[Rei78] R. Reiter. Deductive question-answering on relational databases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 149–176. Plenum Press, New York, 1978.

[Rob65] A. Robinson. A Machine oriented Logic Based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[Sti85] M. Stickel. Automated Deduction by Theory Resolution. *Journal of Automated Reasoning*, 4:333–356, 1985.

[Win93] G. Winskel. *The Formal Semantics of Programming Languages.* MIT press, 1993.

[BM88] R.S. Boyer and J.S. More. Integrating decision procedures into heuristic theorem provers: A case study of linear arithmetic. *Machine Intelligence*, 11:83–124, 1988.

[DG79] B. Dreben and W.D. Goldfarb. *The Decision problem - Solvable classes of quantificational formulas.* Addison-Wesley Publishing Company Inc., 1979.

[DLL62] M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[Gil60] P.C. Gilmore. A proof Method for Quantification Theory: its Justification and Realization. *IBM Journal on Research and Development*, 4:28–35, 1960.

[Giu92] F. Giunchiglia. The `GETFOL` Manual - `GETFOL` version 1. Technical Report 9204-01, DIST - University of Genova, Genoa, Italy, 1992. Forthcoming IRST-Technical Report.

[HHT91] F. Harche, J.N. Hooker, and G.L. Thompson. A Computational Study of Satisfiability Algorithms for Propositional Logic. Working Paper 1991-27, June 1991.

[Jer88] R.G. Jeroslow. Computation-Oriented Reduction of Predicate to Propositional Logic. *Decision Support System*, 4:183–197, 1988.

[Joy76] W.H. Joyner. Resolution startegies as decision procedures. *Journal of the ACM*, 23(3):398–417, 1976.

[LP90] S.J. Lee and D.A. Plaisted. Eliminating Duplication with the Hyper-Linking Strategy. Technical Report TR90-032, The University of North Carolina - Dept. of Computer Science, August 1990.

[MR] N.V. Murray and E. Rosenthal. Dissolution: Making Paths Vanish. *Journal of the ACM.* To appear.

[NO78] G. Nelson and D.C. Oppen. Simplification by Cooperating Decision Procedures. Technical Report STAN-CS-78-652, Stanford Computer Science Department, April 1978.

[Pel86] F.J. Pelletier. Seventy-Five Problems for Testing Automatic Theorem Provers. *Journal of Automated Reasoning*, 2:191–216, 1986. See also Errata Corrige in Journal of Automated Reasoning, 4:235–236, 1988.

[Pra60] D. Prawitz. An improved proof procedure. *Theoria*, 26:102–139, 1960.

[Pra65] D. Prawitz. *Natural Deduction - A proof theoretical study.* Almquist and Wiksell, Stockholm, 1965.

The number of literals of the corresponding negative normal forms is given by $L(n) = 4^n + 6 \cdot \sum_{k=0}^{n-2} 4^k$ (the number of literals of the corresponding clausal forms grows even faster). Hence the translation process alone is of exponential complexity both in time and in space. On the other hand the space required by `PTAUT` grows only polynomially with n, while the time grows as 2^n.

Future work will be focussed on:

- carrying out experimental comparisons of the effects of different unblocking functions on the performance of the procedure;
- incorporating in `FOLTAUT` the partitioning technique [AG93] in order to avoid useless duplication of parts of the formula;
- adapting the general strategy in figure 2 to effectively deal with specific theories. The extension of `FOLTAUT` to deal with equality by using `PTAUTEQ` (instead of `PTAUT`) will be the first step in such a direction.

5 Acknowledgements

We are grateful to Andrea Parodi and Fulvio Rappa who implemented `FOLTAUT` as part of their master thesis. Roberto Sebastiani is also thanked for some useful comments on the final version of the paper. Fausto Giunchiglia and Mauro Di Manzo provided invaluable support and encouragement. The Mechanized Reasoning Group is a great environment for doing research: we thank all the members not explicitly cited above. This work has been supported by the Italian National Research Council (CNR), Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo (Special Project on Information Systems and Parallel Computing).

References

[AG93] A. Armando and E. Giunchiglia. Embedding Complex Decision Procedures inside an Interactive Theorem Prover. *Annals of Mathematics and Artificial Intelligence*, 8(3–4):475–502, 1993.

[AM90] M. Abadi and Z. Manna. Nonclausal deduction in first-order temporal logic. *Journal of the ACM*, 37(2):279–317, 1990.

[And81] P.B. Andrews. Theorem Proving via General Matings. *Journal of the ACM*, 28(2):193–214, 1981.

[BB92] M. Buro and H. Kleine Büning. Report on a SAT competition. Technical Report Nr. 110, FB 17 – Mathematik/Informatik Universität Paderborn, November 1992.

[Bib82] W. Bibel. *Automated Theorem Proving.* Vieweg, Braunschweig, 1982.

$\alpha) \rightarrow \beta^+ \vee \bigvee_{i=1}^{n'} \beta_i') = F$ and v is not blocked w.r.t. $\overline{\beta} \cup \{\beta_1', \ldots, \beta_n'\}$. Since $v(\alpha \rightarrow \beta^+ \vee \bigvee_{i=1}^{n'} \beta_i') = F$, the lemma holds taking $\beta_1', \ldots, \beta_{n'}'$ as the enumeration of instances and v as the assignment.

Q.E.D.

The main result can now be easily established by exploiting the previous lemmas.

Theorem 3.3 `FOLTAUT`$(\beta(\vec{x}), True, \{\beta(\vec{x})\}) =$ `TRUE` *if and only if* $\models \exists \vec{x}.\beta(\vec{x})$.

Proof

$\Rightarrow$) It follows from theorem 3.1 and lemma 2.

$\Leftarrow$) It follows from theorem 3.2, lemma 3 and the termination property of `FOLTAUT`.

Q.E.D.

4 Final remarks and conclusions

Considering `FOLTAUT` per se as a decision procedure, there are two issues we have to discuss: applicability and effectiveness.

applicability : As already pointed out, the formulas in the Bernays-Schönfinkel class (which represent one of the widest classes known to be decidable) trivially reduce to existential formulas via skolemization. As far as real applications are concerned, a decision procedure for the Bernays-Schönfinkel class can be used for query formation and answering in relational databases (but see [Rei78, Jer88] for more details on this topic). Furthermore, we already poited out that a much wider class of formulas can be reduced to the existential fragment via the `reduce` procedure (see section 2).

effectiveness (compared with present, state-of-the-art decision procedures): It is obvious that `FOLTAUT` performance strictly relies on the performance of the propositional decider employed. With this respect it is worth recalling that `PTAUT` is a generalization of DPL to deal with non clausal formulas which retains all the optimizations of the original procedure. Recent comparative studies on different propositional deciders working on clausal formulas [LP90, HHT91, BB92] confirm that DPL is to date one of the most efficient. The generalization to directly manage any input formula provides many advantages w.r.t. the deciders requiring the input formula to be translated in some normal form (either clausal form or negative normal form). An extreme example is given by the *U-problems* presented in [Pel86]:

$$
\begin{aligned}
&U_1 : (P_1 \leftrightarrow P_1)\\
&U_2 : (P_1 \leftrightarrow (P_2 \leftrightarrow (P_1 \leftrightarrow P_2)))\\
&U_3 : (P_1 \leftrightarrow (P_2 \leftrightarrow (P_3 \leftrightarrow (P_1 \leftrightarrow (P_2 \leftrightarrow P_3)))))\\
&\cdot\\
&\cdot\\
&\cdot\\
&U_n : (P_1 \leftrightarrow (P_2 \leftrightarrow (P_3 \leftrightarrow \ldots \leftrightarrow (P_n \leftrightarrow (P_1 \leftrightarrow (P_2 \ldots \leftrightarrow P_n) \ldots)
\end{aligned}
$$

