Um Provador de Teoremas Multi-Estratégia
A Multi-Strategy Theorem Prover

Adolfo Gustavo Serra Seca Neto
DAINF - UTFPR
`http://www.dainf.ct.utfpr.edu.br/~adolfo`

Data desta versão: 14 de novembro de 2008
Date of this version: November 14th, 2008

Sobre

Este é um capítulo da minha tese de Doutorado intitulada "Um Provador de Teoremas Multi-Estratégia". Esta tese, na área de Ciência da Computação, foi defendida em 30 de janeiro de 2007 no Instituto de Matemática e Estatística (IME) da Universidade de São Paulo (USP). Meu orientador foi o Prof. Dr. Marcelo Finger. O texto completo desta tese está disponível em
`http://www.teses.usp.br/teses/disponiveis/45/45134/tde-04052007-175943/`

About

This is a chapter of my Ph.D. thesis entitled "A Multi-Strategy Theorem Prover". This Computer Science thesis was defended on January 30th, 2007 at the Institute of Mathematics and Statistics (IME) of the University of São Paulo (USP). My advisor was Prof. Dr. Marcelo Finger. Thesis full text is available at
`http://www.teses.usp.br/teses/disponiveis/45/45134/tde-04052007-175943/`.
Only the first chapter was written in Portuguese. All the following appendices were written in English.

Apêndice C

KEMS Design and Implementation

In this appendix we will discuss **KEMS** design and implementation. We first present tableau provers and the basic ideas behind **KEMS**, a multi-strategy tableau prover. Then we show some extensions to the **KE** methods discussed in Appendix B that were motivated by implementation issues. After that we present a brief description of the system, discussing its architecture and showing some class diagrams. Finally we briefly discuss each of the implemented **KEMS** strategies.

C.1 Tableau Provers

Theorem provers are computer programs that prove formal theorems, and *tableau provers* are theorem provers based on tableau methods (see Appendix B). Theorem provers receive a problem as input, where a *problem* [95] is a list of logic formulas (or, for signed tableau methods, a list of signed formulas) that represents a sequent.

A tableau prover output can be a 'closed' or an 'open' answer. A 'closed' answer means that the refutation of the problem sequent was successful and a closed tableau (see Section C.2.1) was obtained; an 'open' means that the sequent refutation failed and an open and completed tableau (see Section C.2.1) was obtained. Some tableau provers, besides this closed-open answer, provide a proof tree and maybe a countermodel. Let us explain this in more detail. The prover uses tableau expansion rules on problem formulas (and on formulas later generated by these rules) to construct a *proof tree* (also called *proof* *object*). If the prover cannot close a tableau branch, the search for a refutation fails, and the proof tree represents an open tableau from which we can obtain a *countermodel* for the problem. Otherwise, if the prover closes all tableau branches, the search for a refutation succeds and no countermodel can be given. The resulting *closed tableau* is a refutation for the sequent, that is, an object that explains why the sequent is not valid[1]. Therefore an 'open' answer may be accompanied by an open proof tree and a countermodel. And a 'closed' answer may be accompanied by a closed proof tree.

In other words, tableau methods can be seen as search procedures for countermodels meeting certain conditions [52]. If we use a tableau prover to search for a model in which a sequent X is false, and we produce a closed tableau, no such model exists, so X must be valid. Tableau methods can be used to generate counter-examples: if we do not produce a closed tableau, then we have a countermodel for X.

Many tableau provers for several logics were described in the literature [104]: leanTAP [4], leanKE [96], linTAP [73], LOTREC [46], and jTAP [3], among others. According to [75], "tableau and sequent calculi are the basis for most popular interactive theorem provers for hardware and software verification. Yet, when it comes to decision procedures or automatic proof search, tableaux are orders of magnitude slower than Davis-Putnam, SAT based procedures or other techniques based on resolution." But tableau provers have two advantages over the Resolution method and the DLL procedure. First, they usually do not require conversion to any normal form. Most implementations of Resolution and DLL require problem formulas to be in clausal form. Second, there are tableau systems available for several non-classical logics, while the Resolution method and the DLL procedure are deeply linked to classical logic.

C.2 KEMS—A Multi-Strategy Tableau Prover

In this thesis we investigate the construction of **KEMS**, a **KE**-based multi-strategy tableau prover. In a multi-strategy theorem prover we can vary the strategy without

[1]Many SAT provers, zChaff [53] for instance, do not give any justification when they found a problem to be unsatisfiable.

modifying the core of the implementation. Our main objective was to be able to test and compare strategies with respect to the time spent by the proof search and the size of the proof obtained. In **KEMS**, a *strategy* will be responsible, among other things, for: *(i)* choosing the next rule to be applied, *(ii)* choosing the formula on which to apply the (PB) rule, and *(iii)* verifying branch closure.

In **KEMS**, we are able to implement different strategies for the same logical system. Then we use benchmarks to compare the results obtained by these strategies. A secondary objective was to investigate if proof strategies for tableau provers could be well modularized by using object-oriented and aspect-oriented programming.

The first step towards **KEMS** construction was to study and make some modifications (discussed in [86]) on an object-oriented framework for **KE**-based provers [38]. The second step was the implementation of a single-strategy **KE**-based object-oriented prover [85]. After that, we implemented a multi-strategy **KE**-based object-oriented prover for an extended **CPL KE** system [89]. In this system, besides using object orientation, we implemented some aspects [43], a new programming construct. Finally, we extended this system to deal with **mbC** and **mCi**, two logics of formal inconsistency, and implemented strategies for the three logical systems. Here we describe **KEMS** design and implementation as well as some related issues.

C.2.1 KE Proof Search Procedure

As **KEMS** is a **KE**-based prover, we describe here the proof search procedure for this system. This procedure builds a **KE** *tableau* (also called **KE** *proof tree*) for a target sequent $A_1, A_2, \ldots, A_m \vdash B_1, B_2, \ldots, B_n$ (the sequent we want to prove or refute). A *sequent* is an expression of the form $\Gamma \vdash \Delta$, where Γ and Δ are finite sets of formulas. The symbols $\bigwedge \Gamma$ and $\bigvee \Gamma$ stand for, respectively, the conjunction and the disjunction of all formulas in Γ. That is, if $\Gamma = \{A_1, A_2, \ldots, A_n\}$, then $\bigwedge \Gamma = (A_1 \wedge (A_2 \wedge (\ldots \wedge (A_{n-1} \wedge A_n))))$ and $\bigvee \Gamma = (A_1 \vee (A_2 \vee (\ldots \vee (A_{n-1} \vee A_n))))$, keeping in mind that $\wedge$ and $\vee$ are left-associative. So, a sequent should be read as "from $\bigwedge \Gamma$ we can deduce $\bigvee \Delta$". A sequent $\Gamma \vdash \Delta$ is *valid* when the formula "$\bigwedge \Gamma \to \bigvee \Delta$" is a tautology. A sequent $\Gamma \vdash \Delta$ is *satisfiable* when "$\bigwedge \Gamma \rightarrow \bigvee \Delta$" is satisfiable.

The **KE** tableau for $A_1, A_2, \ldots, A_m \vdash B_1, B_2, \ldots, B_n$ is an ordered binary tree whose nodes contain finite *sets* of signed formulas. The proof search procedure starts by placing the following signed formulas

$$\mathtt{T}\, A_1, \mathtt{T}\, A_2, \ldots, \mathtt{T}\, A_m, \mathtt{F}\, B_1, \mathtt{F}\, B_2, \ldots, \mathtt{F}\, B_n$$

in the root node. These formulas represent the falsification of the target sequent.

The proof search proceeds by expanding the tableau. The **KE** method is an *expansion system* whose rules for **CPL** are presented in Figure B.2. An *expansion rule R of type* $\langle n \rangle$, with $n \geq 1$, is a computable relation between sets of signed formulas and n-tuples of sets of signed formulas satisfying the following condition:

$$R(S_o, (S_1, \ldots, S_n)) \Longrightarrow \text{ for every truth-set } S, \text{ if } S_0 \subseteq S, \text{ then } S_i \subseteq S \text{ for } 1 \leq i \leq n.$$

A *truth-set* or *saturated set* [29] is a set of signed formulas corresponding to **CPL** valuations. Given any **CPL** valuation v, there exists a saturated set $\mathcal{S}_v$ such that, for any formula A, if $v(A) = 1$ then $A \in \mathcal{S}_v$. For instance, if v is a valuation such that $v(A \wedge B) = 1$, then $\mathcal{S}_v = \{A, B, A \wedge B\}$ is the truth-set corresponding to v, because of **(v1)** in Definition B.1.1.

The **KE** expansion rules define what one can do, not what one must do. That is, at a given time during the construction of the tree one may have several rules that can be applied. To introduce signed formulas in a node, we can apply linear expansion rules that take as premises one or more signed formulas that already appear in that node or in some other node of the same branch. These new signed formulas are obviously logical consequences of the premises. We can always adjoin two nodes as successors of a given node, by applying the (PB) rule, which is a branching rule without premises. We only have to choose the formula to be used in (PB).

The proof search terminates when the tableau is closed or completed. A **KE** tableau is *closed* when all its branches are closed. We say that a branch is *closed* if, for some formula X, $\mathtt{T}\ X$ and $\mathtt{F}\ X$ appear in the same branch, possibly not in the same node. Otherwise it is *open*. That is, a branch is closed when we arrive at a contradiction and a tableau is closed when we arrive at a contradiction in all branches of the generated tree. If this happens, the sequent we were trying to falsify is valid. Therefore, the resulting **KE** tableau is a **KE**-*refutation* (or *proof*) of $A_1, A_2, \ldots, A_m \vdash B_1, B_2, \ldots, B_n$.

We say that a signed formula $\mathcal{S}A$ was *analyzed* in a branch θ when:

- A is an atomic formula or

- $\mathcal{S}A$ was used as the main premise in the application of some rule in θ.

A branch is *completed* when all its signed formulas have been analyzed. A **KE** tableau is *completed* when at least one of its branches is completed and open.

When a tableau is completed, the sequent we were trying to falsify is not valid. In the **KE** systems presented here, as in most (if not all) tableau systems, there is a method for obtaining a counter-model of the target sequent ($A_1, A_2, \ldots, A_m \vdash B_1, B_2, \ldots, B_n$) from a completed tableau. A counter-model for a sequent is a valuation that assigns true to all formulas in the left side and false to the formulas in the right side. By analyzing any completed open branch[2] it is possible to obtain a valuation such that for $1 \leq i \leq m$, $v(A_i) = 1$ and for $1 \leq j \leq n$, $v(B_j) = 0$.

We define the *size* of a tableau proof as the sum of the sizes of all its nodes. The size of a node is the sum of the size of all its signed formulas. Besides that, the size of a list of signed formulas is the sum of its components' sizes. The size of a signed formula $\mathcal{S}A$ is defined as the size of A. And finally, the *size* $s(A)$ of a formula A is defined as [51]:

- $s(A) = 1$ if A is a propositional atom;

- $s(\odot A) = 1 + s(A)$, where A is a formula and $\odot$ is a unary connective;

- $s(A \oslash B) = 1 + s(A) + s(B)$, where $\oslash$ is a binary connective, and A and B are formulas.

[2] A branch is a sequence of nodes that goes from the root branch to a leaf node (a node without successors).

The *height* of the proof tree and the *number of nodes* in the tree are other important dimensions for evaluating the efficiency of a proof search procedure. These are defined as usually for trees [26].

Example C.2.1. In Figures C.1 and C.2, we present two different proofs of the same sequent: the third instance of the Γ family [12] of problems (see Section D). The Γ_3 problem instance is represented by the following valid sequent:

$$(p_1 \vee q_1), (p_1 \rightarrow (p_2 \vee q_2)), (q_1 \rightarrow (p_2 \vee q_2)), (p_2 \rightarrow (p_3 \vee q_3)), (q_2 \rightarrow (p_3 \vee q_3)),$$

$$(p_3 \rightarrow (p_4 \vee q_4)), (q_3 \rightarrow (p_4 \vee q_4)) \vdash (p_4 \vee q_4)$$

In both proofs, the first step was to include the signed formulas[3] numbered 1 to 8 (representing the falsification of the sequent) in the origin. In Figure C.1, the next step was to apply all linear rules that could be applied. This generated formulas 9 to 12. Then, we had to choose the first formula to apply the (PB) rule. In this case, we would do better by choosing a formula that could be used as an auxiliary premise with one of the five formulas (1-5) that were not yet used as main premises. By choosing the left subformula of 2, the best result is a proof with size 71 and 31 nodes.

