The Mission

*The*orem proving by *M*odel el*i*mination with *S*ubsumpt*ion**

User's Guide

M. Kühn, B. Thomas
Institut für Informatik
Universität Koblenz
{bthomas,kuehn}@informatik.uni-koblenz.de

March 7, 1995

1 What The Mission is (and what it isn't)

The Mission is an automated theorem prover for a tableau variant of Loveland's model elimination calculus [12] as described in [6], extended by a general treatment of tableau subsumption. Besides of the usual inferences in model elimination it offers a regularity check and specialized inference rules like factorization, lemma extension and theory extension.

It was designed as an experimental device for examining the use of subsumption in connection with model elimination and so the main requirement was flexibility. Therefore (and due to the need of a sound unification) it was

*We'd like to thank P. Baumgartner for his guide and supervision of this work – and his countless suggestions for improvements!

written in ECRC's Prolog dialect ECLiPSe [1] — apart from the module for tableau retrieval which is an adapted version of [10].

It is *not* a high inference rate theorem prover, but in contrast to other provers it typically needs much less inferences to complete a proof. Nevertheless, the additional expenses of the subsumption tests are – yet – quite significant. On the other hand there is the interesting possibility of tableau weighting and selecting, which is not straightforward in other approaches.

2 Tableau model elimination

As a starting point we use a model elimination calculus that differs from the original one presented by [12]; it is described in [11] as the base for the prover SETHEO. In [4] this calculus is discussed in detail by presenting it in a consolution style [9] and comparing it to various other calculi. This model elimination manipulates trees by extension- and reduction-steps.

In the following we use a formal presentation of the calculus along the lines of [4]. Instead of trees we manipulate multisets of paths, where paths are sequences of literals.

A clause is a multiset of literals, usually written as the disjunction $L_1 \vee \ldots \vee L_n$. A *connection* in a set of clauses is a pair of literals, written as (K, L), which can be made complementary by application of a substitution. A *path* is a sequence of literals, written as $p = \langle L_1, \ldots, L_n \rangle$. L_n is called the *leaf* of p, which is also denoted by $leaf(p)$; similarly, the first element L_1 is also denoted by $first(p)$. '$\circ$' denotes the append function for literal sequences. Multisets of paths are written with caligraphic capital letters.

Given a set of clauses S.

- The inference rule *extension* is defined as follows:

$$\frac{\mathcal{P} \cup \{p\} \quad L \vee R}{\mathcal{R}}$$

  where all the following holds:

  1. $\mathcal{P} \cup \{p\}$ is a path multiset, and $L \vee R$ is a variable disjoint variant of a clause in S; L is a literal and R denotes the rest literals of $L \vee R$.

[1] To get ECLiPSe contact `sepia@ecrc.de`

2. $(leaf(p), L)$ is a connection with MGU σ.
3. $\mathcal{R} = (\mathcal{P} \cup \{p \circ \langle K \rangle \mid K \in R\})\sigma$.

- The inference rule *reduction* is defined as follows:

$$\frac{\mathcal{P} \cup \{p\}}{\mathcal{P}\sigma}$$

where

1. $\mathcal{P} \cup \{p\}$ is a path multiset, and
2. there is a literal L in p such that $(L, leaf(p))$ is a connection with MGU σ.

- A sequence $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ is called a *model elimination derivation* iff

  $\mathcal{P}_1$ is a path multiset $\{\langle L_1 \rangle, \ldots, \langle L_n \rangle\}$ consisting of paths of length 1, with $L_1 \vee \ldots \vee L_n$ in S (also called the *goal clause*), and

  $\mathcal{P}_{i+1}$ is obtained from $\mathcal{P}_i$ by means of an extension step with an appropriate clause C, or

  $\mathcal{P}_{i+1}$ is obtained from $\mathcal{P}_i$ by means of a reduction step.

The path p is called *selected path* in both inference rules. Finally, a *refutation* is a derivation where $\mathcal{P}_n = \{\}$.