- $\boxed{\texttt{FOLTAUT}(\bigvee_{\beta' \in \overline{\beta'}} \beta', \mu^*, \overline{\beta} \cup \overline{\beta'}) = \texttt{TRUE} \text{ and } \texttt{FOLTAUT}(\beta, \neg\mu^* \wedge \alpha, \overline{\beta}) = \texttt{TRUE}}$ where $\overline{\beta'} = \texttt{unblock}(\mu, \overline{\beta})$. By the induction hypothesis, since both $\langle \bigvee_{\beta' \in \overline{\beta'}} \beta', \mu^*, \overline{\beta} \cup \overline{\beta'} \rangle \prec \langle \beta^+, \alpha, \overline{\beta} \rangle$ and $\langle \beta^+, \neg\mu^* \wedge \alpha, \overline{\beta} \rangle \prec \langle \beta^+, \alpha, \overline{\beta} \rangle$, there exists an enumeration $\beta'_1, \ldots, \beta'_{n'}$ of instances of β such that $\mu^* \rightarrow \bigvee_{i=1}^{n'} \beta'_i$ is a tautology and an enumeration $\beta''_1, \ldots, \beta''_{n''}$ of instances of β such that $(\neg\mu^* \wedge \alpha) \rightarrow \bigvee_{i=1}^{n''} \beta''_i$ is a tautology. Now we show that there exists an enumeration $\beta'''_1, \ldots, \beta'''_{n'''}$ of instances of β such that $\alpha \rightarrow \bigvee_{i=1}^{n'''} \beta'''_i$ is a tautology. Let $\beta'''_1, \ldots, \beta'''_{n'''}$ be an enumeration of instances of β, such that $\beta'''_i = \beta'_i$ for $i = 1 \ldots n'$ and $\beta'''_{n'+i} = \beta''_i$ for $i = 1 \ldots n''$. By reductio ad absurdum we want to prove that $\alpha \rightarrow \bigvee_{i=1}^{n'''} \beta'''_i$ is a tautology. Let v be an assignment that falsifies $\alpha \rightarrow \bigvee_{i=1}^{n'''} \beta'''_i$. It follows that $v(\alpha) = T$ and $v(\bigvee_{i=1}^{n'''} \beta'''_i) = F$. Now extend v to an assignment v' such that it gives a value to all the atomic formulas in μ^*. There are two cases:

  - $v'(\mu^*) = T$, then it follows that $v'(\mu^* \rightarrow \bigvee_{i=1}^{n'} \beta'_i) = F$. We get an inconsistency, because $\mu^* \rightarrow \bigvee_{i=1}^{n'} \beta'_i$ is a tautology.
  - $v'(\mu^*) = F$, then it follows that $v'((\neg\mu^* \wedge \alpha) \rightarrow \bigvee_{i=1}^{n''} \beta''_i) = F$. We get an inconsistency, because $(\neg\mu^* \wedge \alpha) \rightarrow \bigvee_{i=1}^{n''} \beta''_i$ is a tautology.

  Q.E.D.

Lemma 3 (completeness) *Let β^+ be the disjunction of the elements of a non-empty subset of $\overline{\beta}$. If* $\texttt{FOLTAUT}(\beta^+, \alpha, \overline{\beta}) = \texttt{FALSE}$ *then there exists an enumeration $\beta_1, \ldots, \beta_n$ of instances of β and an assignment v such that $v(\alpha \rightarrow (\beta^+ \vee \bigvee_{i=1}^{n} \beta_i)) = F$ and v is not blocked (w.r.t. the set $\overline{\beta} \cup \{\beta_1, \ldots, \beta_n\}$).*

Proof Assume that $\texttt{FOLTAUT}(\beta^+, \alpha, \overline{\beta}) = \texttt{FALSE}$. Under this hypothesis we must prove that there exists an enumeration $\beta_1, \ldots, \beta_n$ of instances of β and an assignment v such that $v(\alpha \rightarrow (\beta^+ \vee \bigvee_{i=1}^{n} \beta_i)) = F$ and v is not blocked (w.r.t. the set $\overline{\beta} \cup \{\beta_1, \ldots, \beta_n\}$).

The proof follows the case analysis structure of the program.

- $\boxed{\text{There is an assignment falsifying } (\alpha \rightarrow \beta^+)}$ that is not blocked w.r.t. $\overline{\beta}$. Trivial.

- $\boxed{\texttt{FOLTAUT}(\bigvee_{\beta' \in \overline{\beta'}} \beta', \mu^*, \overline{\beta} \cup \overline{\beta'}) = \texttt{FALSE}}$ where $\overline{\beta'} = \texttt{unblock}(\mu, \overline{\beta})$ and μ falsifies β^+. By the induction hypothesis, since $\langle \bigvee_{\beta' \in \overline{\beta'}} \beta', \mu^*, \overline{\beta} \cup \overline{\beta'} \rangle \prec \langle \beta^+, \alpha, \overline{\beta} \rangle$, there exists an enumeration $\beta'_1, \ldots, \beta'_{n'}$ of instances of β and an assignment v such that $v(\mu^* \rightarrow ((\bigvee_{\beta' \in \overline{\beta'}} \beta') \vee (\bigvee_{i=1}^{n'} \beta'_i))) = F$ and v is not blocked w.r.t. $\overline{\beta} \cup \overline{\beta'} \cup \{\beta'_1, \ldots, \beta'_n\}$. Let $\beta''_1, \ldots, \beta''_{n''}$ be an enumeration of the elements of $\overline{\beta'}$. Let $\beta'''_1, \ldots, \beta'''_{n'''}$ be an enumeration of instances of β, such that $\beta'''_i = \beta'_i$ for $i = 1 \ldots n'$ and $\beta'''_{n'+i} = \beta''_i$ for $i = 1 \ldots n''$. The lemma holds taking $\beta'''_1, \ldots, \beta'''_{n'''}$ as the enumeration of instances of β and v as the assignment (notice that $v(\mu^*) = T$ and hence $v(\beta^+) = F$).

