ADOLFO GUSTAVO SERRA SECA NETO

CONSTRUÇÃO DE PROVADORES AUTOMÁTICOS DE TEOREMAS UTILIZANDO DESENVOLVIMENTO DE SOFTWARE ORIENTADO A ASPECTOS

SÃO PAULO - 2003

Sumário

1 Introdução

Neste projeto pretendemos estudar a construção de provadores automáticos de teoremas baseados em Tableaus. Para isto, iremos utilizar orientação a objetos, padrões de projeto, orientação a aspectos e algumas técnicas de programação extrema. Veremos abaixo o que são cada um desses itens. O resultado será um provador para lógica proposicional clássica baseado em tableaus. Ele será implementado em Java e AspectJ e durante seu desenvolvimento pretendemos utilizar, sempre que for adequado, padrões de projeto e algumas técnicas de programação extrema.

1.1 Provadores Automáticos de Teoremas

A construção de Provadores Automáticos de Teoremas (PAT´s) é tema de pesquisa na área de Ciência da Computação já há bastante tempo [29] e foi tema de diversos livros como [36] e [16]. São várias as aplicações dos PAT´s, dentre as quais podemos destacar [5]:

- prova de teoremas matemáticos
  - verificação de provas: detectar e corrigir erros em provas
  - prova de teoremas: encontrar novas provas automaticamente
  - auxílio ao ensino de matemática
- apoio ao desenvolvimento de software confiável
  - verificação: encontrar erros ou provar correção
  - otimização: aumentar o desempenho
  - síntese: gerar código a partir de especificações
- motor de inferência (*inference engine*) para sistemas em IA
  - resolvedor geral de problemas
  - planejamento de robôs
  - agentes inteligentes

Atualmente existem vários PAT´s baseados em diferentes técnicas (Resolução, Dedução Natural, Tableaus) e implementados em diferentes linguagens (C, Prolog, Java, entre outras). Uma listagem bastante abrangente de sistemas conhecidos pode ser encontrada em [25]. Neste trabalho iremos nos concentrar em PAT's baseados em Tableaus.

1.1.1 Tableaus

O método dos tableaus é um procedimento de prova formal que tem muitas variantes e existe para diferentes lógicas [17]. Entre outras aplicações importantes, é utilizado para o desenvolvimento de PAT´s. Foi com Smullyan [32] que os tableaus ficaram conhecidos. Em seu livro ele apresenta um sistema chamado "Tableaus Analíticos". O método dos tableaus é um procedimento refutacional. Isto é, para provar que uma fórmula X é válida nós tentamos mostrar que ela não é válida. Tendo isto em mente, nós aplicamos um procedimento para inferir as conseqüências lógicas das fórmulas presentes no tableau. Se algum ramo da árvore de prova gerada permanecer aberto, temos uma refutação de X.

Dois provadores automáticos de teoremas baseados em tableaus servirão como base para nossos estudos: o sistema apresentado em [11] e o jTAP [2]. Além de serem baseados em tableaus, ambos foram implementados em linguagens de programação orientadas a objetos.

1.2 Orientação a Objetos

O paradigma de desenvolvimento orientado a objetos é considerado atualmente o paradigma dominante em desenvolvimento de software. Um objetivo importante da orientação a objetos é o reuso. Reuso de código, assim como de análise e de projeto. Para isto, diversos conceitos básicos estão presentes nas linguagens de modelagem e programação orientadas a objetos, como herança, encapsulamento, abstração e composição. A principal linguagem de modelagem de sistemas, a UML (*Unified Modeling Language* – linguagem de modelagem unificada) [10], assim como as linguagens de programação mais utilizadas hoje em dia (como C++ e Java) baseiam-se neste paradigma.

1.2.1 Padrões de Projeto

A utilização de padrões de projeto [19] têm se tornado uma dos mais importantes ferramentas para os projetistas de sistemas orientados a objetos. O surgimento dos padrões de projeto foi motivado pela observação seguinte: "projetar software orientado a objetos é difícil, e projetar software orientado a objetos *reusável* é mais difícil ainda" [19]. Padrões de projeto são descrições abstratas de soluções de projeto que se repetem em sistemas bem projetados. Eles tornam mais fácil a reutilização de arquiteturas e projetos bem-sucedidos. O livro [19] apresentou os primeiros 23 padrões de projeto. Atualmente, na literatura podem ser encontrados inúmeros outros padrões para as mais diferentes aplicações.

