A Multi-Strategy Tableau Prover

Adolfo Gustavo Serra Seca Neto and Marcelo Finger

Departamento de Ciência da Computação,
Instituto de Matemática e Estatística,
Universidade de São Paulo,
Rua do Matão, 1010, São Paulo SP, Brazil 05315-970, + 55 11 30916122
e-mail: [adolfo, mfinger]@ime.usp.br

A Multi-Strategy Tableau Prover

Abstract

One of the most important open research problems in computer science nowadays is the "P=NP?" question [5].The answer to this question corresponds to knowing if decision problems that can be solved by a polynomial-time nondeterministic algorithm can also be solved by polynomial-time deterministic algorithm. The inference rules of automated deduction systems are typically non-deterministic in nature. 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 [2]. In this paper we present the design and implementation of a multi-strategy theorem prover based on the KE Tableau System [9]. A multi-strategy theorem prover is a theorem prover where we can vary the strategy without modifying the core of the implementation. It 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 new strategies and compare them with others. 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 method, aspect orientation [12], that allows a better modularization of crosscutting concerns such as strategies. The aspect-oriented Multi-Strategy Tableau Prover whose implementation we present here obtained excellent results compared with a similar tableau-based theorem prover [11].

1 Introduction

An algorithm is a sequence of computational steps that takes a value (or set of values) as input and produces a value (or set of values) as output [6]. A nondeterministic algorithm is an algorithm that is allowed, at certain times, to choose between more than one possible step. Nondeterministic algorithms compute the same class of functions as deterministic algorithms, but the complexity may be much smaller. Nondeterministic algorithms are used in several areas such as automated theorem proving, term-rewriting systems, protocol specification, formal specification, optimization, pattern recognition, and decision making. Every nondeterministic algorithm can be turned into a deterministic algorithm, possibly with exponential slow down. One of the most important open research problems in computer science nowadays is the "P=NP?" question [5].Informally speaking, the answer to this question corresponds to knowing if decision problems that can be solved by a polynomial-time nondeterministic algorithm can also be solved by polynomial-time deterministic algorithm.

When a computer program is written to implement a nondeterministic algorithm, it must have a strategy for choosing the next step that is going to be performed. An interesting example of nondeterminism and the use of strategies is the method of tableaux. It is a formal proof procedure that has many variants and exists for several logics [15]. Automated theorem provers were one of the first applications of computers [10] and still have many applications such as hardware and software verification. Their history is almost as old as that of computing; the first provers were implemented nearly 50 years ago. Most provers discussed in the literature are based on the resolution method [30], but tableau methods have also been found to be a convenient formalism for automating deduction in various non-standard logics as well as in classical logic. The inference rules of automated deduction systems in general and tableau provers in particular are typically non-deterministic in nature. 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 [2].

In this paper we present the design and implementation of a multi-strategy theorem prover based on the KE Tableau System [9]. 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 new strategies and compare them with others. 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 method: aspect orientation [12]. This is another contribution of our work, as we know of no other aspect-oriented theorem prover. We are using aspect-orientation together with object-orientation [29], a well-established software development method. 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. The Multi-Strategy Tableau Prover (MSTP) whose implementation we present here obtained excellent results compared with a similar tableau-based theorem prover [11].

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 [9], was presented as an improvement, in the computational sense, over Analytic Tableaux [32]. 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 [32, 14] for an introduction. Let us see some concepts and definitions used throghout this paper. For propositional logic we construct formulae A, B, C from a set of *propositional atoms* (or *atomic formulas*) $p, q, \ldots \in \mathcal{P}$ and the boolean constants $\top$ and $\bot$ combined with the binary logical *connectives* $\wedge, \vee, \neg, \rightarrow$ and $\leftrightarrow$. We use Γ and Δ to denote sets of formulas. $\bigwedge \Gamma$ and $\bigvee \Gamma$ are, respectively, the conjunction and the disjunction of all formulas in Γ. A sequent is a judgement of the form $\Gamma \vdash \Delta$. It should be read as "from $\bigwedge \Gamma$ we can deduce $\bigvee \Delta$". A sequent $\Gamma \vdash \Delta$ is valid when "$\bigwedge \Gamma \rightarrow \bigvee \Delta$" is a tautology.

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 $\mathtt{T}\,A$).

Following [14], we define the size $s(A)$ of a formula A 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 size of a signed formula is defined as the size of its formula and the size of a list of signed formulas is the sum of the size of all its signed formulas.

