Using Aspect-Oriented Programming in the Development of a Multi-strategy Theorem 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

Using Aspect-Oriented Programming in the Development of a Multi-strategy Theorem Prover

Abstract

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. Automated Theorem Provers (ATP's) usually implement nondeterministic algorithms, therefore the representation of strategies is a very important part of their design. A multi-strategy theorem prover is an ATP where we can vary the strategy without modifying the implementation. In this paper we present some remarks towards the development of an aspect-oriented multi-strategy prover.

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 [2]. A nondeterministic algorithm is an algorithm that is allowed, at certain times, to choose between more than one possible steps. Nondeterministic algorithms compute the same class of functions as deterministic algorithms, but the complexity may be much less. Nondeterministic algorithms are used in several areas such as automated theorem proving, term-rewriting systems and protocol specification.

Every nondeterministic algorithm can be turned into a deterministic algorithm, possibly with exponential slow down. For instance, there are some problems for which no polynomial-time deterministic algorithm is known, but there is an exponential-time deterministic algorithm that is obtained by testing all possibilities of a polynomial-time nondeterministic algorithm. One of the most important open research problems in computer science nowadays is the "P=NP?" question. 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 [5]. It is a refutational procedure. That is, in order to prove that a formula X is valid we try to show that it is not valid. Having this in mind, we apply a procedure for inferring the logical consequences of the formulas present in the tableaux. This procedure applies a strategy for choosing the next expansion rule to be applied amongst possibly many applicable rules. A tree is generated by this procedure and if all branches of the tree close (a branch is closed when we find a contradiction) then we have a proof of X. Otherwise, if we apply all possible rules and at least one branch of tree remains open, we have a refutation of X.

Automated theorem provers were one of the first applications of computers 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 almost 50 years ago. Most provers discussed in the literature are based on the resolution method, 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.
We have just finished the implementation of a single-strategy object-oriented version of a tableau prover, implemented in Java [8]. Our prover is based on the KE Tableau System [3], developed by Marco Mondadori and Marcello D'Agostino. We know of two other object-oriented tableau-based provers: the system presented in [4] and jTAP [1].
In our work we are investigating the construction of multi-strategy theorem provers. A multi-strategy theorem prover is a prover where we can vary the strategy without modifying the implementation. Usually different theorem provers (using distinct implementations techniques) are compared by using benchmarks [10]. Our first objective is to be able to test different strategies in the same implementation. Another objective is to investigate if proof strategies for tableau provers can be well modularized by using object-oriented and aspect-oriented software development. Right now we have three questions:

- how can we use object-orientation and aspect-orientation to achieve modularity in the definition of proof strategies?
- how should we represent a proof strategy: as an object, an aspect or a composition of aspects?
- can we represent features of proof strategies as aspects?

In this paper we will present our initial ideas towards the construction of aspect-oriented multi-strategy provers. In Section 2 we present the KE System and an example showing the use of strategies in that system. Sectionr̃efinitial presents some remarks on the design and implementation of a multi-strategy prover using aspect-orientation. Section 4 concludes and points to some future work.

2 The KE System

The KE System, a tableau method presented by Marco Mondadori and Marcello D'Agostino [3], is an improvement, in the computational sense, over traditional tableaux [9]. It is a refutation system for classical propositional logic that is sound and complete.
Informally, a proof in the KE System is a tree whose nodes are signed formulas. A *signed formula* is an expression $T\,X$ or $F\,X$ where X is a (unsigned) formula and the symbols T and F represent the truth-values true and false. Every proof of a formula X begins by starting a tree with $F\,X$ as the root node. Then, we use expansion rules that take as premises one or more signed formulas that already appear in the tree and introduce one or more new signed formulas. These new signed formulas are 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 use. Notice also that there is only one rule that divides the tree into branches, the PB rule, corresponding to the principle of bivalence; all other rules are linear.
When does a proof terminate? It terminates when all branches of a tree are closed. A branch is closed if it contains $T\,X$ and $F\,X$ for some formula X, that is, when we arrive at a contradiction. Therefore, the method can be described as follows: if we want to prove a formula X, we start by supposing it is false. Then, we apply a procedure for inferring the logical consequences of that supposition (that is done by applying expansion

