Implementing a Multi-Strategy Theorem Prover

Adolfo Gustavo Serra Seca Neto[1*], **Marcelo Finger**[1]

[1]Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo
Rua do Matão, 1010, São Paulo SP 05315-970

`[adolfo, mfinger]@ime.usp.br`

Abstract. *We describe the implementation of MSTP, a multi-strategy theorem prover based on the KE Tableau System. Strategies are responsible for the* control *of (nondeterministic) inference rules of automated deduction systems. MSTP is a theorem prover where we can vary the strategy without modifying the core of the implementation. To achieve the goal of constructing a well-designed and efficient multi-strategy theorem prover, we are using a new software development method, aspect orientation, that allows a better modularization of crosscutting concerns such as strategies. MSTP obtained excellent results compared with a similar tableau-based theorem prover.*

1. Introduction

Since the 1950s, automated deduction has been an active area of research. The early developments within the automated deduction field have had a profound effect on the artificial intelligence area. Automated Theorem Proving (ATP) deals with the development of computer programs that show that some statement (the conjecture) is a logical consequence of a set of statements (the axioms and hypotheses). ATP systems are used in a wide variety of domains [Sutcliffe 2001], such as mathematics, software generation, software verification, security protocol verification and hardware verification.

Most automated theorem provers nowadays are based either on the resolution principle [Robinson 1965] or on the DPLL procedure [Davis et al. 1962], but other methods can also be used. Tableau methods [Smullyan 1968, Fitting 1999] are particularly interesting for theorem proving since they exist in many varieties and for several logics. Besides that, contrary to resolution and DPLL, they do not require conversion of input problems to clause form.

The inference rules of automated deduction systems in general and tableau provers in particular are typically non-deterministic in nature. They say what can be done, not what must be done [Fitting 1999]. Thus, in order to obtain a mechanical procedure, inference rules need to be complemented by another component, usually called *strategy* or *search plan*, which is responsible for the *control* of the inference rules [Bonacina and de la Tour 2004]. That is, the inference rules of a proof method define a nondeterministic algorithm for finding a proof, and a strategy is a (deterministic) algorithm for finding proofs in this method. For each proof method, many strategies can be

[*] Partially supported by a graduate fellowship from CAPES.

defined. Several features of the proofs obtained by a strategy such as size of the proof and time spent by the proof procedure, can vary greatly as different strategies are used.

In this paper we present the design and implementation of MSTP, a multi-strategy theorem prover based on the KE Tableau System [D'Agostino and Mondadori 1994]. The KE System is a refutation system for finding proofs which is close to Analytic Tableaux (AT) [Smullyan 1968] but, instead of being cut free, includes a classical cut rule which is not eliminable. Because of this, KE was proven to be more efficient than AT (KE linearly simulates AT) and allows the insertion of several new proof strategies.

A multi-strategy theorem prover is a theorem prover where we can vary the strategy without modifying the core of the implementation. A multi-strategy theorem prover can be used for three purposes: educational, exploratory and adaptive. For educational purposes, it can be used to illustrate how the choice of a strategy can affect performance of the prover. As an exploratory tool, a multi-strategy theorem prover can be used to test strategies and make comparisons between them. And we can also think of an adaptive multi-strategy theorem prover that changes the strategy used according to features of the problem presented to it.

To achieve the goal of constructing a well-designed and efficient multi-strategy theorem prover, we are using a new software development paradigm: aspect orientation [Elrad et al. 2001]. This is a contribution of our work, as we know of no other aspect-oriented theorem prover. We are using aspect-orientation together with object-orientation, a well-established software development paradigm. The use of these two technologies allows a better modularization of strategies in the development of the prover than that achieved using object-orientation only. MSTP obtained excellent results compared with a similar tableau-based theorem prover [Dias 2002].

1.1. Overview

In Section 2 we present the KE System and an example showing the use of strategies in that system. Section 3 discusses the design of the prover and Section 4 presents some remarks on the implementation. Section 5 shows the problems used to evaluate our system and the results obtained. Finally, Section 6 concludes and points to future work.

