two valid ground clauses. If $\gamma = \alpha_1 \cup \beta_1 \in Res_T(\alpha, \beta)$, then γ is valid.*

Proof. Let M be a T-model of α and β ($M \models T \wedge \{\alpha_1 \cup \alpha_2, \beta_1 \cup \beta_2\}$). Suppose M is not a model of $\alpha_1 \cup \beta_1$ (i.e., both α_1 and β_1 are false in M). Since M is a model for T, from the T-resolution rule (Definition 1.3) it follows that at least one of α_2 and β_2 must be false in M. Hence, at least one of α and β must be false in M. This yields a contradiction. Therefore M is a T-model of γ.

□

We will never consider valid clauses containing *extraneous* literals, i.e., literals whose atoms do not occur in a given set of clauses; in fact the following holds:

Proposition 2.2 *Given a T-unsatisfiable set of ground clauses S, there exists a tautology-free set of valid clauses T' such that $S \cup T'$ is unsatisfiable, and every atom of literals occurring in T' also occurs (possibly in a literal with different polarity) in S. Moreover, each predicate symbol occurring in T' is a symbol of $\mathcal{L}(T)$.*

Proof. Without lost of generality, we can assume that S is finite and T is expressed in clausal form. Since S is T-unsatisfiable it follows that $S \cup T$ is an unsatisfiable set of clauses. By the Herbrand Theorem there exists a finite set T' of ground instances of clauses of T such that $S \cup T'$ is unsatisfiable. Obviously, each literal in T' has a predicate symbol in $\mathcal{L}(T)$ (note that this might not be the case for the constant and the function symbols).

Let $A_1, \ldots, A_n$ be all the atoms occurring in S. Let $\mathcal{T}$ be a binary semantic tree such that all the edges of the i-th level are labeled by the literal A_i or $\neg A_i$, for $i = 1, \ldots, n$. Given a node $\mathbf{m}$, let $Context(\mathbf{m})$ be the conjunction of the literals labeling the edges in the path from the root to $\mathbf{m}$.

Since $S \cup T'$ is unsatisfiable, each branch b of $\mathcal{T}$ contains a failure node $\mathbf{m}_b$ belonging to a level h_b of $\mathcal{T}$. Each $\mathbf{m}_b$ is a failure node either because it falsifies a clause of S (i.e., $Context(\mathbf{m}_b)$ contains the complement of each literal occurring in that clause), or because $Context(\mathbf{m}_b)$ is a T-unsatisfiable conjunction of literals (i.e., it falsifies a clause of T'). Let us say $\mathbf{m}_b$ to be a C-failure node in the former case, and a T-failure node in the latter. We can assume that T' is such that each atom occurring in T' also occurs in S. To verify this claim observe that each C-failure node must belong to a level $j \leq n$; therefore, if there are two sibling nodes $\mathbf{m}_1$ and $\mathbf{m}_2$ belonging to a level $i > n$, then such nodes must be T-failure nodes. Let α_1 and α_2 be the two clauses in T' falsified by $Context(\mathbf{m}_1)$ and $Context(\mathbf{m}_2)$ respectively. An ancestor $\mathbf{m}'$ of $\mathbf{m}_1$ can be made a T-failure node by simply adding to T' the standard resolvent of α_1 and α_2 obtained resolving on the literals labeling the edges of level i.

Since no atom occurs twice in any branch of $\mathcal{T}$, no clause in T' is tautological.

□

2.2 Linearity and T-resolution.

In the context of standard resolution, linear derivations have a very natural and readable format, and the restriction of the search space to such derivations produces a significant saving.

Imposing some form of linearity surely guarantees some advantages in the context of T-resolution as well, since, after all, standard resolution is a special case of T-resolution. However, we will show below that simple attempts to extend linear restrictions to T-resolution encounter some pitfalls.

We will first discuss the sort of linearity that it is reasonable to impose on T-resolution; then, from the completeness of SL-T-resolution (a generalization of s-linear resolution), we will obtain the completeness of linear T-resolution as well as the compatibility of T-resolution with the set-of-support strategy.

Definition 1.5 of T-resolution is too general and is not suitable to be employed, for example, together with linear refinements. This is easily seen since, for every pair of clauses α and β, putting $\alpha = \alpha_1$, $\alpha_2 = \beta_1 = \emptyset$

and $\beta = \beta_2$, α turns out to be a legal T-resolvent of α and β. Hence, we can systematically map every T-derivation into a "linear" one in the following manner: suppose $\mathcal{D}$ is a derivation of γ, let α and β be its parent clauses; having two "linear" derivations $\mathcal{D}_1$ and $\mathcal{D}_2$ of α and β respectively, we could combine them so as to obtain a "linear" derivation of γ simply deriving the top clause δ_2 of $\mathcal{D}_2$ from α and δ_2 (see Figure 1, where, for simplicity, prefixes are not displayed).

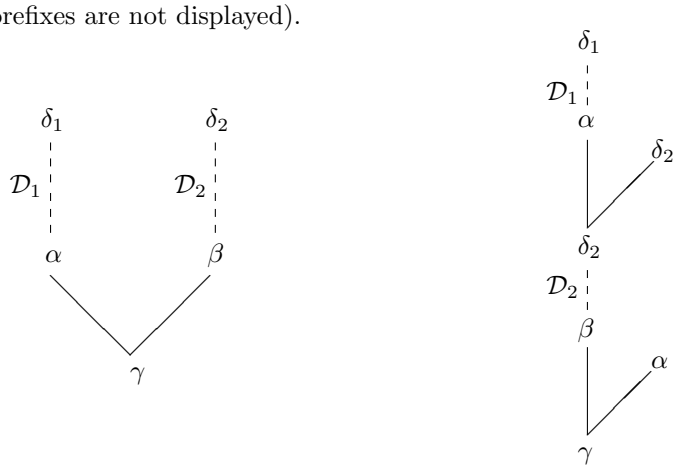

Figure 1: Trivial linearization.

Remark. To see that an analogous situation arises in theory resolution, consider the following two clauses (see Definition 1.8): $\alpha_1 = \alpha_{1,1} \cup \alpha_{1,2}$ and $\alpha_2 = \alpha_{2,1} \cup \alpha_{2,2}$; suppose $n = 1$, $\alpha_{1,1} = \emptyset$ and $\alpha_{2,2} = \{L\}$ (for $L \in \alpha_2 \neq \emptyset$). Choose $M_1 = L^c$, then $\{\alpha_{1,2}, \alpha_{2,2}, \{M_1\}\} \supseteq \{\{L\}, \{L^c\}\}$ is T-unsatisfiable. Hence, we have $\alpha_2 = \alpha_{1,1} \cup \alpha_{2,1} \cup \{M_1^c\}$ as theory resolvent of α_1 and α_2.

In order to avoid these trivially linear derivations, which are not in the spirit of what is the primary purpose of linear refinements (i.e., to always proceed *using* the last derived clause), we need to restrict the kind of inferences allowed.

A very natural way to avoid the above problem seems to be the introduction of the following restriction[3]: for $\alpha = \alpha_1 \cup \alpha_2$ and $\beta = \beta_1 \cup \beta_2$, consider only those $\gamma = \alpha_1 \cup \beta_1$ such that

$$T \models \alpha_2 \wedge \beta_2 \rightarrow \alpha_1 \vee \beta_1 \text{ and } \alpha_2 \neq \emptyset \text{ and } \beta_2 \neq \emptyset$$

Unfortunately, this restriction does not solve the problem since the same undesired effect can be reached in a more subtle way using the loading operation which, on the other hand, cannot be omitted (see the example in Section 1.3). The critical situation is exhibited in the following fragment of a derivation:

δ_1	
$\vdots\ \mathcal{D}_1$	
α	
$\vdots$	loading of the literals of δ_2 in α
$\alpha \cup \delta_2$	
δ_2	because $\delta_2 \in Res_T(\alpha \cup \delta_2, \delta_2)$ even with the above restriction.
$\vdots\ \mathcal{D}_2$	
β	
γ	T-resolvent of α and β

In Definition 2.2, below, an inference rule is proposed that is strong enough to avoid this phenomenon while still ensuring completeness,

[3]Actually, this is the restriction imposed by definition in Stickel's rule.

Definition 2.2 (Ground Strict T-resolvent) *γ is a* STRICT T-RESOLVENT *of α and β if and only if*

R1 *$\gamma \in Res_T(\alpha, \beta)$ and neither α nor β subsumes γ (that is, in the ground case: $\alpha \not\subseteq \gamma$ and $\beta \not\subseteq \gamma$); or*

R2 *$\gamma = \alpha \cup \{L\}$ where either $L \in \beta$ or $L^c \in \beta$.*

Rule **R1** is still a generalization of standard ground resolution. From now on we will consider Definition 2.2 as the actual definition of T-resolvents.

In the trivial linearization just displayed, the linear derivation is constructed connecting $\mathcal{D}_1$ and $\mathcal{D}_2$ with a sequence of loading steps of the literals of δ_2, followed by a single (non-strict) T-resolution step which uses δ_2 as parent clause. The next example clarifies how the new definition avoids situations of this kind.

Example. Let $T = \emptyset$ and $S = \{\{A, B, C\}, \{\neg A, B, D\}, \{\neg B\}\}$. Consider the derivation depicted in Figure 2 where $\alpha = \{A, C\}$, $\beta = \{\neg A, D\}$, $\delta_1 = \{A, B, C\}$, $\delta_2 = \{\neg A, B, D\}$ and $\gamma = \{C, D\}$. If only strict T-resolvents are allowed, it is impossible to linearize $\mathcal{D}'$ in the manner described above (i.e., deriving $\{\neg A, B, D\}$ from $\{A, C\}$, by first loading literals from δ_2 and then performing a T-resolution step with δ_2 as parent clause).

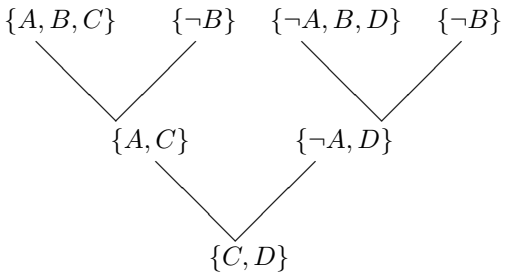

Figure 2: $\mathcal{D}'$.

2.2.1 Linear T-resolution

After introducing the concept of linear T-derivation, we present a result (Lemma 2.1) essential for the following exposition and representing a basic connection point between resolution and T-resolution. In complete analogy with the standard case we have:

Definition 2.3 *A* LINEAR DERIVATION *(by T-resolution) $\mathcal{D}$ of α from S is a finite sequence $\beta_1, \ldots, \beta_{r+n}$ of clauses such that:*

- *$\beta_{r+n} = \alpha$;*
- *$\beta_1, \ldots, \beta_r$ are clauses in S;*
- *for all i, $r + 1 \leq i \leq r + n$, β_i is a strict T-resolvent of β_{i-1} and β_j with $j < i$;*
  the clause β_{i-1} is called the NEAR PARENT *and β_j is called the* FAR PARENT.

β_r is called the TOP CLAUSE *of $\mathcal{D}$. prefix($\mathcal{D}$) is defined as usual. The clauses $\beta_{r+1}, \ldots, \beta_{r+n}$ (*DERIVED CLAUSES*) constitute the* SUFFIX *(suffix($\mathcal{D}$)) of the derivation. A* LINEAR REFUTATION *of S is a linear derivation of a T-unsatisfiable clause.*

The derivation shown in Section 1.3 is a linear T-derivation.

In order to prove the completeness of linear T-resolution, we will transform a (classical) linear derivation from $T' \cup S$ (where T' is the set of ground instances of clauses of T) into a linear derivation from S by T-resolution. The following lemma is relative to the basic step of such a transformation.

Lemma 2.1 *Given a clause $\alpha = \alpha' \cup \{L\}$ in a tautology-free derivation $\mathcal{D}$ from S, and a valid clause $\beta = \beta' \cup \{L^c\}$, such that $\gamma = \alpha' \cup \beta'$ is non-tautological, there exists a tautology-free derivation $\mathcal{D}'$ of γ from S.*

Moreover:

- *if $\mathcal{D}$ is a linear derivation of α, then $\mathcal{D}'$ is a linear derivation of γ;*
- *if γ is not valid, and $\mathcal{D}$ is validity-free, then so is $\mathcal{D}'$.*

Proof. Since $T \models \beta$, we have $T \models \beta' \vee L^c \vee \alpha'$. Choosing an arbitrary clause δ occurring in $\mathcal{D}$, we also have $T \models \beta' \vee L^c \vee \alpha' \vee \neg\delta$ and $T \models L \wedge \delta \rightarrow \alpha' \vee \beta'$, from which it follows that $\gamma = \alpha' \cup \beta'$ is a T-resolvent of $\alpha' \cup \beta' \cup \{L\}$ and δ. The clause $\alpha' \cup \beta' \cup \{L\}$ is derivable from $\alpha = \alpha' \cup \{L\}$ with $\mid (\beta' \setminus \alpha) \mid$ steps, loading every literal in $(\beta' \setminus \alpha)$ (this can be done because, as mentioned in the preceding section, we assume all literals in β to have an atom occurring in S). Since γ does not contain complementary pairs of literals, $\mathcal{D}'$ is tautology-free. Clearly, if $\mathcal{D}$ is a linear derivation of α then $\mathcal{D}'$ is a linear derivation of γ.