We define a *proof* in the KE System as a tree whose nodes are signed formulas. Every proof of a sequent $A_1, A_2, \ldots, A_m \vdash B_1, B_2, \ldots, B_n$ begins by starting a tree with a root branch containing $\mathtt{T}\,A_1, \mathtt{T}\,A_2, \ldots, \mathtt{T}\,A_m \vdash \mathtt{F}\,B_1, \mathtt{F}\,B_2, \ldots, \mathtt{F}\,B_n$. That means we are trying to falsify $A_1, A_2, \ldots, A_m \vdash B_1, B_2, \ldots, B_n$. Then, we can use expansion rules that take as premises one or more signed formulas that already appear in the proof and introduce one or more new signed formulas. These new signed formulas must be logical consequences of the premises. We can only introduce (in a given branch of the tree) signed formulas that can be produced from signed formulas that appear in that same branch.

The set of expansion rules for the KE System is presented in Figure 1. 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, that divides the tree into two branches. This rule is related to the Cut rule in Gentzen sequent presentations [17]. Notice also that some rules have two premises, some have one premise and the PB rule is a zero premise rule. We say that the rules with two premises have a *main premise* (the one at the top) and an *auxiliary premise* (the other). For the rules with only one premise, this premise is also referred as *main premise*.

When does a proof terminate? It terminates when all branches of a tree are closed. It important to notice that closure is not a rule, but a definition [28]. A branch is closed if it

Disjunction Rules

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

Conjunction Rules

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

Implication Rules

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

Negation Rules

$$\dfrac{T\,\neg A}{F\,A}\ (T\neg) \qquad \dfrac{F\,\neg A}{T\,A}\ (F\neg)$$

Principle of Bivalence

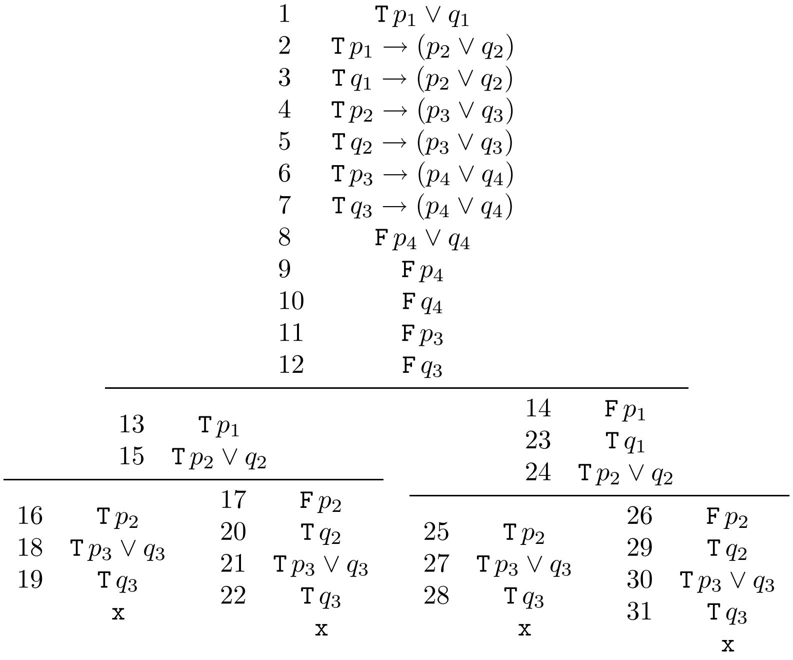
$$\dfrac{}{T\,A \mid F\,A}\ (\mathrm{PB})$$

Figure 1: KE tableau expansion rules

contains T X and F X for some formula X, that is, when we arrive at a contradiction. That is, if we arrive at a contradicition in all branches of the generated tree, then the sequent is valid. Otherwise, it is not valid.

We define the *size* of a tableau proof as the number of nodes in the proof tree generated by the use of expansion rules. The *height* of the proof tree and the number of nodes in the tree are other important dimensions of a proof. These are defined as usually for trees [6].

Let us give an example of proof in the KE System (see Figure 2) that will help to illustrate the use of strategies in tableaux. Suppose we want to prove the Γ_3 problem (see subsection 5.1.1). 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 Figure 3.