2. The KE System

The KE System, a tableau method developed by Marco Mondadori and Marcello D'Agostino [D'Agostino and Mondadori 1994], was presented as an improvement, in the computational sense, over Analytic Tableaux [Smullyan 1968]. Here we discuss the version for classical propositional logic, a refutation system that is sound and complete.

We assume familiarilty with the syntax and semantics of propositional classical logic. See [Smullyan 1968] for an introduction. Let us see some conventions used throghout this paper. A *signed formula* is an expression $S\ X$ where S is called the *sign* and X is a propositional *formula*. The symbols $\mathtt{T}$ and $\mathtt{F}$, respectively representing the truth-values true and false, can be used as signs. The *conjugate* of a signed formula $\mathtt{T}\,A$ (or $\mathtt{F}\,A$) is $\mathtt{F}\,A$ (or $\mathtt{T}\,A$).

We define a *proof* in the KE System as a tree whose nodes are lists of signed formulas, here called *branch nodes*. The *root branch node* is the only branch node that does not have a parent branch node. All branch nodes can have two children: the *left branch node* child and the *right branch node* child. A *leaf branch node* is childless. A *branch* is a sequence of branch nodes starting at the root and finishing in a leaf branch node.

When we want to prove that the formulas $B_1, B_2, \ldots, B_n$ follow from $A_1, A_2, \ldots, A_m$, we start a tree with a root branch node containing $\mathtt{T}\,A_1, \mathtt{T}\,A_2, \ldots, \mathtt{T}\,A_m, \mathtt{F}\,B_1, \mathtt{F}\,B_2, \ldots, \mathtt{F}\,B_n$. That means we are trying to falsify $(A_1 \land A_2 \land \ldots \land A_m) \rightarrow (B_1 \lor B_2 \lor \ldots \lor B_n)$. The set of expansion rules for the KE System is presented in Figure 1. For each branch node, we can use expansion rules that take as premises one or more signed formulas that already appear in the branch of this branch node and introduce one or more new signed formulas. These new signed formulas are logical consequences of the premises. The PB rule has no premiss and introduces two branch nodes as children of a given branch node.

The 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. Notice also that all rules are linear, except the PB rule, corresponding to the principle of bivalence. This rule is related to the Cut rule in Gentzen sequent presentations [Gentzen 1969].

$$\begin{array}{l} T\,A \lor B \\ F\,A \\ \hline T\,B \end{array}\ (T\lor 1) \qquad \begin{array}{l} T\,A \lor B \\ F\,B \\ \hline T\,A \end{array}\ (T\lor 2) \qquad \begin{array}{l} F\,A \lor B \\ \hline F\,A \\ F\,B \end{array}\ (F\lor)$$

$$\begin{array}{l} F\,A \land B \\ T\,A \\ \hline F\,B \end{array}\ (F\land 1) \qquad \begin{array}{l} F\,A \land B \\ T\,B \\ \hline F\,A \end{array}\ (F\land 2) \qquad \begin{array}{l} T\,A \land B \\ \hline T\,A \\ T\,B \end{array}\ (T\land)$$

$$\begin{array}{l} T\,A \rightarrow B \\ T\,A \\ \hline T\,B \end{array}\ (T\rightarrow 1) \qquad \begin{array}{l} T\,A \rightarrow B \\ F\,B \\ \hline F\,A \end{array}\ (F\rightarrow 2) \qquad \begin{array}{l} F\,A \rightarrow B \\ \hline T\,A \\ F\,B \end{array}\ (F\rightarrow)$$

$$\begin{array}{l} T\,\neg A \\ \hline F\,A \end{array}\ (T\neg) \qquad \begin{array}{l} F\,\neg A \\ \hline T\,A \end{array}\ (F\neg) \qquad \begin{array}{c} \\ \hline T\,A \,|\, F\,A \end{array}\ (\text{PB})$$

Figure 1. KE tableau expansion rules

A proof terminates when all branches of a tree are closed. It important to notice that closure is not a rule, but a definition. A branch is closed if, for some formula X, $\mathtt{T}\,X$ appears in some branch node and $\mathtt{F}\,X$ also appears in some branch node (possibly not the same) of the branch. That is, a branch is closed when we arrive at a contradiction. If we arrive at a contradicition in all branches of the generated tree, then the sequent is valid. Otherwise, it is not valid.