In Figure C.2, we used a different strategy. We did not apply all linear rules that could be applied (formula 8 was not expanded), generating only 9 and 10. After that, we chose the left subformula of 4 to apply the (PB) rule, and the result was a proof with size 61 and 25 nodes.

C.2.2 Extended CPL KE System

Instead of the original **CPL KE** system (see Section B.2.2), **KEMS** implements an extended **CPL KE** system, which we present here. First let us introduce four symbols to the **CPL** language (**L**): $\top$ (called 'top'), $\bot$ (named 'bottom'), $\leftrightarrow$ (called 'bi-implication') and $\oplus$ (named 'exclusive or'). Σ^s will denote the signature obtained by the addition of

[3] From now on we will use the term *s-formula* to refer to signed formulas.

$$
\begin{array}{lc}
1 & \mathtt{T}\, p_1 \lor q_1 \\
2 & \mathtt{T}\, p_1 \to (p_2 \lor q_2) \\
3 & \mathtt{T}\, q_1 \to (p_2 \lor q_2) \\
4 & \mathtt{T}\, p_2 \to (p_3 \lor q_3) \\
5 & \mathtt{T}\, q_2 \to (p_3 \lor q_3) \\
6 & \mathtt{T}\, p_3 \to (p_4 \lor q_4) \\
7 & \mathtt{T}\, q_3 \to (p_4 \lor q_4) \\
8 & \mathtt{F}\, p_4 \lor q_4 \\ \hline
9 & \mathtt{F}\, p_4 \\
10 & \mathtt{F}\, q_4 \\
11 & \mathtt{F}\, p_3 \\
12 & \mathtt{F}\, q_3
\end{array}
$$

$$
\begin{array}{lc@{\qquad\qquad\qquad\qquad}lc}
13 & \mathtt{T}\, p_1 & 14 & \mathtt{F}\, p_1 \\
15 & \mathtt{T}\, p_2 \lor q_2 & 23 & \mathtt{T}\, q_1 \\
 & & 24 & \mathtt{T}\, p_2 \lor q_2
\end{array}
$$

$$
\begin{array}{lc@{\qquad}lc@{\qquad}lc@{\qquad}lc}
16 & \mathtt{T}\, p_2 & 17 & \mathtt{F}\, p_2 & 25 & \mathtt{T}\, p_2 & 26 & \mathtt{F}\, p_2 \\
18 & \mathtt{T}\, p_3 \lor q_3 & 20 & \mathtt{T}\, q_2 & 27 & \mathtt{T}\, p_3 \lor q_3 & 29 & \mathtt{T}\, q_2 \\
19 & \mathtt{T}\, q_3 & 21 & \mathtt{T}\, p_3 \lor q_3 & 28 & \mathtt{T}\, q_3 & 30 & \mathtt{T}\, p_3 \lor q_3 \\
 & \times & 22 & \mathtt{T}\, q_3 & & \times & 31 & \mathtt{T}\, q_3 \\
 & & & \times & & & & \times
\end{array}
$$

Figure C.1: A **CPL KE** proof of Γ_3.

$$
\begin{array}{ll}
1 & \mathtt{T}\, p_1 \vee q_1 \\
2 & \mathtt{T}\, p_1 \to (p_2 \vee q_2) \\
3 & \mathtt{T}\, q_1 \to (p_2 \vee q_2) \\
4 & \mathtt{T}\, p_2 \to (p_3 \vee q_3) \\
5 & \mathtt{T}\, q_2 \to (p_3 \vee q_3) \\
6 & \mathtt{T}\, p_3 \to (p_4 \vee q_4) \\
7 & \mathtt{T}\, q_3 \to (p_4 \vee q_4) \\
8 & \mathtt{F}\, p_4 \vee q_4 \\
\hline
9 & \mathtt{F}\, p_3 \\
10 & \mathtt{F}\, q_3
\end{array}
$$

11	$\mathtt{T}\, p_2$					12	$\mathtt{F}\, p_2$		
13	$\mathtt{T}\, p_3 \vee q_3$								
14	$\mathtt{T}\, q_3$								
	$\times$	15	$\mathtt{T}\, q_2$			16	$\mathtt{F}\, q_2$		
		17	$\mathtt{T}\, p_3 \vee q_3$						
		18	$\mathtt{T}\, q_3$						
			$\times$	19	$\mathtt{T}\, p_1$			20	$\mathtt{F}\, p_1$
				21	$\mathtt{T}\, p_2 \vee q_2$			23	$\mathtt{T}\, q_1$
				22	$\mathtt{T}\, p_2$			24	$\mathtt{T}\, p_2 \vee q_2$
					$\times$			25	$\mathtt{T}\, p_2$
									$\times$

Figure C.2: A smaller **CPL KE** proof of Γ_3.

these two zeroary ($\top$ and $\bot$) and two binary ($\leftrightarrow$ and $\oplus$) connectives to the original **CPL** signature (Σ), and For^s will denote the algebra of formulas for this signature.

The following axioms have to be added to **CPL** axiomatization (see Section B.1.1) to deal with the new connectives:

(Ax12) $(A \leftrightarrow B) \to ((A \wedge B) \vee ((\neg A) \wedge (\neg B)))$;

(Ax13) $(A \oplus B) \to ((A \wedge (\neg B)) \vee ((\neg A) \wedge B))$;

(Ax14) $\bot \to A$;

(Ax15) $A \to \top$.

Then we extend **CPL**-valuations (see Definition B.1.1) by adding the following clauses:

(v5) $v(A \leftrightarrow B) = 1$ if and only if $v(A) = v(B)$;

(v6) $v(A \oplus B) = 1$ if and only if $v(A) = 1$ and $v(B) = 0$, or $v(A) = 0$ and $v(B) = 1$.

(v7) $v(\top) = 1$;

(v8) $v(\bot) = 0$.

Finally, we have to add the rules shown in Figure C.3, Figure C.4 and Figure C.5 to the original set of **CPL KE** rules (see Figure B.2). We call **e-CPL-KE** this extended **CPL KE** system .

$$\frac{}{\mathtt{T}\,\top}\ (\mathtt{T}\,\top) \qquad \frac{}{\mathtt{F}\,\bot}\ (\mathtt{F}\,\bot)$$

Figure C.3: ‘Top’ and ‘bottom’ **KE** rules.

C.2.3 Simplification Rules

To obtain a more efficient system, we can add a set of simplification rules [75] to the extended **CPL KE** system. These simplification inference rules do not cause branching and in some cases may even prevent it. They play for tableau methods the same role of unit propagation for DLL and subsumption for resolution. We adapt Massacci’s simplification

$$\frac{\begin{array}{c}\mathtt{T}\,A \leftrightarrow B\\ \mathtt{T}\,A\end{array}}{\mathtt{T}\,B}\;(\mathtt{T} \leftrightarrow_1) \qquad \frac{\begin{array}{c}\mathtt{T}\,A \leftrightarrow B\\ \mathtt{T}\,B\end{array}}{\mathtt{T}\,A}\;(\mathtt{T} \leftrightarrow_2)$$

$$\frac{\begin{array}{c}\mathtt{T}\,A \leftrightarrow B\\ \mathtt{F}\,A\end{array}}{\mathtt{F}\,B}\;(\mathtt{T} \leftrightarrow_3) \qquad \frac{\begin{array}{c}\mathtt{T}\,A \leftrightarrow B\\ \mathtt{F}\,B\end{array}}{\mathtt{F}\,A}\;(\mathtt{T} \leftrightarrow_4)$$

$$\frac{\begin{array}{c}\mathtt{F}\,A \leftrightarrow B\\ \mathtt{T}\,A\end{array}}{\mathtt{F}\,B}\;(\mathtt{F} \leftrightarrow_1) \qquad \frac{\begin{array}{c}\mathtt{F}\,A \leftrightarrow B\\ \mathtt{T}\,B\end{array}}{\mathtt{F}\,A}\;(\mathtt{F} \leftrightarrow_2)$$

$$\frac{\begin{array}{c}\mathtt{F}\,A \leftrightarrow B\\ \mathtt{F}\,A\end{array}}{\mathtt{T}\,B}\;(\mathtt{F} \leftrightarrow_3) \qquad \frac{\begin{array}{c}\mathtt{F}\,A \leftrightarrow B\\ \mathtt{F}\,B\end{array}}{\mathtt{T}\,A}\;(\mathtt{F} \leftrightarrow_4)$$

Figure C.4: ‘Bi-implication’ **KE** rules.

$$\frac{\begin{array}{c}\mathtt{T}\,A \oplus B\\ \mathtt{T}\,A\end{array}}{\mathtt{F}\,B}\;(\mathtt{T}\oplus_1) \qquad \frac{\begin{array}{c}\mathtt{T}\,A \oplus B\\ \mathtt{T}\,B\end{array}}{\mathtt{F}\,A}\;(\mathtt{T}\oplus_2)$$

$$\frac{\begin{array}{c}\mathtt{T}\,A \oplus B\\ \mathtt{F}\,A\end{array}}{\mathtt{T}\,B}\;(\mathtt{T}\oplus_3) \qquad \frac{\begin{array}{c}\mathtt{T}\,A \oplus B\\ \mathtt{F}\,B\end{array}}{\mathtt{T}\,A}\;(\mathtt{T}\oplus_4)$$

$$\frac{\begin{array}{c}\mathtt{F}\,A \oplus B\\ \mathtt{T}\,A\end{array}}{\mathtt{T}\,B}\;(\mathtt{F}\oplus_1) \qquad \frac{\begin{array}{c}\mathtt{F}\,A \oplus B\\ \mathtt{T}\,B\end{array}}{\mathtt{T}\,A}\;(\mathtt{F}\oplus_2)$$

$$\frac{\begin{array}{c}\mathtt{F}\,A \oplus B\\ \mathtt{F}\,A\end{array}}{\mathtt{F}\,B}\;(\mathtt{F}\oplus_3) \qquad \frac{\begin{array}{c}\mathtt{F}\,A \oplus B\\ \mathtt{F}\,B\end{array}}{\mathtt{F}\,A}\;(\mathtt{F}\oplus_4)$$

Figure C.5: ‘Exclusive or’ **KE** rules.

rule definition in [75] and define the following schema for **KE** simplification rules:

$$\frac{\begin{array}{c}\mathcal{S}_1\ \Phi(\Theta(X))\\ \mathcal{S}_2\ X\end{array}}{\mathcal{S}_1\ \Phi(\Theta(X)/\text{simpl}(\Theta(X),\mathcal{S}_2\ X))}$$

where:

1. $\mathcal{S}_1\ \Phi(\Theta(X))$ is the major premise;

2. $\mathcal{S}_2\ X$ is the minor premise;

3. $\mathcal{S}_1\ \Phi(\Theta(X)/\text{simpl}(\Theta(X),\mathcal{S}_2\ X))$ is the conclusion

and

- $\mathcal{S}_1$ and $\mathcal{S}_2$ are signs;

- for any Z, $\Phi(Z)$ is a formula where Z appears (one or more times) as a subformula;

- $\Theta(X)$ is either $\neg X$ or $X \oslash Y$ or $Y \oslash X$, for any X and Y, where $\oslash \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$;

- $\Phi(\Theta(X)/\text{simpl}(\Theta(X),\mathcal{S}_2\ X))$ means that we substitute every occurrence of $\Theta(X)$ in $\Phi(\Theta(X))$ by $\text{simpl}(\Theta(X),\mathcal{S}_2\ X)$.