- $\boxed{\texttt{FOLTAUT}(\beta^+, \neg\mu^* \wedge \alpha, \overline{\beta}) = \texttt{FALSE}}$ where $\overline{\beta'} = \texttt{unblock}(\mu, \overline{\beta})$ and μ falsifies β^+. By the induction hypothesis, since $\langle \beta^+, \neg\mu^* \wedge \alpha, \overline{\beta} \rangle \prec \langle \beta^+, \alpha, \overline{\beta} \rangle$, there exists an enumeration $\beta'_1, \ldots, \beta'_{n'}$ of instances of β and an assignment v, such that $v((\neg\mu^* \wedge$

Definition 3.3 (Unblocking function) *Let f be a function defined from assignments and sets of instances of β to sets of instances of β. We say that f is an* unblocking function *if and only if when μ is blocked w.r.t. $\overline{\beta}$ and $f(\mu, \overline{\beta}) = \overline{\beta'}$, then $Bl_\mu^{\overline{\beta}} \cap Bl_\mu^{\overline{\beta} \cup \overline{\beta'}} \subset Bl_\mu^{\overline{\beta}}$.*

Informally speaking an unblocking function reduces the number of "old" blockages, and it possibly introduces "new" ones. We say that an unblocking function f is *strong* if $Bl_\mu^{\overline{\beta} \cup \overline{\beta'}} = \emptyset$ for each μ and $\overline{\beta}$, otherwise we say that f is *weak*. A trivial strong unblocking function is the one that generates the set of all the instances of β regardless the particular μ and $\overline{\beta}$. A simple example of weak unblocking procedure can be sketched as follows. Let μ be blocked w.r.t. $\overline{\beta}$, we choose a blockage $\langle P_1, P_2, \sigma_1, \sigma_2, \beta_{i_1}, \beta i_2 \rangle \in Bl_\mu^{\overline{\beta}}$ and we define $f(\mu, \overline{\beta}) = \{\beta_{i_1}\sigma_1, \beta_{i_2}\sigma_2\}$. Let us do this for any μ and $\overline{\beta}$ (if μ is not blocked we may define $f(\mu, \overline{\beta}) = \emptyset$). It can be easily verified that f is a weak unblocking function, since $\langle P_1, P_2, \sigma_1, \sigma_2, \beta_{i_1}, \beta i_2 \rangle \notin Bl_\mu^{\overline{\beta} \cup f(\mu, \overline{\beta})}$. Notice that we do not make any commitment on the particular choice of the element in $Bl_\mu^{\overline{\beta}}$. Unblocking functions have the simple property stated in the lemma below.

Lemma 1 *Let f be an unblocking function and let μ be blocked w.r.t. $\overline{\beta}$, then*

$$f(\mu, \overline{\beta}) \not\subset \overline{\beta}.$$

Proof The proof of this lemma follows trivially from definitions 3.2 and 3.3. Q.E.D.

In the following we assume that the function `unblock` of `FOLTAUT` is a generic unblocking function. This suffices to prove that `FOLTAUT` always terminates. In fact we can prove that the arguments of both the recursive calls in the body are $\prec$-smaller than the arguments of the header of the procedure. Let μ be a falsifying assignment for $\alpha \rightarrow \beta^+$ and let μ be blocked w.r.t. $\overline{\beta}$. From lemma 1 we have that (since $\overline{\beta} \cup \overline{\beta'} \supset \overline{\beta}$)

$$\langle \bigvee_{\beta' \in \overline{\beta'}} \beta', \mu^*, \overline{\beta} \cup \overline{\beta'} \rangle \prec \langle \beta^+, \alpha, \overline{\beta} \rangle$$

and from the fact that $\mu(\alpha) = T$ (since μ falsifies $\alpha \rightarrow \beta^+$, hence $\neg\mu^* \wedge \alpha$ has "less" models than α) we have that

$$\langle \beta^+, \neg\mu^* \wedge \alpha, \overline{\beta} \rangle \prec \langle \beta^+, \alpha, \overline{\beta} \rangle.$$

Lemma 2 (correctness) *Let β^+ be the disjunction of the elements of a non-empty subset of $\overline{\beta}$. If* `FOLTAUT`$(\beta^+, \alpha, \overline{\beta})$ = `TRUE` *then there exists an enumeration $\beta_1, \ldots, \beta_n$ of instances of β such that $(\alpha \rightarrow \bigvee_{i=1}^n \beta_i)$ is a tautology.*

Proof Assume that `FOLTAUT`$(\beta^+, \alpha, \overline{\beta})$ = `TRUE`. Under this hypothesis we must prove that there exists an enumeration $\beta_1, \ldots, \beta_n$ of instances of β such that $\alpha \rightarrow \bigvee_{i=1}^n \beta_i$ is a tautology. The proof follows the case analysis structure of the program.

- There is no assignment falsifying $(\alpha \rightarrow \beta^+)$. Trivial, since (by definition) $(\alpha \rightarrow \beta^+)$ is itself a tautology.

$T, P(x) \leftarrow F, P(y) \leftarrow F\}$ is such an assignment. As already pointed out in the example following definition 3.2, $\mu = \{P(a) \leftarrow T, P(x) \leftarrow F, P(y) \leftarrow F\}$ is not blocked by $\sigma = \{x/a, y/y\}$. However μ is blocked with blocking substitution $\sigma = \{x/x, y/a\}$. The blockage relative to this case forces the generation of $\beta(a,a) = (P(a) \rightarrow (P(a) \wedge P(a)))$ and a new recursive call to the procedure (line (3>) in figure 3). Since $\beta(a,a)$ is a tautology, we have a successful termination of that branch of the search.

There are three final considerations to take into account. First of all notice that the "appealing" idea to define an assignment as blocked if it assigns opposite truth values to unifying atomic formulas does not work. To see that this is the case, notice that the falsifying assignment $\mu = \{P(a) \leftarrow T, P(x) \leftarrow F, P(y) \leftarrow F\}$ in line (2>) of figure 3 would be blocked with blocking substitution $\sigma = \{x/a, y/y\}$. In this case `unblock` would generate once more the instance $\beta(a,y)$, thereby compromising the termination of the algorithm. Secondly it is worth stressing that variants and optimizations to the basic procedure we have described can be proposed. Each modification, if suitably implemented, could make the procedure effective on particular classes of formulas. However what is relevant in this context is to notice how easily such strategical considerations can be carried out and eventually implemented in the simplified (*i.e.* high level) framework of the procedure here presented. Finally notice that `FOLTAUT` has the claimed modularity. It does not depend on the particular propositional decider used to determine the falsifying assignment. Furthermore, the extension to treat equality in a special way can be accomplished by using `PTAUTEQ` instead of `PTAUT` and suitably modifying the definition of blocked assignment.

3.3 Correctness and completeness

In this section we prove that `FOLTAUT` is correct and complete, *i.e.* that $\exists \vec{x} \beta(\vec{x})$ is valid if and only if $\texttt{FOLTAUT}(\beta, True, \{\beta\}) = \texttt{TRUE}$. The proofs are by well-founded induction w.r.t. the well-founded relation $\prec$ defined as follows:

$$\langle \beta, \alpha, \overline{\beta} \rangle \prec \langle \beta', \alpha', \overline{\beta'} \rangle \equiv (\overline{\beta} \supset \overline{\beta'} \vee (\overline{\beta'} = \overline{\beta} \wedge models(\alpha) \subset models(\alpha')))$$

where $models(\alpha)$ is the set of assignments μ such that $\mu(\alpha) = T$. The relation $\prec$ is the lexicographic product of the superset relation between the third elements and the subset relation on the models of the second elements. Since the lexicographic product of well-founded relations is well-founded (see [Win93]), in order to establish the well-foundedness of $\prec$ it suffices to prove the well-foundedness of the superset relation between sets of instances of β and the well-foundedness of the subset relation on the *models* of the second elements of $\prec$. The superset relation between sets of instances of β is well-founded since the set of instances of β is finite. The bottom element w.r.t. this relation is the set of all the instances of β. From the fact that the set of instances of β is finite, it follows that the set of atomic formulas A_β is finite and hence that the set of assignments is finite. This guarantees that the subset relation on the *models* of the second elements of $\prec$ is well-founded.