Note that this calculus does not assume a special selection function which determines which path is to be extended or reduced next. Correctness and completeness of this calculus follows immediately from a result proved in [1].

3 Tableau subsumption

The notion of *(clause) subsumption* has been developed in the area of the resolution calculus and showed to be very useful in that context (see [14]). It is defined as a relation between two clauses in the following way: a clause C_1 is said to *subsume* a clause C_2 *if and only if* there is a substitution σ such that $C_1\sigma \subseteq C_2$ [2].

The main idea is that clauses that are subsumed by another do not have to be considered in a refutation, i.e. the search space can be considerably pruned.

[2] where clauses are defined as multisets of literals

The question we were interested in was: how can we define a corresponding relation not for clauses but for tableaux in model elimination and what search space pruning can we achieve this way?

It soon becomes obvious that it is not sufficient to take only the leaves of two tableaux and to compare them as you would do for clause subsumption, e.g. to check if there is a substitution σ, such that $\{A, \neg D\}\ \sigma \subseteq \{A, \neg D, \neg C\}$ (Fig. 1). The reason for this is that the tableaux in model elimination do not only represent derived clauses, but also the history of their derivation, which is encoded in the *ancestor context* of the leaves. If the ancestor context is ignored for tableau subsumption, completeness of the model elimination will be lost.

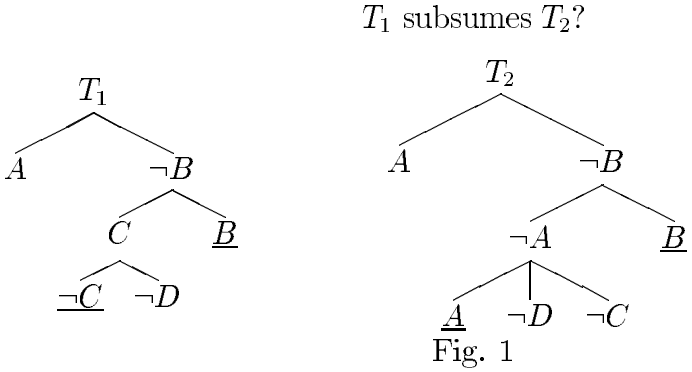

Fig. 1

But if we have to consider all the ancestors of the tableaux, the case that *tableau subsumption* indeed holds will be rare. Our solution is that it suffices to consider only positive ancestors which come from an input clause with at least one another positive literal. As a special case of this *positive disjunctive refinement* remember that for a horn clause theory no ancestors ever have to be taken into consideration as they are all negative (for a further discussion see [3]).

The only literal in our example that fulfills this requirement is C in T_1. That means, the ancestor context that must be considered is $\{C\}$ in T_1. The ancestor context of T_2 is empty, as there are no positive disjunctive literals at all.

Now, how is this ancestor context incorporated into our concept of tableau subsumption? First let us define our concept of *path subsumption*: a path P_1 with leaf L_1 and the set of positive disjunctive ancestors A_1 *subsumes* a path P_2 with leaf L_2 and the positive disjunctive ancestors A_2 *if and only if* there is a substition σ, such that $L_1\sigma = L_2$ and $A_2 \subseteq A_1\sigma$. We call σ the *subsuming substitution.*

The definition of *tableau subsumption* is now straightforward: a tableau T_1 (that is, a multiset of paths) *subsumes* a tableau T_2 *if and only if* there is a substitution σ and an injection f that maps every occurence P of every path in T_1 into a occurence of a path in T_2, such that P subsumes $f(P)$ with the subsuming substitution σ.

It is important to notice the restriction introduced by using an injection between multi-sets: eg. the tableau $T_1 = \{A, A\}$ does *not* subsume the tableau $T_2 = \{A\}$ (whereas T_2 subsumes T_1).