Let $\mathcal{D}$ be validity-free and let $\gamma = \alpha' \cup \beta'$ be not valid. Since $T \models \beta$, if $\alpha \cup \beta'$ were valid so should be γ (as the resolvent of two valid clauses, cf. Proposition 2.1), which is a contradiction. Hence, $\alpha \cup \beta'$ is not valid. Since, for all $\beta'' \subseteq \beta'$, $\alpha \cup \beta'' \subseteq \alpha \cup \beta'$, it follows that no clause in the derivation of $\alpha \cup \beta'$ from α is valid. Hence $\mathcal{D}'$ is validity-free.

□

Remark. If β contains only literals having predicate symbol in $\mathcal{L}(T)$ (in general, this does not follow from the validity of β), then in the proof of Lemma 2.1 we always load literals having interpreted predicate symbol (i.e., a symbol in $\mathcal{L}(T)$, cf. Proposition 2.2).

2.2.2 *SL-T*-resolution.

Following [Lov78] we briefly recall the basic notions regarding ground s-linear resolution and generalize them to our case (notice that we are treating ground linear refinements):

Definition 2.4 *A (ground)* s-RESOLVENT *("s" stands for subsumption) of the near parent α and the far parent β, is a resolvent γ of α and β such that $\gamma \subseteq \alpha$. An* s-LINEAR DERIVATION *of a (ground) clause α from the set of (ground) clauses S is a tautology-free derivation of α from S such that each clause not in S has a far parent in S or is an s-resolvent.*

Definition 2.5 *Given two ground clauses, α and β, α T-*SUBSUMES *β if and only if $T \models \alpha \rightarrow \beta$. A T-*S-RESOLVENT *of the near parent α and the far parent β, is a T-resolvent γ of α and β such that γ T-subsumes α. A SL-T-*DERIVATION *of a clause α from S is a linear and tautology-free T-derivation of α from S such that each clause not in S has a far parent in S or is a T-s-resolvent.*

Given two ground clauses α and β, if α subsumes β then α T-subsumes β. Hence, any s-linear derivation is an SL-T-derivation.

Using Lemma 2.1 we can prove:

Lemma 2.2 *Given an s-linear derivation $\mathcal{D}$ of γ from $S \cup T'$ with top clause in S, there exists an SL-T-derivation $\mathcal{D}'$ of γ from S, such that each clause derived in $\mathcal{D}$ occurs in $\mathcal{D}'$.*

Proof. By induction on the length n of *suffix*$(\mathcal{D})$.

Base case. $n = 0$. There is nothing to prove since $\gamma \in S$ is the top clause of $\mathcal{D}$.

Inductive step. Assume the lemma holds for each derivation with suffix shorter that $n > 0$. Let $\mathcal{D} = \mathcal{D}_1 \circ \gamma$, with $\mathcal{D}_1$ s-linear derivation of α, and let $\gamma = \alpha' \cup \beta'$ be the resolvent of $\alpha = \alpha' \cup \{L\}$ and $\beta = \beta' \cup \{L^c\}$.

By the induction hypothesis on $\mathcal{D}_1$ there exists an SL-T-derivation $\mathcal{D}'_1$ of α from S. There are two possible cases:

1. if $\beta \in S \cup$ *suffix*$(\mathcal{D}_1)$ then $\mathcal{D}' = \mathcal{D}'_1 \circ \gamma$ is an SL-T-derivation.

2. Otherwise, $\beta \in T' \setminus \mathit{suffix}(\mathcal{D}_1)$. By hypothesis, α is non-tautological, and therefore, since β is valid, the derivation $\mathcal{D}$ is obtained from $\mathcal{D}'_1$ by Lemma 2.1.

Using Lemma 2.1 is correct because, without loss of generality, we can suppose that in each T-resolution step it introduces, it is always possible to choose an appropriate far parent clause from the given set S (this is because each literal introduced with loading operations has an atom occurring in S; in each loading step the clause containing that atom will be chosen as far parent). Hence, the derivation obtained fulfills the requirements of Definition 2.5.

□

As a consequence, Theorem 2.1 below gives the completeness of SL-T-resolution.

Theorem 2.1 (Completeness of SL-T-resolution) *Let there be given a set S of ground clauses. If S is T-unsatisfiable, then there exists an SL-T-refutation of S.*

Proof. Immediate if S contains a T-unsatisfiable clause.

Otherwise, there exists a finite subset S' of S and a finite set T' of ground valid clauses such that $S' \cup T'$ is minimally unsatisfiable.

From the completeness of s-linear resolution ([CL73, Lov78]), it follows that there exists an s-linear refutation of $S' \cup T'$ for each top clause in $S' \cup T'$, and then for each top clause in S', (which must be non-empty since T' is satisfiable).

By Lemma 2.2 we can conclude the proof.

□

From Theorem 2.1 the completeness of linear T-resolution follows:

Corollary 2.1 (Completeness of linear T-resolution) *Let there be given a set S of ground clauses. If S is T-unsatisfiable, then there exists a linear and tautology-free T-refutation of S.*

Proof. From Theorem 2.1, observing that each SL-T-refutation of S is a linear and tautology-free T-refutation of S.

□

Remark. The refutations whose existence is ensured by Theorem 2.1 and Corollary 2.1 exist for each top clause α in S, provided α belongs to a T-unsatisfiable subset of S.

An immediate consequence of Corollary 2.1 is the completeness of the set-of-support strategy applied to T-resolution, as the following theorem indicates:

Theorem 2.2 (Completeness of T-resolution with Set-of-support) *Let S be a T-unsatisfiable set of ground clauses, and let $R \subseteq S$ be such that $S \setminus R$ is T-satisfiable. Then there exists a T-refutation of S with set-of-support R.*

Proof. There must exist a clause $\alpha \in R$ belonging to a minimally T-unsatisfiable subset of S. By Corollary 2.1 (see also the remark following it) there exists a linear T-refutation of S with top clause $\alpha \in R$. Such a T-refutation has set-of-support $\{\alpha\}$ and hence it has set-of-support R.

□

2.3 Ordered T-resolution

As with linearity, the T-resolution rule can be combined with standard ordering techniques, properly generalizing O_A-resolution in a very natural manner. Moreover, the completeness of the new calculus is proved following the same approach adopted for the linear refinements.

In this section we apply to ground T-resolution a simple technique based on a fixed ordering of predicate symbols (see, for example [KH69], or [Lov78] for a more exhaustive treatment of ordered refinements of standard resolution). A different approach, based on ordering rules which require that the ordering of literals in parent clauses be inherited in resolvents, will be adopted in Section 3.

Let us identify the kind of orderings we are interested in:

Definition 2.6 *Let S be a set of ground clauses. An A-*ORDERING *$\leq_A$ over S is a (partial) ordering over the set of atoms occurring in clauses of S. If L_1 and L_2 are two literals in S and l_1 and l_2 are the corresponding atoms, then we extend the A-ordering to the literals assuming that $L_1 \leq_A L_2$ if and only if $l_1 \leq_A l_2$.*

*An O_A-*clause *is an ordered sequence of distinct literals $L_1, \ldots, L_n$ such that $\forall i, j = 1, \ldots, n,\ i \neq j$, $L_i \leq_A L_j \rightarrow i \leq j$. An A-*ordering rule *O_A is a rule which assign to each clause at least one O_A-clause.*

Given a clause γ, $max(\gamma)$ is the set of maximal literals in γ. If α is an O_A-clause then $cl(\alpha)$ denotes the corresponding clause.

Some preliminary concepts related to the ordered refinement of standard resolution (O_A-resolution):

Definition 2.7 *Given two ground O_A-clauses, α and β, with $L \in \alpha$ and $L^c \in \beta$, their* RESOLVENT *is defined as the conventional resolvent, provided that L and L^c are rightmost literals in α and β, respectively.*

Definition 2.8 *Let S be a set of ground clauses and $\leq_A$ an A-ordering over S. An O_A-*DERIVATION *of α from S, is a sequence of non-tautological O_A-clauses $\beta_1, \ldots, \beta_{r+n}$ such that:*

- *$\beta_{r+n} = \alpha$;*
- *$\beta_1, \ldots, \beta_r$ are O_A-clauses of clauses in S;*
- *for all i, $r+1 \leq i \leq r+n$, β_i is an O_A-clause of the resolvent of β_j and β_k for $j, k < i$.*

*An O_A-*REFUTATION *is an O_A-derivation of the empty clause.*

The completeness of the ordered refinement of resolution can be shown (see for example [KH69]).

Let us introduce our (T-)generalization of O_A-resolution:

Definition 2.9 *Given two O_A-clauses, $\alpha = \alpha_1 \cup \alpha_2$ and $\beta = \beta_1 \cup \beta_2$, their T-*RESOLVENT *$\alpha_1 \cup \beta_1$ is defined as the conventional T-resolvent, provided that either*

- *α (β) is $L_1, \ldots, L_n$ and α_2 (resp. β_2) is $L_s, \ldots, L_n$, for a suitable $1 \leq s \leq n$; or*
- *the step is a loading operation;*

Definition 2.10 *Let S be a set of ground clauses and $\leq_A$ an A-ordering over S. An O_A-T-*DERIVATION *of α from S, is a finite sequence of non-valid O_A-clauses $\beta_1, \ldots, \beta_{r+n}$ such that:*

- *$\beta_{r+n} = \alpha$;*
- *$\beta_1, \ldots, \beta_r$ are O_A-clauses of clauses in S;*
- *for all i, $r+1 \leq i \leq r+n$, β_i is an O_A-clause of a T-resolvent of β_j and β_k for $j, k < i$.*

*The prefix and suffix of the O_A-T-derivation are defined as before. An O_A-T-*REFUTATION *is an O_A-T-derivation of a T-unsatisfiable clause.*

By Definition 2.10, an O_A-T-derivation is always validity-free and hence tautology-free.

Example. Let $T = \{\{\neg P, \neg Q, R\}, \{\neg P, Q, \neg R\}, \{P, R\}\}$ and

$$S = \{\{Q, R\}, \{\neg L, \neg R\}, \{L, P\}, \{L, \neg P \neg Q\}\}.$$

Let us choose the following ordering for atoms: $L \leq_A P \leq_A Q \leq_A R$ (notice that L does not belong to $\mathcal{L}(T)$).

The following is an O_A-T-refutation of S; the prefix lists the O_A-clauses of clauses of S.

1) R, Q
2) $\neg R, \neg L$
3) P, L

4)	$\neg Q, \neg P, L$	
5)	$\neg R, P$	O_A-clause of the resolvent of 2, 3
6)	$\neg R, \neg Q, \neg P$	O_A-clause of the resolvent of 2, 4
7)	$\neg R, \neg Q$	O_A-clause of the resolvent of 5, 6
8)	$\neg R, Q, P$	by loading of Q in 5
9)	$\neg R, Q$	from 8 and an auxiliary O_A-clause because $T \models \neg R \vee Q \vee \neg P$
10)	$\neg R$	O_A-clause of the resolvent of 7, 9
11)	R	from 1 and a auxiliary O_A-clause because $T \models \neg Q \vee R$
12)	$\Box$	O_A-clause of the resolvent of 10, 11

Using Lemma 2.1 we can prove:

Lemma 2.3 *Given an O_A-derivation $\mathcal{D}$ of a non-valid clause γ from $S \cup T'$, there exists an O_A-T-derivation $\mathcal{D}'$ of γ from S, such that every non-valid derived clause in $\mathcal{D}$ also occurs in $\mathcal{D}'$.*

Proof. By induction on the length n of *suffix*$(\mathcal{D})$.

Base case. $n = 0$. There is nothing to prove since, by hypothesis, $cl(\gamma) \in S$.

Inductive step. Assume the lemma holds for all O_A-derivations with suffix shorter than $n > 0$. It is easy to see that there must be a non-valid O_A-clause preceding γ in $\mathcal{D}$. Let δ be the last non-valid O_A-clause in $\mathcal{D}$, different from γ, and let $\mathcal{D}_1$ be the initial part of $\mathcal{D}$ ending with δ.

By the induction hypothesis on $\mathcal{D}_1$ (whose suffix is shorter than n) there exists an O_A-T-derivation $\mathcal{D}'_1$ of δ from S. Let $cl(\gamma)$ be the resolvent of $\alpha = \alpha' \cup \{L\}$ and $\beta = \beta' \cup \{L^c\}$ where α and β are two O_A-clauses in $\mathcal{D}$ having L and L^c as rightmost literal respectively.

Either α or β must be non-valid (because γ is); suppose without loss of generality that α is non-valid. By induction hypothesis α occurs in $\mathcal{D}'_1$. There are two possible cases:

1. β non-valid. Then like α, β also occurs in $\mathcal{D}'_1$. A further O_A-T-resolution step allows one to obtain γ. To finish put $\mathcal{D}' = \mathcal{D}'_1 \circ \gamma$.
2. β is valid. We can finish by applying Lemma 2.1 to $\mathcal{D}'_1$.