Now we precisely define the notion of unblocking function. Let μ be an assignment blocked w.r.t. $\overline{\beta}$. We define $Bl_\mu^{\overline{\beta}}$ to be the set of blockages w.r.t. μ and $\overline{\beta}$, *i.e.* the set of 6-tuples $\langle P_1, P_2, \sigma_1, \sigma_2, \beta_{i_1}, \beta_{i_2} \rangle$ satisfying clauses 1-3 in definition 3.2.

In figure 3 we show part of an execution of FOLTAUT when $\beta(x,y) = (P(a) \rightarrow (P(x) \wedge P(y)))$. For simplicity only the (recursive) calls to FOLTAUT are listed. The initial call FOLTAUT$(\beta(x,y), True, \{\beta(x,y)\}))$ corresponds to line (1>). The falsifying assignment found (line 2 in figure 2) is $\mu = \{P(a) \leftarrow T, P(x) \leftarrow F\}$. According to definition 3.2, μ is blocked with blocking substitution $\sigma = \{x/a, y/y\}$.

(1>) FOLTAUT$(\beta(x,y), True, \{\beta(x,y)\})$
  blocked assignment: $\mu_1 = \{P(a) \leftarrow T, P(x) \leftarrow F\}$
  unblocking: $\{\beta(a,y)\} = \{(P(a) \rightarrow (P(a) \wedge P(y)))\}$
  (2>) FOLTAUT$(\beta(a,y), \mu_1^*, \{\beta(x,y), \beta(a,y)\})$
    blocked assignment $\mu_2 = \{P(a) \leftarrow T, P(x) \leftarrow F, P(y) \leftarrow F\}$
    unblocking: $\{\beta(a,a)\} = \{(P(a) \rightarrow (P(a) \wedge P(a)))\}$
    (3>) FOLTAUT$(\beta(a,a), \mu_2^*, \{\beta(x,y), \beta(a,y), \beta(a,a)\})$
      No falsifying assignment
    (3<) FOLTAUT yields TRUE
    (3>) FOLTAUT$(\beta(a,y), \neg\mu_2^* \wedge \mu_1^*, \{\beta(x,y), \beta(a,y)\})$
      No falsifying assignment
    (3<) FOLTAUT yields TRUE
  (2<) FOLTAUT yields TRUE
  (2>) FOLTAUT$(\beta(x,y), \neg\mu_1^*, \{\beta(x,y)\})$
    blocked assignment: $\mu_2 = \{P(a) \leftarrow T, P(x) \leftarrow T, P(y) \leftarrow F\}$
    unblocking: $\{\beta(x,a)\} = \{(P(a) \rightarrow (P(x) \wedge P(a)))\}$
    (3>) FOLTAUT$(\beta(x,a), \mu_2^*, \{\beta(x,y), \beta(x,a)\})$
      no falsifying assignment
    (3<) FOLTAUT yields TRUE
    (3>) FOLTAUT$(\beta(x,y), \neg\mu_2^* \wedge \neg\mu_1^*, \{\beta(x,y)\})$
      No falsifying assignment
    (3<) FOLTAUT yields TRUE
  (2<) FOLTAUT yields TRUE
(1<) FOLTAUT yields TRUE

Notation: $(i>)$ represents a recursive call occurring at the i-th level of nesting and $(i<)$ specifies the returned value. Recursive calls and returned values are indented according to the level of nesting.

Figure 3: Trace of the execution with $\beta(x,y) = (P(a) \rightarrow (P(x) \wedge P(y)))$.

This prevents the execution of the terminating branch of line 6 in figure 2 and forces the execution of unblock (line 7). The instance generated by unblock is $\beta(x,y)\sigma = (P(a) \rightarrow (P(a) \wedge P(y)))$. At this point a new recursive call (corresponding to line 8 in figure 2, and to line (2>) in figure 3) is executed with arguments: $\beta^+ = (P(a) \rightarrow (P(a) \wedge P(y)))$, $\alpha = (P(a) \wedge \neg P(x))$ and $\overline{\beta} = \{(P(a) \rightarrow (P(x) \wedge P(y))), (P(a) \rightarrow (P(a) \wedge P(y)))\}$. Such a recursive call looks for a falsifying assignment for $(\alpha \rightarrow \beta^+)$. In our case, $\mu = \{P(a) \leftarrow$

1: FOLTAUT($\beta^+, \alpha, \overline{\beta}$)
2: **if** there is no assignment falsifying $(\alpha \rightarrow \beta^+)$
3: **then return** TRUE
4: **else let** μ be an assignment falsifying $(\alpha \rightarrow \beta^+)$
5: **if** μ is not blocked w.r.t. $\overline{\beta}$
6: **then return** FALSE
7: **else let** $\overline{\beta'}$ = unblock($\mu, \overline{\beta}$)
8: **return** (FOLTAUT($\bigvee_{\beta' \in \overline{\beta'}} \beta', \mu^*, \overline{\beta} \cup \overline{\beta'}$) **and**
9: FOLTAUT($\beta^+, \neg\mu^* \wedge \alpha, \overline{\beta}$))

NOTE: FOLTAUT has to be invoked with $\beta^+ = \beta(\vec{x})$, $\overline{\beta} = \{\beta(\vec{x})\}$ and $\alpha = True$.

Figure 2: FOLTAUT

strategy can be realized simply by swapping the two recursive calls). By μ^* we mean the formula obtained from the conjunction of the literals in the set

$$\{P : P \in A_\beta, \mu(P) = T\} \cup \{\neg P : P \in A_\beta, \mu(P) = F\}.$$

For example, if μ is such that $\mu(P(x)) = T$, $\mu(P(a)) = F$ and μ is undefined elsewhere, then $\mu^* = (P(x) \wedge \neg P(a))$. Let us now discuss in detail the two recursive calls.

1. First recursive call (line 8), *i.e.* FOLTAUT($\bigvee_{\beta' \in \overline{\beta'}} \beta', \mu^*, \overline{\beta} \cup \overline{\beta'}$). The first argument is given by the disjunction of the new instances generated by unblock. There is no need to include the previous disjunct (namely β^+) since it is falsified under the current assignment (encoded in μ^*). Notice that in the execution of such a recursive call, the formula to be falsified at line 2 is $\gamma = (\mu^* \rightarrow \bigvee_{\beta' \in \overline{\beta'}} \beta')$. Falsifying γ requires to make μ^* true and therefore any possible falsifying assignment for γ must be necessarily an extension of μ. For this reason, before applying a decider for discovering a falsifying assignment, $\bigvee_{\beta' \in \overline{\beta'}} \beta'$ can be simplified w.r.t. μ. However, if PTAUT is used, this simplification is automatically carried out as first step since all the literals in μ^* are TLDO in γ.

2. Second recursive call (line 9), *i.e.* FOLTAUT($\beta^+, \neg\mu^* \wedge \alpha, \overline{\beta}$). It continues the search by looking for falsifying assignments that are extensions of α, but are different from μ (this fact is encoded in the second argument, *i.e.* $\neg\mu^* \wedge \alpha$). Hence during the execution of the recursive call, the formula to be falsified (at line 2) is $\gamma = ((\neg\mu^* \wedge \alpha) \rightarrow \bigvee_{\beta' \in \overline{\beta}} \beta')$. Notice that the new generated instances (*i.e.* $\overline{\beta'}$) are not included in the third argument of the procedure. This approach avoids cluttering the latest calls with possibly useless instances. On the other hand it has the drawback that the same instance might be generated several times in different branches of the search.