What does this mean for our example (Fig. 1)? Tableau T_1 consists of a path with leaf A and no ancestors and one with leaf $\neg D$ and positive disjunctive ancestor context $\{C\}$. In tableau T_2 there are no positive disjunctive ancestors at all and so there are three paths with leaves A, $\neg D$, $\neg C$ and empty (positive disjunctive) ancestestor context. So for every path of T_1 there is a corresponding path of T_2 whose ancestor context is a subset of the former (as it is empty). In other words, T_1 *subsumes* T_2.

4 Search space pruning with subsumption

In a refutation the so defined tableau subsumption relation can be applied in two directions. Let $T_1, \ldots, T_n$ be the tableau enumeration in a refutation. We say that T_i *subsumes forward* T_j *if and only if* $i < j$ and T_i *subsumes backward* T_j *if and only if* $i > j$ (with $i, j \in \{1, \ldots, n\}$) (see also [13]).

So when a tableau is subsumed forward it can be omitted as it is not needed for the refutation. But what is the effect of a backward subsumption? The way we handle this is that we cancel the subsumed tableau and all fowolling tableaux which were derived from it. As this is a rather strong pruning we have to restrict our original definitions in two ways to preserve completeness:

- As we have to make sure that we don't get any backward subsumption loops, we demand that the number of paths of the backward subsuming tableau is strictly less than that of the subsumed tableau.
- Consider the case that a tableau T_1 that has subsumed forward another tableau T_2 is cancelled by backward subsumption. This is only legal if T_2 is a successor tableau of (i.e. derived from) T_1. Therefore, forward subsumption is restricted to successor tableaux.

The behaviour of the two subsumption tests can be customized by various flags (see Appendix).

5 Tableau weighting and selection

One advantage we have by explicitly enumerating the tableaux is that we have the possibility to select from the so far derived tableaux the next one which is used to derive new tableaux.

For doing this we label every derived tree with a weight which is computed by simple heuristics (e.g. such as the depth of the tableau multiplied with its width). The tableau selection now picks always a tableau with the lowest weight and replaces it in the enumeration list by the (again weighted) tableaux which can be derived from it in one step.

The idea is that tableaux which are more likely to be successful are treated first. We have implemented a few weighting functions to choose from (see Appendix).

6 Tableau Enumeration

While enumerating the tableaux in order to get a refutation there are two places tableaux are stored: one pool with tableaux that have already been visited (ie. there are already tableaux derived from them) — this pool we call in this section O — and another pool A with the tableaux that will be visited next.

So the overall tableau enumeration in THE MISSION can be described as follows:

1. Initialize O with $\emptyset$ and A with the set of the initial goal tableau.
2. While A is not the empty set:
   - 2.1. Select the lightest tableau T and remove it from A. Repeat this if T is irregular.
   - 2.2. Check if any tableau in O is subsumed backward by this tableau.
   - 2.3. While a new tableau T' can be derived from T in one step:
     - 2.3.1. If T' is the empty tableau, terminate with 'Proof found'. Otherwise:
     - 2.3.2. If T' is not subsumed forward by a tableau from A, compute the weight of the tableau T' and store it in A.
3. Otherwise terminate with 'No Proof found'.

7 Regularity

The *regularity* condition says that you never have to derive a tableau in which a path contains two identical literals. For reasons of efficiency we do a somewhat weaker check: whenever the leaf of a path is identical to one of its ancestors, this tableau is dismissed, i.e. there will be no tableaux derived from it.

Furthermore, we do not check all ancestors in a path but only the positive disjunctive ones and the ancestors upto the last positive disjunctive literal (*blockwise regularity*, [5]). This version preserves completeness.

The regularity check can be disabled by a flag (see Appendix).

8 Factorization

Factorization is an inference that tries to avoid redundant inferences in a tableau. It says that a path can be closed if the leaf occurs also a brother literal of the leaf or of one of its ancestors. If no substitution is envolved, i.e. the literals are identical , this is the only inference made with this tableau (*ground factorization*).