Finally, 'simpl$(\mathcal{F}_1, \mathcal{S}\ \mathcal{F}_2)$', where $\mathcal{F}_2$ if a subformula of $\mathcal{F}_1$ and $\mathcal{S}$ is a sign, is the following rewrite function:

1. $\text{simpl}(\neg X, \mathtt{T}\ X) \rightsquigarrow \bot$;

2. $\text{simpl}(\neg X, \mathtt{F}\ X) \rightsquigarrow \top$;

3. $\text{simpl}(X \wedge Y, \mathtt{T}\ X) = \text{simpl}(Y \wedge X, \mathtt{T}\ X) \rightsquigarrow Y$;

4. $\text{simpl}(X \wedge Y, \mathtt{F}\ X) = \text{simpl}(Y \wedge X, \mathtt{F}\ X) \rightsquigarrow \bot$;

5. $\text{simpl}(X \vee Y, \mathtt{T}\ X) = \text{simpl}(Y \vee X, \mathtt{T}\ X) \rightsquigarrow \top$;

6. $\text{simpl}(X \vee Y, \mathtt{F}\ X) = \text{simpl}(Y \vee X, \mathtt{F}\ X) \rightsquigarrow Y$;

7. $\mathrm{simpl}(X \to Y, \mathtt{T}\ X) \rightsquigarrow Y$;

8. $\mathrm{simpl}(X \to Y, \mathtt{F}\ Y) \rightsquigarrow \neg X$;

9. $\mathrm{simpl}(X \to Y, \mathtt{F}\ X) \rightsquigarrow \top$;

10. $\mathrm{simpl}(X \to Y, \mathtt{T}\ Y) \rightsquigarrow \top$;

11. $\mathrm{simpl}(X \leftrightarrow Y, \mathtt{T}\ X) \rightsquigarrow Y$;

12. $\mathrm{simpl}(X \leftrightarrow Y, \mathtt{T}\ Y) \rightsquigarrow X$;

13. $\mathrm{simpl}(X \leftrightarrow Y, \mathtt{F}\ X) \rightsquigarrow \neg Y$;

14. $\mathrm{simpl}(X \leftrightarrow Y, \mathtt{F}\ Y) \rightsquigarrow \neg X$;

15. $\mathrm{simpl}(X \oplus Y, \mathtt{T}\ X) \rightsquigarrow \neg Y$;

16. $\mathrm{simpl}(X \oplus Y, \mathtt{T}\ Y) \rightsquigarrow \neg X$;

17. $\mathrm{simpl}(X \oplus Y, \mathtt{F}\ X) \rightsquigarrow Y$;

18. $\mathrm{simpl}(X \oplus Y, \mathtt{F}\ Y) \rightsquigarrow X$.

For instance, below we have an application of a simplification rule:

$$\frac{\begin{array}{c}\mathtt{T}\ D \to ((A \to B) \wedge (C \to E)) \\ \mathtt{T}\ A\end{array}}{\mathtt{T}\ D \to (B \wedge (C \to E))}$$

where

- $\Theta(A) = A \to B$;
- $\Phi(Z) = D \to (Z \wedge (C \to E))$, therefore, $\Phi(\Theta(A)) = D \to ((A \to B) \wedge (C \to E))$;
- $\mathrm{simpl}(A \to B, \mathtt{T}\ A) \rightsquigarrow B$; and
- '$D \to (B \wedge (C \to E))$' is the result of substituting '$A \to B$' by 'B' in '$D \to ((A \to B) \wedge (C \to E))$'.

Given this schema for simplification rules we can define an extension of **e-CPL-KE** that includes simplification rules for every **CPL** connective. As an example we show in Figure C.6 the simplification rules for conjunction. We call this system **s-CPL-KE**.

$$\frac{\mathcal{S}_1\ \Phi(A \wedge B) \quad \mathtt{T}\ A}{\mathcal{S}_1\ \Phi(B)} \qquad \frac{\mathcal{S}_1\ \Phi(B \wedge A) \quad \mathtt{T}\ A}{\mathcal{S}_1\ \Phi(B)}$$

$$\frac{\mathcal{S}_1\ \Phi(A \wedge B) \quad \mathtt{F}\ A}{\mathcal{S}_1\ \Phi(\bot)} \qquad \frac{\mathcal{S}_1\ \Phi(B \wedge A) \quad \mathtt{F}\ A}{\mathcal{S}_1\ \Phi(\bot)}$$

Figure C.6: Simplification **CPL KE** rules for the conjunction connective.

A logical system must have some properties, such as the replacement property, to accept the definition of such rules. In **mbC** and **mCi**, for instance, the replacement property is not valid [18], so it is not possible to have general simplification rules for these logical systems.

C.2.4 Extended mbC and mCi KE Systems

As we have done with **CPL**, in **KEMS** we work with extended versions of the **mbC KE** (see Section B.2.3) and **mCi KE** (see Section B.2.4) systems. For the extended versions of these systems, called respectively **e-mbC-KE** and **e-mCi-KE**, we only introduced the '$\top$' and '$\bot$' connectives; we did not include '$\leftrightarrow$' and '$\oplus$'. For this inclusion we followed the same steps presented in Section C.2.2, restricting ourselves to '$\top$' and '$\bot$' axioms, valuation clauses and rules.

Derived Rules

To achieve better performance with some problems, we can add derived rules to the two extended systems for **mbC** and **mCi**. In Figure C.7 we show some of the rules that can be derived from **C^{3}M mbC** rules (see Figure B.3) and that can be added to the extended systems.

Note that the ($\mathtt{T}\circ'$) rule was in fact presented in [18] as a **C^{3}M C$_1$** rule. The ($\mathbf{F}_{\text{formula}}$)

rule can be used to shorten proofs when we have, for instance, $\mathtt{F}\ A$ and $\mathtt{T}\ (\neg A) \to X$. Without this rule, in such a situation we must apply (PB) on $\{\mathtt{F}\ \neg A, \mathtt{T}\ \neg A\}$. From $\mathtt{F}\ \neg A$ we obtain $\mathtt{T}\ A$ and close the left branch. So we proceed in the right branch with $\mathtt{T}\ \neg A$, exactly like the ($\mathrm{F}_{\text{formula}}$) rule does without branching.

The current **KEMS** version has **mbC** and **mCi** strategies that use only ($\mathtt{T}\circ''$) and ($\mathtt{T}\neg''$) as additional rules, but strategies can be implemented to use the other two rules, or even other derived rules.

$$\frac{\begin{array}{c}\mathtt{T}\circ A\\ \mathtt{T}\, A\end{array}}{\mathtt{F}\,\neg A}\ (\mathtt{T}\circ'') \qquad \frac{\begin{array}{c}\mathtt{T}\,\neg A\\ \mathtt{T}\, A\end{array}}{\mathtt{F}\circ A}\ (\mathtt{T}\neg'')$$

$$\frac{\mathtt{F}\, A}{\mathtt{T}\,\neg A}\ (\mathrm{F}_{\text{formula}}) \qquad \frac{\begin{array}{c}\mathtt{T}\circ A\\ \mathtt{T}\, B \to A\\ \mathtt{T}\, B \to \neg A\end{array}}{\mathtt{F}\, B}\ (\mathtt{T}\circ')$$

Figure C.7: Derived **mbC KE** rules.

C.3 System Description

In this section we present a description of the implemented **KEMS** system. We start with the system architecture which is presented in Figure C.8. The digram describes that the user presents as input to **KEMS** a problem instance and a prover configuration. The prover configuration must contain values for four major **KEMS** parameters:

1. the logical system for the proof search procedure (as we see in Appendix D, some problems can be submitted to provers for different logical systems);

2. the analyzer used to lexically analyse and parse the problem;

3. the strategy chosen to search a proof for the problem;

4. the sorter chosen to be used with the strategy.

Besides that, the prover configuration can be used to give values to other four minor parameters:

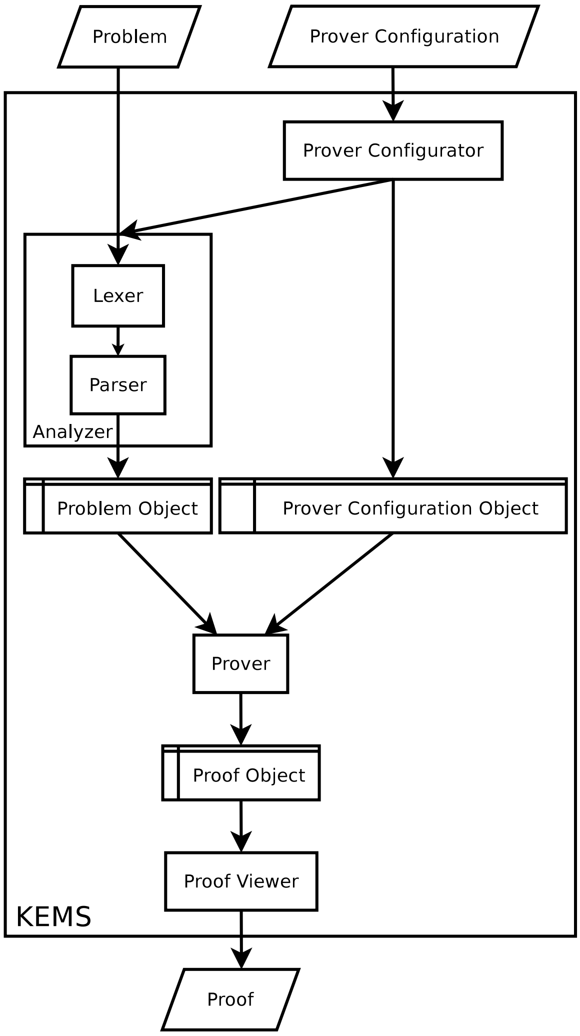

Figure C.8: System architecture.

- the number of times the prover must run the proof search procedure with the given problem[4];
- the time limit for the proof search procedure (if this time limit is exceeded, the prover is interrupted);
- a boolean option to determine if the prover must save s-formula origins or not;
- a boolean option to determine if the prover must discard closed branches or not;
- a boolean option to determine if the prover must save discarded branches in disk (to be restored after the proof procedure finishes) or not.

Given these inputs, the system outputs a proof that contains, among other things:

- the open/closed final status of the tableau;
- the tableau proof tree, that can be partial if the discard closed branches option is set to true;
- the problem size;
- the time spent by the prover while building a proof for the input problem;
- the proof size;
- a counter-model valuation, if the tableau is open.

Let us see an example. The user can present a PHP_4 instance (see Section D.1.1) with a prover configuration establishing '**mbC**' (see Section B.1.3) as the logical system, '**LFI** analyzer'[5] as the problem analyzer, '**mbC** Simple Strategy' (see Section C.4.4) as the strategy, and 'Insertion Order' (see Section C.4.2) as the sorter. Then **KEMS** uses the chosen analyzer (which contains a lexer and a parser) to build a problem object. And it builds a prover configuration object from user options. These two objects are used by the prover module to build a proof object. This proof object is given as input to the proof viewer. Finally, the proof viewer shows the proof to the user.

[4] This is useful for prover evaluation, in order to get a better estimate of the time spent to find a proof.

[5] A problem analyzer for **mbC** and **mCi** that we implemented.

C.3.1 Class Diagrams

Here we present and discuss some simplified class diagrams for **KEMS**. The first of these diagrams is depicted in Figure C.9. It describes the classes we implemented to represent formulas and signed formulas. The Formula class uses the Composite design pattern [55]: a formula can be either an AtomicFormula or a composition of AtomicFormula objects. A SignedFormula object contains references of one FormulaSign object and one Formula object.

The FormulaFactory and SignedFormulaFactory classes use the Flyweight design pattern [55]. This pattern prevents the multiplication of objects representing formulas and signed formulas as well as makes it easier to implement rule choice and application. That is, there is only one instance of each formula and each signed formula. This allows us to save space as well as serves to simplify the search for subformulas of a formula, and for signed formulas where a formula appears. Using this pattern, when we want to compare two formulas, we have only to compare pointers instead of character strings or formula structure.

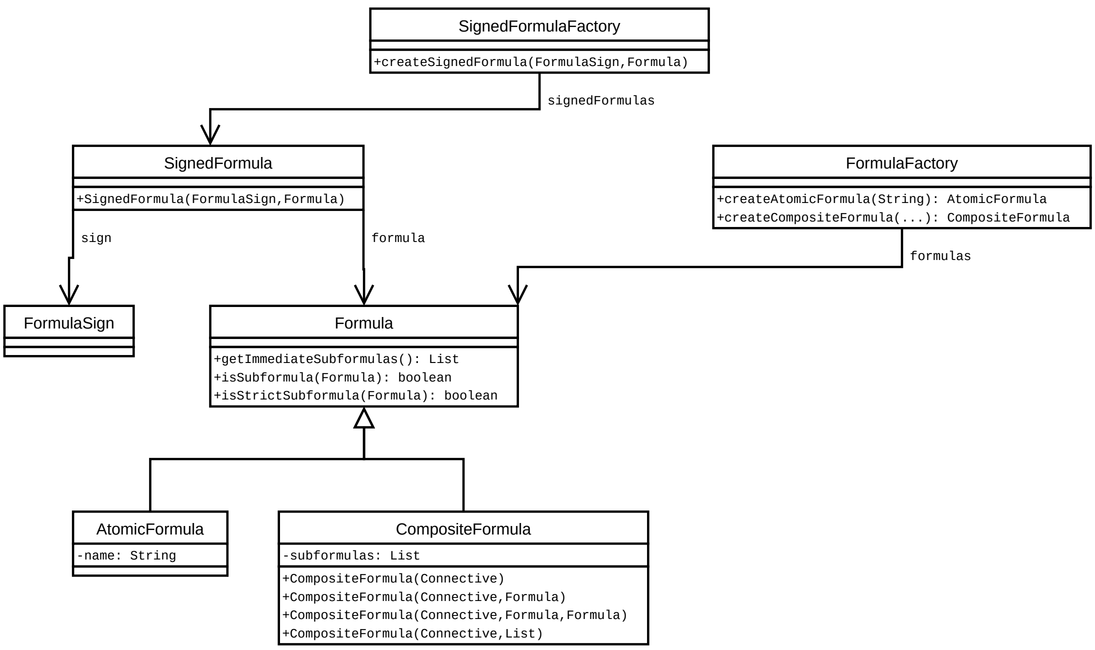

Figure C.9: Formula and Signed Formula class diagram.