Disjunction Rules

$$\frac{T\,A \vee B \quad F\,A}{T\,B}\ (\mathrm{E}T\vee 1) \qquad \frac{T\,A \vee B \quad F\,B}{T\,A}\ (\mathrm{E}T\vee 2) \qquad \frac{F\,A \vee B}{\begin{array}{l}F\,A\\F\,B\end{array}}\ (\mathrm{E}F\vee)$$

Conjunction Rules

$$\frac{F\,A \wedge B \quad T\,A}{F\,B}\ (\mathrm{E}F\wedge 1) \qquad \frac{F\,A \wedge B \quad T\,B}{F\,A}\ (\mathrm{E}F\wedge 2) \qquad \frac{T\,A \wedge B}{\begin{array}{l}T\,A\\T\,B\end{array}}\ (\mathrm{E}T\wedge)$$

Implication Rules

$$\frac{T\,A \to B \quad T\,A}{T\,B}\ (\mathrm{E}T\to 1) \qquad \frac{T\,A \to B \quad F\,B}{F\,A}\ (\mathrm{E}F\to 2) \qquad \frac{F\,A \to B}{\begin{array}{l}T\,A\\F\,B\end{array}}\ (\mathrm{E}F\to)$$

Negation Rules

$$\frac{T\,\neg A}{F\,A}\ (\mathrm{E}T\neg) \qquad \frac{F\,\neg A}{T\,A}\ (\mathrm{E}F\neg)$$

Principle of Bivalence

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

Figure 1: KE tableau expansion rules

rules). If we arrive at a contradicition in all branches of the generated tree, then the formula is a tautology. Otherwise, it is satisfiable.

1	$\mathtt{F}\ (((A \wedge (A \rightarrow B) \wedge (A \rightarrow C)) \wedge (A \rightarrow D)) \rightarrow C$
2	$\mathtt{T}\ ((A \wedge (A \rightarrow B) \wedge (A \rightarrow C)) \wedge (A \rightarrow D)$
3	$\mathtt{F}\ C$
4	$\mathtt{T}\ (A \wedge (A \rightarrow B) \wedge (A \rightarrow C))$
5	$\mathtt{T}\ A \rightarrow D$
6	$\mathtt{T}\ A$
7	$\mathtt{T}\ (A \rightarrow B) \wedge (A \rightarrow C)$
8	$\mathtt{T}\ A \rightarrow B$
9	$\mathtt{T}\ A \rightarrow C$
10	$\mathtt{T}\ D$
11	$\mathtt{T}\ B$
12	$\mathtt{T}\ C$
	X

Figure 2: A proof of $(((A \wedge (A \rightarrow B) \wedge (A \rightarrow C)) \wedge (A \rightarrow D)) \rightarrow C$

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 $(((A \wedge (A \rightarrow B) \wedge (A \rightarrow C)) \wedge (A \rightarrow D)) \rightarrow C$ in the KE System. This formula is a tautology in propositional classical logic, therefore it can be proved in the KE System. We begin the tableau by putting $\mathtt{F}\ (((A \wedge (A \rightarrow B) \wedge (A \rightarrow C)) \wedge (A \rightarrow D)) \rightarrow C$ as the root node, that is, suuposing that it is false. Then by using KE rules we add formulas 2-9 to the tableau. Among other things, we arrive at $\mathtt{T}\ A$ (node 6) and $\mathtt{T}\ A \rightarrow C$ (node 9). Using "E$T \rightarrow 1$" rule with these two formulas, we can obtain $\mathtt{T}\ C$ and arrive at a contradiction with $\mathtt{F}\ C$ in node 10. But as we had other rules that could be applied, and the strategy used here applies all rules that can be applied top-down, the contradiction is introduced only in node 12. If we had used a different strategy, the proof could be shorter.