The *size* of a tableau proof is defined as the sum of the sizes of all branch nodes of the proof tree generated by the use of expansion rules. The size of a branch node is the sum of the size of all its signed formulas. The size of a signed formula is the size of its formula. Finally, the size $s(A)$ of a formula A is defined as:

- $s(A) = 1$ if A is a propositional atom;
- $s(\neg A) = 1 + s(A)$, where A is a formula and
- $s(A \circ B) = 1 + s(A) + s(B)$, where $\circ$ is a binary connective, and A and B are formulas.

The *height* of the proof tree and the number of branch nodes in the tree are other important dimensions of a proof. These are defined as usually for trees.

Let us give an example of proof in the KE System (see the first proof in Figure 2) that will help to illustrate the use of strategies in tableaux. The formula below, called Γ_3, is a tautology:

$$\begin{gathered}((p_1 \vee q_1) \wedge \\ (p_1 \rightarrow (p_2 \vee q_2)) \wedge (q_1 \rightarrow (p_2 \vee q_2)) \wedge \\ (p_2 \rightarrow (p_3 \vee q_3)) \wedge (q_2 \rightarrow (p_3 \vee q_3)) \wedge \\ (p_3 \rightarrow (p_4 \vee q_4)) \wedge (q_3 \rightarrow (p_4 \vee q_4))) \rightarrow \\ (p_4 \vee q_4)\end{gathered}$$

Suppose we want to prove this formula, representing it as the signed formulas 1-8 in Figure 2. First all linear rules are applied. This generates formulas 9-12. Then, one has to choose a formula to apply the PB rule. It is clever to choose a formula that can be used as an auxiliary premise with one of the five formulas (1-5) that were not yet used as main premises. If we first choose the left subformula of 2, the result is a proof with size 71 and 31 nodes. If we use a different strategy, do not expand formula 8 and choose the left subformula of 4 to apply the PB rule, the result is a proof with size 61 and 25 nodes, as can be seen in the second proof of Figure 2.

3. Design

The main purpose of the design of our system is to have a prover where we can vary the proof strategy with the minimum amount of changes in the rest of the system. A strategy for a KE tableau prover will be responsible for: (i) choosing the next rule to be applied, (ii) choosing the formula on which to apply the PB rule and (iii) checking the closure of branches. These features can be scattered in the prover if strategies are not considered first-class citizens. With the use of object-orientation along with aspect-orientation (see Section 4), a strategy can influence several classes of the prover and still be defined in a modular way.

Let us see how some other object-oriented tableau-based provers treat strategies. The here called WDTP [Dias 2002], written in C++, implements Analytic [Smullyan 1968] and KE propositional tableaux methods while jTAP [Beckert et al. 1999] 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 subclass one or more classes of the system, as well as modify existing ones, to implement a new strategy. LOTREC [del Cerro et al. 2001] 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