9 Theory extension

THE MISSION can handle special theories like e.g. equality which are linear completed [2]. The advantage of this is that these theories can be handled more efficiently than theories that are encoded as input clauses. For a detailed discussion see [1].

10 Lemma Extension

The key idea of our *lemma extension* is to expand the given input clause set by derived tableau fronts. These fronts, called lemmas, can be used in later extension steps. We perform this expansion after every inference step on a tableau.
To restrict these expansions we built in the Option to specify the maximum length of the front to be added (see appendix).

11 Concluding Remarks

As we said before, we did not expect THE MISSION to be a high rate inference theorem prover. Rather our aim was to decrease the total number of inferences needed in a proof by the usage of subsumption checks. This indeed has been the case, if you compare our results with the number of inferences made by high inference rate theorem provers (see Appendix E).

On the other hand we made the astonishing experience, that the tableau selection we had implemented alone decreased the number of inferences drastically and that the additional usage of subsumption partly undid the good results. One explanation for this is that by subsumption also tableaux that might be finished in few inference steps get eliminated.

Nevertheless, for more difficult problems the subsumption check showed to be necessary in order to find a proof at all. So the interesting question is whether it is possible to make use of tableau subsumption without pruning too many good solutions. The advantages of such a global search space pruning might be worth-while.

A How to run the THE MISSION

To start THE MISSION first run eclipse and consult the file *mission.pl*:

```
[eclipse 1]: [mission].
```

Now you can load your file describing your problem to be proven by:

```
[eclipse 2]: mission('myfile').
```

The filename must have the extension *.tme*, but you have to omit its extension when loading it.
To start the proof type:

```
[eclipse 3]: proof.
```

If you want to set some additional flags, which you have not set in the file before, you have to do it before starting the proof, in the later on described way. By consulting a new file all flags are set back to their default values, if there are not any settings given in the file (see *syntax tme-file*). To restart the proof of an already consulted file just type *proof* again.

B Options

To perform some necessary and useful functions we have implemented some flags with following options, which can be invoked by:

```
[eclipse 4]: flag(Flagname,Flagvalue).
```

To view which flags are activated use :

```
[eclipse 5]: flags.
```

flags just lists those flags which have a non-default setting.
Valid boolean values are: *true,false,on,off,yes,no*

B.0.1 Trace-Flags

trace **boolean**
Trace the proof.

pause **boolean**
Expects a key pressed to proceed the trace:
ESC=stop
RET=next tree(more detailed)
SPACE=next tree

spy **boolean**
Special trace for subsumption-activities.

monitor_storage **boolean**
Shows the momentary amount of stored trees, where the first number indicates how many trees have been created and the second number how many trees are still kept in memory.

statistic **boolean**
Shows a statistic about the inferences,subsumptions etc. made during the proof.

history **boolean**
List the found proof.

B.0.2 Answer-Flags

answer **boolean**
Display the found answer corresponding to the goal.

definite_answer **boolean**
Find a proof with only definite answers.

solutions **boolean,integer**
To get a specified number of answers use:
true,0=to get all answers
N=to get just the first N answers

B.0.3 Prove-Flags

lemma **(boolean,integer)**
To use lemmas within your proof, you can specify the maximum length of a front to be used as a lemma.
true=length is 1
false=no lemmas
N=fronts with N literals.

regularity **boolean**
Check whether the paths in the Trees are regular (no occurence of one literal twice).

ground_reduce **boolean**
Check whether there is a reduction with a ground literal, if so *cut* (omit the other possible reduction alternatives).

ground_factor **boolean**
Check whether there is a factorisation with a ground literal, if so *cut*.

reduce **boolean**
Use reduction.

factor **boolean**
Check wether there is a factorisation.

weight,(boolean,depth,width,depth_width)
Use a selection function with following options:

- false: no selection function
- depth: trees are assorted by its longest path
- width: trees are assorted by its longest front
- true=depth_width: trees are assorted by (depth * width)

in increasing order.