It remains to be shown that no application of Lemma 2.1 violates the A-ordering rule. At point 2 above, the literals L and L^c are rightmost in α and β respectively. By the definition of A-ordered resolvent we have that for every literal $L' \in max(\alpha' \cup \beta')$, $L' \not>_A L$ (L^c). It follows that to every clause introduced in the proof of Lemma 2.1 there corresponds a suitable O_A-clause, with L as rightmost literal, which will be used in the final O_A-T-derivation.

□

Theorem 2.3 (Completeness of O_A-T-resolution) *Given a set S of clauses and an A-ordering $\leq_A$, if S is T-unsatisfiable, then there exists an O_A-T-refutation of S.*

Proof. Immediate if S contains a T-unsatisfiable clause.

Otherwise, there exists a finite subset S' of S and a finite set T' of ground valid clauses such that $S' \cup T'$ is minimally unsatisfiable.

From the completeness of O_A-resolution, it follows that there is an O_A-refutation of $S' \cup T'$. By Lemma 2.3 we can obtain the desired result.

□

As with linear refinement, completeness is preserved if loading is restricted to literals having predicate symbol in $\mathcal{L}(T)$ (see the remark after Lemma 2.1).

A further step can be employed in controlling loading operations. If the A-ordering is such that all literals in $\mathcal{L}(T)$ are $>_A$-smaller than those not in $\mathcal{L}(T)$ (remember that clauses in S may be written in a language $\mathcal{L}^\star$ which is an extension of $\mathcal{L}(T)$), then the refutation obtained has the property that loading operations

are never performed on clauses containing literals not in $\mathcal{L}(T)$. Let us consider how the O_A-T-refutation $\mathcal{D}'$ of S' is built using Lemma 2.1, starting from the O_A-refutation $\mathcal{D}$ of $S' \cup T'$. Each application of this lemma introduces in $\mathcal{D}'$ a sequence of loading steps of literals of β' corresponding to an ordered resolution step employing the O_A-clauses $\alpha = \alpha' \cup \{L\}$ and $\beta = \beta' \cup \{L^c\} \in T$, and resolving on L and L^c. If $\leq_A$ has the mentioned property (remember that the A-ordering is the same for both derivations), then, since $L \in max(\alpha)$, α contains only literals in $\mathcal{L}(T)$ (otherwise L could not be maximal in α). It follows that all loading operations needed to reach completeness are those performed on O_A-clauses consisting of literals in $\mathcal{L}(T)$.

Remark. Even if O_A-T-resolution is restricted in a narrow fashion (i.e., resolving out just one literal from each clause), it remains complete and never employs T-valid clauses. The analogous refinement proposed for narrow theory resolution (cf. [Bau92b]) has to call into play even tautological clauses in order to guarantee completeness; this happens even for simple cases, as the following example (taken from ([Bau92b]) illustrates.

Example. Let $S = \{\underline{A} \vee B, \underline{\neg A} \vee \neg B\}$ where the maximal literals are underlined, and suppose T implies the logical equivalence of A and B. S is T-unsatisfiable and the following is a ("narrow") O_A-T-refutation of S.

1)	B, A	
2)	$\neg B, \neg A$	
3)	B	from 1 and an arbitrary O_A-clause since $T \models \neg A \vee B$
4)	$\neg B$	from 2 and an arbitrary O_A-clause since $T \models A \vee \neg B$
5)	$\square$	O_A-clause of the resolvent of 3, 4

2.4 Loading

As mentioned in the previous sections, loading plays a fundamental role in the use of the T-resolution rule. It is needed to ensure completeness (Section 1.3), and it is one of the features that guarantee the complete separation of the background reasoner from the foreground reasoner in our approach to T-theorem proving (Section 1.4). Unfortunately however, it introduces some inefficiency in the derivation process, preventing any full application of subsumption techniques; nevertheless such techniques can actually be applied "locally" (this is effectively done below in our T-generalization (T-ME) of Model Elimination), considering only the sections of T-derivation not involving loading steps.

Some simple restriction strategies on loading that are easily shown to preserve completeness are available. For example:

- loading can be restricted to a pre-processing phase, where a new input set S' is obtained from the given set S of clauses, through loading steps only.

The main disadvantage of this heuristic is that, in general, even starting with a minimally T-unsatisfiable set S, the set S' is not necessarily minimally T-unsatisfiable. Moreover, since $S \subseteq S'$ the number of input clauses may increase.

In the case of non-linear refinements, an equivalent approach which does not directly modify the given set of clauses, is as follows: allow loading only on clauses which either are input clauses or are obtained by loading steps.

Other possibilities, which preserve completeness, have been noted in previous sections, namely:

- we can avoid resolving two clauses each of which results from loading steps;
- we can load only literals whose atoms occur in the given T-unsatisfiable set of clauses (Proposition 2.2);
- from Lemma 2.1 (see also the remark following Lemma 2.1) it follows that, even if the given set of clauses is written in a language $\mathcal{L}^\star$ (extension of $\mathcal{L}(T)$), loading can be restricted to those literals having predicate symbol in $\mathcal{L}(T)$.

Some further heuristics to handle loading which we are currently studying are:

- load each literal at most once;
- load only literals that do not occur in the given set of clauses, but whose complements do (i.e., allow loading of L if and only if L^c occurs in S but L does not);
- perform all loading steps considered to be useful on a single input clause $\alpha \in S$ (namely: obtain α' by loading enough literals on α so that each atom will occur with both polarities in the set $S \cup \{\alpha'\}$).

A more specific heuristic, not directly related to loading, is employed in our implementation of T-ME. There we made intensive use of the (unit) lemmas obtained in searching for the refutation, to simplify the given set of clauses. This technique actually broadens the class of problems solvable by T-resolution without performing loading (see Section 3.3 for technical details).

3 T-resolution and Model Elimination

In this section we illustrate our T-generalization of ground Model Elimination. Beginning with a suitable set of inference rules, we follow the same approach we used with the previous refinements. However, there is a significant increase of complexity.

3.1 The T-ME Calculus

Model Elimination, as proposed in [Lov78], is essentially a calculus ensuring soundness and completeness even dealing with input derivations only (i.e., linear derivations where in each inference step one of the parent clauses comes from the given input set). This feature is realized by storing information about the inference steps in particular literals (the so-called A-literals) in clauses being derived.

In this context the concept of clause is replaced by that of *chain* which is an ordered sequence of literals. The literals can be of two kind: A-literals and B-literals. The A-literals are bracketed (as in $[A]$) to distinguish them from B-literals. Not all literal sequences are legitimate chains: a chain is *acceptable* if and only if no two literals, of any kind, are identical or complementary and the rightmost literal is a B-literal.

In what follows, we borrow from [Lov78] (refer to it for the technical details) the concept of *accepting transformation* $\mathcal{T}$, which can be viewed as a mapping from chains to chains. The basic actions of the accepting transformations on a given chain are: merging left for identical B-literals; deletion of the rightmost literal in the chain, whenever it is an A-literal; and removal of those B-literals occurring to the right of a complementary A-literal (*ground reduction* step). Since we deal with the ground case only, the last kind of action is the mechanism that allows the calculus to simulate the inference steps involving previously derived chains (namely, the removal of a B-literal corresponds to a resolution step involving a previously derived chain).

Given a set of clauses S, let $\mathcal{T}(S)$ be $\{\mathcal{T}(\alpha) \mid \alpha \text{ is a chain of a clause in } S\}$.
For the ground case, a single inference rule is employed:

extension rule. Given two acceptable chains, α_1 (parent chain) and α_2 (input chain), if their rightmost literals are complementary, then this rule yields a chain $\mathcal{T}(\alpha_3)$, where α_3 is formed by making the rightmost literal in α_1 an A-literal, and placing all literals in α_2, except the rightmost, to the right of α_1 according to the specified ordering rule.

In this section, unless otherwise specified, we will use A, B, perhaps with subscripts, to denote A-literals, B-literals, respectively. As above, $L, N, M, \ldots$ will be used for generic literals. Since chains are ordered sequences of literals, given two chains α and β, $\alpha \circ \beta$ is the chain obtained by their concatenation.

Given a chain α, with $B_1, \ldots, B_q$ as B-literals and $[A_1], \ldots, [A_p]$ as A-literals (in the order shown) we write $\mathcal{B}(\alpha)$ and $\mathcal{A}(\alpha)$ for $B_1, \ldots, B_q$ and $A_1, \ldots, A_p$ respectively, and write $\bar{\mathcal{A}}(\alpha)$ for $A_1^c, \ldots, A_p^c$. Although $\mathcal{B}(\alpha)$, $\mathcal{A}(\alpha)$, and $\bar{\mathcal{A}}(\alpha)$ are defined as chains, we will, if need be, regard them as sets or as disjunctions of literals.

The key point in adapting the framing mechanism to the T-case is that we are forced to permit the framing of literals in arbitrary positions in the chain, when justified by the theory. These steps are guaranteed by the loading and unloading rules introduced below. However, in order to keep such "T-framing" under control (and to preserve soundness), the ordering in which these steps take place must be kept track of while the derivation is being constructed. For this reason we define T-chains as pairs whose second component encodes this ordering; thus the framing process is regulated using an ordered list of B-literals that serve as witnesses to the framings.

Definition 3.1 *A chain α is a* B-CHAIN *if $\mathcal{A}(\alpha) = \emptyset$. A* T-CHAIN *$\bar{\alpha}$ is a pair $(\alpha; \tau_\alpha)$ of (possibly empty) chains (referred to as* first *and* second part *of the T-chain, respectively).*

Given $\bar{\alpha} = (\alpha; \tau_\alpha)$, a B-literal in α is said to be *paired* if there exists an identical (*pairing*) B-literal in τ_α. A paired B-literal B may be made into an A-literal $[B]$ (*framing*) which will still be said to be *paired*.

We now introduce the (T)-generalizations of the concepts of *acceptable chain* and *accepting transformation*:

Definition 3.2 *A T-chain $\bar{\alpha}$ is* ACCEPTABLE *if and only if*

- *α is an acceptable chain and each A-literal in α is paired;*

- *τ_α is a B-chain such that for each B-literal B in τ_α, either B or $[B]$ occurs in α.*

An INPUT T-CHAIN *is a T-chain $\bar{\alpha} = (\alpha; \tau_\alpha)$ where α is an input chain and τ_α is empty. A T-chain $(\Box; \Box)$ is called a* REFUTING *or* EMPTY T-*chain.*

In what follows $\mathcal{B}(\bar{\alpha})$ will stand for $\mathcal{B}(\alpha)$.

Definition 3.3 *A* T-TRANSFORMATION *$\mathcal{T}_T$ maps $\bar{\alpha}$ into $\mathcal{T}_T(\bar{\alpha})$ according to the following rules:*

R0 *merge left for identical B-literals in α;*

R1 *if a paired A-literal $[A]$ occurs to the left of a B-literal A^c in α, then remove A^c from α;*

R2 *if an A-literal $[A]$ is the rightmost literal in α then remove it and the corresponding B-literal A in τ_α;*

The resulting T-chain is produced only if it is acceptable.

From now on we assume as given a ME-ordering rule O (see [Lov78]), which determines the input T-chains and the ordering of literals in T-extension steps.

We next introduce the inference rules of our T-ME-calculus. Note that since $\mathcal{T}_T$ is employed in those rules, the resulting T-chain (if any) is always acceptable:

T-extension rule. Let $\bar{\alpha} = (\alpha; \tau_\alpha)$ be a T-chain such that $\alpha = \alpha_1 \circ \alpha_2$ with $\alpha_1 = (L_1, \ldots, L_n)$ and $\alpha_2 = (B_q)$, and let $\bar{\beta} = (\beta; \Box)$ be an input T-chain such that $\beta = \beta_1 \circ \beta_2$ with $\beta_1 = (H_1, \ldots, H_{r-1})$ and $\beta_2 = (H_r, \ldots, H_t)$. Moreover, let us assume that

$$T \models \alpha_2 \wedge \beta_2 \rightarrow \mathcal{B}(\alpha_1) \vee \bar{\mathcal{A}}(\alpha_1) \vee \beta_1 \tag{4}$$

and let $\bar{\gamma} = \mathcal{T}_T(\bar{\gamma'})$ with $\bar{\gamma}' = (L_1, \ldots, L_n, [B_q], H_1, \ldots, H_{r-1}; \tau_\gamma)$ and $\tau_\gamma = \tau_\alpha$ if B_q is paired in $\bar{\alpha}$, or $\tau_\gamma = B_q \circ \tau_\alpha$ otherwise. Then, we say that $\bar{\gamma}$ is DERIVED from $\bar{\alpha}$ and $\bar{\beta}$ provided that neither $\mathcal{B}(\bar{\alpha})$ nor $\mathcal{B}(\bar{\beta})$ subsumes $\mathcal{B}(\bar{\gamma})$.