1 $\mathtt{T}\, p_1 \vee q_1$
2 $\mathtt{T}\, p_1 \rightarrow (p_2 \vee q_2)$
3 $\mathtt{T}\, q_1 \rightarrow (p_2 \vee q_2)$
4 $\mathtt{T}\, p_2 \rightarrow (p_3 \vee q_3)$
5 $\mathtt{T}\, q_2 \rightarrow (p_3 \vee q_3)$
6 $\mathtt{T}\, p_3 \rightarrow (p_4 \vee q_4)$
7 $\mathtt{T}\, q_3 \rightarrow (p_4 \vee q_4)$
8 $\mathtt{F}\, p_4 \vee q_4$
9 $\mathtt{F}\, p_4$
10 $\mathtt{F}\, q_4$
11 $\mathtt{F}\, p_3$
12 $\mathtt{F}\, q_3$
13 $\mathtt{T}\, p_1$
15 $\mathtt{T}\, p_2 \vee q_2$
14 $\mathtt{F}\, p_1$
23 $\mathtt{T}\, q_1$
24 $\mathtt{T}\, p_2 \vee q_2$
16 $\mathtt{T}\, p_2$
18 $\mathtt{T}\, p_3 \vee q_3$
19 $\mathtt{T}\, q_3$
x
17 $\mathtt{F}\, p_2$
20 $\mathtt{T}\, q_2$
21 $\mathtt{T}\, p_3 \vee q_3$
22 $\mathtt{T}\, q_3$
x
25 $\mathtt{T}\, p_2$
27 $\mathtt{T}\, p_3 \vee q_3$
28 $\mathtt{T}\, q_3$
x
26 $\mathtt{F}\, p_2$
29 $\mathtt{T}\, q_2$
30 $\mathtt{T}\, p_3 \vee q_3$
31 $\mathtt{T}\, q_3$
x

Figure 2: A proof of Γ_3

3 Design

Our first step towards the design and implementation of our MSTP was the implementation of an object-oriented single-strategy tableau prover [25]. That prover was also based on the KE Tableau System [9]. We know of only two other object-oriented tableau-based provers: the here called WDTP [11] and jTAP [1]. WDTP was written in C++. It implements Analytic [32], KE and KE-S_3 [13] propositional tableaux methods. And jTAP is a propositional tableau prover written inJava. It is based on the method of signed Analytic Tableaux. Both systems have some strategies implemented and can be extended with new strategies. In both cases one has to subclass one or more classes of the system, as well as modify some others, to implement a new strategy. Strategies are not well modularized.

Our purpose in the design of this 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. Having this in mind, let us first try to make it clear what a strategy for a KE tableau prover will be responsible for. Maybe the most important task of a strategy to choose the next rule to be applied. The ordering between rules and between formulas must be taken into consideration.

A strategy must also decide when to apply the PB rule and with which formula. It seems natural that a PB rule is applied so that at least one of the pair of conjugate formulas introduced

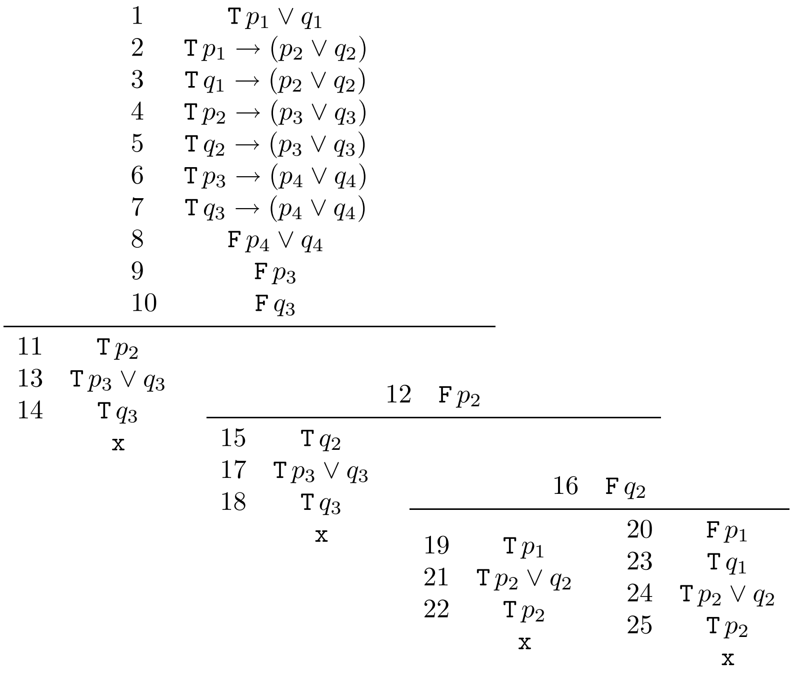


Figure 3: Another proof of Γ_3

in the two new branches can be used as an auxiliary premise of a two-premise rule. So the strategy can chose the PB formula based on the formulas that can be auxiliary premise of a two-premise rule that were not yet analysed.