By a *blockage* (w.r.t. μ and $\overline{\beta}$) we mean a 6-tuple $\langle P_1, P_2, \sigma_1, \sigma_2, \beta_{i_1}, \beta_{i_2}\rangle$ satisfying clauses 1-3 of the previous definition. Whether an assignment μ is blocked or not depends on the set of instances considered. Continuing the previous example, if $\beta(\vec{x}) = (P(a) \rightarrow (P(x) \wedge P(y)))$ and $\overline{\beta} = \{\beta(\vec{x})\}$, then the assignment $\mu = \{P(a) \leftarrow T, P(x) \leftarrow F\}$ is blocked. The same assignment is not blocked if we consider the set of instances $\overline{\beta} = \{\beta(x,y), \beta(a,y)\}$. Notice that in the first case, μ falsifies the disjunction of the instances in $\overline{\beta}$, but this is no longer true in the second case. There are further assignments that falsify the disjunction of $\beta(x,y)$ and $\beta(a,y)$, but they are blocked w.r.t. $\{\beta(x,y), \beta(a,y)\}$. For example the assignment $\mu = \{P(a) \leftarrow T, P(x) \leftarrow F, P(y) \leftarrow F\}$ is blocked w.r.t. $\{\beta(x,y), \beta(a,y)\}$ with blocking substitution $\sigma = \{x/a, y/y\}$.

The following result allows to conclude that the input formula is not valid as soon as an unblocked falsifying assignment is found.

Theorem 3.2 *If there is a truth value assignment v such that $v(\bigvee_{\beta' \in \overline{\beta}} \beta') = F$ and v is not blocked w.r.t. $\overline{\beta}$ then $\not\models \exists \vec{x} \beta(\vec{x})$.*

3.2 The algorithm

The algorithmic structure of `FOLTAUT` is showed in figure 2. The arguments of the algorithm have the following meaning: $\overline{\beta}$ is the set of instances generated so far (initially set to $\{\beta(\vec{x})\}$), β^+ is the disjunction of a non empty subset of $\overline{\beta}$ (initially set to $\beta(\vec{x})$) and α is a propositional formula embodying the search state (initially set to the sentential constant $True$). The algorithm determines whether there exists an enumeration $\beta_1, \ldots, \beta_n$ of instances of β such that $(\alpha \rightarrow \bigvee_{i=1}^{n} \beta_i)$ is a tautology (this is formally established in section 3.3). It is immediate to verify that in the initial case (*i.e.* when $\overline{\beta} = \{\beta(\vec{x})\}$, $\beta^+ = \beta(\vec{x})$ and $\alpha = True$) this amounts to determine the existence of an enumeration $\beta_1, \ldots, \beta_n$ of instances of β such that $\bigvee_{i=1}^{n} \beta_i$ is a tautology. By theorem 3.1, this guarantees that the existential closure of $\beta(\vec{x})$ is valid.

Let us analyze in detail the main steps of the procedure (see figure 2). In line 2 a truth value assignment falsifying $(\alpha \rightarrow \beta^+)$ is sought (this activity can be carried out by `PTAUT`). If such an assignment does not exist, then the algorithm halts returning a positive answer (we have determined the enumeration $\beta_1, \ldots, \beta_n$ of instances of β such that $(\alpha \rightarrow \bigvee_{i=1}^{n} \beta_i)$ is a tautology). If such an assignment (say μ) does exist, then (in line 5) `FOLTAUT` checks whether μ is blocked. If it is not blocked, the algorithm halts (in line 6) and returns a negative answer (*i.e.* it asserts that the formula is not provable). This step is justified by theorem 3.2. If it is blocked, then in general many blockages may exist relative to a given blocked assignment. The unblocking strategy (implemented by `unblock`) chooses one or more blockages and then generates new instances according to the chosen blockages (line 7). In the following examples we assume that `unblock` finds (without any commitment to a particular strategy) a blockage $\langle P_1, P_2, \sigma_1, \sigma_2, \beta_{i_1}, \beta_{i_2}\rangle$ and generates the instances $\beta_{i_1}\sigma_1$, $\beta_{i_2}\sigma_2$. This simple strategy suffices to guarantee the correctness and completeness of `FOLTAUT`. It is clear that the unblocking strategy greatly affects the performance of the procedure. However, this issue is out of the scope of this paper.

The two recursive calls of the algorithm (lines 8 and 9) implement a depth-first search for an assignment falsifying the disjunction of the instances in $\overline{\beta}$ (but a breadth-first search

Theorem 3.1 *If there is no truth value assignment μ such that $\mu(\bigvee_{\beta' \in \overline{\beta}} \beta') = F$ then $\models \exists \vec{x} \beta(\vec{x})$.*

If such an assignment does exist, then we are not entitled to conclude that the input formula is not valid, unless $\overline{\beta}$ contains the set of all the ground instances. However, Jeroslow [Jer88] provides a sufficient condition for determining whether the input formula is not valid even in such cases. Intuitively the main idea is to consider each instance $\beta_i \in \overline{\beta}$ as the representative of a set of instances, called the set of instances "directly covered" by β_i. Let us suppose that we find an assignment μ falsifying the disjunction of the instances in $\overline{\beta}$. We say that μ is "not blocked", if to each occurrence of an atomic formula in β_j (*not* necessarily in $\overline{\beta}$) we can assign the same truth value of the corresponding (in the construction trees of the two formulas) atomic formula in β_i, where $\beta_i \in \overline{\beta}$ and β_i directly covers β_j. If this is the case then we can conclude that $\exists \vec{x}.\beta(\vec{x})$ is not provable. This is ensured by the fact that the union of the set of the instances directly covered by the instances in $\overline{\beta}$, is the set of all the instances (and hence it contains the set of all the ground instances) of $\beta(\vec{x})$. In order to state precisely this result, we need to refine the notions of directly covered instance and blocked assignment. Let $\leq$ be the binary relation on the set of instances of $\beta(\vec{x})$ such that $\beta(\vec{x})\sigma_i \leq \beta(\vec{x})\sigma_j$ if and only if there exists a substitution σ_k such that $\sigma_i = \sigma_j \sigma_k$ (where composition of substitutions is defined in the usual way).

Definition 3.1 (Directly covered instance) *An instance $\beta(\vec{t_i}) \in \overline{\beta}$* directly covers *an instance $\beta(\vec{t})$ if and only if $\beta(\vec{t}) \leq \beta(\vec{t_i})$ and there is no $\beta(\vec{t_j}) \in \overline{\beta}$ (with $i \neq j$) such that $\beta(\vec{t}) \leq \beta(\vec{t_j}) \leq \beta(\vec{t_i})$.*

From definition 3.1, it is evident that (since $\beta(\vec{x}) \in \overline{\beta}$) the union of the set of instances directly covered by the instances in $\overline{\beta}$ is the set of *all* the instances. Furthermore, it is also apparent that the set of instances directly covered by an instance $\beta' \in \overline{\beta}$, strictly depends on the set of instances $\overline{\beta}$ under consideration. Supposing $\overline{\beta} = \{\beta(\vec{x})\}$, then $\beta(\vec{x})$ *directly covers* the set of all the instances. As soon as we consider a new instance $\beta(\vec{x})\sigma_0$ (*i.e.* $\overline{\beta} = \{\beta(\vec{x}), \beta(\vec{x})\sigma_0\}$), $\beta(\vec{x})$ no longer directly covers the whole set of instances, but only the instances which are not directly covered by $\beta(\vec{x})\sigma_0$. On the other hand $\beta(\vec{x})\sigma_0$ directly covers all the instances $\beta(\vec{x})\sigma_0\sigma_1$ for any substitution σ_1. As an example, consider $\beta(x,y) = (P(a) \rightarrow (P(x) \wedge P(y)))$. If we take $\overline{\beta} = \{\beta(x,y)\}$, then the set of instances directly covered by $\beta(x,y)$ is the set of all the instances, *i.e.* $\{\beta(x,y), \beta(x,a), \beta(a,y), \beta(a,a)\}$. Let now $\overline{\beta} = \{\beta(x,y), \beta(a,y)\}$ then the set of instances directly covered by $\beta(a,y)$ is $\{\beta(a,y), \beta(a,a)\}$ and $\beta(x,y)$ directly covers the set of the remaining instances (*i.e.* $\{\beta(x,y), \beta(x,a)\}$).