B.0.4 Subsumption-Flags

sub_forward **boolean,only_prec**
To restrict the forward subsumption check:
only_prec=check only the predecessors, this setting must be choosen to be complete in combination with backward subsumption.
true=check all stored trees
false=no forward subsumption

sub_backward **boolean,only_prec**
To restrict the backward subsumption check:
only_prec=check only the predecessors
true=check all stored trees
false=no backward subsumption

mark_sub_back **boolean,only_long**
If a tree is subsumed backward his successors are marked subsumed, according to the following settings:
true=all successors are marked
false=no successors are marked
only_long= only those successors with a longer front than the subsumed tree are marked.

C Syntax of the *tme*-file

Syntax	Meaning
Literals	
literal	positive literal
$\sim literal$	not(literal)
true	true
false	false
Clauses	
$C :- P_1, \ldots, P_n.$	$C \vee not(P_1) \vee \ldots \vee not(P_n)$
$C_1; \ldots; C_m :- P_1, \ldots, P_n.$	$C_1 \vee \ldots \vee C_m \vee not(P_1) \vee \ldots \vee not(P_n)$
$C_1, \ldots, C_m :- P_1, \ldots, P_n.$	same
$L_1, \ldots, L_n.$	$L_1 \vee \ldots \vee L_n$
$L_1; \ldots; L_n.$	same
$? - Q_1, \ldots, Q_n.$	$not(Q_1) \vee \ldots \vee not(Q_n)$ Query (must be given)
Theory	
$P_1, \ldots, P_m -> C.$	theory clause
$P_1, \ldots, P_m -> false.$	conclusio extension is treated as reduction
Flags	
mission_flag(Flagname,Flagvalue).	set Flag(*Flagname*) to specified Value(*Flagvalue*)
Miscellaneous	
%	comment until end of line
/*	beginning of comment
*/	end of comment
read(Complete_Filename).	include a file

Important: Literals must not contain numbers as arguments, i.e. you have to write p(one) instead of p(1)!

D Flags

Flagname	Function	Default	Options
	Trace-Flags		
trace	show trace	false	boolean
pause	pause after each trace output	true	boolean
spy	special subsumption trace output	false	boolean
monitor_storage	show actual number of stored trees	false	boolean
statistic	display statistic about subsumptions inferences etc.	true	boolean
history	display found proof	true	boolean
	Answer-Flags		
answer	display answer of query	true	boolean
definite_answer	prove only definite_answers	false	boolean
solutions	display specified numbers of answers	0	0=all solutions interactive N=first N solutions
	Prove-Flags		
lemma	use lemmas with specified length(front-length)	false	true=with length 1 N=with length N
regularity	check blockwise regularity	true	boolean
ground_reduce	ground_reduction check	true	boolean
ground_factor	ground_factorisation check	true	boolean
reduce	use reduction	true	boolean
factor	use factorisation	false	boolean
weight	weighting selection_function	true	false= no selection true=depth_width width, depth

Flagname	Function	Default	Options
	Subsumption-Flags		
sub_forward	restrict forward subsumption	only_prec	true=check of all stored trees false=no check only_prec=only predecessor trees
sub_backward	restrict backward subsumption	true	true=check of all stored trees false=no check only_prec=only predecessor trees
mark_sub_back	specify marking of backward subsumption	only_long	true=mark all successors false=just mark subsumed tree only_long=mark only longer successors

