DCC - UFMG

Uma biblioteca de analisadores sintáticos

Aluno: Thadeu Lima de Souza Cascardo
Orientadora: Mariza Andrade da Silva Bigonha
Data: 2 de dezembro de 2005

Relatório de Projeto Orientado I

por Thadeu Lima de Souza Cascardo

Orientadora: Mariza Andrade da Silva Bigonha
Data: 02 de Dezembro de 2005

Tabela de conteúdos

1 Introdução

A produção de compiladores, interpretadores de linguagens de programação e outras tarefas como a leitura de arquivos de configuração ou de entrada para qualquer aplicação, ou mesmo a interpretação de linguagens naturais envolvem um processo denominado análise sintática. Em geral, esse processo envolve a especificação da linguagem em uma gramática, na maioria das vezes, em uma forma denominada BNF (Backus-Naur Form). Ferramentas de geração de compiladores são utilizadas para traduzir uma linguagem fonte na forma de uma gramática em código objeto. Especificamente, em relação à análise sintática, tem-se como exemplos o bison [1] e styx [2] para C, Javacc [3], Cup [4] e ANTLR [5] para Java, entre muitos outros.

Entretanto, esse processo pode oferecer algumas limitações. Uma delas é que o processo é executado em tempo de compilação, o que impediria uma possível extensão ou adaptação da linguagem em tempo de execução. Outra limitação é o algoritmo utilizado para a análise sintática ou a geração do analisador. Alguns algoritmos são mais eficientes em termos de espaço ou tempo e uma escolha poderia ser feita por parte do programa ou do usuário em tempo de execução dependendo dos recursos disponíveis no ambiente. Por fim, algumas gramáticas não podem ser utilizadas com alguns analisadores, e, geralmente, o desenvolvedor deve reescrevê-la utilizando alguns algoritmos conhecidos.

Uma biblioteca de analisadores e geradores não teria estas limitações. A implementação de diferentes algoritmos permitiria a escolha entre otimizar tempo e espaço por parte do desenvolvedor em tempo de execução, utilizando o analisador mais apropriado. Este trabalho se propõe a implementar uma biblioteca contendo alguns dos algoritmos aqui enumerados sob uma mesma interface, permitindo uma maior flexibilidade para o desenvolvedor de aplicações.

2 Conceitos Básicos

As linguagens podem ser classificadas segundo seu poder de expressão. Entre elas, temos as linguagens regulares e as linguagens livres de contexto, as mais utilizadas e referenciadas na literatura, pois podem ser analisadas de forma eficiente por programas de computador. Estas linguagens podem ser representadas de diversas formas. Entre as representações mais populares, temos a BNF (Backus Naur Form) [6]. Para linguagens livres de contexto, há ainda a TDPL (top-down parsing language) [7] e, recentemente, a PEG (Parsing Expression Grammar) [8]. Para as linguagens regulares, existem as expressões regulares.

As BNF's permitem a descrição de linguagens ambíguas, enquanto as PEG's não. As PEG's também podem ser utilizadas para descrever algumas linguagens que não são livres de contexto, mais poderosas. Estas linguagens podem ser analisadas utilizando diferentes algoritmos. Para as linguagens livres de contexto, temos duas classes de algoritmos, os "top-down" e os "bottom-up". Geralmente, os "bottom-up" são mais eficientes, enquanto os "top-down" são mais restritivos.

Entre os algoritmos "top-down", temos o "recursive descent parser" [9] [10], que executa em tempo exponencial, e o "packrat parser" [11], que, apesar de executar em tempo linear, consome muito espaço em memória. Ambos algoritmos não reconhecem gramáticas recursivas à esquerda e ambíguas.

Os algoritmos "bottom-up" utilizam uma técnica denominada "shift-reduce". Esses algoritmos são muito eficientes, mas dependem de ferramentas para a geração das tabelas de análise sintática.

Há ainda o "Earley parser" [12] e o "CYK parser" [13] [14] [15], que podem analisar as linguagens descritas por qualquer gramática e executam em tempo cúbico.