Definition 3.2 (Blocked assignment) *An assignment μ is* blocked *w.r.t. $\overline{\beta}$ if and only if there exist two atomic formulas P_1 and P_2 (occurring in $\beta_{i_1} \in \overline{\beta}$ and $\beta_{i_2} \in \overline{\beta}$ respectively) and two substitutions σ_1 and σ_2* (blocking substitutions) *such that:*

1. *μ is defined both on P_1 and on P_2 and $\mu(P_1) \neq \mu(P_2)$,*
2. *$P_1\sigma_1 = P_2\sigma_2$,*
3. *β_{i_j} directly covers $\beta_{i_j}\sigma_j$ $(j = 1, 2)$.*

1. a decider for a quantifier-free theory (*e.g.* PTAUT) is applied to the input formula[1] in order to determine whether it is a tautologous formula;

2. if the previous step yields a positive answer, then the procedure halts reporting that the initial formula is valid; otherwise,

3. given a falsifying assignment, the procedure searches for a set of instances whose disjunction is true on all the possible extensions of the falsifying assignment. If such a set of instances is not found then the procedure halts reporting that the initial formula is falsifiable. Otherwise a new falsifying assignment is searched and the whole process iterated.

Notice the similarities between FOLTAUT and Andrews' procedure. Restricting to formulas in negative normal form (as Andrews does), a falsifying assignment corresponds to a vertical path. The generation of new instances and the search for the next falsifying assignment correspond to amplification and to the search for a new vertical path respectively.

Before presenting FOLTAUT in details it is worth emphasizing that neither FOLTAUT nor anyone of the previously described procedures require the input formula to be in a particular normal form. Besides complexity considerations, the translation in any normal form often obscures the form of the original formula [And81, AM90] and this is particularly bad in an interactive theorem proving setting.

3 The Proof Strategy

3.1 Preliminary notions and results

Hereinafter we assume that $\exists x_1, \ldots, x_m.\beta(x_1, \ldots, x_m)$ denotes the closed formula to be proven, where $\beta(x_1, \ldots, x_m)$ is a quantifier-free formula without function symbols. We define D_β to be the set of constants in $\beta(x_1, \ldots, x_m)$ (if there are none, we pick out a newly introduced constant c). By a *substitution* $\sigma = \{x_1/t_1, \ldots, x_m/t_m\}$ we mean a map from the variables of $\beta(x_1, \ldots, x_m)$ to terms such that each t_i is either x_i or an element in D_β. By an *instance* of $\beta(x_1, \ldots, x_m)$ we mean a formula $\beta(t_1, \ldots, t_m) = \beta(x_1, \ldots, x_m)\sigma$ where $\sigma = \{x_1/t_1, \ldots, x_m/t_m\}$ (and $\beta(x_1, \ldots, x_m)\sigma$ is defined in the usual way). We say that an instance is *ground* if and only if it does not contain free variables. For the sake of brevity, we write $\beta(\vec{x})$ for $\beta(x_1, \ldots, x_m)$, $\beta(\vec{t_i})$ for $\beta(t_{i_1}, \ldots, t_{i_m})$ (*i.e.* for the instance $\beta(x_1, \ldots, x_m)\sigma$ where $\sigma = \{x_1/t_{i_1}, \ldots, x_m/t_{i_m}\}$) and $\overline{\beta} = \{\beta(\vec{t_0}), \ldots, \beta(\vec{t_n})\}$ for an arbitrary set of instances of $\beta(\vec{x})$ such that $\beta(\vec{x}) \in \overline{\beta}$.

Let A_β be the set of the atomic formulas occurring in some instance of β. By a *truth value assignment* (in brief an assignment) μ we mean a partial map from A_β to boolean values (*i.e.* T or F). Given an assignment μ and a quantifier-free formula α, we define the evaluation of α under μ (denoted by $\mu(\alpha)$) in the standard way. Notice that since μ is a partial map, $\mu(\alpha)$ can be undefined.

Given a set of instances $\overline{\beta}$, it is a trivial consequence of Herbrand's Theorem that, if the disjunction of the instances in $\overline{\beta}$ is a tautology, then the input formula $\exists \vec{x}.\beta(\vec{x})$ is a valid formula. This fact is formally stated by the following theorem.

[1] We assume that a preliminary step of skolemization has been already carried out.

2.2 Defining new proof procedures

New powerful proof procedures can be easily built by exploiting the functionalities provided by the predefined procedures. A straightforward way to achieve such a result is to combine the procedures in a sequential way as suggested by the straight arrows in figure 1. For instance, a decider for the existential fragment can be assembled by combining `phexp` with a propositional decider (*e.g.* either `PTAUT` or `PTAUTEQ`). The resulting decider can easily solve the formula (3) by applying `PTAUT` to the result of the application of `phexp` (*i.e.* the formula (4)). Extending such a procedure with an application to `reduce` (*i.e.* the proof strategy invoking `reduce`, `phexp` and `PTAUT` in sequence) we obtain a decider for the monadic calculus.

A further example is given by the combination of `reduce`, `tautren` and `PTAUT`. Formula (5) (*i.e.* problem 29 from [Pel86]) is solved by the resulting procedure: `reduce` rewrites (5) into (1), `tautren` maps (1) into (2) which is finally decided by `PTAUT`.

However, even if such procedures considerably enrich the inferential capabilities of the prover, they suffer of intrinsic limitations. In particular, even if smaller in size than the standard Herbrand's expansion, the formula generated by `phexp` may still have (in the worst case) exponential dimension in the number of existential variables occurring in the input formula. This severely limits the applicability of procedures defined in terms of `phexp`. More effective proof strategies can be synthesized by observing that Herbrand's theorem suggests an algorithmic schema characterized by the iteration of the following two distinguished phases:

- generation of sets of instances of the initial formula,
- propositional testing to establish whether the disjunction of the instances so far generated is a tautology.

The proof procedures proposed in the past differ in the degree of interplay between the two steps. Gilmore's procedure [Gil60] is a straightforward realization of the previous schema. It builds an enumeration $S_1, S_2, \ldots$ of sets of ground instances and at each step applies a tautology checker. The major drawback of the approach resides in the fact that the invocation of the tautology checker does not exploit the computation carried out in previous calls. Prawitz (in [Pra60]) is apparently the first to recognize that a significant gain in effectiveness could be achieved by making the instances generation phase be driven by the propositional analysis. A further step is then accomplished by Robinson [Rob65] who collapses the two steps into a unique and uniform inference rule (*i.e. resolution*). However resolution is inherently a *local* inference rule. The major drawback of this fact is that even trivial theorems require several applications of the rule. The latest generation of procedures [And81, Bib82, MR] have rediscovered the separation between the two steps of instance generation and propositional analysis (which - adopting Andrews' terminology - are called *amplification* and search over the set of *vertical paths* respectively).