1.3 Orientação a Aspectos

O paradigma orientado a objetos tem algumas limitações como o entrelaçamento e o espalhamento de código com diferentes propósitos. Parte dessas limitações podem ser compensadas com o uso de padrões de projeto. O desenvolvimento de software orientado a aspectos (mais conhecido como Programação Orientada a Aspectos) é uma tentativa de solucionar estes e outros problemas identificados no paradigma orientado a objetos. É uma técnica que busca obter mais modularidade em situações onde a orientação a objetos e a utilização de padrões de projeto não é suficiente.

A Programação Orientada a Aspectos foi proposta como uma técnica para melhorar a separação de interesses (*separation of concerns*) em software [23]. Ela aumenta a modularidade separando código que implementa funções específicas e afeta diferentes partes do sistema, as chamadas "preocupações ortogonais" (*crosscutting concerns*) [33]. Com o aumento da modularidade, busca-se obter sistemas que sejam mais fáceis para projetar, compreender e manter.

AspectJ [23] é uma extensão orientada a aspectos, de propósito geral, à linguagem de programação Java. Nela são implementadas diversas construções que permitem a programação orientada a aspectos com uma sintaxe similar à de Java. É uma das principais implementações da orientações a aspectos atualmente.

1.4 Programação Extrema

Programação Extrema (XP) é uma metodologia de desenvolvimento de software que considera a codificação a principal tarefa no desenvolvimento [21]. Dentre as chamadas "doze práticas de XP" apresentadas em [21], iremos utilizar neste projeto as seguintes:

1. "Projeto simples": A idéia por trás do "Projeto simples" é manter o código simples. O projeto mais simples possível não tenta resolver problemas futuros tentando adivinhar necessidades futuras.

2. "Testes": A prática de testes é essencial para XP. Os testes devem ser automatizados para que o código possa ser modificado e refatorado com segurança, sem que haja a inclusão de defeitos no sistema.

3. "Refatoração" (*Refactoring*): O ato de refatorar permite que os desenvolvedores acrescentem características mantendo o código simples. A idéia é não duplicar código nem escrever código "feio". O ato de refatorar depende

dos testes para confirmar que o código ainda funciona. Refatoração não é limitada à adição de características; ela pode ser feita também para deixar o código mais claro.

2 Objetivo

Objetivo geral:

- Estudar a viabilidade e a adequação da utilização do desenvolvimento de software orientado a aspectos (análise, projeto e implementação) na implementação de provadores automáticos de teoremas baseados em Tableaus.

Objetivos específicos:

- Implementar um PAT´s baseado em Tableaus eficiente e flexível quanto à definição das estratégias de prova
- Comparar o provador obtido com outros provadores
- Estudar o impacto da utilização de padrões de projeto e de técnicas de programação extrema no desenvolvimento de PAT´s

3 Procedimentos Metodológicos

1. Estudo dos artigos e livros descrevendo:
   (a) Tableaus e Provadores Automáticos de Teoremas baseados em Tableau
   (b) Orientação a Objetos
   (c) Padrões de Projeto
   (d) Programação Orientada a Aspectos
   (e) Programação Extrema
2. Testes com as ferramentas a serem utilizadas no desenvolvimento:
   (a) Eclipse - Ambiente de desenvolvimento integrado com suporte a Java, AspectJ, JUnit e Refatoração
   (b) Java - linguagem de programação orientada a objetos
   (c) AspectJ - linguagem de programação orientada a aspectos
   (d) JUnit - ferramenta para testes automáticos
3. Desenvolvimento incremental do provador automático de teoremas:
   (a) 1^a versão: implementação de tableaus analíticos para lógica clássica proposicional, utilizando estratégias de cima para baixo (*top-down*) e de baixo para cima (*bottom-up*) [11]
   (b) 2^a versão: implementação do sistema KE de tableaus [8] e de diferentes estratégias para este sistema apresentadas em [11]
   (c) 3^a versão: implementação das estratégias como aspectos ortogonais às classes que descrevem o provador de teoremas