Compare (4) with

$$T \models \alpha_2 \wedge \beta_2 \rightarrow \alpha_1 \vee \beta_1, \tag{5}$$

which is the general condition to perform a T-resolution step, introduced in Section 1.3. The only significant difference is the presence of $\bar{\mathcal{A}}(\alpha_1)$, namely the disjunction of negations of A-literals which have been produced in previous deduction steps. These A-literals (let us call them $[A'_1], \ldots, [A'_m]$) testify that in building the

derivation yielding $\bar{\alpha}$ we have generated m T-chains $(\bar{\delta}_1, \ldots, \bar{\delta}_m)$, such that, for each $i = 1, \ldots, m$, $A'_i \in \mathcal{B}(\bar{\delta}_i)$ and $\mathcal{B}(\bar{\delta}_i) \setminus \{A'_i\}$ T-subsumes $\mathcal{B}(\bar{\alpha})$. The T-extension step (which is binary) makes use of information obtained from these T-chains. This is a generalization of a simpler situation existing in Model Elimination. In that case, the presence of an A-literal $[A]$ in a chain γ, ensures that there exists a previous derived chain γ' (called A-*ancestor* chain) such that the B-literal A is rightmost in γ' and each B-literal in γ' occurs to the left of $[A]$ in γ. Hence, if a B-literal A^c occurs to the right of $[A]$ in γ, its removal (reduction step) corresponds to a resolution step between γ' and γ, on the literals A and A^c, respectively.

The subsumption requirement on the resulting T-chain, in the above rule, comes from the same argument already exploited for linear T-resolution (cf. Section 2.2).

Let us consider a particular case of the T-resolution rule: when $\alpha = \beta = \alpha_1 \cup \{L\}$, $\alpha_1 = \beta_1$, and $\alpha_2 = \beta_2 = \{L\}$. Condition (5) then becomes $T \models L \rightarrow \alpha_1$. In this case the theory T permits the discarding of L from the clause α, thus obtaining $\alpha_1 \in Res_T(\alpha, \alpha)$ (or more generally $\alpha_1 \in Res_T(\alpha, \delta)$, for each clause δ, so we can see this kind of step as an unary step).

In T-ME a similar kind of step is a particular case of the T-extension rule: when L is rightmost in the first part α of a T-chain $\bar{\alpha}$, $\bar{\beta}$ is arbitrary, $\alpha_2 = \{L\}$, and $\beta_1 = \Box$.

The following is a rule generalizing this idea:

unloading rule. Let $\bar{\alpha} = (\alpha; \tau_\alpha)$ be a T-chain such that $\alpha = \alpha' \circ \alpha''$, $L \in \alpha'$ is a B-literal followed (in α') by non paired B-literals only, and α'' is empty or starts with a paired literal (A-literal or B-literal). If

$$T \models L \rightarrow \mathcal{B}(\alpha' \setminus \{L\}) \vee \bar{\mathcal{A}}(\alpha'), \tag{6}$$

then let $\bar{\alpha}_1$ be obtained from $\bar{\alpha}$ in the following manner:

- if L is paired with the leftmost literal in τ_α and $\alpha'' = \emptyset$, then replace L with $[L]$ in α,
- if L is not paired then remove it from α.

We say that $\bar{\gamma} = \mathcal{T}_T(\bar{\alpha}_1)$, is DERIVED (by unloading L) from $\bar{\alpha}$.

Remark. When L is paired but not with the leftmost literal in τ_α, removing it would yield a non-acceptable T-chain; so, in this case, no T-chain is produced.

To ensure the completeness of the calculus (see Lemma 3.4 below) we need to allow unloading of B-literals even if they are not rightmost, provided that the step is justified by condition (6) using only literals occurring to the left of α''. Using literals in α'' would introduce unsound steps into the derivation. This is intuitively clear recalling the intuitive idea behind the T-extension step outlined above: if we unload literals by means of α'', then we cannot ensure the existence of the mentioned T-chains $\bar{\delta}_1, \ldots, \bar{\delta}_m$ in the previous part of the derivation (cf. Lemma 3.1).

From another point of view, the unloading rule is a T-generalization of the Model Elimination reduction rule: following the philosophy of the T-resolution approach, syntactical identity is replaced by logical consequence in T.

The following rule introduces loading in our calculus:

loading rule. Let $\bar{\alpha} = (\alpha; \tau_\alpha)$ with B the rightmost literal in α, and let B' be a B-literal. The T-chain $\bar{\gamma} = \mathcal{T}_T((\alpha \circ B'; \tau))$, with $\tau = \tau_\alpha$ if B is paired in $\bar{\alpha}$ or $\tau \in \{\tau_\alpha, B \circ \tau_\alpha\}$ otherwise, is DERIVED (by loading B') from $\bar{\alpha}$.

Remark. The non-determinism of the loading rule could be avoided admitting loading of a sequence of literals in a single step. In this case taking $\tau = B \circ \tau_\alpha$ when B is not paired in $\bar{\alpha}$ is sufficient to ensure completeness (see Lemma 3.4).

Given a set of non tautological input chains S, then $\bar{S} = \mathcal{T}_T(S)$ contains the T-chain $(\alpha; \Box)$ for each $\alpha \in S$. Moreover, if R is a set of clauses, then $\alpha \in \mathcal{T}(R)$ implies that $(\alpha; \Box) \in \mathcal{T}_T(R)$, because the ordering rule is the same. These facts will be implicitly used below.

We now give a formal definition of derivation in T-ME:

Definition 3.4 *A T-ME-*DERIVATION *of a T-chain $\bar{\alpha}_{r+n}$ from the set of input T-chains $\bar{S}$ is a finite sequence $\bar{\alpha}_1, \ldots, \bar{\alpha}_{r+n}$ of acceptable T-chains such that:*

- *$\bar{\alpha}_1, \ldots, \bar{\alpha}_r$ are input T-chains;*
- *for all $i, r+1 \leq i \leq r+n$, $\bar{\alpha}_i$ is obtained from $\bar{\alpha}_{i-1}$ by:*
  - *T-extension rule, using an input T-chain $\bar{\alpha}_j$ for $j \leq r$; or*
  - *unloading rule; or*
  - *loading rule, loading a literal whose atom occurs in $\bar{S}$.*

The chain $\bar{\alpha}_r$ is the TOP CHAIN *of the T-ME-derivation. Prefix and suffix are defined as usual. A T-ME-refutation is a T-ME-derivation of an empty T-chain.*

Example. Consider the set of clauses: $S = \{\{P, Q\}, \{\neg Q, R\}, \{\neg P, R\}\}$, and let T be defined as follows: $T = \{\{\neg Q, \neg R\}, \{\neg P \neg R\}\}$. Suppose the ordering adopted allows the set of input T-chain $\bar{S}$ listed in the prefix. Then the following is a T-ME-refutation of $\bar{S}$:

1) $(P, Q; \Box)$
2) $(R, \neg Q; \Box)$
3) $(R, \neg P; \Box)$
4) $(P, [Q], R; Q)$ by T-extension of 1 with 2
5) $(P; \Box)$ by unloading of R (since $T \models P \vee \neg Q \vee \neg R$) and application of $\mathcal{T}_T$
6) $([P], R; P)$ by T-extension of 5 with 3
7) $(\Box; \Box)$ by unloading of R (since $T \models \neg P \vee \neg R$) and application of $\mathcal{T}_T$

Given a T-chain $\bar{\alpha}$ and a model M, we write $M \models \bar{\alpha}$ if there is a B-literal in α true in M (i.e., $\bar{\alpha}$ is satisfied in M); moreover, given a set $\bar{S}$ of T-chains, we write $M \models \bar{S}$ for $\forall \bar{\alpha} \in \bar{S}\ M \models \bar{\alpha}$.

3.2 Soundness and Completeness of T-ME

Formally, T-ME is not a refinement of T-resolution. Hence, as occurs also with Model Elimination in the classical setting, we have to prove the soundness of its inference rules. The following technical lemma, exploiting the peculiarities of our inference rules, guarantees that the characteristic property of an ME-calculus holds for the T-ME-calculus.

Lemma 3.1 *Given a T-ME-derivation $\mathcal{D}$ of a T-chain $\bar{\alpha} = (\alpha' \circ [A] \circ \alpha''; \tau_\alpha)$ from $\bar{S}$, there exists a T-ME-derivation $\mathcal{D}'$ of $\bar{\gamma} = (\alpha' \circ A; \tau_\gamma)$ from $\bar{S}$, shorter than $\mathcal{D}$.*

Proof. The occurrence of $[A]$ in $\bar{\alpha}$ could have been introduced in two ways:

1. by an unloading step on a T-chain $\bar{\alpha}_1 = (\alpha'_1 \circ A \circ \alpha''_1; A \circ \tau_{\alpha_1})$ where $A \circ \alpha''_1$ is made of B-literals only, and the occurrence of A in the second part of $\bar{\alpha}_1$ has been introduced by a previous loading step on a T-chain $\bar{\alpha}_2 = (\alpha'_2 \circ A; \tau_{\alpha_2})$, yielding $\bar{\alpha}_3 = (\alpha'_2 \circ A, B; A \circ \tau_{\alpha_2})$ (where B is a B-literal).

   Let us call $\mathcal{D}_1$ the segment of $\mathcal{D}$ starting from $\bar{\alpha}_3$ and yielding $\bar{\alpha}$. In $\mathcal{D}_1$ no framing of paired B-literals to the left of A can be performed using the unloading rule, because the corresponding pairing literal would not be leftmost. Moreover, neither T-extension steps nor loading steps can alter the sequence of literals to the left of A in the first components of T-chains in $\mathcal{D}_1$. Hence, the transformation performed on the prefix α'_2 (of $\bar{\alpha}_2$) to obtain α'_1 (and then α') are independent from literals of the suffix "$A \ldots$" (resp. "$[A] \ldots$"). This is true because being A (resp. $[A]$) paired, each removal of B-literals caused by the unloading rule (see (6)) has to be justified using only literals to the left of A (resp. $[A]$). Therefore, starting from $\bar{\alpha}_2$, just mimicking $\mathcal{D}_1$, it is possible to derive the T-chain $\bar{\gamma} = (\alpha' \circ A; \tau_\gamma)$. The T-ME-derivation $\mathcal{D}'$ obtained is shorter than $\mathcal{D}$ because we can ignore all the steps in $\mathcal{D}_1$ not modifying literals to the left of A; obviously, the first step in $\mathcal{D}_1$ (loading of B) is one of them.

2. by a T-extension step on the T-chain $\bar{\alpha}_1 = (\alpha'_1 \circ A; \tau_{\alpha_1})$ with the input T-chain $\bar{\alpha}_2 = (\alpha'_2 \circ \alpha''_2; \Box)$ obtaining $\bar{\alpha}_3 = (\alpha'_1 \circ [A] \circ \alpha'_2; A \circ \tau_{\alpha_1})$. Afterwards, $\bar{\gamma}$ is produced processing $\bar{\alpha}_3$; let us call $\mathcal{D}_1$ the part of T-ME-derivation yielding $\bar{\gamma}$ from $\bar{\alpha}_1$. Since in each T-chain of $\mathcal{D}_1$ the literal $[A]$ is paired (every A-literal is paired) we can conclude the proof as in the previous case. In this case $\mathcal{D}'$ is shorter than $\mathcal{D}$ because at least the extension step producing $\bar{\alpha}_3$ can be skipped in $\mathcal{D}'$.

□

Lemma 3.2 *Let $\bar{S}$ be a set of input T-chains. For each T-chain $\bar{\alpha}$ T-ME-derived from $\bar{S}$ and for each T-model M, we have that $M \models \bar{S} \rightarrow M \models \bar{\alpha}$.*

Proof. Let $\mathcal{D}$ denote the T-ME-derivation of $\bar{\alpha}$ and let M be a T-model of $\bar{S}$. The proof is by induction on the length of $\mathcal{D}$.

Base case. $\bar{\alpha} \in \bar{S}$: $M \models \bar{S}$ then $M \models \bar{\alpha}$.

Inductive step. Let $\bar{\alpha}_1$ be $\mathcal{T}_T(\bar{\alpha})$ and suppose $M \models \bar{\alpha}$. Clearly, merging left of identical B-literals (rule **R0**), as well as rule **R2**, is sound. Let us show that so is rule **R1**.

Let $\bar{\alpha} = (\alpha' \circ [A] \circ \alpha'' \circ A^c \circ \alpha'''; \tau_\alpha)$ and $\bar{\alpha}_1$ be obtained removing A^c from $\bar{\alpha}$. From Lemma 3.1 it follows that there exists a T-ME-derivation $\mathcal{D}$ of the T-chain $\bar{\gamma} = (\alpha' \circ A; \tau_\gamma)$ from $\bar{S}$ shorter than $\mathcal{D}$. Hence, $M \models \bar{\gamma}$. Since $M \models \bar{\alpha}$ and $M \models \bar{\gamma}$, it follows that $M \models \mathcal{B}(\alpha') \vee A^c \vee \mathcal{B}(\alpha'') \vee \mathcal{B}(\alpha''')$ and $M \models \mathcal{B}(\alpha') \vee A$, hence, we have $M \models \mathcal{B}(\alpha' \circ \alpha'' \circ \alpha''')$. Therefore $M \models \bar{\alpha}_1$.