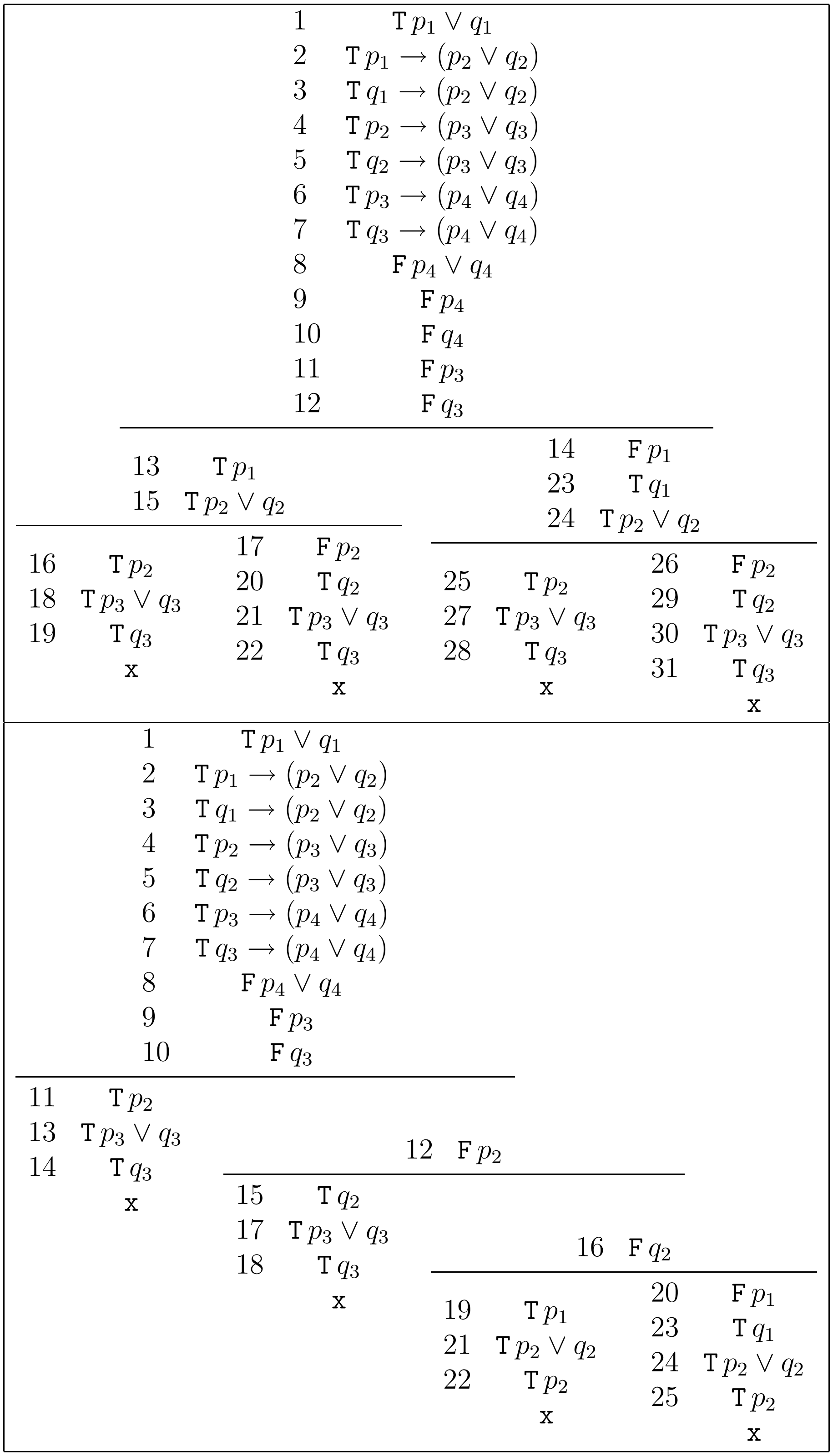

Figure 2. Two proofs of Γ_3.

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.

In our system, instead of the rules in Figure 1, we are using rules based on simplification rules in the style of [Massacci 1998]. These simplification inference rules do not cause branching and in some cases may even prevent it.

For the definition of simplification rules, we use the following notation: $\Phi(A)$ means a formula where A appears as a subformula. For instance, $\Phi(A \rightarrow B)$ can be $A \rightarrow B$, $(A \rightarrow B) \rightarrow B$ or even $(C \rightarrow D) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow B))$. Let us see an example of the development of simplification rules for KE. Instead of "$T \vee 1$" rule in Figure 1, in MSTP we have the "$X \vee F$" rule in Figure 3. Besides that, other rules such as the one in Figure 4 may be added to reduce the size of some formulas. And rules such as the ones in Figure 5 have to be incorporated to deal with the appearance of $\top$ and $\bot$ in some formulas. In MSTP, the "$T \vee 1$" (but not the "$T \vee 2$") rule in Figure 1 is still used, but only for choosing a formula for the application of PB.