4. Execução dos testes descritos em [11] e comparação (*benchmarks*) dos resultados dos testes com outros provadores de teoremas

5. Escrita de monografia descrevendo as bases teóricas, o desenvolvimento, os resultados dos testes, e procurando responder as seguintes perguntas:

   (a) a utilização da orientação a objetos, dos padrões de projeto, da programação orientada a objetos e da programação orientada a aspectos atingiu os objetivos de flexibilidade e reusabilidade ao longo do processo de desenvolvimento?

   (b) as técnicas de programação extrema utilizadas permitiram um desenvolvimento mais rápido e estável do sistema?

   (c) a eficiência do sistema é comparável à de provadores automáticos de teoremas existentes?

   (d) através do uso da programação orientada a aspectos, foi possível modularizar melhor as estratégias de prova para tableaus?

Referências

[1] Kent Beck and Ralph Johnson. Patterns generate architectures. *Lecture Notes in Computer Science*, 821:139–??, 1994.

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

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

[4] K. Broda, M. D'Agostino, and M. Mondadori. A Solution to a Problem of Popper, 1995.

[5] Robert Constable and Cristoph Kreitz. Automated Reasoning, 2002. Slides from a presentation available at http://www.cs.cornell.edu/Info/People/kreitz.

[6] 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.

[7] Marcello D'Agostino. Are Tableaux an Improvement on Truth-Tables? Cut-Free proofs and Bivalence. 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] Carlos J. P. de Lucena. Separation of concerns and multi-agent systems group, 2003. http://www.teccomm.les.inf.puc-rio.br/SoCagents/.

[10] José Eduardo Zindel Deboni. *Modelagem orientada a objetos com a UML*. Futura, 2003.

[11] 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.

[12] Elrad, Filman, and Bader. Aspect-oriented programming. *CACM: Communications of the ACM*, 44, 2001.

[13] U. Endriss. A KE based theorem proving assistant, 1996. Master's thesis, Department of Computing, Imperial College, London, 1996.

[14] 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.

[15] R. Filman and D. Friedman. Aspect-oriented programming is quantification and obliviousness, 2000.

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

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

[18] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Adisson-Wesley Longman, 1999.

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

[20] R. Hähnle. Tableaux and related methods. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 3, pages 100–178. Elsevier Science, 2001.

[21] Richard Hightower and Nicholas Lesiecki. *Java Tools for eXtreme Programming - Mastering Open Source Tools including Ant, JUnit and Cactus*. Wiley, 2002.

[22] Louis H. Kaufmann. The Robbins Problem: computer proofs and human proofs. *Kybernetes - The International Journal of Systems and Cybernetics*, 30, 2001.

[23] 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.

[24] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[25] Michael Kohlhase and Carolyn Talcott. Database of existing mechanized reasoning systems, 2003. Available at http://www-formal.stanford.edu/clt/ARS/systems.html.

[26] William F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, Urbana-Champaign, IL, USA, 1992.

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

[28] Don Roberts and Ralph Johnson. Evolving Frameworks. *????*, 12(1):23–41, January 1965. Avaliable at http://st-www.cs.uiuc.edu/users/droberts/evolve.html.

[29] J. A. Robinson. A machine-oriented logic based on the resolution principle. *JACM*, 12(1):23–41, January 1965. Reprinted in [31].

[30] Start Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995. Preis: 70,- DM.

[31] Jörg Siekmann and Graham Wrightson, editors. *Automation of Reasoning: Classical Papers in Computational Logic 1957–1966*, volume 1. Springer-Verlag, 1983.

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

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

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

[35] Andrei Voronkov. Algorithms, Datastuctures, and Other Issues in Efficient Automated Deduction. In *International Joint Conference on Automated Reasoning*, LNCS, pages 13–28. Springer-Verlag, 18-23 juin 2001.

[36] Larry Wos, Ross Overbeek, Ewing Lusk, and Jim Boyle. *Automated Reasoning: Introduction and Applications.* Prentice-Hall, 1984.