This proves that the transformation $\mathcal{T}_T$ is sound.

It remains to prove the soundness of the inference rules. This is immediate for loading. Let us consider T-extension and unloading rules:

- let $\bar{\gamma}'$ be obtained as indicated in the definition of the T-extension rule[4]; let $M \models \bar{\alpha}$ and $M \models \bar{\beta}$. From (4) it follows that
$$M \models \mathcal{B}(\alpha_1) \vee \bar{\mathcal{A}}(\alpha_1) \vee \beta_1 \vee \neg\alpha_2 \vee \neg\beta_2. \tag{7}$$
Let $\mathcal{A}(\alpha_1) = (A_1, \ldots, A_p)$ and let $\alpha^{(i)}$ be the prefix of $\bar{\alpha}_1$ to the left of $[A_i]$ $(i = 1, \ldots, p)$. By Lemma 3.1 there exists a T-ME-derivation of $\bar{\alpha}^{(i)} = (\alpha^{(i)} \circ A_i; \tau_{\alpha^{(i)}})$ from $\bar{S}$ such that $M \models \bar{\alpha}^{(i)}$ for each $i = 1, \ldots, p$. Hence, $M \models \mathcal{B}(\alpha^{(i)}) \vee A_i$ $(i = 1, \ldots, p)$ from which we have:
$$M \models \mathcal{B}(\alpha_1) \vee A_i \vee \beta_1 \vee \neg\beta_2 \vee \neg\alpha_2 \tag{8}$$
for each $i = 1, \ldots, p$.
From (7) and (8) it follows that:
$$M \models \mathcal{B}(\alpha_1) \vee \beta_1 \vee \neg\beta_2 \vee \neg\alpha_2 \tag{9}$$
Now we use the hypothesis on $\bar{\alpha}$ and $\bar{\beta}$: the fact that $M \models \mathcal{B}(\bar{\alpha})$ and $M \models \bar{\beta}$, together with (9), gives $M \models \mathcal{B}(\alpha_1) \vee \beta_1$. Therefore $M \models \gamma'$.
- Let $\bar{\alpha}_1$ be obtained as indicated in the definition of the unloading rule[5] and let $M \models \bar{\alpha}$. From (6) it follows that $M \models \mathcal{B}(\alpha' \setminus L) \vee \bar{\mathcal{A}}(\alpha') \vee L^c$. Similarly to the previous case, supposing that $\mathcal{A}(\alpha') = (A_1, \ldots, A_p)$, for each $i = 1, \ldots, p$, it is the case that $M \models \mathcal{B}(\alpha' \setminus L) \vee A_i$. It follows that $M \models \mathcal{B}(\alpha' \setminus L) \vee L^c$. By the hypothesis on $\bar{\alpha}$ we have $M \models \mathcal{B}(\alpha') \vee \mathcal{B}(\alpha'')$. So, $M \models \mathcal{B}(\alpha' \setminus L) \vee \mathcal{B}(\alpha'')$, and hence $M \models \bar{\alpha}_1$.

□

The following proposition gives the soundness of T-ME.

[4]Refer to it for symbolism.
[5]Refer to it for symbolism.

Proposition 3.1 (Soundness of T-ME) *Any T-chain $\bar{\alpha}$ T-ME-derived from a T-satisfiable set of T-chains $\bar{S}$ is T-satisfiable.*

Proof. Let M be a T-model of $\bar{S}$ which must exist because $\bar{S}$ is T-satisfiable. From Lemma 3.2 if follows that $M \models \bar{\alpha}$; hence there must exist at least one B-literal in α true in M.

□

To prove the completeness we translate a standard derivation of a chain α into a T-ME-derivation of the T-chain $(\alpha; \Box)$.

Lemma 3.3 *Given α, β acceptable chains (β input chain) such that γ is obtained by the extension rule from them, and a T-ME-derivation $\mathcal{D}'_1$ of $\bar{\alpha} = (\alpha; \tau_\alpha)$, there exists a T-ME-derivation $\mathcal{D}'$ of $\bar{\gamma} = (\gamma; \tau_\gamma)$.*

Proof. Let $\alpha = L_1, \ldots, L_n, B$ and $\beta = H_1, \ldots, H_{t-1}, B^c$. $\mathcal{D}'$ will be obtained with a further step of T-extension from $\mathcal{D}'_1$. Since β is an input chain there exists an input T-chain $\bar{\beta} = (\beta; \Box)$. The extension step produces a chain γ' framing B and appending $H_1, \ldots, H_{t-1}$ (possibly reordered by O) to its right; γ' is then transformed by $\mathcal{T}$, which (modulo merging of B-literals) may:

1. delete a B-literal L because of a complementary A-literal $[L^c]$ to its left (ground reduction step);
2. remove the rightmost literal if it is an A-literal.

In a T-extension step we can produce the T-chain $(\gamma'; B \circ \tau_\alpha)$ which will be transformed by $\mathcal{T}_T$. Rules **R0**–**R2** allow $\mathcal{T}_T$ to mimic $\mathcal{T}$. It is then possible to derive $\bar{\gamma} = (\gamma; \tau_\gamma)$.

□

The role played by Lemma 2.1 for the refinements described in the previous sections, is here exploited by Lemma 3.4 which relates Model Elimination steps and T-ME steps.

Lemma 3.4 *Given α, β acceptable chains (β valid input chain) such that γ is obtained by the extension rule from them, and a T-ME-derivation $\mathcal{D}'_1$ of $\bar{\alpha} = (\alpha; \tau_\alpha)$, there is a T-ME-derivation $\mathcal{D}'$ of $\bar{\gamma} = (\gamma; \tau_\gamma)$.*

Proof. Let $\alpha = L_1, \ldots, L_n, B$ and $\beta = H_1, \ldots, H_{t-1}, B^c$. As with Lemma 3.3 the extension step produces $\gamma' = L_1, \ldots, L_n, [B], H_1, \ldots, H_{t-1}$ (modulo O and merging of B-literals) and transforms it using $\mathcal{T}$, which performs the two usual kinds of actions. We can derive $(L_1, \ldots, L_n, B, H_1, \ldots, H_{t-1}; B \circ \tau_\alpha)$, by $t-1$ loading steps, from $\bar{\alpha}$; then the unloading rule is applied to make B an A-literal ($[B]$), using the fact that $T \models \neg(H_1 \vee \cdots \vee H_{t-1}) \rightarrow \neg B$ holds. Now, as was done in the proof of Lemma 3.3, let $\mathcal{T}_T$ simulate $\mathcal{T}$ yielding $\bar{\gamma}$.

□

Lemma 3.5 and the subsequent theorem guarantee the completeness of T-ME.

Lemma 3.5 *If there exists a ME-derivation $\mathcal{D}$ of γ from $\mathcal{T}(S) \cup \mathcal{T}(T')$ with top chain in $\mathcal{T}(S)$, then there exists a T-ME-derivation $\mathcal{D}'$ of $\bar{\gamma} = (\gamma; \tau_\gamma)$ from $\mathcal{T}_T(S)$.*

Proof. By induction on the length n of *suffix*$(\mathcal{D})$.

Base case. $n = 0$. There is nothing to prove since, by hypothesis, $\bar{\gamma} \in \mathcal{T}_T(S)$.

Inductive step. Assume the lemma holds for all ME-derivations with suffix shorter than $n > 0$. Let $\mathcal{D} = \mathcal{D}_1 \circ \gamma$ and let $\mathcal{D}_1$ be a ME-derivation of α, and let γ obtained by extension from α and an input chain $\beta \in \mathcal{T}(S) \cup \mathcal{T}(T')$.

By induction hypothesis on $\mathcal{D}_1$, there exists a T-ME-derivation $\mathcal{D}'_1$ of $\bar{\alpha} = (\alpha; \tau_\alpha)$ from $\mathcal{T}_T(S)$. There are two possibilities:

1. if $\beta \in \mathcal{T}(S)$ then $\bar{\beta} = (\beta; \Box) \in \mathcal{T}_T(S)$, α and β are acceptable and $\mathcal{D}'_1$ is a T-ME-derivation of $\bar{\alpha}$; then by Lemma 3.3 we are through.
2. if $\beta \in \mathcal{T}(T')$ then since α and β are acceptable, β is valid and $\mathcal{D}'_1$ is a T-ME-derivation of $\bar{\alpha}$; then by Lemma 3.4 we are through.

□

Lemma 3.5 is employed in proving our main result:

Theorem 3.1 (Completeness of T-ME) *Let O be an ME-ordering rule. Given a T-unsatisfiable set S of clauses, there exists a T-ME-refutation of $\bar{S}' = \mathcal{T}_T(S')$ for $S' \subseteq S$.*

Proof. Immediate if S contains a T-unsatisfiable clause α. In fact, starting from any T-chain of α, a sequence of unloading steps will give an empty T-chain.

Otherwise, there exist a finite subset S' of S and a finite set T' of ground valid clauses such that $S' \cup T'$ is minimally unsatisfiable. Obviously, if S' is T-unsatisfiable so is $\mathcal{T}(S')$, and then $\mathcal{T}(S') \cup \mathcal{T}(T')$ is an unsatisfiable set of chains such that at least one chain in $\mathcal{T}(S')$ is essential for unsatisfiability. By the completeness of ME, it follows that there exists an ME-refutation of $\mathcal{T}(S') \cup \mathcal{T}(T')$ with top chain in $\mathcal{T}(S')$.

By Lemma 3.5 we can obtain the desired result, since for each acceptable T-chain $\bar{\alpha}$ it is the case that $\mid \tau_\alpha \mid \leq \mid \alpha \mid$, considering the T-chain $\bar{\gamma}$ whose existence is ensured by Lemma 3.5; if $\gamma = \Box$ then $\bar{\gamma} = (\Box; \tau_\gamma) = (\Box; \Box)$.

□

3.3 Implementation and experimental results: a case study

In this section we will briefly describe an implementation of T-ME written in SETL2 ([SDDS86]) and run on a Sun Sparc 10, and we will present some of our experimental results. Since our implementation is just a prototype, we have been mainly interested in the number of steps performed for a given example rather than with CPU time or other technical details.

As is the case with Model Elimination a number of optimizations and features can be added to the original calculus. Most of them have analogues used with Model Elimination (see for example [FLSY74]) although some do take advantage of the specific properties of the T-resolution rule.

Search strategy. The search space can be viewed simply as a tree (possibly, with multiple occurrences of equal sub-trees) whose nodes are T-chains and such that each child of a node can be obtained from its parent by T-extension, loading, or unloading.

The fact that the T-ME procedure (as Model Elimination does) essentially deals with input derivations suggests an immediate optimization: the procedure can avoid processing the same T-chain more than once (i.e., visiting equal sub-trees).

The search strategy used is a depth-first iterative deepening combined with a best-first heuristic search. Basically, the algorithm proceeds by performing a sequence of bounded depth-first searches, with increasing bounds. The bound is not placed on the depth reached in the search tree but rather on the total cost of a path, the cost of a path taking into account the distance from the root (i.e., the node corresponding to the top T-chain) to the currently visited node n and the length of the path from n to a leaf-node (possibly corresponding to a refuting T-chain). This length has to be heuristically estimated and the simple heuristic employed is the number of B-literals in the first part of the current T-chain, which is in fact a lower bound to the length of the remaining part of the path. At each iteration a depth-first search is performed cutting off each branch whose total cost exceeds some given value. The value employed in the subsequent iteration will be the lowest cost among those exceeding the bound in the current search.

The algorithm is initially provided with a list of input T-chains. The selection of the top T-chains is made following the given order. Unloading is attempted before T-extension, which precedes loading operations. For each input T-chain $\bar{\alpha}$ (suppose its first part contains h B-literals);then, at most h contrapositives can be used as side T-chains in T-extension steps. These are obtained from $\bar{\alpha}$ by moving each B-literal to the right end of the first part. This fulfills Loveland's requirement that each literal should occur in the rightmost position somewhere in the input set (see [Lov78]). Control parameters (see below) usually prevent the exploration of all h alternatives.

Control parameters and rejection rules. A set of control parameters can be employed to reduce the number of derivable T-chains:

- bounds on the number of A-literals and B-literals occurring in the first part of a derived T-chain;

- bound on the length of the first part of a derived T-chain;

- for each input T-chain $\bar{\alpha}$, there is a bound on the number of T-extension steps that can employ a contrapositive of $\bar{\alpha}$ as side T-chain in the same derivation;

- the set of input T-chains is partitioned in classes (a similar strategy is employed in the implementation of Model Elimination described in [FLSY74]) according to the length of the first parts. There are four classes, respectively for input T-chains made of 2, 3, 4 and more than 4 B-literals. Units are treated in a special way, as will be explained below.

  It is possible to impose a bound on the total number of side T-chains used in the same derivation and coming from the same class.