The diagram in Figure C.10 shows that in the ProverFacade class we have used the

Facade design pattern [55] to provide a higher-level interface to the classes that implement prover functionality. The SignedFormulaCreator class uses instances of subclasses of Lexer and Parser classes to build a Problem object. The Prover class has a method that receives as input a Problem object and outputs a Proof object. And the ProofVerifier class has a method that gets as input a Proof object and outputs an ExtendedProof object, containing a tableau proof tree and additional information about the proof and the proof search procedure.

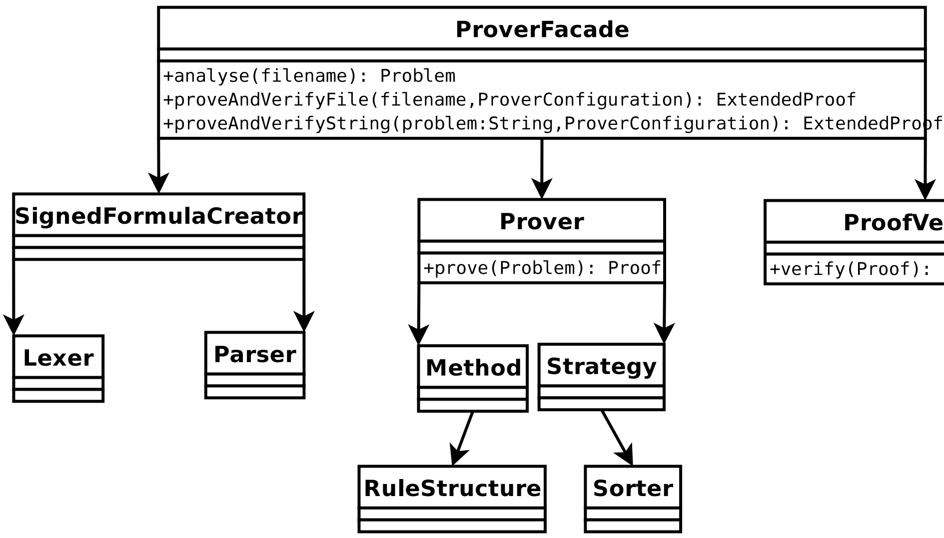

Figure C.10: Prover class diagram.

The Prover class uses the Strategy design pattern [55] to be able to make proof strategies interchangeable. In Figure C.11 we see that an IStrategy interface was defined. All strategies must implement this interface. Besides that we defined an ISimpleStrategy that defines several methods that are common to all strategies implemented in **KEMS** current version (but that future strategies need not implement). We implemented the functionality which is common to all implemented **KEMS** strategies in AbstractSimpleStrategy. This class has three subclasses: SimpleStrategy, MemorySaverStrategy and ConfigurableSimpleStrategy. These three classes define three strategies whose features we discuss in Section C.4.3. And we used SimpleStrategy as a basis for several other strategies that are discussed in Sections C.4.4, C.4.5 and also in Section C.4.3.

A subset of the classes and interfaces written to implement **KE** rules is shown in

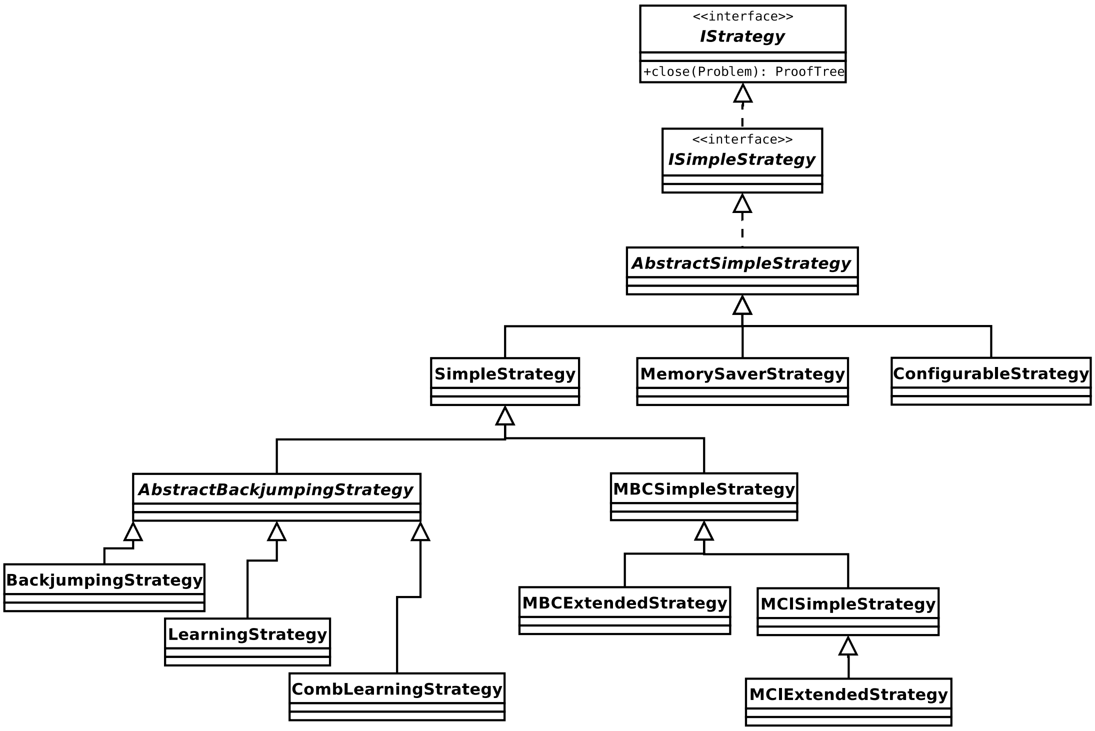

Figure C.11: Strategy class diagram.

Figure C.12. We have classes for one-premise one-conclusion rules, two-premise one-conclusion rules and one-premise two-conclusion rules. All of these classes are subclasses of an abstract class Rule that implements the IRule interface. A RuleStructure contains one or more lists of rules and there is a map that assigns a name to each RuleList object. In this way, a RuleStructure can have subsets of the chosen **KE** system rules so that a strategy can choose which subset to apply first. Besides these classes, we have several classes to implement rule premise patterns and rule conclusion actions.

C.3.2 Programming Languages Used

Current **KEMS** version is written in Java 1.5 [57] with some aspects written in AspectJ 1.5 [65, 64]. Java was chosen because it is a well established object-oriented programming (OOP) language for which there is an extension, called AspectJ, that supports a new software development paradigm: aspect-orientation. At the time of our choice, 2003, AspectJ was the best aspect-oriented language. As we had a plan to make use of aspects in our implementation, we chose to work with Java and AspectJ.

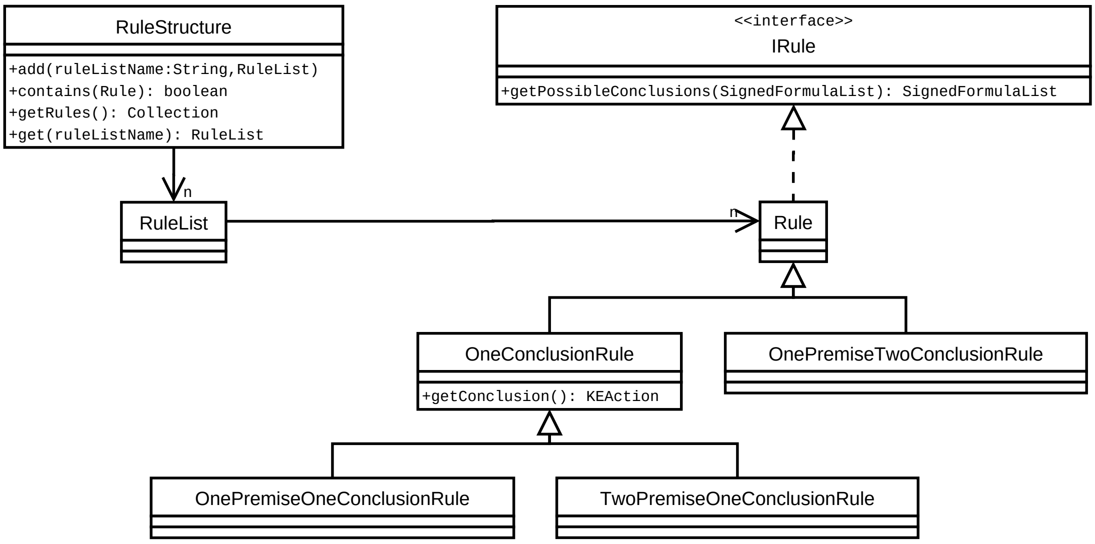

Figure C.12: Rule class diagram.

In aspect-oriented systems, classes are blueprints for the objects that represent the main concerns of a system while aspects represent concerns that are orthogonal to the main concerns and that may have impact over several classes in different places in the class hierarchy. The use of aspects, among other advantages, leads to less scattered code. That is, lines of code that implement a given feature of the system can rest in the same file. Next we present in more detail AspectJ and Aspect-oriented programming (AOP), and discuss briefly some aspects we have implemented in **KEMS**.

AspectJ

Although the object oriented paradigm is dominant nowadays, it has some limitations. For instance, in object oriented systems, code with different purposes can become scattered and tangled. Part of these limitations can be overcome with the use of design patterns [55] or traits [103]. Aspect oriented programming [43] is an attempt to solve these and other problems identified in the object oriented paradigm. It is a technique that intends to achieve more modularity in situations where object orientation and the use of design patterns is not enough.

The main motivation for the development of AOP was the alleged inability of OOP and other current programming paradigms to fully support the *separation of concerns* principle [65]. According to AOP proponents, AOP solves some problems of OOP by allowing an adequate representation of the so-called *crosscutting concerns* [107]. With AOP, code that implements crosscutting concerns, i.e. that implements specific functions that affect different parts of a system and would be scattered and tangled in an OOP implementation, can be localized, increasing modularity. With this increase in modularity, one can achieve software that is more adaptable, maintainable and evolvable in the face of changing requirements [97]. Other expected benefits of using AOP are a more readable and reusable code and a more natural mapping of system requirements to programming constructs.

To work with AOP one needs an aspect-oriented programming language or an aspect-oriented extension/framework to an existing language. AspectJ is a general purpose, seamless aspect-oriented extension to Java that enables the modular implementation of a wide range of crosscutting concerns. We have chosen to use AspectJ because it seems to be the most promising approach to aspect orientation. It adds support for aspects to the well-established Java language and is regarded as the most mature approach to AOP at the time of writing.

An AspectJ program is a Java program with some extra constructs. These included language constructs allow the implementation of AOP features such as *pointcuts*, *advice*, *inter-type declarations* and *aspects*. Pointcuts and advice dynamically affect program flow, while inter-type declarations statically affect the class hierarchy of a program. Aspects are the modularization unit in AspectJ, just as a common concern's implementation in OOP is called a class. They are units of modular crosscutting implementation, composed of pointcuts, advice, and ordinary Java member declarations [65]. Aspects are defined by aspect declarations, which have a form similar to that of class declarations.