The key characteristic of `FOLTAUT` is a neat separation between such two activities which can be naturally implemented in a modular way. Roughly speaking the procedure implements the following strategy:

propositional letters. For instance the formula

$$\begin{array}{l}(\exists x.F(x) \wedge \exists x.G(x)) \rightarrow ((\forall x.(F(x) \rightarrow H(x)) \wedge \forall x.(G(x) \rightarrow J(x))) \leftrightarrow \\ \qquad ((\exists y.G(y) \rightarrow \forall x.(F(x) \rightarrow H(x))) \wedge \\ \qquad (\exists x.F(x) \rightarrow \forall y.(G(y) \rightarrow J(y))))\end{array} \tag{1}$$

is mapped into the propositional formula

$$(A \wedge B) \rightarrow ((C \wedge D) \leftrightarrow ((B \rightarrow C) \wedge (A \rightarrow D))) \tag{2}$$

The relation between the decision problems of the input formula (say α) and of the output formula (say α') is that $\vdash \alpha'$ only if $\vdash \alpha$.

A more careful reduction to the quantifier-free fragment is performed by `phexp`. `phexp` maps an existential formula α into a quantifier-free formula α' such that $\vdash \alpha'$ if and only if $\vdash \alpha$. The formula α' is an improved version of the Herbrand's expansion of α [DG79]. An application of `phexp` to the following formula:

$$(((\mathrm{P(x)} \wedge \neg \mathrm{Q(y)}) \vee ((\mathrm{Q(a)} \vee \mathrm{R(a)}) \wedge (\neg \mathrm{Q(b)} \vee \neg \mathrm{S(b)}))) \vee ((\mathrm{F(z)} \wedge \neg \mathrm{G(z)}) \wedge \mathrm{S(v)})) \vee ((\neg \mathrm{P(c)} \vee \neg \mathrm{F(c)}) \vee \mathrm{G(c)}) \tag{3}$$

yields

$$\begin{array}{l}((((P(a) \vee P(b) \vee P(c)) \wedge (\neg Q(a) \vee \neg Q(b) \vee \neg Q(c))) \vee \\ ((Q(a) \vee R(a)) \wedge (\neg Q(b) \vee \neg S(b)))) \vee \\ (((F(a) \wedge \neg G(a)) \vee (F(b) \wedge \neg G(b)) \vee (F(c) \wedge \neg G(c))) \wedge \\ (S(a) \vee S(b) \vee S(c)))) \vee ((\neg P(c) \vee \neg F(c)) \vee G(c))\end{array} \tag{4}$$

In [AG93] it is shown that, the size of (4) is 44 times smaller than the size of the standard Herbrand's expansion of (3).

`reduce`

`reduce` tries a set of rewriting rules on the input formula aiming at rewriting it into a logically equivalent formula that can be easily turned into an existential one via skolemization. The rewriting rules employed by `reduce` are the usual rules expressing the distributivity of quantifiers through propositional connectives and the commutativity and associativity of propositional connectives. For instance, a single application of `reduce` turns the formula

$$\begin{array}{l}(\exists x.F(x) \wedge \exists x.G(x)) \rightarrow ((\forall x.(F(x) \rightarrow H(x)) \wedge \forall x.(G(x) \rightarrow J(x))) \leftrightarrow \\ \qquad (\forall x.\forall y.((F(x) \wedge G(y)) \rightarrow (H(x) \wedge J(y)))))\end{array} \tag{5}$$

into (1). `reduce` considerably enlarges the set of formulas which can be solved by using the system of deciders. In particular, any prenex first order formula

$$\forall \vec{y}_n \exists \vec{x}_n \ldots \forall \vec{y}_i \exists \vec{x}_i \ldots \forall \vec{y}_1 \exists \vec{x}_1 . \Phi$$

such that each literal in Φ contains no variables in $\vec{y}_k$ and in $\vec{x}_l$ with $k < l$, or in $\vec{x}_k$ and in $\vec{x}_l$ with $k \neq l$ can be "reduced" to an existential formula. On the basis of the previous result it is a trivial consequence that the *monadic calculus* can be reduced to the existential fragment by means of `reduce`.

application of PTAUT. PTAUT is a generalization of the Davis-Putnam-Loveland procedure (hereon DPL) [DP60, DLL62] to non clausal formulas. The core of PTAUT is a procedure capable of partially evaluating the input formula w.r.t. a partial assignment of truth-values to the atomic subformulas. A step of statistical analysis (of polynomial time complexity) collects information about the *polarity* of the subformulas and the existence of *Top-Level Disjunctive Occurrences* (TLDO) of atomic subformulas. A formula α occurs as a TLDO in β if and only if β can be rewritten into a formula either of the form $(\alpha \vee \gamma)$ or $(\neg\alpha \vee \gamma)$ by means of rules expressing the usual properties of the propositional connectives such as the associativity, commutativity and distributivity of the propositional connectives. The notion of positive (negative) subformula occurrence is inductively defined as follows: α occurs positively in α, α occurs negatively in $\neg\alpha$; α and β occur positively in $(\alpha \wedge \beta)$ and $(\alpha \vee \beta)$; α occurs negatively and β occurs positively in $(\alpha \rightarrow \beta)$; finally α and β occur both positively and negatively in $(\alpha \leftrightarrow \beta)$. A subformula α is positive (negative) in β if and only if each occurrence of α occurs positively (negatively) in β.

The statistical analysis may suggest a partial assignment μ (w.r.t. which the formula can be simplified) according to the following criteria:

- for each positive (negative) atomic formula α occurring in β, $\mu(\alpha) = F$ $(\mu(\alpha) = T)$;
- if β contains a positive (negative) TLDO of α and there are no negative (positive) TLDO of α, then $\mu(\alpha) = F$ $(\mu(\alpha) = T)$.

If μ is not completely undefined, then PTAUT simplifies the formula in input w.r.t. μ and recurs on the resulting (simplified) formula. If the input formula contains both a positive and a negative TLDO of an atomic formula the input formula is a tautology. These optimizations generalize the *Affirmative-Negative Rule* and the *Rule for the Elimination of One-Literal Clauses* of DPL. If μ is totally undefined, then an atomic formula is chosen, two partial assignments are created (one assigning T, the other F to the chosen atomic formula), the formula is partially evaluated w.r.t. such assignments and finally the procedure branches recurring on the two simplified formulas. This last step generalizes the *Splitting Rule* of DPL.