Rejection rules are employed to eliminate useless branches of the search space:

- acceptability of derived T-chains is imposed, so tautologies are always avoided;

- any T-chain whose first part has the first part of an input T-chain, or of a previously derived T-chain (in the current derivation), as prefix is rejected;

- T-chains already derived are rejected;

- rejection of valid T-chains can be optionally imposed.

Lemmas and units. Note that in T-ME the (ground) reduction operation is incorporated in the T-extension step. In fact, it can be viewed as a special case of the unloading operation. However, obtaining lemmas from reduction steps as is possible in Model Elimination is not very useful in T-ME because the characteristics of the T-resolution rule force one to consider *each* B-literal in the first part of the T-chains as part of the lemmas.

Actions produced by the T-extension and unloading rules on B-literals are justified in general by the presence of several B-literals (see (4) and (6)) in the first parts of the T-chains. Extending the "scope-mechanism" (see [Lov78]) to T-ME would require the capability to determine which literals in the T-chains are sufficient to perform the T-derivation step (in order to obtain short lemmas). Obviously, handling this "minimality" requirement would make lemma production not so advantageous. On the other hand, if this requirement is ignored, each literal (the A-literals complemented) of the first part of a T-chain would have to be included in each lemma produced from it, making lemmas substantially useless.

Nevertheless, each derived T-chain generates a lemma consisting of the B-literals occurring in its first part. Lemmas of this kind are actually used in our implementation with some restrictions:

- only lemmas with less than a predefined number of literals are retained;

- a non-unit lemma is retained only if it subsumes at least one input T-chain, in which case the subsumed T-chains are replaced by the new lemma;

- unit lemmas are intensively employed. Together with unit input T-chains, they form a conjunction of literals which will be always considered in each T-satisfiability test performed by the procedure (in general this technique tends to reduce the number of T-models to be considered by the T-decider). Moreover, whenever a new unit is obtained, a step similar to the Davis-Putnam's unit rule is applied to the input T-chains: those subsumed are removed (they will not be used anymore), whereas each occurrence of literals complementary to the derived unit is deleted from the T-chains (this process can generate new units which will be employed in the same manner to simplify the given set of T-chains). Clearly, if a T-unsatisfiable conjunction of literals is obtained, the initial set of T-chains is declared T-unsatisfiable. We notice that it may happen that the procedure finds a T-satisfiable conjunction of literals satisfying each input T-chain, in which case the set is declared T-satisfiable.

  The strategy just outlined takes advantage of previously derived T-chains and the use of units corresponds to permitting implicit non-binary steps (as in the case of total theory resolution) but only

when all but one of the side T-chains are units and complementary to a literal occurring in the other two parent T-chains.

From another point of view, the conjunction of units gives the T-decider supplementary knowledge about the entire problem faced by the theorem prover, making the T-satisfiability tests more restrictive, "simplifying" the work of the T-decider and increasing global efficiency.

Further features and heuristics. A few strategies and heuristics were employed mainly to increase efficiency and to reduce the number of calls to the T-satisfiability decider:

- ground-reduction and merging left of identical B-literals are handled automatically;
- unloading is always tried first;
- each rejection test is done as soon as possible, possibly before invoking the T-satisfiability decider;
- standard resolution steps are intercepted without invoking the T-satisfiability decider;
- the set of side T-chains (their contrapositives) is heuristically ordered before trying T-extension, using an estimate for the number of B-literals of each potential resolvent. Moreover, resolution steps may be attempted before proper T-extension steps;
- loading operations can be restricted in one of the following ways:
  - completely avoided;
  - allowed only if the loaded literal does not occur in the initial set of T-chains, but its complement does.

Remark. Even if loading is avoided, the use of units as described above actually increases the power of the procedure. For example the T-unsatisfiable set of clauses of the example in Section 1.3 is found to be T-refutable by our implementation even without loading operations.

Remark. Though we restrict our treatment to the ground case of T-ME, our implementation is able to deal with with certain non-ground problems. This capability arises from the presence of the T-decider: an appropriate modeling of the problem and a convenient choice of the theory T may enable one to incorporate the non-ground part of the problem in T.

An example. We applied our procedure to Lewis Carroll's "Salt and Mustard Problem" ([Car77]).

The problem is about five friends (Barry, Cole, Dix, Lang and Mill) that agreed to have lunch together. They devised the following rules, to be observed whenever beef appeared on the table. The rules state:

1. If Barry takes salt, then either Cole or Lang takes one only of the two condiments, salt and mustard; if he (Barry) takes mustard, then either Dix takes neither condiment, or Mill takes both.
2. If Cole takes salt, then either Barry takes only one condiment, or Mill takes neither; if he takes mustard, then either Dix or Lang takes both.
3. If Dix takes salt, then either Barry takes neither condiment, or Cole takes both; if he takes mustard, then either Lang or Mill takes neither.
4. If Lang takes salt, then either Barry or Dix takes only one condiment; if he takes mustard, then either Cole or Mill takes neither.
5. If Mill takes salt, then either Barry or Lang takes both condiments; if he takes mustard, then either Cole or Dix takes only one condiment.

Moreover, it is assumed that:

- phrases like "Barry takes salt" leave open two possibilities: "Barry takes salt only" or "Barry takes both condiments";

- similarly, phrases like "either Cole or Lang takes one only" allow three possible cases: "Cole takes one only, Lang takes both or neither", "Lang takes one only, Cole takes both or neither", and "Cole takes one only, Lang takes one only";

- it is also assumed that every rule is to be understood as implying the worlds "*and vice versa*" (i.e., the first rule should be completed by: "and, if either Cole or Lang takes only one condiment, then Barry takes salt", and so on).

The original problem was to discover if these rules are compatible, i.e., satisfiable. A (non-ground) clause formulation of the problem is used in [MB88] to shown that the conditions are in fact satisfiable as follows:

Barry takes both condiments.

Cole and Dix take neither salt nor mustard.

Lang takes mustard but not salt.

Mill takes salt but not mustard.

Let us see how to deal with this problem in the context of T-ME. Employing a theory T, which is essentially a finite axiomatic set theory modeling only membership, union, intersection and symmetric difference of sets, it is possible to formulate the Salt and Mustard Problem at the ground level.

The language $\mathcal{L}(T)$, is $2LS$ ([FO78]). It is a simplified case of MLS, (*Multi Level Syllogistic*) which is known to be decidable ([FOS80]). A $2LS$-decider has actually been implemented.

We have $\mathcal{L}(T) \supseteq \{in, union, inter, sdiff\}$. In this particular problem we will model the five friends as individuals, while sets will identify who takes which condiment. Thus, if the friends are represented by the letters a, b, c, l, m, and if S and M represent the set of those who take salt or mustard, respectively, then the set of those who take both condiments is given by $S \cap M$. We use the atom $inter(interMS, S, M)$ to represent the assertion that $interMS$ is the intersection of S and M.

The following three unit T-chains define respectively, the sets of those who take only one, at least one, and both condiments:

1) $(< sdiff(deltaMS, M, S) >; <>)$
2) $(< union(unionMS, M, S) >; <>)$
3) $(< inter(interMS, M, S) >; <>)$

The five rules describing the problem can be reformulated as the following set of T-chains:

4) $(< \neg in(b, M), \neg in(d, unionMS), in(m, interMS) >; <>)$
5) $(< in(d, unionMS), in(b, M) >; <>)$
6) $(< \neg in(l, deltaMS), in(b, S) >; <>)$
7) $(< \neg in(b, S), in(c, deltaMS), in(l, deltaMS) >; <>)$
8) $(< in(b, S), \neg in(c, deltaMS) >; <>)$
9) $(< in(b, M), \neg in(m, interMS) >; <>)$
10) $(< \neg in(c, S), \neg in(m, unionMS), in(b, deltaMS) >; <>)$
11) $(< \neg in(b, deltaMS), in(c, S) >; <>)$
12) $(< in(c, S), in(m, unionMS) >; <>)$
13) $(< \neg in(c, M), in(l, interMS), in(d, interMS) >; <>)$
14) $(< \neg in(d, interMS), in(c, M) >; <>)$
15) $(< \neg in(l, interMS), in(c, M) >; <>)$
16) $(< \neg in(d, S), in(c, interMS), \neg in(b, unionMS) >; <>)$
17) $(< in(d, S), in(b, unionMS) >; <>)$
18) $(< \neg in(c, interMS), in(d, S) >; <>)$
19) $(< \neg in(d, M), \neg in(m, unionMS), \neg in(l, unionMS) >; <>)$
20) $(< in(d, M), in(l, unionMS) >; <>)$

21) $(< in(m, unionMS), in(d, M) >; <>)$
22) $(< \neg in(l, S), in(b, deltaMS), in(d, deltaMS) >; <>)$
23) $(< \neg in(b, deltaMS), in(l, S) >; <>)$
24) $(< \neg in(d, deltaMS), in(l, S) >; <>)$
25) $(< \neg in(l, M), \neg in(c, unionMS), \neg in(m, unionMS) >; <>)$
26) $(< in(c, unionMS), in(l, M) >; <>)$
27) $(< in(m, unionMS), in(l, M) >; <>)$
28) $(< \neg in(m, S), in(b, interMS), in(l, interMS) >; <>)$
29) $(< \neg in(b, interMS), in(m, S) >; <>)$
30) $(< \neg in(l, interMS), in(m, S) >; <>)$
31) $(< \neg in(m, M), in(d, deltaMS), in(c, deltaMS) >; <>)$
32) $(< \neg in(c, deltaMS), in(m, M) >; <>)$
33) $(< \neg in(d, deltaMS), in(m, M) >; <>)$

To build a T-unsatisfiable set of clauses we need a further T-chain, expressing the negation of the answer (i.e., eliminating the only model of the given set of clauses):

34) $(< \neg in(m, S), in(m, M), \neg in(l, M), \neg in(b, interMS), in(d, unionMS), in(c, unionMS), in(l, S) >; <>)$

Our implementation of T-ME was able to find a T-refutation of the given set of T-chains with various settings of the control parameters, in particular when the parameters were defined as follows:

- bound on the number of A-literals: 6;
- bound on the number of B-literals: 2;
- bound on the length of the first part: 7;
- at most one contrapositive for each input clause in the same derivation;
- no loading operation is allowed, and no retention of lemmas;
- forced rejection of valid resolvents

The procedure derived a T-unsatisfiable conjunction of unit T-chains, declaring the conjunction of the 34 input clauses to be T-unsatisfiable. This was done producing 150 new T-chains and trying T-extension on 91 of them. The T-decider was called 6263 times on 2097 different conjunction of literals (further optimization could have been realized by intercepting repeated calls).

It should be noticed that the choice of the top T-chains along with the retention of valid T-chains considerably influences the search for the refutation. The most useful parameters for reducing the portion of the search space explored seem to be the bounds on length and number of literals as well as an appropriate choice of the initial cost bound.

4 Applications of T-resolution

This section illustrates how an approach based on T-resolution can be profitably adopted in various situations. In particular, we show how the features of the T-resolution rule can be exploited to define a deduction framework which generalizes the CLP approach to Logic Programming. A second application is a theorem prover for modal logic employing an implementation of the T-ME calculus to realize a (semi)-decision procedure for a wide class of modal logics.

4.1 T-resolution as a deduction scheme

A general deduction scheme based on T-resolution was proposed in [DFP97]. The starting points are the previously illustrated features of the T-resolution rule, namely:

- the existence of non-trivial linear refinements;
- the capability of integrating domain specific knowledge;
- the clear distinction and a strong separation between the background and the foreground reasoners.

These features are in strict analogy with similar properties of CLP ([JL86, JM94]), arising from the integration of Logic Programming and (independently developed) constraint solvers from which its usefulness derives. In this context the background and the foreground reasoners realize the calculation level (i.e., constraint solving) and the deduction level of the system, respectively.

The generality of the theories T allowed in our development is very advantageous. Ordinarily, to formalize a real problem in terms of a set of clauses to be processed by an automated deduction system, it is necessary to design a theory suitable to express the characteristics of the specific domain of knowledge of interest.

If the system is based on a pure Logic Programming approach (e.g., Prolog), then the axioms of that theory have to be added to the description of the problem. This considerably increases the number of the clauses to be taken into account.

In the systems based on the CLP-schemes, where an equational theory is built-in, we do not need to consider the standard equality axioms as part of the problem description. This can substantially simplify the set of clauses and provide higher efficiency. However, not every problem can easily be expressed in an equational theory, and even when this is the case, it might be more appropriate to employ a theory more directly related to the specific domain of knowledge from which the problem is taken.

Usually, in schemes based on CLP the only way to share/exchange information between the theory and the program is via the equality relation. The T-resolution approach allows a better and more useful cooperation between the foreground reasoner and the T-decider which, in general, can handle a non-equational theory.

The deduction scheme we propose has the standard CLP-scheme as a special case and offers several improvements which arise from the possibility of using interpreted predicate symbols (i.e., symbols of $\mathcal{L}(T)$) in the head of the clauses (this feature is ordinarily not present in CLP-schemes).

4.1.1 T Logic Programming

In this subsection we introduce the syntax and semantics of our deduction scheme, T Logic Programming (TLP for short). We will adopt notation taken from [JL86] (which may be consulted for a detailed description of CLP).