To check the closure of branches is another responsibility of strategies. Some systems close a branch only when encounter T A and F A where A is an atomic formula. Others allow closing on any kind of formula. This second approach is more difficult to implement and uses more memory but produces smaller proofs.

3.1 Simplification rules

In our implementation, instead of the rules in Figure 1, we are using rules based on simplification rules in the style of [24]. 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 notations: $\Phi(A)$ means a formula where A appears as a subformula. For instance, $\Phi(A)$ can be A, $A \to B$, $C \leftrightarrow (D \to A)$ or even $(D \to D) \to (A \to A)$. $\Phi(A \to B)$ can be $A \to B$, $(A \to B) \to B$, $C \leftrightarrow (D \to (A \to B))$ or even $(D \to D) \to ((A \to B) \to (A \to B))$. A, B, C, D can be any formula (not only atomic formulas).

Let us see an example of the development of simplification rules for KE. Instead of "$T \lor 1$" rule in Figure 1, in MSTP we have the "$X\Phi \lor F$ left" rule in Figure 4. Besides that, other rules such as the one in Figure 5 may be added to reduce the size of some formulas. And rules such as the ones in Figure 6 have to be incorporated to deal with the appearance of $\top$ and $\bot$ in some formulas. In MSTP, the "$T \lor 1$" (but not the "$T \lor 2$") rule in Figure 1 is still used, but only for choosing a formula for the application of PB.

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

Figure 4: Simplification rule that replaces $T \lor 1$ rule

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

Figure 5: Simplification rule for $\vee$

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

Figure 6: Two auxiliary simplification rules for $\vee$

3.2 Architecture of the system

The architecture of the system is depicted in Figure 7. 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.

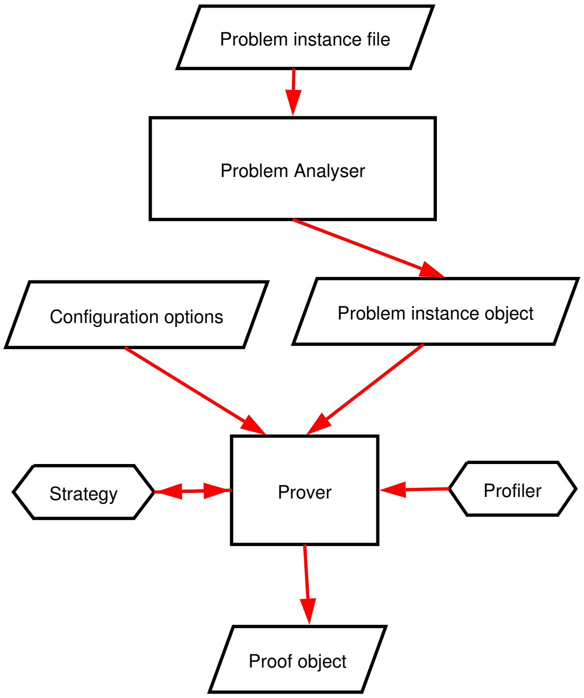


Figure 7: Architecture of the Multi-Strategy tableau Prover

4 Implementation

Our multi-strategy tableau prover was implemented in Java [18] and AspectJ [22, 21]. 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.

Let us describe how the prover works. It takes as input a list of signed formulas and outputs a proof tree. Therefore the first thing to be done is to parse a problem text file. This is done by the problem analyser module, that generates a list of signed formulas objects in memory. Each signed formula is an object composed of two other objects: a sign and a formula. For classical logic there are only two signs and these are shared by all signed formulas.

The formulas are also shared because we implemented the Flyweight design pattern [16]. That is, there is only one instance of every formula (and also of every signed formula). This allowed us to save space as well as served to simplify the search for subformulas of a formula, and for signed formulas where a formula appeared.

The first strategy we 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 [25, 11].