Implemented Aspects

In **KEMS**, we have a few aspects implementing secondary functionalities:

- the Complexity aspect introduces an `int getComplexity()` method in the classes that represent formulas and their lists and factories, and signed formulas and their lists and factories. This method calculates the size of the objects of these classes;

- the Formula Parent Introduction (FPI) aspect adds data structures (as attributes) and methods to the Formula class. With these structures and methods, a given formula can hold references to its parents (i.e to all formulas of which this formula is a subformula) and to its signed counterparts (i.e to all signed formulas that have this formula). This aspect works together with the Formula Parents aspect, which intercepts Formula and Signed Formula object creations and uses FPI aspect methods to add those extra references to all formulas;

- the Memory Usage Tracker aspect inspects memory usage and when necessary forces the java virtual machine to perform a garbage collection on its heap memory;

- the Proof Tree Size aspect includes attributes and methods in the Proof Tree class to get the size, the number of branches and the number of nodes of a proof tree;

- finally, the Prover Thread aspect interrupts running problems. That is, when a problem is being runned, there is always a thread of the prover which was initiated only for this purpose. When the user asks **KEMS** to interrupt a running problem, or when a prespecified time limit is reached, the Prover Thread aspect captures the runnning thread and interrupts it.

We had planned to make more use of AOP in **KEMS**, but we find out it was not as productive as we thought it would be (in line with what was described in [109]). This happened, in part, due to the evolutionary nature of our development and to the lack of adequate tools for refactoring aspects (to aspects and to classes). Therefore we decided to use it only on secondary features. But as it was described in [90], it is still possible to use AOP to design future **KEMS** strategies.

C.4 Strategies

As we have said in Section C.2, a **KEMS** strategy is responsible, among other things, for: *(i)* choosing the next rule to be applied, *(ii)* choosing the formula on which to apply the (PB) rule, and *(iii)* verifying branch closure. In other provers, these features can be scattered in several prover modules if strategies are not designed as first-class citizens. In **KEMS**, strategies are first-class citizens. This is **KEMS** most important feature. In **KEMS**, the core of the implementation is shared by all strategies and each strategy is defined in one main class and possibly some auxiliary classes and aspects. In this way we can prove the same problem with several different strategies and compare the proofs obtained.

The idea of having several strategies implemented in the same prover and being able of varying the strategy used is not new: in [72], in the context of first-order theorem proving, this idea is clearly present. **KEMS** is a multi-strategy tableau prover for **KE** systems. As **KE** is a tableau method that is available for several logics, **KEMS** can be used to provide effective provers for many different logical systems, as well as to enable the comparison between strategies for these systems.

Let us see how some other object-oriented tableau-based provers represent strategies. The prover developed by Wagner Dias, which we call WDTP [38], was written in C++ and implements Analytic [106] and KE propositional tableaux methods. jTAP [3] is a propositional tableau prover, written in Java, based on the method of signed Analytic Tableaux. Both systems have some strategies implemented and can be extended with new ones, but strategies are not well modularized since one has to create subclasses of one or more classes of the system, as well as modify existing ones, to implement a new strategy. LOTREC [46] is a generic tableau prover for modal and description logics (MDLs). It aims at covering all logics having possible worlds semantics, in particular MDLs. It is implemented in Java. Logic connectives, tableau rules and strategies are defined in a high-level language specifically designed for this purpose. In LOTREC, strategies are described using a very simple language, not in a programming language. They are limited to establishing the order and the number of times the rules will be applied.

C.4.1 Strategy Implementation

As we have said above, in **KEMS** a strategy is implemented by writing a main class and possibly some auxiliary classes and aspects. We will describe later each strategy implemented in **KEMS**. Let us discuss now some features that are common to all strategies.

We are going to see now the basic procedure for applying rules in a node. The procedure which is the basis for all strategies is the **KE** canonical procedure described in [29]. This generic procedure establishes the following order for **KE** rule applications:

1. all one-premise rules;

2. all two-premise rules;

3. (PB) rule.

This sequence is rather obvious since (PB) rule applications are computationally more expensive.

Another important feature of **KEMS** strategies is that they choose which rules are going to be applied by analyzing signed formula structure. For instance, suppose a **CPL KE** strategy is analyzing a node that has the following list of signed formulas: [$\mathtt{T}\ A \rightarrow B$, $\mathtt{T}\ C \vee A$, $\mathtt{F}\ C$, $\mathtt{T}\, D \rightarrow E$]. By iterating over this list we first analyse '$\mathtt{T}\ A \rightarrow B$' and see that it can be used as the main premise in application of the ($\mathtt{T}\!\rightarrow_1$) and ($\mathtt{T}\!\rightarrow_2$) rules[6]. These rules tell us that the minor premise could be either '$\mathtt{T}\ A$' or '$\mathtt{F}\ B$', but as none of these signed formulas in our list, we cannot apply any of these rule.

Next the strategy analyses the '$\mathtt{T}\ C \vee A$' s-formula and discovers that it can be the major premise of ($\mathtt{T}\vee_1$) and ($\mathtt{T}\vee_2$) rule applications. These rules tells us that the minor premise could be either '$\mathtt{F}\ C$' or '$\mathtt{F}\ A$'. As '$\mathtt{F}\ C$' is in our list, we apply the ($\mathtt{T}\vee_1$) rule and include '$\mathtt{T}\ A$' in our list. Now we can apply ($\mathtt{T}\!\rightarrow_1$) rule and obtain '$\mathtt{T}\ B$'.

Some s-formulas can be the main premise of two-premise rule applications, some cannot. In **CPL KE** all s-formulas whose connective is binary can be the main premise of a two-premise rule application, but that can vary from logic to logic. When we start the proof search procedure for a problem, we create a list of *not analyzed main candidates*

[6] Because the analyzed s-formula's sign is $\mathtt{T}$ and its main connective is '$\rightarrow$'.

(namc) that contains all problem s-formulas that can be main premise. Every time we use one of these s-formulas as main premise in some rule application, we remove it from the list. If, after we have applied all possible linear rules, the tableau is still open, and the *namc* list is not empty, we can choose one of the s-formulas on the list to serve as a basis for a (PB) application. In **KEMS**, after choosing a s-formula $\mathcal{S}\ \mathcal{F}$ from the list, where $\mathcal{F}$ is something like '$A \oslash B$' and '$\oslash$' is a binary connective, all strategies apply the (PB) rule by branching on '$\mathcal{S}_1\ A$' and '$\mathcal{S}_2\ A$'[7]. And $\mathcal{S}_1$ is the sign that allows the application of a two-premise rule on the left successor node.

Continuing our example, after iterating over the list we have only one formula in our *namc* list: '$\mathtt{T}\ D \rightarrow E$'. By following the procedure described in the previous paragraph, we apply (PB) with $\{\mathtt{T}\ D,\ \mathtt{F}\ D\}$ and the result is the following:

$$\mathtt{T}\ D \rightarrow E$$

$$\begin{array}{cc} \mathtt{T}\ D & \mathtt{F}\ D \\ \mathtt{T}\ E & \end{array}$$

Therefore, from the node we were analyzing we created two new successor node. Now let us describe how strategies deal with nodes. Every strategy has to keep a stack of open nodes. When a strategy starts the proof search procedure for a problem, the root branch, containing the s-formulas of the problem, is put on the top of this stack. Only one node is being expanded at a given time. We call this node the *current* node.

The procedure for dealing with nodes is as follows:

1. Remove the node which is on the top of the open node stack and make it the current node. If the stack is empty, finish the procedure;

2. Apply all possible linear rules to the current node. If the node closes, go to the first step. If it remains open, apply the (PB) rule, put the two newly created nodes on the stack (first the right node and after that the left, so that the left goes to the top) and go back to the first step. If no (PB) rule can be applied, finish the procedure.

[7]That is, they always choose the left subformula to be the motivation for (PB) application, that is, the auxiliary formula of a two-premise rule application.

When the procedure finishes, the strategy checks if the root node is closed. If it is, that is because all of its child nodes are also closed. Therefore the tableau is closed. If the root node is not closed, this happened because at least one of its child nodes remained open and completed, thus the tableau is open.

C.4.2 Sorters

The order in which s-formulas are analyzed is an aspect that can have a strong influence on a strategy performance. As we have already said, at every moment in the proof search procedure, the prover has a list of not-analyzed s-formulas in the current node from which it is going to choose the next formula to be analyzed. As the s-formulas in the beginning of the list are analyzed first, if we sort the s-formulas we can change the strategy behavior.

In **KEMS**, the s-formulas are sorted before rules are applied by one *signed formula sorter*. Let us describe each of the thirteen sorters we have implemented[8]. The first two sorters are related to the order in which signed formulas are inserted in **KE** tableau nodes' signed formula list:

1. insertion order - most recently inserted s-formulas go to the end of the list;

2. reverse order - most recently inserted s-formulas go to the beginning of the list.

The five connective sorters behave similarly: the s-formulas where a given connective appears as the main connective are put in the beginning of list:

1. and connective;

2. or connective;

3. implication connective;

4. bi-implication connective;

5. exclusive or connective.

[8]In future versions we may have more sorters and even combine two or more in a prover configuration.

The two sign sorters behave in a similar way: the s-formulas with a given sign are put in the beginning of list:

1. true sign;

2. false sign.

In the two complexity sorters, the s-formulas are sorted according to their size:

1. increasing complexity - the smaller s-formulas appear first;

2. decreasing complexity - the more complex s-formulas appear first.

In the two string sorters, the s-formulas are sorted according to the string that represents them:

1. string order - sorts s-formulas in alphabetical order;

2. reverse string order - sorts s-formulas in reverse alphabetical order.

The two sorters above were implemented only to be compared with others.

The results presented in Section D.2 will illustrate the impact of sorters on **KEMS** prover configurations performance.

C.4.3 CPL Strategies

Here we will discuss the six strategies implemented for **CPL**.

Simple Strategy

The first implemented strategy is called *Simple Strategy* because it uses simplification rules (it implements the **s-CPL-KE** system). This is the order of rule applications in Simple Strategy:

1. all one-premise rules;

2. all simplification rules in which '**T** $\top$' or '**F** $\bot$' is the minor premise;

3. all other simplification rules[9];

4. (PB) rule.

Tableau proof trees are represented by the ProofTree class. A ProofTree contains a list of nodes and three references to other ProofTree objects: a reference to its left child, a reference to its right child, and a reference to its parent. Each of these references may be null. Only the root proof tree does not have a parent. Every ProofTree either has two children or none.

ClassicalProofTree is a subclass of ProofTree in which each node contain a list that we call s-formula container. A s-formula container includes a s-formula, a state and an origin. The state of a signed formula container (sf-container) can be either *not analyzed*, *analyzed*, and *fulfilled*. The first state is for those containers that contain signed formulas that can be used as the major premise in some rule application, but were not yet used. The second is for the containers that contain signed formulas that were used as major premise. And the last state is for the containers that contain signed formulas that cannot be used as major premise. A container origin may contain references to the rule that originated that container as well as the premises used in that rule application. Besides that, a ClassicalProofTree contains a *namc* list and has additional methods for dealing with node closure.

To apply simplification rules, we designed a subclass of the ClassicalProofTree class called FormulaReferenceClassicalProofTree. This class contains data structures that keep track of all references between signed formula containers. For instance, if we have a sf-container that contains the '$\mathtt{T}\ A \rightarrow B$' s-formula, we can find all s-formulas that have '$A \rightarrow B$' as subformula so that we can apply a simplification rule. And all this data is kept in memory, what makes this strategy consume a lot of memory.

Let us see an example of this strategy in action. In this example we will work with labelled signed formulas '$l : \mathcal{S}\ \mathcal{F}$' where l is a label that contains the '$\mathcal{S}\ \mathcal{F}$' signed formula's origin. Signed formulas which come from the problem are labelled p_i, where i is an index. $\mathrm{app}^p_c(\mathrm{R}, \mathcal{S}_1\ \mathcal{F}_1, t)$ is the label associated with the application of a basic (non simplification)

[9]Notice that no non-simplification two-premise rule is used in this strategy.

rule R (that has p premises and c conclusions), where '$\mathcal{S}_1\ \mathcal{F}_1$' is the main premise and the third parameter (t) can either not appear, or be '$\mathcal{S}_2\ \mathcal{F}_2$', the auxiliary premise (for two-premise rules), or even be an i which is the number of the conclusion associated with the label (for two-conclusion rules).

To indicate when a node is closed, we have the close(sfl_1, sfl_2) label, where sfl_1 and sfl_2 are the labels of the s-formulas that justify the closure. Finally,

$$\text{simplSubst}(sfl_m, sfl_a, \Theta(X))$$

is the label of the s-formula which is the result of applying a simplification rule, where sfl_m is the main premise label, sfl_a is the auxiliary premise label, and $\Theta(X)$ is the subformula of the main premise to which the simplification rule is applied (see Section C.2.3 for simplification rule definition).