Let $\Sigma = \Sigma_C \cup \Sigma_P$ be a first-order signature. Σ_C is the CONSTRAINT signature and Σ_P is the PROGRAM signature. We require: $\Sigma_C \cap \Sigma_P = \emptyset$, $\Sigma_C = \Pi_C \cup \mathcal{F}_C$, and $\Sigma_P = \Pi_P \cup \mathcal{F}_P$, where Π denotes a set of predicate symbols and $\mathcal{F}$ a set of function (and constant) symbols. A Π-ATOM is an atom $p(t_1, \ldots, t_n)$, where $p \in \Pi$ and $t_1, \ldots, t_n$ are terms built from $\mathcal{F}_C \cup \mathcal{F}_P$ and a denumerable set of variables. A Π-LITERAL is either a Π-atom or the negation of a Π-atom. There are cases in which T deals with all possible functional symbols (e.g., standard Clark Equality Theory); in such cases, clearly, $\mathcal{F}_P = \emptyset$.

The general form of a T-Logic Programming PROGRAM CLAUSE is $B_0 \leftarrow B_1, \ldots, B_n$ where B_i can be either a Π_P-atom, or a Π_C-literal[6]. If B_0 is a Π_C-literal, then the clause is said to be a T-HEADED CLAUSE (such clauses are not allowed in CLP-schemes). If $B_0, B_1, \ldots, B_n$ all are Π_C-literals, then the clause is said to be a CONSTRAINT CLAUSE. A GOAL is a clause with empty head.

Unlike predicate symbols of Π_P, predicates in Π_C can be (partially) defined in T, which can be any first order theory. Moreover, they can occur in a negative literal either in the head or in the body of a TLP-program clause.

[6] Hence, negative Π_P-literals can not occur in any clause

In CLP there is no way to act on the theory T by modifying semantics of the predicate and function symbols of Σ_C and of $\mathcal{F}_P$. For instance, having a $CLP(\mathcal{R})$ system, for two uninterpreted functional symbols $f, g \in \mathcal{F}_P$, we have no means of requiring that $g(X, X) < f(X)$ when $0 < X < 1$.

The possibility of writing TLP-clauses with interpreted head makes it possible to overcome such restrictions. There are three types of program clauses of this kind:

1. clauses that add known (entailed) information to T. For instance, if T is a theory of natural numbers and the symbol '$<$' belongs to Π_C: $(0 < X) \leftarrow (1 < X)$.
2. Clauses which cause the inconsistency of $T \cup P$. With respect to the above theory T, for instance: $(x * x < 0) \leftarrow (0 < x)$.
3. Clauses which contain the functional symbols in $\mathcal{F}_P$ or $\mathcal{F}_C$ and consistently extend T, such as, for instance: $(f(X) < f(Y)) \leftarrow (X < Y)$.

Clearly, situations of the type 2) should be avoided. This task is left to the programmer.

Considering constraint clauses only, we can characterize these three kind of clauses in model-theoretic terms: clauses of type 1) have, as models, all the models of T; none of the models of clauses of type 2) is model of T; only a proper subset of the models of T are models of clauses of kind 3). Clauses of type 1) could be introduced in order to give a higher priority to some theorems of the theory T in the inference process.

An issue offered by TLP is the possibility of using program-defined predicates, to provide a semantics to new constants.

Example. Consider a program containing the TLP-clause:

$$X \in c \leftarrow X \in a, r(X)$$

where $r(X)$ is a program defined atom (i.e., a Π_P-atom). The above TLP-clause characterizes the constant c by requiring that in each T-model of the program P it is the case that:

$$\{X \in a \mid r(X)\} \subseteq c.$$

By means of this feature it is possible, for instance, to force the semantics of a constant (ω in the following example) in a such a way that in every model of the program it is interpreted as the set of answers to certain goals ($\leftarrow num(X)$).

Example. Let T be a fragment of set theory, and consider the program:

$$num(\emptyset) \leftarrow$$
$$num(X \cup \{X\}) \leftarrow num(X)$$
$$(X \in \omega) \leftarrow num(X)$$

The (minimal) semantics for the constant symbol ω is exactly the (infinite) set of all numerals intended *à la* Von Neumann.

The following definition introduces the rules of TLP, each of them being an instance of the general T-resolution rule (cf. Definition1.6). In particular, **R1** is a generalization of the standard Prolog-like inference rule and it is sufficient to simulate the CLP inference rule, provided the satisfiability checker is implemented (hence, TLP has CLP as a special case).

Definition 4.1 *Let μ be a substitution and $\leftarrow H_1, \ldots, H_k$ be a goal* [7].

[7] For the sake of simplicity, we assume the body of a clause to be a multiset of literals. This guarantees that selecting H_1 will be like considering a generic element H_i of the multiset.

R1: *If $B_0 \leftarrow B_1, \ldots, B_n$ is a TLP-clause, and B_0 and H_1 are Π_P-atoms,*

$$\frac{B_0 \leftarrow B_1, \ldots, B_n \quad \leftarrow H_1, \ldots, H_k}{\leftarrow (B_1, \ldots, B_n, H_2, \ldots, H_k)\mu}$$
$$T \models \vec{\forall}(((B_0 \wedge \neg H_1) \rightarrow (\neg B_1 \vee \cdots \vee \neg B_n) \vee (\neg H_2 \vee \cdots \vee \neg H_k))\mu)$$

R2: *If $\leftarrow B_0, \ldots, B_n$ is a T-headed clause (i.e., $\neg B_i \leftarrow B_0, \ldots, B_{i-1}, B_{i+1}, \ldots, B_n$) or a previously derived goal,*

$$\frac{\leftarrow B_0, \ldots, B_n \quad \leftarrow H_1, \ldots, H_k}{\leftarrow (B_0, \ldots, B_s, H_1, \ldots, H_r)\mu}$$
$$T \models \vec{\forall}((\quad (\neg B_{s+1} \vee \cdots \vee \neg B_n) \wedge (\neg H_{r+1} \vee \cdots \vee \neg H_k) \rightarrow (\neg B_0 \vee \cdots \vee \neg B_s) \vee (\neg H_1 \vee \cdots \vee \neg H_r))\mu)$$

where $H_{r+1}, \ldots, H_k, B_{s+1}, \ldots, B_n$ are all Π_C-literals.

Loading: *Let α be program clause, and L be a Π_C-literal,*

$$\frac{\alpha \quad \leftarrow H_1, \ldots, H_k}{(\leftarrow H_1, \ldots, H_k, L)\mu} \quad L \text{ in } \alpha \text{ or } \neg L \text{ in } \alpha$$

Rules **R1** and **R2** constitute a proper refinement of the general T-resolution rule. **R1** operates on pairs of Π_P-literals in a classical fashion; however the rule generalizes the standard Prolog inference step since it uses knowledge embedded in the theory and in the whole parent-clauses. Rule **R2**, on the other hand, uses the full power of T-resolution and deals only with the "T-part" of the TLP-program.

Notice that the parts of a TLP-derivation (see below) built using rule **R1** actually have a linear-input character (just as Horn clauses —pure Prolog— derivations have). This is the main reason justifying the introduction of two different instances (i.e., **R1** and **R2**) of the general T-resolution rule.

Observe that strictness condition —i.e., never derive a clause including a variant of the parent clauses (this restriction is analogous to that imposed on ground linear T-resolution, see Section 2.2)— holds for rule **R1**. Moreover, we require that only strict applications of **R2** are allowed. This does not affect completeness and guarantees a proper linearity of the derivation.

Definition 4.2 *Given a program P and a goal G, a TLP-*DERIVATION *is a sequence of goals $G = G_0, \ldots, G_n$ such that, for each $i \in \{1, \ldots, n\}$, G_i is obtained from G_{i-1} and:*

- *a program clause using rule* **R1** *or* **Loading**,
- *a T-headed program clause, or a goal G_k, $k < i$, using rule* **R2**.

*A TLP-*REFUTATION *is a derivation of a goal $\leftarrow H_1, \ldots, H_k$, with H_i a Π_C-literal (for all $i \in \{1, \ldots, k\}$), such that $H_1 \wedge \cdots \wedge H_k$ is T-satisfiable. For each $i \in \{1, \ldots, n\}$, let μ_i be the substitution employed to obtain G_i from G_{i-1}. The pair consisting of $\mu_1 \circ \cdots \circ \mu_n$ and $H_1 \wedge \cdots \wedge H_k$ is the* COMPUTED ANSWER.

Example. Let T be a theory over the reals dealing with $\leq$ and $\in$. Consider the simple program P defined by the following clauses:

$$\begin{aligned} X \in a &\leftarrow 2 \leq X \\ p(X) &\leftarrow X \in a, X \leq 2. \end{aligned}$$

The first clause provides semantics for the constant symbol a in terms of $\in$, stating that it must be interpreted as a set containing at least all numbers greater than or equal to 2. The second clause defines a Π_P-literal (namely, $p(X)$) in terms of the new constraint $X \in a$. Submitting the goal: $\leftarrow p(X)$, a few inference steps yield the goal $\leftarrow 2 \leq X, X \leq 2$.

The following theorems give soundness and completeness of TLP:

Theorem 4.1 (Soundness) *Rules* **R1**, **R2**, *and* **Loading** *preserve T-satisfiability.*

Proof. It is immediate that the three rules are instances of the general T-resolution inference rule. Therefore, the soundness follows from Theorem 1.5. □

Theorem 4.2 (Completeness) *Let P be a TLP-program, and G be a goal. If $P \cup \{G\}$ is a T-unsatisfiable set of clauses, then there exists a TLP-refutation of $P \cup \{G\}$.*

Proof. The proof will follow the classical pattern of proving ground completeness and then lifting the result to the general case. Ground completeness follows easily from the completeness of ground SL-T-resolution (see [DFP97] for a detailed proof). □

Even though the above result is a direct consequence of the completeness of SL-T-resolution, the restrictions imposed on the T-resolution rule and on the kind of program clauses allowed in the context of TLP, make Theorem 4.2 rather significant. As a matter of fact, the two main differences between TLP and SL-T-resolution are the following: 1) TLP deals with sets of clauses built from two different (disjoint) sets of symbols (namely, Σ_C and Σ_P); 2) the part of the derivation relative to Π_P-literals is in fact a linear-input derivation. In other words, the inference process relative to Π_P-literals proceeds in a "Prolog-like fashion". Neither of these characteristics is achievable in the general context of SL-T-resolution.

4.2 A Modal Theorem Prover

The main problem addressed here is to determine the derivability of a modal formula from a given set of modal axioms. Most of the methods proposed in the past have followed one of two different approaches.

- DIRECT METHODS work directly on the modal formula in analogy with the methods developed for classical logic;
- INDIRECT METHODS solve the problem by translating it into an equivalent problem in classical logic and then applying usual deduction algorithms.

The key point in the approach outlined below is a new translation technique introduced in [DMP95] (and subsequently studied in [BDMP96, BDMP97]), that enables one to express modal formulas in set-theoretic terms. In this manner the problem of deciding the validity of a (propositional) modal formula is solved by deciding the validity of a first-order formula with respect to a suitable ground-decidable set theory (i.e., MM, see below).

We first recall some basics about modal logic (for a detailed description see [Ben85]). In the following we will use a fairly standard syntax for propositional modal logic, consisting of: propositional variables (or letters) $\mathsf{P}_1, \mathsf{P}_2, \ldots$; logical connectives $\wedge, \neg$; the modal operator $\Box$. Derived symbols, to be used as abbreviations, are $\vee$ and $\Diamond$ (defined as $\neg\Box\neg$). Well-formed formulas are defined as usual with the $\Box$ as a unary operator.

The starting point for the set-theoretic translation is the notion of *Kripke frame* semantics: a FRAME F is a pair (W, R) in which W represents the set of (possible) WORLDS and R is a binary relation on W called ACCESSIBILITY RELATION. A VALUATION of a propositional variable is a subset of W. Relative to a valuation of all variables, one defines, for all $w \in W$, the notion $w \models \varphi$ by induction on the structural complexity of the modal formula φ (see [Ben85]).

A formula φ is:

- VALID IN A FRAME (W, R) if and only if, for all $w \in W$ and for any valuation $\models$, $w \models \varphi$ holds;
- VALID if it is valid in all frames;
- a FRAME LOGICAL-CONSEQUENCE of a formula ψ ($\psi \models_f \varphi$) if and only if, for all frames F, if ψ is valid in F, then φ is valid in F.