$$\frac{\begin{array}{l} X\,\Phi(A \vee B) \\ F\,A \end{array}}{X\,\Phi(B)} \quad (X \vee F)$$

Figure 3. Simplification rule that replaces $T \vee 1$ rule

$$\frac{\begin{array}{l} X\,\Phi(A \vee B) \\ T\,A \end{array}}{X\,\Phi(\top)} \quad (X \vee T)$$

Figure 4. Simplification rule for $\vee$

$$\frac{X\,\Phi(\top \vee A)}{X\,\Phi(\top)} \quad (X \vee \top) \qquad \frac{X\,\Phi(\bot \vee A)}{X\,\Phi(A)} \quad (X \vee \bot)$$

Figure 5. Two auxiliary simplification rules for $\vee$

The architecture of the system is depicted in Figure 6. The main input to the system is a text file that contains a description of an instance of a problem (see Section 5). The Problem Analyser module parses this file and constructs the object that is going to be used by the Prover module. The Prover module also receives as input some configuration options such as the strategy and set of rules to be used. It is this module that actually asks the Strategy module to try to construct a closed proof tree for the problem. The Strategy module contains classes and aspects. It is responsible for changing the behavior of classes in the prover according to the features of the strategy chosen. A better separation of concerns is achieved because the strategy is not in the prover, but rather works alongside the prover. The Profiler module also contains classes and aspects. It tracks and records the performance of the Prover by checking information collected while the code is being executed.

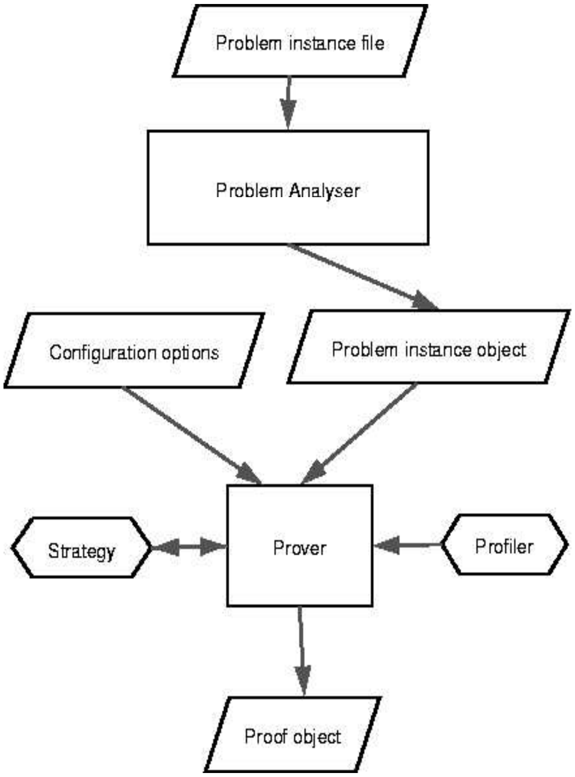

Figure 6. Architecture of the Multi-Strategy tableau Prover

4. Implementation

Our multi-strategy tableau prover was implemented in Java [Gosling et al. 1996] and AspectJ [Kiczales et al. 2001]. Java is a well established object-oriented programming language and AspectJ is an extension of Java that supports a new software development paradigm: aspect-orientation (AO). 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.

By using AO we are able to better modularize strategies. In MSTP, a strategy is represented as a collection of classes and aspects. More precisely, for each strategy implemented, there is a Strategy object that is responsible for the main concerns of that strategy and some aspects that implement other features of the strategy.

The first strategy implemented is called SimpleStrategy. In this strategy, no rule is tried with a signed formula (or pair of signed formulas) if it cannot be applied to it, different from what is done in [Dias 2002]. The strategy keeps a list of PB candidates for every branch. These are the signed formulas that were not used as main premiss of any rule in a given branch (a formula can be used as main premiss in different branches). When a formula is used as main premiss in a branch we say that it was *analysed* in that branch. In the beginning every formula is on the list, except $T\top$ and $F\bot$, that cannot be analysed.