Since we are working with closure as a definition, in the beginning, the formulas $T\top$ and $F\bot$ are added as the first formulas of the proof tree. Therefore, anytime $T\bot$ or $F\top$ is added to any branch, it becomes closed.

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 that means that all of its child branches are also closed. If it is not then at least one branch remained open and exhausted — there were left no possible rules to apply).

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. Other strategies can be implemented either from scratch or as variations over this strategy. 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.

4.1 Technologies used

Besides Java and AspectJ, we have used the JFlex [23] package for implementing the lexical analysis of problems and the CUP [19] package for syntactical analysis. Using these packages we have built parsers for SATLIB SAT format [31], SATLIB CNF format [31] as well as the format used by [11]. In addition, we have also used a package called JDOM [20] for generating XML files that represent proof trees.

5 Evaluation

Theorem provers are usually compared by using benchmarks [34]. We have chosen to evaluate our system using the same families of problems used in [11]. These families contain explicit propositional valid sequents whose proofs in Sequent Calculus [17] or Analytic Tableaux [32] tend to be exponential. The families of problems and the results obtained by our prover compared with those obtained by WDTP [11] are presented below.

5.1 Families of problems

5.1.1 Γ formulas

The n-th instance of the Γ family [4] has $2n$ propositional variables and $2n+2$ formulas. It is possible to find both exponential and non-exponential (in n) proofs of instances of this family using Analytic Tableaux. For the n-th instance, Γ_n, the sequent to be proved is:

$$p_1 \vee q_1, \gamma_n \vdash p_{n+1} \vee q_{n+1}$$

where

$$\gamma_n = \{p_i \rightarrow (p_{i+1} \vee q_{i+1}), q_i \rightarrow (p_{i+1} \vee q_{i+1}) | 1 \leq i \leq n\}$$

5.1.2 H formulas

For this family, the sequent to be proved for the n-th instance is

$$\vdash H_n$$

where H_n is constructed in the following way:

$$\begin{aligned}
&H_1 = p_1 \vee \neg p_1 \\
&H_2 = (p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2) \vee (\neg p_1 \wedge p_2) \vee (\neg p_1 \wedge \neg p_2) \\
&H_3 = (p_1 \wedge p_2 \wedge p_3) \vee \ldots \vee (\neg p_1 \wedge \neg p_2 \wedge \neg p_3) \\
&\vdots
\end{aligned}$$

These formulas (also called “truly fat” formulas) were defined in [7] and used to prove that the Analytic Tableaux method cannot polynomially simulate the truth-table method. Each instance of this family has n propositional variables, but the size of the formula H_n grows exponentially in n (because each instance has 2^n clauses $p_1 \wedge \ldots \wedge p_n$).

5.1.3 Statman formulas

Statman formulas [33] can be constructed as follows. Consider

$$\begin{aligned}
&A_k = \textstyle\bigwedge_{j=1}^{k} (p_j \vee q_j) \\
&B_1 = p_1 \\
&C_1 = q_1
\end{aligned}$$

and, inductively:

$$B_{i+1} = A_i \rightarrow p_{i+1} \qquad C_{i+1} = A_i \rightarrow q_{i+1}$$

The sequent to be proved for the n-th instance of this family is:

$$B_1 \vee C_1, \ldots, B_n \vee C_n \vdash p_n \vee q_n$$

Statman has proved that this family of formulas has polynomial proofs in sequent calculus if we use the cut rule, but only exponential size cut-free proofs [4].

5.1.4 Pigeon Hole Principle formulas

The Pigeon Hole Principle [27, 4] states that given $n-1$ pigeon holes and n objects to be put in these holes, there is always one hole that will receive at least two objects.

The sequent to be proved is

$$\vdash \mathrm{PHP}_n$$

where

$$\mathrm{PHP}_n = A_n \rightarrow B_n$$

and

$$A_n = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n-1} p_{ij}$$

$$B_n = \bigvee_{i=1}^{n-1} \bigvee_{k=i}^{n} \bigvee_{j=1}^{n-1} (p_{ij} \wedge p_{kj})$$

where p_{ij} expresses that object i was inserted in the j hole, A_n that every object goes to some hole and B_n that at least one hole receives 2 objects.

This problem leads to a lot of branching in Analytic Tableaux and Sequent Calculus. In Sequent Calculus, cut-free proofs are exponential but there is an extremely complicated polynomial proof with cut [3].