E Results

Problem	sub_back	sub_for	Inferences	Subsumptions		Time
MSC006-1	true	only_prec	1167	for:89	back:42	26.94
" (lemma)	true	only_prec	878	for:70	back:29	19.65
" (theory)	true	only_prec	1655	for:177	back:64	35.57
"	false	false	382	for:0	back:0	2.80
SYN071-1	true	only_prec	2199	for:442	back:20	19.9
" (lemma)	true	only_prec	322	for:80	back:8	2.44
" (theory)	true	only_prec	3206	for:808	back:110	16.67
"	only_prec	only_prec	1567	for:352	back:132	24.90
"	false	false	279	for:0	back:0	0.95
ALL4	true	only_prec	4635	for:1468	back:340	62.69
"	only_prec	only_prec	4716	for:1388	back:614	80.44
"	false	false	2590	for:0	back:0	8.78
EDER23	true	only_prec	230	for:0	back:32	2.15
" (lemma)	true	only_prec	66	for:0	back:20	0.65
"	only_prec	only_prec	690	for:0	back:506	41.78
"	false	false	126	for:0	back:0	0.5
EDER45	true	only_prec	2714	for:0	back:319	50.95
" (lemma)	true	only_prec	640	for:0	back:319	12.1
"	only_prec	only_prec	–	–		∞
GRP004-1	true	only_prec	38	for:0	back:1	0.62
"	false	false	38	for:0	back:0	0.22
HEN001-1	true	only_prec	8	for:1	back:0	0.02
"	only_prec	only_prec	38	for:0	back:1	0.58
SYN086-1.005*	true	only_prec	2644	for:1408	back:0	23.43
"	false	true	596	for:256	back:0	4.43
SYN087-1.005*	false	true	600	for:288	back:0	7.26

Comments
All examples are taken from the TPTP-Library and have been tested on a Sun Sparc 2.
The * marked problems are satisfiable problems. Only lemmas with one literal have been used (*unit lemmas*).

References

1. P. Baumgartner, *A Model Elimination Calculus with Built-in Theories*, in Proceedings of the 16-th German AI-Conference (GWAI-92), H.-J. Ohlbach, ed., Springer, 1992, pp. 30–42. LNAI 671.
2. ——, *Linear Completion: Combining the Linear and the Unit-Resulting Restrictions*, Research Report 9/93, University of Koblenz, 1993. (to appear in the "Journal of Automated Reasoning").
3. ——, *Refinements of Theory Model Elimination and a Variant without Contrapositives*, in 11th European Conference on Artificial Intelligence, ECAI 94, A. Cohn, ed., Wiley, 1994.
4. P. Baumgartner and U. Furbach, *Consolution as a Framework for Comparing Calculi*, Journal of Symbolic Computation, 16 (1993). Academic Press.
5. ——, *Model Elimination without Contrapositives and its Application to PTTP*, Fachbericht Informatik 12/93, Universität Koblenz, 1993.
6. ——, *Model Elimination without Contrapositives*, in 'Automated Deduction – CADE-12', A. Bundy, ed., vol. 814 of LNAI, Springer, 1994, pp. 87–101.
7. S. Brüning, *Techniques for Avoiding Redundancy in Theorem Proving Based on the Connection Method*, PhD thesis, FB Informatik, TH Darmstadt, Nov. 1994.
8. N. Dershowitz, *Termination of Rewriting*, Journal of Symbolic Computation, 3 (1987), pp. 69–116.
9. E. Eder, *Consolution and its Relation with Resolution*, in Proc. IJCAI '91, 1991.
10. P. Graf, *ACID User Manual - version 1.0*, Technical Report MPI-I-94-DRAFT, Max-Planck-Institut, Saarbrücken, Germany, June 1994.
11. R. Letz, J. Schumann, S. Bayerl, and W. Bibel, *SETHEO: A High-Performance Theorem Prover*, Journal of Automated Reasoning, 8 (1992).
12. D. Loveland, *Mechanical Theorem Proving by Model Elimination*, JACM, 15 (1968).
13. ——, *Automated Theorem Proving - A Logical Basis*, North Holland, 1978.
14. R. Overbeek and L. Wos, *Subsumption, a Sometimes Undervalued Procedure*, Technical Report MCS-P93-0789, Argonne National Laboratory, Argonne, IL., 1989.