3 Design

Here we will describe some of our initial ideas towards the design and implementation of an aspect-oriented multi-strategy theorem prover. A simplified class diagram is presented in Figure 3. An object of the KETableau class is instantiated for a formula whose tableau we want to build. For instance, we can create a KETableau for the $A \rightarrow B$ formula. Then, the $FA \rightarrow B$ formula is included as the root of the tableau. The tree that represents the tableau is an object of the ProofTree class. Every object of this class can have a right and/or a left child, as well as a list of formulas.
In the instantiation of an object of the KETableau class, we must also pass as parameters two factory objects: a FormulaFactory object and a SignedFormulaFactory object. This happens because we are using the Flyweight design pattern [6] to prevent the multiplication of objects representing formulas and signed formulas as well as making it easier to implement the choice and application of rules. By using this pattern, when we want to compare two formulas, we have only to compare two pointers instead of two strings.

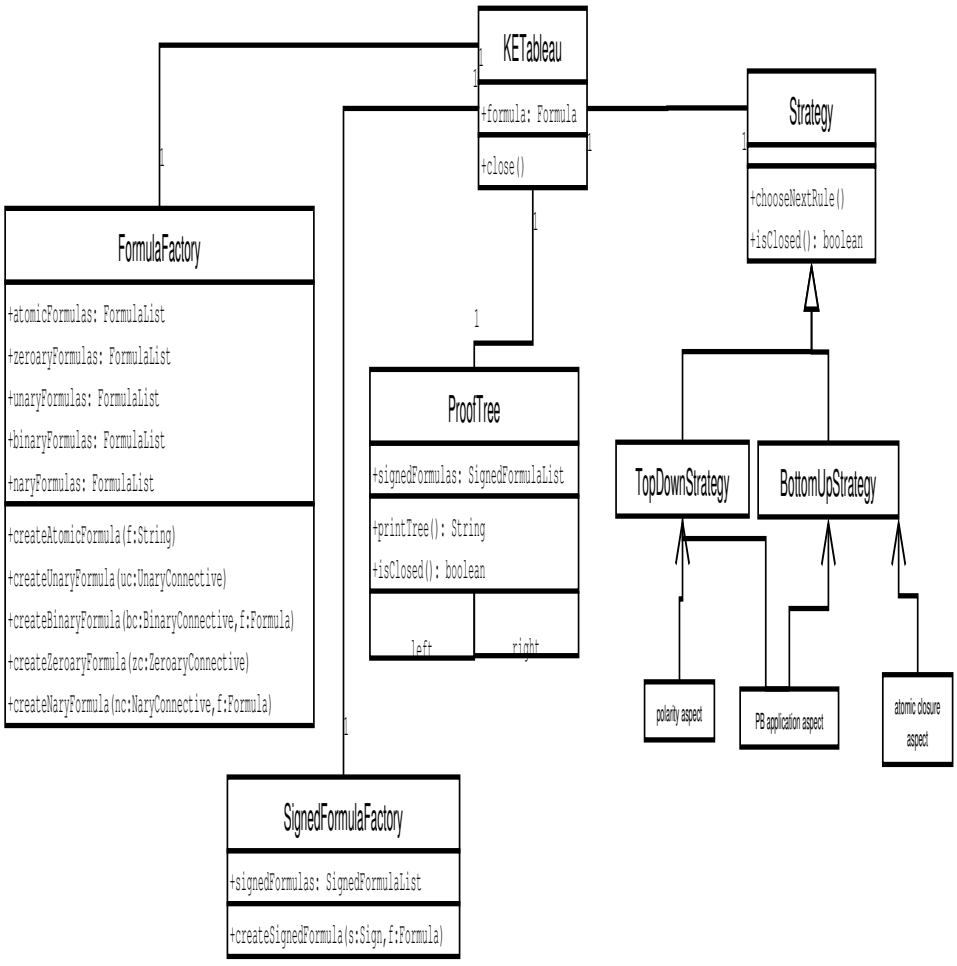

Figure 3: Simplified class diagram of the KE Tableau Prover.