The minimal modal logic K_s consists of a set of propositional axioms complete for classical logic, the modal axiom

$$\Box(\mathsf{P}_1 \to \mathsf{P}_2) \to (\Box\mathsf{P}_1 \to \Box\mathsf{P}_2),$$

and the rules of modus ponens, substitution (i.e., infer $\varphi^{\mathsf{P}_i}_{\psi}$ from φ), and necessitation (infer $\Box\varphi$ from φ). Derivability of φ from ψ in K_s is defined as usual and denoted by $\psi \vdash_{K_s} \varphi$. A formula ψ is said to be COMPLETE if and only if, for all φ,

$$\psi \vdash_{K_s} \varphi \quad \Leftrightarrow \quad \psi \models_f \varphi.$$

Other modal logics can be obtained considering a set H of modal axioms extending the minimal system K_s. For instance, here are some common modal axiom schemes:

$$\begin{array}{rl} T: & \Box P \to P \\ 4: & \Box P \to \Box\Box P \\ B: & P \to \Box\Diamond P \\ 5: & \Diamond P \to \Box\Diamond P \end{array}$$

The basic idea of the set-theoretic translation is to replace the accessibility relation (R) by the membership relation ($\in$). On this basis, a world v accessible from w becomes an *element* of w; a further step from v using the accessibility relation R will be like *looking into* v in order to reach one of its elements.

This straightforward encoding of the relation R as membership has a number of interesting consequences:

1. worlds, frames, as well as valuations of propositional variables are simply *sets* (of worlds);
2. a frame F can be identified with its support W, being the accessibility relation implicitly defined as the membership relation on W;
3. since we clearly want that all worlds v accessible from a given world w in a frame W are themselves elements of W, it is natural to require that all frames are *transitive* sets.[8]

As a valuation for a propositional variable is but a set of worlds, the standard definition of $\models$ will allow us to associate a set of worlds with each propositional formula. This will be the collection of those worlds in the frame in which the formula holds true. At this point the relation $\models$ can be entirely replaced by the membership relation $\in$.

The translation involves a specific theory, MM, which is defined by the following set of axiom schemes:

$$\begin{array}{l} x \in y \cup z \leftrightarrow x \in y \vee x \in z \\ x \in y \setminus z \leftrightarrow x \in y \wedge x \notin z \\ x \subseteq y \leftrightarrow \forall z(z \in x \to z \in y) \\ Pow(x) \cap Pow(y) \subseteq Pow(x \cap y) \\ x \subseteq y \to Pow(x) \subseteq Pow(y) \end{array}$$

where $\varphi_1 \cap \varphi_2$ stands for $\varphi_1 \setminus (\varphi_1 \setminus \varphi_2)$.

The theory MM has been shown to be ground-decidable (see [Tur96], where a decision algorithm for MM is given). The following result holds:

Theorem 4.3 *Given any unquantified formula φ of $\mathcal{L}(MM)$, it is decidable whether or not there exists a model of MM in which $\vec{\exists}\varphi$ is satisfied.*

A given modal formula $\varphi(P_1, \ldots, P_n)$ is translated into a formula $\varphi^*(x, x_1, \ldots, x_n)$ of $\mathcal{L}(MM)$ using the so called $\Box$-AS-Pow TRANSLATION, which acts following these rules:

- $P_i^* = x_i$;

[8] A set is *transitive* if it contains the members of all of its members.

- $\top^* = x$;
- $(\varphi_1 \vee \varphi_2)^* = \varphi_1^* \cup \varphi_2^*$;
- $(\varphi_1 \wedge \varphi_2)^* = \varphi_1^* \cap \varphi_2^*$;
- $(\neg\varphi)^* = x \setminus \varphi^*$;
- $(\varphi_1 \rightarrow \varphi_2)^* = (x \setminus \varphi_1^*) \cup \varphi_2^*$;
- $(\Box\varphi)^* = Pow(\varphi^*)$.

Moreover, $\Diamond$ is translated as $\neg\Box\neg$, and $\varphi_1^* \cap \varphi_2^*$ stands for $\varphi_1^* \setminus (\varphi_1^* \setminus \varphi_2^*)$.

If $\psi(x_1, \ldots, x_m)$ is the conjunction of the modal axioms defining the modal theory H, and $\varphi(P_1, \ldots, P_n)$ is a modal formula, then the problem $\psi \models_f \varphi$ is faced in solving the corresponding problem obtained by means of the $\Box$-as-Pow translation:

$$MM \models (\forall x(x \subseteq Pow(x) \wedge Axiom_H(x) \rightarrow \forall\vec{z}(x \subseteq \varphi^*(x, \vec{z}))))$$

where $Axiom_H(x)$ corresponds to the formula $\forall x_1, \ldots, x_m(x \subseteq \psi^*(x, x_1, \ldots, x_m))$.

The following result ensures soundness and completeness of the above approach (see [Tur96]):

Theorem 4.4 *Let* $(\psi, \varphi)^* = \forall x(x \subseteq Pow(x) \wedge Axiom_H(x) \rightarrow \forall\vec{z}(x \subseteq \varphi^*(x, \vec{z})))$*, then:*

- $\psi \vdash_{K_s} \varphi \Rightarrow MM \models (\psi, \varphi)^*$*;*
- $MM \models (\psi, \varphi)^* \Rightarrow \psi \models_f \varphi$.

Moreover, if H *is complete, then the following holds:* $MM \models (\psi, \varphi)^* \Leftrightarrow \ \psi \vdash_{K_s} \varphi$.

The (prototypal) modal theorem prover described in [Tur96] consists of three different modules combined to constitute a semi-decider for modal logic. Notice that the modal logic can be viewed as a parameter of the modal theorem prover. The modules are (see Figure 3):

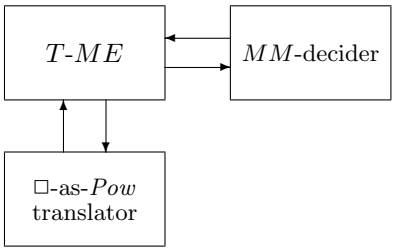

Figure 3: The MTP architecture.

- a translator exploiting the $\Box$-as-Pow translation outlined above;
- a decider for the theory MM;
- an implementation of the Model Elimination refinement of linear T-resolution (T-ME).

The interaction between these modules is rather simple: given a modal formula ψ (representing the set H of modal axiom-schemes defining a modal theory), and a modal formula φ, the translator generates a first-order formula of $\mathcal{L}(MM)$ in CNF. In general, the formula obtained is non-ground; therefore, an instantiate/check method is employed to generate a sequence of ground instances of the problem. The check phase is exploited by T-ME, using the MM-decider, to decide the MM-satisfiability of each generated ground instance until a MM-unsatisfiable set of clauses is found (if any).

5 Future work and open problems

The results presented in this paper suggest several open problems and research directions. For example the relationship between T-resolution and other important theory-reasoning methods (e.g., theory connection calculus or theory consolution) and different approaches to theorem proving such as $PTTP$, which have not been addressed here.

Refinements expressly designed for T-resolution offer interesting research starting points; they involve T-validity freeness, T-subsumption, T-factoring, etc.. Further efforts should be done in controlling loading operations, following the ideas outlined in Section 2.4, and exploring their compatibility with such techniques and linearity.

As mentioned at the end of Section 1.3, the problem of lifting T-derivations is still not settled in a satisfactory manner: given a theory, every extension $\mathcal{L}^\star$ of $\mathcal{L}(T)$ must be considered in order to guarantee a (rather standard) lifting of T-derivations. A most important problem in this area is the reach of deductive methods based on T-resolution allowing the statement of (uniform) conditions on the theory T to guarantee the lifting. Other important classes of open problems arise from attempts to lift all the presented results to the non-ground case; consider, for example, the problem of maintaining T-validity properties .

Section 3.3 describes a first attempt at implementing and testing a theorem prover based on T-resolution. Further experimentation with T-resolution should include the design of a new implementation of T-ME taking advantage from the experience previously gained, in particular regarding heuristics on the treatment of T-validity, lemmas, and loading. Hence, in the context of this task we have in mind two goals among others:

- realization of a foreground reasoner using a more efficient programming language (e.g., for instance, C++) ensuring a greater and easier integrability with other languages; this is strictly connected with the following point:
- integration of the main foreground reasoner with background reasoners already available, and design of a greater number of T-deciders for different theories in order to build a larger set of scenarios for testing T-resolution.

Acknowledgments

The authors would like to thank Martin Davis, Agostino Dovier, and Eugenio Omodeo for suggestions and enlightening discussions on the topics presented in this paper, and the anonymous referees for their suggestions.

This work is partially supported by the CNR of Italy, coordinated project SETA, and by MURST 40%, "Tecniche speciali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di programmi".

References

[Bau92a] P. Baumgartner. A Model Elimination Calculus with Built-in Theories. In H. J. Ohlbach, editor, *Proceedings of the 16th German Conference on Artificial Intelligence*. Springer-Verlag LNAI, 671, 1992.

[Bau92b] P. Baumgartner. An Ordered Theory Resolution Calculus. In *Proceedings of the Conference on Logic Programming and Artificial Intelligence*. Springer-Verlag LNAI, 624, 1992.

[BFP92] P. Baumgartner, U. Furbach, and U. Petermann. A Unified Approach to Theory Reasoning. Technical Report 15/92, University of Koblenz, 1992.

[BDMP96] J. F. A. K. van Benthem, G. D'Agostino, A. Montanari, and A. Policriti. Modal Deduction in Second-Order Logic and Set Theory-I. *Journal of Logic and Computation*, 7(2):251–265, 1996.

[BDMP97] J. F. A. K. van Benthem, G. D'Agostino, A. Montanari, and A. Policriti. Modal Deduction in Second-Order Logic and Set Theory-II. *Notre Dame Journal of Formal Logic*, to appear, 1997.

[Ben85] J. F. A. K. van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, Napoli and Atlantic Heights (N.J.), 1985.

[CFO89] D. Cantone, A. Ferro, and E. G. Omodeo. *Computable Set Theory, Vol. 1*. International Series of Monographs on Computer Science. Clarendon Press, Oxford, 1989.

[Car77] L. Carroll. *Lewis Carroll's Symbolic Logic*. C. N. Potter, New York, 1977.

[CL73] C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[DFP97] A. Dovier, A. Formisano, and A. Policriti. *On T-Logic Programming*. in J.Maluszynski ed. *Proceedings of the International Logic Programming Symposium 1997. ILPS97*. MIT Press.

[DMP95] G.D'Agostino, A.Montanari, and A.Policriti. A Set-Theoretic Translation Method for Polimodal Logics. *Journal of Automated Reasoning*, 15, 1995.

[FLSY74] S. Fleisig, D. Loveland, A. K. Smiley III, and D. L. Yarmush. An implementation of the model elimination proof procedure. *Journal of the ACM*, 21(1), 1974.

[FO78] S. Ferro and E. G. Omodeo. An efficient validity test for formulae in extensional two-level syllogistic. *Le Matematiche*, XXXIII(1):130–137. Catania, Italy, 1978.

[FOS80] A. Ferro, E. G. Omodeo, and J. T. Schwartz. Decision Procedures for Elementary Sublanguages of Set Theory I. Multilevel Syllogistic and Some Extensions. *Communications on Pure and Applied Mathematics*, 33:599–608, 1980.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory,Languages and Computation*. Addison-Wesley Publishing Co. , Inc., Reading, Mass., 1979.

[JL86] J. Jaffar and J. L. Lassez. Constraint Logic Programming. Tech. rep., Department of Computer Science, Monash University, June 1986.

[JM94] J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *The Journal of Logic Programming 19–20* (1994), 503–581.

[KH69] R. Kowalsky and P. J. Hayes. *Semantic Trees in Automatic Theorem-Proving*. Machine Intelligence 4. Meltzer and Michie, eds., Edinburgh University Press, 1969.

[Lov78] D. W. Loveland. *Automated Theorem Proving*. North-Holland, 1978.

[MB88] R. Manthey and F. Bry. Satchmo: a theorem prover implemented in Prolog. *Lecture Notes in Computer Science*, 310:415–434, 1988. CADE 9, 9th Conference on Automated Deduction.

[Men79] E. Mendelson. *Introduction to Mathematical Logic*. D. Van Nostrand Company, New York, second edition, 1979.

[Pet90] U. Petermann. Towards a Connection Procedure with Built-in Theories. In *European Workshop on Logic in AI*. Springer-Verlag LNAI, 478, 1990.

[Pet93] U. Petermann. Completeness of the pool calculus with an open built-in theory. In G. Gottlob, A. Leitsch, and A. Mundici, editors, *Computational Logic and Proof Theory. Proceedings of the 3rd K. Gödel Colloquium*. Springer-Verlag LNCS, 713, 1993.

[PS95] A. Policriti and J. T. Schwartz. *T* Theorem Proving I. *Journal of Symbolic Computation*, 20:315–342, 1995.

[PT92] A. Policriti and P. Tetali. On the Satisfiability Problem for the Ground Case of First Order Theories. Technical Report 38, DIMACS center, 1992.

[SDDS86] J. T. Schwartz, R. K. B. Dewar, E. Dubinsky, and E. Schonberg. *Programming with Sets: An introduction to SETL*. Texts and Monographs in Computer Science. Springer-Verlag, 1986.

[Sti83] M. E. Stickel. Theory resolution: building in nonequational theories. In *Proceedings of the AAAI-83*, pages 391–397, 1983.

[Sti85] M. E. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1(4):333–355, 1985.

[Tur96] D. Turchetti. Sistemi di deduzione per logiche modali. Master's thesis, Università di Udine, Dip. di Matematica e Informatica, 1996.