In the examples below, the '×' symbol denotates that a node is closed. The '■' symbol is used to state that a node is open and completed, that is, no further rule can be applied and from that node we can find a valuation that falsifies the sequent. The '□' symbol is used to state that a node is open but not completed. When another (a previous) node is found to be open and completed, that is, when we find a valuation that falsifies, we no longer need to analyze other nodes that are in the open node stack. Therefore the other nodes remain open but usually are not completed.

The first example only illustrates the use of labels in a proof of $A, A \to B \vdash B$:

$$\begin{array}{rc}
p_1 : & \mathtt{T}\,A \\
p_2 : & \mathtt{T}\,A \to B \\
p_3 : & \mathtt{F}\,B \\
\hline
g_1 := \text{app}_1^2(\mathtt{T} \to_1, p_2, p_1) : & \mathtt{T}\,B \\
\text{close}(g_1, p_3) : & \times
\end{array}$$

The second example shows three applications of simplification rules. We used the $g_i := l$ notation, stating that g_i abbreviates label l, to simplify the labels in the following open tableau for $A, D \to ((A \to B) \wedge (C \to A)) \vdash B$:

$$
\begin{array}{rrc}
 & p_1 : & \mathtt{T}\,\top \\
 & p_2 : & \mathtt{F}\,\bot \\
 & p_3 : & \mathtt{T}\,A \\
 & p_4 : & \mathtt{T}\,D \to ((A \to B) \wedge (C \to A)) \\
 & p_5 : & \mathtt{F}\,B \\
\hline
g_1 := & \mathrm{simplSubst}(p_5, p_3, A \to B) : & \mathtt{T}\,D \to (B \wedge (C \to A)) \\
g_2 := & \mathrm{simplSubst}(g_1, p_5, B \wedge (C \to A)) : & \mathtt{T}\,D \to \bot \\
g_3 := & \mathrm{simplSubst}(g_2, p_2, D \to \bot) : & \mathtt{T}\,\neg D \\
 & \mathrm{app}_1^1(\mathtt{T}\neg, g_3) : & \mathtt{F}\,D \\
 & & \blacksquare
\end{array}
$$

Our last example uses the $PB(sfl_1, s)$ label to indicate that the signed formula to which this label is associated is the result of applying the (PB) rule to the signed formula whose label is sfl_1. The b parameter indicates if this is the label associated with the left successor node (when $s = l$) or right successor node ($s = r$). The example below is an open and completed tableau for $E \vee C \vdash A \to \neg(C \vee D)$:

$$
\begin{array}{rrc}
 & p_1 : & \mathtt{T}\,E \vee C \\
 & p_2 : & \mathtt{F}\ A \to \neg(C \vee D) \\
 & \mathrm{app}_2^1(\mathtt{F} \to, p_2, 1) : & \mathtt{T}\,A \\
g_1 := & \mathrm{app}_2^1(\mathtt{F} \to, p_2, 2) : & \mathtt{F}\,\neg(C \vee D) \\
g_2 := & \mathrm{app}_1^1(\mathtt{F}\neg, g_1) : & \mathtt{T}\,C \vee D
\end{array}
$$

$$
\begin{array}{rrlrl}
g_3 := & PB(g_2, l) : & \mathtt{F}\ C & PB(g_2, r) : & \mathtt{T}\ C \\
 & \mathrm{app}_1^2(\mathtt{T}\vee, g_2, g_3) & \mathtt{T}\ D & & \square \\
 & \mathrm{app}_1^2(\mathtt{T}\vee, p_1, g_3) & \mathtt{T}\ E & & \\
 & & \blacksquare & &
\end{array}
$$

Memory Saver Strategy

The Memory Saver Strategy implements almost the same algorithm implemented by Simple Strategy but keeps the minimum amount of data structures in memory. For instance, instead of using the FormulaReferenceClassicalProofTree class for keeping references to formulas in memory, this strategy uses an OptimizedClassicalProofTree class (that does not keep those references). And it uses a ReferenceFinder class that has methods for searching the same references stored in a FormulaReferenceClassicalProofTree whenever they are needed.

Backjumping Simple Strategy

Backjumping is a technique used in backtracking algorithms that allows for efficient pruning of search spaces. It has been proposed in the early 1990s, and has been extensively applied in constraint propagation [36], but it has hardly ever been applied to theorem provers [40], specially tableau based ones [63]. Backjumping can be very useful in tableau based theorem provers because it can be used to prevent repeatedly re-solving the same subproblem.

The Backjumping Simple Strategy implements the backjumping technique and is an extension of Simple Strategy. The only difference occurs when nodes are closed. Recall that a **KE**-tableau tree is a tree whose nodes contain a list of signed formulas. A left (right) node is a node which is the left (right) child of another node. The first signed formula of a left node is called a decision. A node that contains a decision is called a *decision node*. In the Backjumping Simple Strategy, whenever a node closes, the decision nodes used (as premises) to close that node must be marked as used. And whenever a right node closes, if any of its grandparents is a not used decision, its sibling can be closed by backjumping.

In Section D.1.1 we show a family of problems used to test the Backjumping Simple Strategy. The tableau proofs (for instances of this family) generated by strategies that do not use backjumping may include redundant sub-trees. This is clear in Figure C.13 where we show an example of a B_PHP^2_n proof (see Section D.1.1). The sub-tree containing the **KE** PHP$_n$ proof ($\mathcal{T}_1$) appears four times.

In Figure C.14 we show a proof of the same problem using backjumping. The $\mathcal{T}_1$ subtree now appears only once because the nodes containing $\mathtt{F}\ A_{1,1}$ and $\mathtt{F}\ A_{2,1}$ are declared

$$\frac{\begin{array}{c}\mathtt{T}\ \neg(A_{1,1} \wedge A_{1,2})\\ \mathtt{T}\ A_{2,1} \rightarrow A_{2,2}\\ \mathrm{PHP}_n\end{array}}{\mathtt{F}\ A_{1,1} \wedge A_{1,2}}$$

$\mathtt{T}\ A_{1,1}$
$\mathtt{F}\ A_{1,2}$

$\mathtt{F}\ A_{1,1}$

$\mathtt{T}\ A_{2,1}$
$\mathtt{T}\ A_{2,2}$

$\mathcal{T}_1$

$\mathtt{F}\ A_{2,1}$

$\mathcal{T}_1$

$\mathtt{T}\ A_{2,1}$
$\mathtt{T}\ A_{2,2}$

$\mathcal{T}_1$

$\mathtt{F}\ A_{2,1}$

$\mathcal{T}_1$

Figure C.13: A proof of B_PHP^2_n.

closed by the backjumping strategy when it is found that the $\mathtt{T}\ A_{1,1}$ and $\mathtt{T}\ A_{2,1}$ decisions were not used to close $\mathcal{T}_1$. That is, the $\mathcal{T}_1$ sub-proof now needs to be generated only once, in the leftmost leaf node. When the strategy verifies that the two open nodes came from unused decisions, it closes these nodes.

$$\frac{\begin{array}{c}\mathtt{T}\ \neg(A_{1,1} \wedge A_{1,2})\\ \mathtt{T}\ A_{2,1} \rightarrow A_{2,2}\\ \mathrm{PHP}_n\end{array}}{\mathtt{F}\ A_{1,1} \wedge A_{1,2}}$$

$\mathtt{T}\ A_{1,1}$
$\mathtt{F}\ A_{1,2}$

$\mathtt{F}\ A_{1,1}$
$\times$

$\mathtt{T}\ A_{2,1}$
$\mathtt{T}\ A_{2,2}$

$\mathcal{T}_1$

$\mathtt{F}\ A_{2,1}$
$\times$

Figure C.14: A proof of B_PHP^2_n using backjumping.

Learning Strategy

In the Learning Strategy we implemented the learning technique [40] used in SAT solvers. The idea is the following: whenever a left node closes we look for the reasons for this closure. The reasons are two s-formulas: $\mathtt{T}\ X$ and $\mathtt{F}\ X$. Then we look for the binary s-formulas that were used as main premise to obtain the closing reasons. Then we apply general resolution [1] to these two formulas and include this *learned formula* in the parent node.

For instance, suppose we close a node of an instance of PHP_3 with $\mathtt{F}\ p_{3,1}$ and $\mathtt{T}\ p_{3,1}$ (see Figure C.15). The s-formulas that gave origin to these formulas are $\mathtt{F}\ p_{2,1} \wedge p_{3,1}$ and $\mathtt{T}\ p_{3,0} \vee p_{3,1}$. Suppose we apply $(\mathtt{F}\wedge_2)$ rule to $\mathtt{F}\ p_{2,1} \wedge p_{3,1}$ and $\mathtt{T}\ p_{3,1}$; the result would be $\mathtt{F}\ p_{2,1}$. And suppose we apply $(\mathtt{T}\vee_2)$ to $\mathtt{T}\ p_{3,0} \vee p_{3,1}$ and $\mathtt{F}\ p_{3,1}$; the result would be $\mathtt{T}\ p_{3,0}$. We take these two results ($\mathcal{S}_1\ \mathcal{F}_1$ and $\mathcal{S}_2\ \mathcal{F}_2$) and create a learned formula (lf) is the following way:

- $\text{lf}_i = \mathcal{F}_i$ if $\mathcal{S}_i = \mathtt{T}$, otherwise $\text{lf}_i = \neg(\mathcal{F}_i)$;
- $\text{lf} = \mathtt{T}\ (\text{lf}_1 \vee \text{lf}_2)$.

This '$\mathtt{T}\ \neg(p_{2,1}) \vee p_{3,0}$' learned formula is then included the node which is the parent of the current node (see Figure C.16) so that it can be used in the proof search on the open nodes.

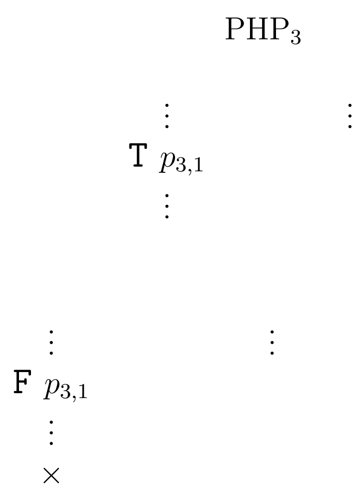

Figure C.15: A proof of PHP_3.

Comb Learning Strategy

This strategy produces a proof whose left branches have a height equal or less than 1, that is, a comb-like proof (see Figure C.17). The idea is the following: whenever we close

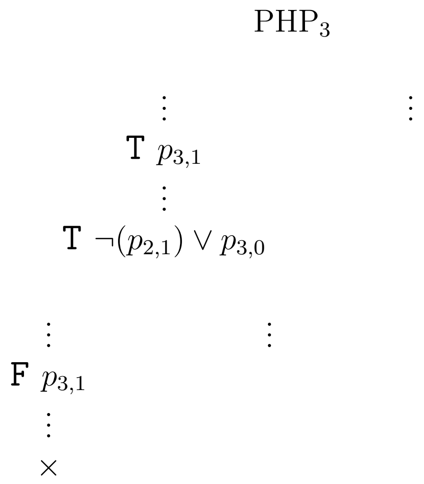

Figure C.16: An example of learning in a proof of PHP$_3$.

a left node, the strategy gathers the used decisions and separates them in two groups: those with a $\mathtt{T}$ sign ($\{\mathtt{T}\ A_1, \ldots \mathtt{T}\, A_n\}$) and those with a $\mathtt{F}$ sign ($\{\mathtt{F}\ B_1, \ldots \mathtt{F}\, B_m\}$). Then it discards the whole left node from the root node (τ) and creates two new child nodes for τ:

$$\tau$$

$$\mathtt{F}\ (A_1 \wedge \ldots \wedge A_n) \to (B_1 \vee \ldots \vee B_m) \qquad \mathtt{T}\ (A_1 \wedge \ldots \wedge A_n) \to (B_1 \vee \ldots \vee B_m)$$

The left node does not have to be expanded, because it will surely close since the '$\mathtt{F}\ (A_1 \wedge \ldots \wedge A_n) \to (B_1 \vee \ldots \vee B_m)$' learned formula will be decomposed into the decisions that led the original left branch to close. So the proof can continue with the right node. It is important to notice that this is a naïve kind of learning that usually produces proofs which are bigger than other strategies' proofs.

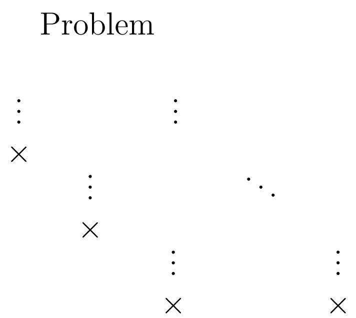

Figure C.17: Sketch of a Comb Learning Strategy proof.