For instance, in the $A \rightarrow (B \vee \neg A)$ we have two occurrences of the atomic formula A. By using the Flyweight pattern, only one object of the AtomicFormula class representing the A formula is created.

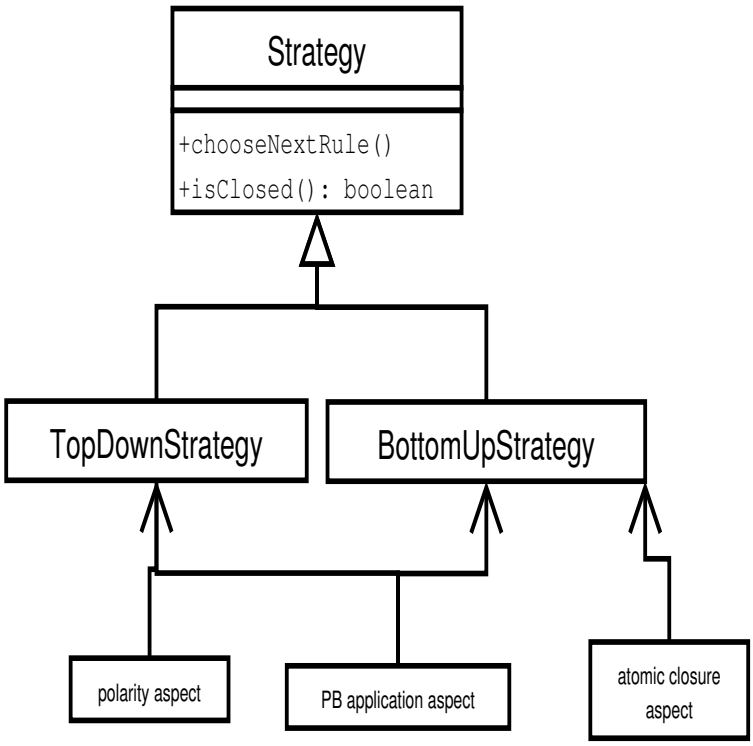

Figure 4: Strategy class hierarchy with aspects.

A strategy is going to be represented as a composition of aspects. More precisely, features of strategies will be aspects that may affect the hierarchy of classes representing strategies as well as other classes in the system. In Figure 4, we have depicted a possible class diagram for the Strategy class. A Strategy object implements an algorithm for choosing the next rule to be applied and for answering if the tableau is closed.
Two examples of simple classes of strategies are top-down and bottom-up strategies. A top-down strategy scans the list of formulas top-down, applying each rule that can be applied, while a bottom-up strategy scans the list from the bottom-up. More sofisticated ways to choose the next rule to be applied could give origin to other classes of strategies. And to every one of these classes we could apply the "atomic closure" aspect, that establishes that tableau branches can be closed only with atomic formulas. Or even the "PB rule application aspect" that decides when and how to apply the PB rule. Another possible aspect would be the "polarity aspect", that chooses the next rule based on a polarity established for the subformulas of the formulas present in the tableau.

4 Conclusion

In this paper we have presented some remarks towards the development of an aspect-oriented multi-strategy prover. Our design represents a strategy a composition of aspects. More precisely, features of strategies will be aspects that may affect the hierarchy

of classes representing strategies as well as other classes in the system.
We are implementing this prover using Java and AspectJ [7]. Soon we will be able to compare the hierarchy of classes in our design with that of jTAP [1] to see if concerns were better separated by using aspect-orientation. As soon as the prover is implemented we will also be able to compare benchmarks for several problems using different strategies. And in the future we may extend the system for classical predicate logic or even non classical logics.

References

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

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

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

[4] Wagner Dias. *Implementações de Tableaux para Raciocínio por Aproximações*. Master's thesis, Departamento de Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo, 2002.

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

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

[7] 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.

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

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

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