PTAUTEQ is the result of adapting PTAUT to take into account the properties of equality. The main difference is that, before a formula is simplified w.r.t. some assignment, the assignment is tested to check whether it is model of the quantifier-free theory of equality. The formula $(P(x) \wedge x = y) \rightarrow (P(y) \wedge y = x)$ can be easily inferred by a single application of PTAUTEQ.

tautren and phexp

The procedures on top of the propositional deciders (namely tautren and phexp) map the first-order formula in input into a quantifier-free formula. The mappings are such that the decision problem of the input (first-order) formula is related to the decision problem of the returned (quantifier-free) formula in a useful way. In particular, tautren atomizes equal (modulo renaming of bound variables) quantified subformulas into newly introduced

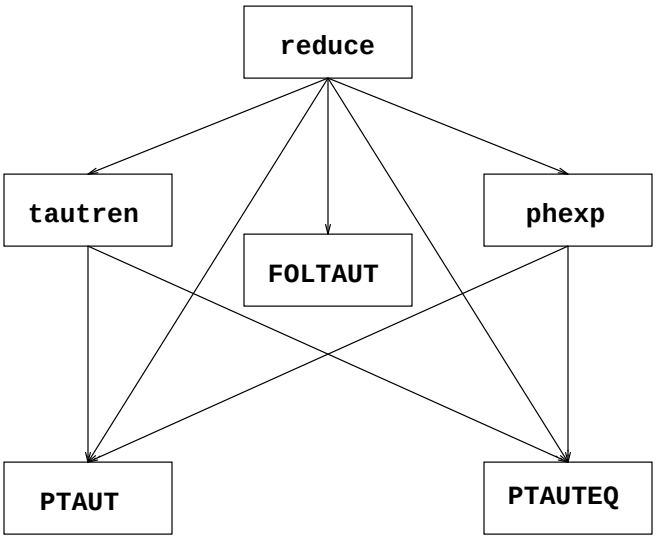

Figure 1: The system of deciders

The work presented in this paper (and in the companion paper [AG93]) is part of a wider project aiming at strengthening the inferential capabilities of interactive theorem provers with complex and reusable proof procedures. In order to give the flavor of the whole project and to make the paper self-contained, section 2 briefly surveys the set of predefined proof procedures presented in [AG93] (and implemented in the `GETFOL` system [Giu92]) and gives some hints on how complex proof procedures can be built out of them. The `FOLTAUT` procedure is presented and discussed in section 3.2, after introducing some preliminary notions and results in section 3.1. The correctness and completeness of the procedure are then proved in section 3.3. Some concluding remarks are finally given in section 4.

2 Building complex proof procedures

The set of procedures of the `GETFOL` system is depicted in figure 1. Each box represents either a decider (`PTAUT`, `PTAUTEQ`, `FOLTAUT`) or a rewriting procedure (`tautren`, `phexp`, `reduce`).

2.1 The predefined proof procedures

`PTAUT` and `PTAUTEQ`

`PTAUT` and `PTAUTEQ` are deciders working on a quantifier-free first order language (hereon by abuse of language we call them propositional deciders). `PTAUT` decides the set of first order formulas provable using only the propositional deductive machinery (moreover it returns a falsifying assignment whenever the input formula is not a tautology). For instance, the formula $(P(x) \wedge R(x,b)) \rightarrow (P(x) \vee R(x,b))$ can be easily inferred by a single

speeding up the system, but mainly as *building blocks* for facilitating the synthesis of higher level proof strategies. In fact the main feature of an interactive theorem prover is that of providing the means for expressing and executing user-defined proof strategies. However in most cases the user has to express his own proof strategies in terms of very simple inference rules (*e.g.* those provided by a standard logical calculus such as Natural Deduction [Pra65]). This often makes the task of synthesizing proof strategies of the complexity needed for real applications very difficult. Enriching the prover with a set of predefined proof procedures is a way to overcome this difficulty.

In this paper we focus on the construction of structured proof procedures built out of deciders for (decidable) quantifier-free theories. The approach has many advantages:

- Deciders for quantifier-free theories provide powerful, high level functionalities which greatly simplify the activity of designing and implementing complex and higher level proof procedures.

- Since most of the proof procedures can be decomposed in steps of propositional reasoning intermixed with steps carrying out higher level strategical functionalities (*e.g.* the expansion of (part of) the formula, the unfolding of definitions, or the choice of the inductive argument), the approach turns out to be of wide applicability.

- Uniform proof procedures - for the sake of generality - often sacrifice decidability and efficiency on important subclasses. As shown in [Joy76], resolution [Rob65] is not a decision procedure for most solvable classes. Moreover, even the refinements proposed in [Joy76], while succeeding in turning resolution into a decider for several solvable classes, are still unable to achieve the same result for the Bernays-Schönfinkel class (*i.e.* the class of prenex universal-existential formulas without function symbols).

- The modularity of the approach allows to modify and/or even substitute the decider for the quantifier-free theory (provided that the input/output behavior is preserved) without affecting the higher level procedures. This implies that any modification in the implementation can be accomplished locally, which is a desirable property from a software engineering standpoint.

As a case study we present and discuss the synthesis of a proof procedure (called `FOLTAUT`) for the existential fragment of first order logic (prenex existential formulas without function symbols) built on top of a propositional decider. Such a fragment is particularly interesting since many important theories can be expressed in it (e.g. partial order, congruence relations). Furthermore, it is one of the widest classes known to be decidable [DG79]. For instance the Bernays-Schönfinkel class can be easily reduced to it via skolemization and in [AG93] it is shown that a much wider class of formulas can be reduced to the Bernays-Schönfinkel class and hence to the existential class, through finitely many applications of truthful preserving rewriting rules. Finally, the existential fragment is a significant testbed for the analysis and development of proof procedures for first order logic. This consideration contrasts with the frequent practice of analyzing the effectiveness of proof procedures for first order logic on the propositional fragment only. In fact the complexity of the full first order case often makes impractical the task of evaluating the computational effectiveness of the procedures.

Structured proof procedures

Enrico Giunchiglia* Alessandro Armando Paolo Pecchiari

Mechanized Reasoning Group
Dipartimento di Informatica, Sistemistica e Telematica
Università di Genova, Italia
{enrico, armando, peck}@dist.unige.it

Keywords: interactive theorem proving, decision procedures

Abstract

In this paper we address the problem of enriching an interactive theorem prover with complex proof procedures. We show that the approach of building complex proof procedures out of deciders for (decidable) quantifier-free theories has many advantages: *(i)* deciders for quantifier-free theories provide powerful, high level functionalities which greatly simplify the activity of designing and implementing complex and higher level proof procedures; *(ii)* this approach is of wide applicability since most of the proof procedures are composed by steps of propositional reasoning intermixed with steps carrying out higher level strategical functionalities; *(iii)* decidability and efficiency are retained on important (decidable) subclasses, while they are often sacrificed by uniform proof strategies for the sake of generality; finally *(iv)*, from a software engineering perspective, the modularity of the procedures guarantees that any modification in the implementation can be accomplished locally. As a case study we present and discuss a proof procedure for the existential fragment of first order logic (prenex existential formulas without function symbols) built on top of a propositional decider.

1 Introduction

In this paper we address the problem of enriching an interactive theorem prover with complex proof procedures. Most of the notable works concerned with the integration of decision procedures inside a theorem proving system have been developed in the setting of completely automated theorem proving (see for example [NO78, BM88, Sti85]). All these works are concerned with building and integrating procedures specialized to deal with particular subproblems w.r.t. a given general and uniform proof strategy.

In interactive theorem proving a change of perspective is required. The heuristic strategy is not fixed and decision procedures are no longer simply regarded as tools for

*Current address: Computer Science Department, University of Texas at Austin.

Structured Proof Procedures

E. Giunchiglia, A. Armando, P. Pecchiari

July 1993
Technical Report # 93-0015

Published in *Proceedings Third Bar-Ilan Symposium on the Foundations of Artificial Intelligence*, Bar-Ilan, Israel, 1993. Also presented at *IJCAI-93 Workshop on Automated Theorem Proving*, Chambery, France, 1993.

università
di genova
facoltà di
ingegneria

dipartimento
informatica
sistemistica
telematica