Configurable Strategy

All previous **CPL** strategies implement the **s-CPL-KE** system. This strategy implements the **e-CPL-KE** system. This is the order of rule applications:

1. all one-premise rules;
2. all two-premise rules;
3. (PB) rule.

The other features are equal to Simple Strategy features.

This strategy is useful because, by using sorters, it allows a higher control over which formulas are analyzed first. Besides that, it is interesting to compare this strategy with the strategies that implement the **s-CPL-KE** system, because the comparison may show when simplification rules are most useful.

C.4.4 mbC Strategies

For **mbC**, we have implemented the following two strategies.

mbC Simple Strategy

This is an extension of **CPL** Simple Strategy for **mbC**. It implements the **e-mbC-KE** system – therefore it does not use any simplification rule. This is the order of rule applications:

1. all **mbC** one-premise rules;
2. all **mbC** two-premise rules;
3. (PB) rule.

An important difference here is that in **mbC**'s ($\mathtt{T}\neg'$) rule the two premises have the same size (in **CPL** two-premise rules the major premise is always bigger than the minor premise). We have proved (see Section B.2.3) that we only need to branch on a '$\circ A$' formula if it already appears as subformula of some formula in this branch. So we had to add this check before applying the (PB) rule. The other features are equal to Simple Strategy features.

mbC Extended Strategy

This is an extension of the previous strategy. It implements the **e-mbC-KE** system with the ($\mathtt{T}\circ''$) and ($\mathtt{T}\neg''$) additional rules (see Section C.2.4). These rules are applied after all other **mbC** two-premise rules and before (PB) rule. The other features are equal to **mbC** Simple Strategy features.

C.4.5 mCi Strategies

For **mCi**, we have implemented the following two strategies.

mCi Simple Strategy

This strategy is very similar to **mbC** Simple Strategy. It implements the **e-mCi-KE** system. This is the order of rule applications:

1. all **mCi** one-premise rules;
2. all **mCi** two-premise rules;
3. (PB) rule.

But here we do not have to restrict the application of the (PB) rule because of the ($\mathtt{T}\neg'$) rule. All other features of this strategy are equal to **mbC** Simple Strategy features.

mCi Extended Strategy

This is an extension of the previous strategy. It implements the **e-mCi-KE** system with two additional rules: ($\mathtt{T}\circ''$) and ($\mathtt{T}\neg''$) (see Section C.2.4). These rules are applied after all other **mCi** two-premise rules and before (PB) rule. The other features are equal to **mCi** Simple Strategy features.

Strategy	Main feature
CPL Simple Strategy	Keeps formula reference data structures in memory
CPL Memory Saver Strategy	Does not keep formula reference data structures in memory
CPL Backjumping Simple Strategy	Implements the backjumping technique
CPL Learning Strategy	Implements a learning technique
CPL Comb Learning Strategy	Implements the comb learning technique
CPL Configurable Strategy	Allows a higher control over which formulas are analyzed first
mbC Simple Strategy	An extension of **CPL** Simple Strategy for **mbC**
mbC Extended Strategy	An extension of **mbC** Simple Strategy that applies derived rules before applying (PB)
mCi Simple Strategy	An extension of **CPL** Simple Strategy for **mCi**
mCi Extended Strategy	An extension of **mCi** Simple Strategy that applies derived rules before applying (PB)

Table C.1: Overview of **KEMS** Strategies.

C.5 Conclusion

KEMS current version implements strategies for three logics: **CPL**, **mbC** and **mCi**. An overview of the implemented strategies is presented in Table C.1. We have shown that to solve a problem with a strategy we must choose a sorter. The implemented sorters are described in Table C.2. In Appendix D we present the results obtained by **KEMS** with several families of problems using these strategies and sorters.

Sorter	Description
Insertion Order	least recently inserted s-formulas are analyzed first
Reverse Order	most recently inserted s-formulas are analyzed first
And	s-formulas with '$\wedge$' as main connective are analyzed first
Or	s-formulas with '$\vee$' as main connective are analyzed first
Implication	s-formulas with '$\rightarrow$' as main connective are analyzed first
Bi-implication	s-formulas with '$\leftrightarrow$' as main connective are analyzed first
Exclusive Or	s-formulas with '$\oplus$' as main connective are analyzed first
True	s-formulas with $\mathbf{T}$ as sign are analyzed first
False	s-formulas with $\mathbf{F}$ as sign are analyzed first
Increasing	smaller s-formulas are analyzed first
Decreasing	bigger s-formulas are analyzed first
String Order	s-formulas whose representation as a string of characters come first in alphabetical order are analyzed first
Reverse String Order	s-formulas whose representation as a string of characters come last in alphabetical order are analyzed first

Table C.2: Overview of **KEMS** sorters.

Referências Bibliográficas

[1] Michael Alekhnovich, Jan Johannsen, Toniann Pitassi, and Alasdair Urquhart. An exponential separation between regular and general resolution. In *STOC '02: Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 448–456, New York, NY, USA, 2002. ACM Press.

[2] Noriko Arai, Toniann Pitassi, and Alasdair Urquhart. The complexity of analytic tableaux. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 356–363. ACM Press, 2001.

[3] Bernhard Beckert, Richard Bubel, Elmar Habermalz, and Andreas Roth. jTAP - a Tableau Prover in Java, February 1999. Universitat Karlsruhe. Available at `http://i12www.ira.uka.de/~aroth/jTAP/`. Last accessed, November 2006.

[4] Bernhard Beckert and Joachim Posegga. leanTAP: Lean tableau-based deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995.

[5] Evert W. Beth. *The Foundations of Mathematics.* North-Holland Publishing Company, Amsterdam, 1959.

[6] Maria Paola Bonacina and Thierry Boy de la Tour. Fifth Workshop on Strategies in Automated Deduction - Workshop Programme, 2004. `http://tinyurl.com/y8dkbj`. Last accessed, November 2006.

[7] Maria Luisa Bonet and Nicola Galesi. A study of proof search algorithms for resolution and polynomial calculus. In *FOCS '99: Proceedings of the 40th Annual*

Symposium on Foundations of Computer Science, page 422, Washington, DC, USA, 1999. IEEE Computer Society.

[8] Krysia Broda, Marcello D'Agostino, and Marco Mondadori. A Solution to a Problem of Popper. In *The Epistemology of Karl Popper*. Kluwer, 1995. `http://citeseer.nj.nec.com/broda95solution.html`. Last accessed, November 2006.

[9] S. R. Buss. Polynomial size proofs of the propositional pigeonhole principle. *Journal of Symbolic Logic*, 52:916–927, 1987.

[10] Liming Cai. *Nondeterminism and optimization*. PhD thesis, Texas A&M University, USA, 1994.

[11] Carlos Caleiro, Walter Carnielli, Marcelo E. Coniglio, and Joao Marcos. Two's company: "The humbug of many logical values". In *Logica Universalis*, pages 169–189. Birkhäuser Verlag, Basel, Switzerland, 2005. Pre-print available at `http://tinyurl.com/yb5qbz`. Last accessed, November 2006.

[12] Alessandra Carbone and Stephen Semmes. *Graphic Apology for Symmetry and Implicitness*. Oxford University Press, 2000.

[13] W. A. Carnielli. Systematization of the finite many-valued logics through the method of tableaux. *The Journal of Symbolic Logic*, 52:473–493, 1987.

[14] W.A. Carnielli and J. Marcos. Ex Contradictione Non Sequitur Quodlibet. *Bulletin of Advanced Reasoning and Knowledge*, 1:89–109, 2001.

[15] W.A. Carnielli and J. Marcos. Tableau systems for logics of formal inconsistency. *Proceedings of the International Conference on Artificial Intelligence (IC-AI'2001)*, pages 848–852, 2001.

[16] Walter Carnielli. How to build your own paraconsistent logic: an introduction to the Logics of Formal (In)Consistency. *Proceedings of the Workshop on Paraconsistent Logic (WoPaLo)*, 2002.

[17] Walter Carnielli, Marcelo Coniglio, and Ricardo Bianconi. *Logic and Applications: Mathematics, Computer Science and Philosophy (in Portuguese).* Unpublished, 2005. Preliminary version. Chapters 1 to 5.

[18] Walter Carnielli, Marcelo E. Coniglio, and Joao Marcos. Logics of Formal Inconsistency. In *Handbook of Philosophical Logic*, volume 12. Kluwer Academic Publishers, 2005. To appear. Pre-print available at `http://tinyurl.com/ybn4yw`. Last accessed, November 2006.

[19] Walter A. Carnielli and Richard L. Epstein. *Computabilidade – Funções Computáveis, Lógica e os Fundamentos da Matemática.* Editora da Unesp, 2006.

[20] Stephen Cook. The P versus NP problem, 2000. `http://tinyurl.com/n5thm`. Last accessed, May 2005.

[21] Stephen Cook. The importance of the P versus NP question. *J. ACM*, 50(1):27–29, 2003.

[22] Stephen Cook and Robert Reckhow. On the lengths of proofs in the propositional calculus (preliminary version). In *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 135–148, New York, NY, USA, 1974. ACM Press.

[23] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM Press.

[24] Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8(4):28–32, 1976.

[25] W. Cook, C. R. Coullard, and G. Turán. On the complexity of cutting-plane proofs. *Discrete Appl. Math.*, 18(1):25–38, 1987.

[26] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms - Second Edition.* MIT Press, 2001.

[27] Newton C. A. da Costa, Décio Krause, and Otávio Bueno. Paraconsistent logics and paraconsistency: Technical and philosophical developments. *CLE e-prints (Section Logic)*, 4(3), 2004. Pre-print available at http://tinyurl.com/yxhon7. Last accessed, November 2006.

[28] Marcello D'Agostino. Are Tableaux an Improvement on Truth-Tables? Cut-Free proofs and Bivalence, 1992. Available at http://citeseer.nj.nec.com/140346.html. Last accessed, May 2005.

[29] Marcello D'Agostino. Tableau methods for classical propositional logic. In Marcello D'Agostino et al., editor, *Handbook of Tableau Methods*, chapter 1, pages 45–123. Kluwer Academic Press, 1999.

[30] Marcello D'Agostino, Dov Gabbay, and Krysia Broda. Tableau methods for substructural logics. In Marcello D'Agostino et al., editor, *Handbook of Tableau Methods*, chapter 6, pages 397–467. Kluwer Academic Press, 1999.

[31] Marcello D'Agostino and Marco Mondadori. The taming of the cut: Classical refutations with analytic cut. *Journal of Logic and Computation*, pages 285–319, 1994.

[32] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[33] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

[34] Sandra de Amo, Walter Carnielli, and João Marcos. A Logical Framework for Integrating Inconsistent Information in Multiple Databases. In Thomas Eiter and Klaus-Dieter Schewe, editors, *Lecture Notes in Computer Science*, volume 2284, pages 67–84. Springer-Verlag, Berlim., 2002.

[35] Eleonora de Moraes. *Lista de discussão gestar bem interior-sp*, 2004. http://br.groups.yahoo.com/group/gestarbeminterior-sp. Last accessed, December 2006.

[36] Rina Dechter. *Constraint Processing.* Morgan Kaufmann, 2003.

[37] Luis Fariñas del Cerro, David Fauthoux, Olivier Gasquet, Andreas Herzig, Dominique Longin, and Fabio Massacci. Lotrec: The generic tableau prover for modal and description logics. In *IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning*, pages 453–458. Springer-Verlag, 2001.

[38] Wagner Dias. Tableaux implementation for approximate reasoning (in portuguese). Master's thesis, Computer Science Department, Institute of Mathematics and Statistics, University of São Paulo, 2002.

[39] Simone Grilo Diniz and Ana Cristina Duarte. *Parto normal ou cesárea? O que toda mulher deve saber (e todo homem também).* Editora da Unesp, São Paulo, 2004.

[40] Heidi Dixon. *Automating Pseudo-Boolean Inference Within a DPLL Framework.* PhD thesis, University of Oregon, USA, Dec 2004. Available at `http://www.cirl.uoregon.edu/dixon/dixonDissertation.pdf`. Last accessed, August 2005.

[41] Amigas do Parto. *Lista de discussão Parto Nosso*, 2003. `br.groups.yahoo.com/group/partonosso`. Last accessed, December 2006.

[42] Itala M. Loffredo D'Ottaviano and Milton Augustinis de Castro. Analytical Tableaux for da Costa's Hierarchy of Paraconsistent Logics $C_n, 1 \leq n \leq \omega$. *Journal of Applied Non-Classical Logics*, 15(1):69–103, 2005.