The strategy keeps a stack of open branches. The first branch is put on the top of this stack. The proof continues until the stack is empty. If the stack becomes empty we have to check if the first branch is closed. If it is, the tableau was successfully closed. Otherwise, it was not. When an open branch is exhausted, the proof search stops. Now, suppose there is at least one open branch on the stack. (a) Then the strategy removes the next branch from the top of stack. This becomes the current branch. After that it does the following: applies as much linear rules as possible. If the branch closes using only linear rules, it stops with this branch and goes back to (a). If all possible linear rules are applied, it choses a signed formula X from PB Candidates. If there is no such signed formula, the proof search finished without success. Otherwise, this formula is going be to the main premiss of one of the two premiss rules in Figure 1. The auxiliary premiss and its conjugate are the formulas that will appear in the new branches of the proof tree. Two branches will be put on the top of stack of open branches. The former current branch and the right branch. The left branch becomes the current branch.

This finishes the description of SimpleStrategy. MemorySaverStrategy is the name of our second strategy. It implements almost the same algorithm of SimpleStrategy but keeps the minimum amount of data structures in memory. For instance, when is closes a left branch, it discards it from memory (SimpleStrategy keeps it for recording the proof object). And instead of keeping references to subformulas of formulas in a map, it searches these references every time they are needed. This strategy uses less memory but can take more time and produces a simplified proof object file.

Other strategies can be implemented either from scratch or as variations over these strategies. To do this, one must create a subclass of the Strategy class and write as many aspects and auxiliary classes as necessary to implement the features of the strategy.

5. Evaluation

Theorem provers are usually compared by using benchmarks such as SATLIB [SAT 2003]. We have chosen to evaluate our system using the same families of problems used in [Dias 2002] as benchmarks: Γ, H, Statman and Pigeon Hole Principle (PHP). These families contain explicit propositional valid formulas whose proofs in Analytic Tableaux [Smullyan 1968] tend to be exponential.

Figure 7 shows the results obtained by MSTP and WDTP with some instances of the problems above. In the tables, “x” means that the prover ran out of memory before finishing the instance with a given strategy and “n/a” is placed when the feature was not measured. The tests were run on a personal computer with an Athlon 1100Mhz processor, 384Mb of memory, running a Linux operating system with a 2.26 kernel. For each instance the following features were measured:

- the time (in seconds) needed to finish the proof (excluding the time to parse the problem),
- the number of *occurrences* of signed formulas in branch nodes of the proof tree,
- the height of the proof tree,

- the size of the proof tree, as defined in Section 2.

The proof trees as well as more data about the problems and the execution are avaliable in [Neto 2005].

The results for WDTP in Figure 7 were obtained running it in the same machine using the KE method option. WDTP was implemented in C/C++, which is usually faster than Java. It uses the rules in Figure 1 plus n-ary versions of the rules for the $\wedge$ and $\vee$ connectives but no simplification rule. Its problem analyser parses a formula such as $A \vee B \vee C$ as an n-ary disjunction while MSTP parser generates $A \vee (B \vee C)$ for the same description. A strategy close to the canonical procedure for KE [D'Agostino 1999] is implemented and it closes only on atomic formulas.

From these results one can see that in all cases MSTP is faster and its proof trees have fewer signed formulas and smaller heights[1]. Besides that, some instances that WDTP was not able to close were proved by MSTP. This happened because of the use of simplification rules and because MSTP closes branches on any kind of formula, not only atomic formulas. Notice that MSTP was able to prove PHP_5, PHP_6 and PHP_5 clausal only with Memory Saver Strategy, which does not produce a detailed proof object.