5.2 Results

Figure 8 shows the results obtained by MSTP and WDTP with some instances of the problems above. 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) to finish the proof (excluding the time to parse the problem),
- the number of 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 [26].

The results for WDTP in Figure 8 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. Therefore, it uses no simplification rules. 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. It implements a strategy close to the canonical procedure for KE [8]. And finally, it closes only on atomic formulas.

From these results one can see that in most cases MSTP is much faster and its proof trees have fewer nodes 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. The only advantage of WDTP happened with an instance of the PHP family. WDTP was able to prove PHP$_5$ while MSTP was not. This happened because the strategy described in Section 4 consumes too much memory. So, MSTP ran out of memory before the proof of PHP$_5$ completed. We are currently implementing a strategy to solve this problem.

		MSTP				WDTP		
family	instance	time	nodes	height	size	time	nodes	height
H	4	0.693	117	3	1665	0.181691	105	18
H	5	1.801	269	4	7459	3.430723	466	51
H	6	10.693	605	5	33313	101.803705	1953	132
Γ	7	0.143	53	0	440	4.319546	2917	12
Γ	80	2.145	645	0	1765	-	-	-
Γ	100	3.788	805	0	2205	-	-	-
Statman	6	0.302	33	0	440	9.807499	5391	16
Statman	21	3.013	258	0	13425	-	-	-
PHP	4	3.089	1127	10	4959	4.247875	483	26
PHP	5	-	-	-	-	229.452014	5409	53
PHP (in clausal form)	4	6.497	2101	10	10860	17.826073	2147	33

Figure 8: Results obtained by MSTP and WDTP

[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 than would be possible by using only object-orientation. The results obtained show 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 modal, approximate and paraconsistent logics.

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. We also want to compare these strategies running them with several families of problems. Usually different theorem provers are compared by using benchmarks. As soon as our work is completed, we will be able to test different strategies using the same core implementation.

References

[1] Bernhard Beckert, Richard Bubel, Elmar Habermalz, and Andreas Roth. jTAP - a Tableau Prover in Java. Universitat Karlsruhe, February 1999.

[2] Maria Paola Bonacina and Thierry Boy de la Tour. Fifth Workshop on Strategies in Automated Deduction - Workshop Programme, 2004. http://www.mpi-sb.mpg.de/∼baumgart/ijcar-workshops/proceedings/PDFs/WS2-final.pdf.

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

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

[5] Stephen Cook. The P versus NP problem, 2000. http://www.claymath.org/millennium/P_vs_NP/Official_Problem_Description.pdf.

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

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

[8] 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.

[9] 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.

[10] M. Davis. The early history of automated deduction. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 1, pages 3–15. Elsevier Science, 2001.

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

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

[13] M. Finger and R. Wassermann. Tableaux for approximate reasoning. In Leopoldo Bertossi and Jan Chomicki, editors, *IJCAI-2001 Workshop on Inconsistency in Data and Knowledge*, pages 71–79, 2001.

[14] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, second edition, 1996.

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

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

[17] 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.

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

[19] Scott Hudson, Frank Flannery, C. Scott Ananian, Dan Wang, and Andrew W. Appel. CUP User's Manual, 1999. Available at http://www.cs.princeton.edu/∼appel/modern/java/CUP/.

[20] JDOM, 2005. Available at http://www.jdom.org. Last accessed in Feb 26th, 2005.

[21] 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.

[22] 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.

[23] Gerwin Klein. JFlex User's Manual, 2001. Available at http://www.jflex.de.

[24] Fabio Massacci. 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, 1998.

[25] Adolfo Gustavo Serra Seca Neto. An Object-Oriented Implementation of a KE Tableau Prover, November 2003. Avaliable at http://www.ime.usp.br/∼adolfo.

[26] Adolfo Gustavo Serra Seca Neto. Multi-Strategy Tableau Prover results page, November 2005. http://www.ime.usp.br/∼adolfo/TableauProver.

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

[28] 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.

[29] Charles Richter. *Designing Flexible Object-Oriented Systems with UML*. Macmillan Technical Publishing, 1999.

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

[31] Satisfiability suggested format, 1993. Available at http://www.satlib.org.

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

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

[34] Geoff Sutcliffe and Christian Suttner. The CADE ATP System Competition, 2003. Available at http://www.cs.miami.edu/∼tptp/CASC.