[43] Tzilla Elrad, Robert E. Filman, and Atef Bader. Aspect-Oriented Programming. *Communications of the ACM*, 44, 2001.

[44] M. Enkin, M. Skiers, J. Nelson, C. Crowder, L. Duly, and E. Hodnett. *A guide to effective care during pregnancy and childbirth.* Oxford, UK: Oxford University Press, 2000.

[45] Fadynha. *Lista de discussão partonatural*, 1999. `br.groups.yahoo.com/group/partonatural`. Last accessed, December 2006.

[46] Luis Fariñas del Cerro, David Fauthoux, Olivier Gasquet, Andreas Herzig, Dominique Longin, and Fabio Massacci. Lotrec: the generic tableau prover for modal and description logics. In *International Joint Conference on Automated Reasoning*, LNCS, page 6. Springer Verlag, 18-23 juin 2001.

[47] M. Finger and R. Wassermann. Approximate Reasoning and Paraconsistency-Preliminary Report. *Proceedings of the Eigth Workshop on Logic, Language, Information and Comunication (WoLLIC), Braslia, Brazil*, 2001.

[48] M. Finger and R. Wassermann. The universe of approximations. *Electronic Notes in Theoretical Computer Science*, 84:1–14, 2003.

[49] M. Finger and R. Wassermann. Anytime Approximations of Classical Logic from Above. *Journal of Logic and Computation*, 2006.

[50] M. Finger and R. Wassermann. The universe of propositional approximations. *Theoretical Computer Science*, 355(2):153–166, 2006.

[51] Melvin Fitting. *First-order logic and automated theorem proving (2nd ed.)*. Springer-Verlag New York, Inc., 1996.

[52] Melvin Fitting. Introduction. In Marcello D'Agostino et al., editor, *Handbook of Tableau Methods*, chapter 1, pages 1–43. Kluwer Academic Press, 1999.

[53] Zhaohui Fu, Yogesh Mahajan, and Sharad Malik. New Features of the SAT'04 versions of zChaff, 2004. `http://www.princeton.edu/~chaff/zchaff/sat04.pdf`. Last accessed, September 2005.

[54] Dov M. Gabbay. *Labelled Deductive Systems, Volume 1.* Oxford University Press, Oxford, 1996.

[55] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Adisson-Wesley, 1994.

[56] Gerhard Gentzen. Investigations into logical deductions, 1935. In M. E. Szabo, editor, *The Collected Works of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1969.

[57] J. Gosling, B. Joy, and G. Steele. *The Java Programming Language*. Addison-Wesley, Reading, MA, 1996.

[58] Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985.

[59] Klaus Havelund and Natarajan Shankar. Experiments in theorem proving and model checking for protocol verification. In *FME '96: Proceedings of the Third International Symposium of Formal Methods Europe on Industrial Benefit and Advances in Formal Methods*, pages 662–681. Springer-Verlag, 1996.

[60] J. Hintikka. Form and content in quantification theory. *Acta Philosophica Fennica*, 8:7–55, 1955.

[61] Jan Holub and Borivoj Melichar. Implementation of nondeterministic finite automata for approximate pattern matching. In *WIA '98: Revised Papers from the Third International Workshop on Automata Implementation*, pages 92–99. Springer-Verlag, 1999.

[62] Scott E. Hudson, Frank Flannery, C. Scott Anaian, Dan Wang, and Andrew Appel. *CUP Parser Generator for Java*, 1999. `http://www2.cs.tum.edu/projects/cup`. Last accessed, November 2006.

[63] Ullrich Hustadt and Renate A. Schmidt. Simplification and backjumping in modal tableau. In *Lecture Notes in Computer Science*, volume 1397 of *Lecture Notes in Computer Science*, pages 187–201, 1998.

[64] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. Getting Started with AspectJ. *Communications of the ACM*, 44:59–65, 2001.

[65] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. *Lecture Notes in Computer Science*, 2072:327–355, 2001.

[66] Gerwin Klein. *JFlex: the fast lexical analyzer generator for Java*, 1998. `http://jflex.de`. Last accessed, November 2006.

[67] S. Kundu and J. Chen. Fuzzy logic or Lukasiewicz logic: A clarification. *Fuzzy Sets and Systems*, 95(3):369–379, 1998.

[68] L. Di Lascio. Analytic fuzzy tableaux. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 5(6):434–439, Dec 2001.

[69] D. W. Loveland. Automated deduction: Some achievements and future directions. Technical report, National Science Foundation, 1997. Available at `http://tinyurl.com/y7mb6p`. Last accessed, May 2005.

[70] D. W. Loveland. Automated deduction: achievements and future directions. *Commun. ACM*, 43(11es):10, 2000.

[71] Wendy MacCaull. Tableau method for residuated logic. *Fuzzy Sets Syst.*, 80(3):327–337, 1996.

[72] Alistair Manning, Andrew Ireland, and Alan Bundy. Increasing the Versatility of Heuristic Based Theorem Provers. In *LPAR'93*, 1993.

[73] Heiko Mantel and Jens Otten. lintap: A tableau prover for linear logic. In *TABLEAUX '99: Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 217–231, London, UK, 1999. Springer-Verlag.

[74] João Marcos. Personal communication by email, October 2006.

[75] Fabio Massacci. Simplification: A general constraint propagation technique for propositional and modal tableaux. In *TABLEAUX '98: Proceedings of the Inter-*

national Conference on Automated Reasoning with Analytic Tableaux and Related Methods, pages 217–231. Springer-Verlag, 1998.

[76] Fabio Massacci. The proof complexity of analytic and clausal tableaux. *Theor. Comput. Sci.*, 243(1-2):477–487, 2000.

[77] William McCune and Larry Wos. Otter - The CADE-13 Competition Incarnations. *J. Autom. Reason.*, 18(2):211–220, 1997.

[78] Elliott Mendelson. *Introduction to Mathematical Logic.* Chapman & Hall, London, UK, fourth edition, 1997.

[79] Paulo Blauth Menezes. *Linguagens Formais e Autômatos.* Instituto de Informática da UFRGS : Editora Sagra Luzzatto, Porto Alegre, 232p., 2005.

[80] Sun Microsystems. Java Runtime Environment (JRE) 5.0 Installation Notes, 2006. `http://java.sun.com/j2se/1.5.0/jre/install.html`. Last accessed, November 2006.

[81] S. H. Mirian and M. Mousavi. Nondeterminism in set-theoretic specifications (in persian). In *Proceedings of Iranian Computer Society Annual Conference (CSICC'02)*, feb 2002.

[82] David G. Mitchell, Bart Selman, and Hector J. Levesque. Hard and easy distributions for SAT problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 459–465, Menlo Park, California, 1992. AAAI Press.

[83] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, June 2001.

[84] John K. Myers. An introduction to planning and meta-decision-making with uncertain nondeterministic action using 2nd-order probabilities. In *Proceedings of the first*

international conference on Artificial intelligence planning systems, pages 297–298. Morgan Kaufmann Publishers Inc., 1992.

[85] Adolfo Neto. An Object-Oriented Implementation of a KE Tableau Prover, nov 2003. Avaliable at `http://tinyurl.com/y3qmkx`. Last accessed, November 2006.

[86] Adolfo Neto. Modifications on the implementation of a framework for tableau methods, jul 2003. Avaliable at `http://tinyurl.com/yjgjnq`. Last accessed, November 2006.

[87] Adolfo Neto and Marcelo Finger. A Multi-Strategy Tableau Prover. In *I Simpósio de Iniciação Científica e Pós-Graduação do IME-USP*. University of São Paulo, 2005. Available at `http://tinyurl.com/tbdd6`. Last accessed, November 2006.

[88] Adolfo Neto and Marcelo Finger. A Multi-Strategy Tableau Prover. In *SeMe-2005. Workshop "Semantics and Meaning"*, IFIP International Federation for Information Processing. Unicamp. Campinas-SP., 2005. Available at `http://tinyurl.com/yzx8ve`. Last accessed, November 2006.

[89] Adolfo Neto and Marcelo Finger. Implementing a multi-strategy theorem prover. In Ana Cristina Bicharra Garcia and Fernando Santos Osório, editors, *Proceedings of the V ENIA (Encontro Nacional de Inteligência Artificial), held in São Leopoldo-RS, Brazil, July 22-29 2005*, 2005. Available at `http://tinyurl.com/yd6n6n`. Last accessed, November 2006.

[90] Adolfo Neto and Marcelo Finger. Using Aspect-Oriented Programming in the Development of a Multi-Strategy Theorem Prover. In *Anais da II Jornada do Conhecimento e da Tecnologia do Univem*, Marília-SP, 2005. Available at `http://www.ime.usp.br/~adolfo/trabalhos/jornada2005.pdf`. Last accessed, November 2006.

[91] Adolfo Neto and Marcelo Finger. Effective Prover for Minimal Inconsistency Logic. In *Artificial Intelligence in Theory and Practice*, IFIP International Federation for Information Processing, pages 465–474. Springer Verlag, 2006. Available at

http://www.springerlink.com/content/b80728w7m6885765. Last accessed, November 2006.

[92] Adolfo Neto and Marcelo Finger. *KEMS - A* **KE** *Multi-Strategy Tableau Prover*, 2006. http://kems.iv.fapesp.br. Last accessed, November 2006.

[93] Michel Odent. *The Caesarean*. Free Association Books, 2004.

[94] Lawrence C. Paulson. *Handbook of logic in computer science (vol. 2): background: computational structures*, chapter Designing a theorem prover, pages 415–475. Oxford University Press, Inc., 1992.

[95] Francis Jeffry Pelletier. Seventy-five problems for testing automatic theorem provers. *J. Autom. Reason.*, 2(2):191–216, 1986.

[96] J. V. Pitt and R. J. Cunningham. Theorem proving and model building with the calculus ke. *Journal of the IGPL*, 4(1):129–150, 1996.

[97] Awais Rashid and Lynne Blair. Editorial: Aspect-oriented Programming and Separation of Crosscutting Concerns. *The Computer Journal*, 46(5):527–528, 2003.

[98] Alexandre Riazanov and Andrei Voronkov. Vampire 1.1 (system description). In *IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning*, pages 376–380. Springer-Verlag, 2001.

[99] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1):107–136, 2006.

[100] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[101] Satisfiability suggested format, 1993. http://www.satlib.org. Last accessed, March 22, 2005.

[102] Satisfiability library, 2003. http://www.satlib.org. Last accessed, March 22, 2005.

[103] N. Scharli, S. Ducasse, O. Nierstrasz, and A.P. Black. Traits: Composable units of behaviour. *Proc. of ECOOP*, 2743:248–274, 2003.

[104] J. Schumann. Tableau-based theorem provers: Systems and implementations. *Journal of Automated Reasoning*, 13(3):409–421, 1994. `http://www.springerlink.com/content/k182u80451306371`. Last accessed, November 4th, 2006.

[105] Bart Selman, Hector J. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California, 1992. AAAI Press.

[106] Raymond M. Smullyan. *First-Order Logic*. Springer-Verlag, 1968.

[107] Sérgio Soares and Paulo Borba. AspectJ - Programação orientada a aspectos em Java. *Tutorial no SBLP 2002, 6o. Simpósio Brasileiro de Linguagens de Programação. 5 a 7 de Junho, PUC-Rio, Rio de Janeiro, Brasil*, pages 39–55, 2002.

[108] R. Statman. Bounds for proof-search and speed-up in the predicate calculus. *Annals of Mathematical Logic*, pages 225–287, 1978.

[109] Friedrich Steimann. The paradoxical success of aspect-oriented programming. *SIGPLAN Not.*, 41(10):481–497, 2006.

[110] Geoff Sutcliffe. An overview of automated theorem proving, 2001. `http://www.cs.miami.edu/~tptp/OverviewOfATP.html`. Last accessed, March 2005.

[111] Geoff Sutcliffe. Thousands of problems for theorem provers, 2001. `http://www.cs.miami.edu/~tptp`. Last accessed, March 2005.

[112] Geoff Sutcliffe and Christian Suttner. The CADE ATP System Competition, 2003. `http://www.cs.miami.edu/~tptp/CASC`. Last accessed, March 2005.

[113] Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987.

[114] Marian Vittek. A compiler for nondeterministic term rewriting systems. In *RTA '96: Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, pages 154–167. Springer-Verlag, 1996.

[115] Wikipedia. *Nondetermnistic algorithm*, 2007. `http://en.wikipedia.org/wiki/Nondeterministic`. Last accessed, February 2007.