MSTP results

family	instance	time		signed formulas	height	size
		MSS	SS			
H	6	1.926	2.885	605	5	33313
Γ	7	0.333	0.284	53	0	440
Γ	100	23.627	2.567	805	0	2205
Statman	6	0.409	0.311	33	0	440
Statman	21	1.274	2.178	258	0	13425
PHP	4	2.264	2.635	1127	10	4959
PHP	5	35.425	x	n/a	n/a	n/a
PHP	6	764.182	x	n/a	n/a	n/a
PHP (clausal)	4	5.551	6.152	2101	10	10860
PHP (clausal)	5	89.781	x	n/a	n/a	n/a

WDTP results

family	instance	time	signed formulas	height
H	6	75.65	35537	133
Γ	7	4.31	8885	13
Statman	6	9.80	19539	17
PHP	4	4.24	2219	27
PHP	5	183.41	26527	54
PHP	6	8092.56	405217	95
PHP (clausal)	4	17.82	9860	34
PHP (clausal)	5	2041.66	151202	65

Figure 7. Results obtained

[1] We were not able to compare the size of the proof trees generated because WDTP does not measure it.

6. Conclusion

We have presented the design and implementation of MSTP as well as the results obtained with our prover. Thanks to the use of aspect-orientation together with object-orientation, the design and implementation of MSTP achieves a better separation of concerns compared to pure object-oriented tableau provers. The results obtained with benchmark problems showed that this is a promising approach to the implementation of multi-strategy theorem provers. We plan to extend the system to be able to deal also with other logics, such as approximate logics and logics of formal inconsistency.

We are currently working on the development of new strategies for MSTP. For instance, we hope to be able to prove bigger instances of PHP (DPLL-based provers can prove up to PHP_{14} clausal in a few days). We also want to try MSTP with some problems taken from SATLIB [SAT 2003]. and compare new strategies running them with other families of problems for studying the computational complexity of ATP's [Pelletier 1986]. Usually different theorem provers are compared by using benchmarks. As soon as our work is completed, we will be able to test many different strategies using the same core implementation. Such a prover will be useful for education as it will enable the student to see how different strategies behave on the same problem. And from this prover it will be possible to implement an adaptive multi-strategy theorem prover that changes the strategy used according to features of the problem being tackled.

6.1. Acknowledgement

This paper has been sponsored by FAPESP Thematic Project Grant ConsRel 2004/14107-2.

References

Beckert, B., Bubel, R., Habermalz, E., and Roth, A. (1999). jTAP - a Tableau Prover in Java. Universitat Karlsruhe.

Bonacina, M. P. and de la Tour, T. B. (2004). Fifth Workshop on Strategies in Automated Deduction - Workshop Programme. Retrieved March 22, 2005, from http://www.mpi-sb.mpg.de/∼baumgart/ijcar-workshops/proceedings/PDFs/WS2-final.pdf.

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

D'Agostino, M. and Mondadori, M. (1994). The taming of the cut: Classical refutations with analytic cut. *Journal of Logic and Computation*, pages 285–319.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397.

del Cerro, L. F., Fauthoux, D., Gasquet, O., Herzig, A., Longin, D., and Massacci, F. (2001). 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.

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

Elrad, T., Filman, R. E., and Bader, A. (2001). Aspect-Oriented Programming. *Communications of the ACM*, 44.

Fitting, M. (1999). Introduction. In et al., M. D., editor, *Handbook of Tableau Methods*, chapter 1, pages 1–43. Kluwer Academic Press.

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

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

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. G. (2001). An overview of AspectJ. *Lecture Notes in Computer Science*, 2072:327–355.

Massacci, F. (1998). Simplification: A general constraint propagation technique for propositional and modal tableaux. In *TABLEAUX '98: Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 217–231. Springer-Verlag.

Neto, A. G. S. S. (2005). Multi-Strategy Tableau Prover results page. Retrieved March 28, 2005, from http://www.ime.usp.br/∼adolfo/TableauProver.

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

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

SAT (2003). Satisfiability library. Retrieved March 22, 2005, from http://www.satlib.org.

Smullyan, R. M. (1968). *First-Order Logic*. Springer-Verlag.

Sutcliffe, G. (2001). An overview of automated theorem proving. Retrieved March 22, 2005, from http://www.cs.miami.edu/ tptp/OverviewOfATP.html.