Os geradores de analisadores são algoritmos que geram código capaz de analisar uma linguagem a partir de uma descrição da mesma, seja em BNF, TDPL ou PEG. A TDPL e a PEG são mais apropriadas para a geração de algoritmos "top-down".`

3 Desenvolvimento

O trabalho foi organizado em etapas, que poderiam ser realizadas paralelamente. São elas:

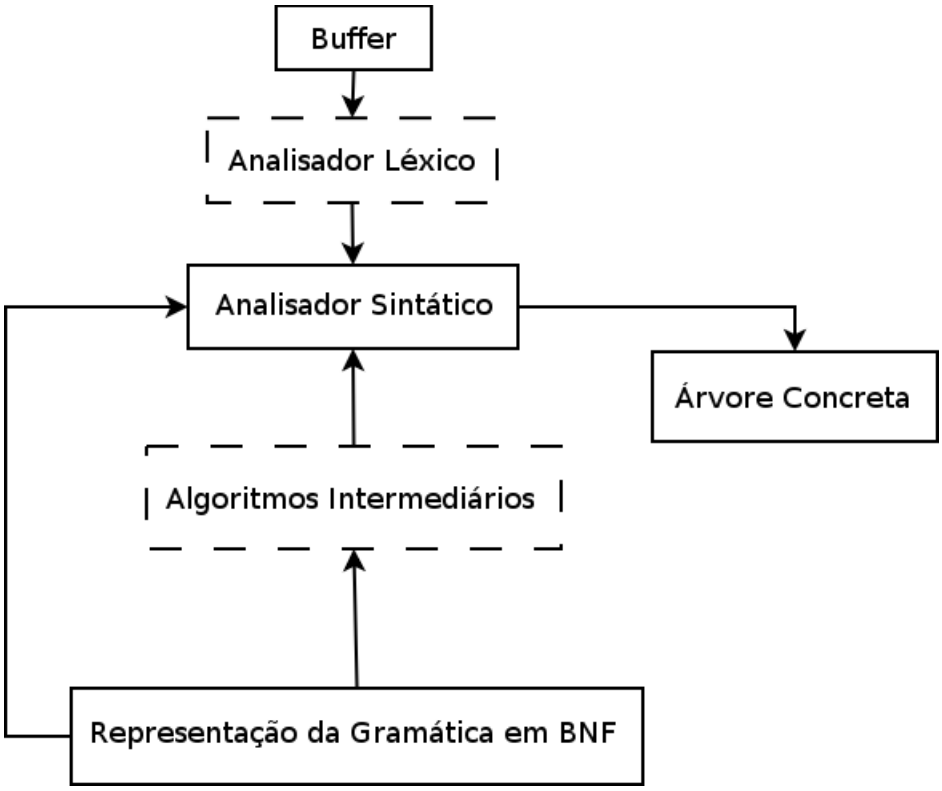
- Desenvolvimento de alguns algoritmos de análise sintática: dentre os candidatos à implementação estão os algoritmos enumerados na seção 2. O algoritmo LR(1) foi implementado e testado. Os algoritmos LR(0), RDP, Packrat e Predictive Parser já tiveram sua implementação iniciada e dependem de correções e testes para funcionarem adequadamente. Otimizações também são possíveis trabalhos.

- Desenvolvimento de interface para representação de linguagens: uma API (Application Programming Interface) para a representação de BNF's foi implementada. Entretanto, é planejada uma revisão desta interface que permita a execução de ações programadas pelo usuário da bilioteca durante a análise sintática. Outras facilidades podem ser implementadas para uma manipulação mais simples das gramáticas que permita sua combinação.

- Desenvolvimento de analisadores de representações como BNF's e PEG's: o armazenamento da gramática em um arquivo já está implementado e a representação de expressões regulares está em fase de desenvolvimento. Ambas utilizam a própria biblioteca para a leitura e análise do texto.

- Testes de linguagens de programação e arquivos de configuração: a implementação de um compilador na disciplina de compiladores está utilizando a biblioteca como forma de testá-la e identificar possíveis melhorias.

Abaixo, segue uma imagem representando a arquitetura da biblioteca.



A representação da gramática é utilizada por algoritmos intermediários que geram informação necessária pelos algoritmos de análise sintática. O algoritmo utiliza como entrada, além dessas informações e da representação da gramática o texto a ser lido. Por fim, uma árvore concreta é dada como saída para o programador.

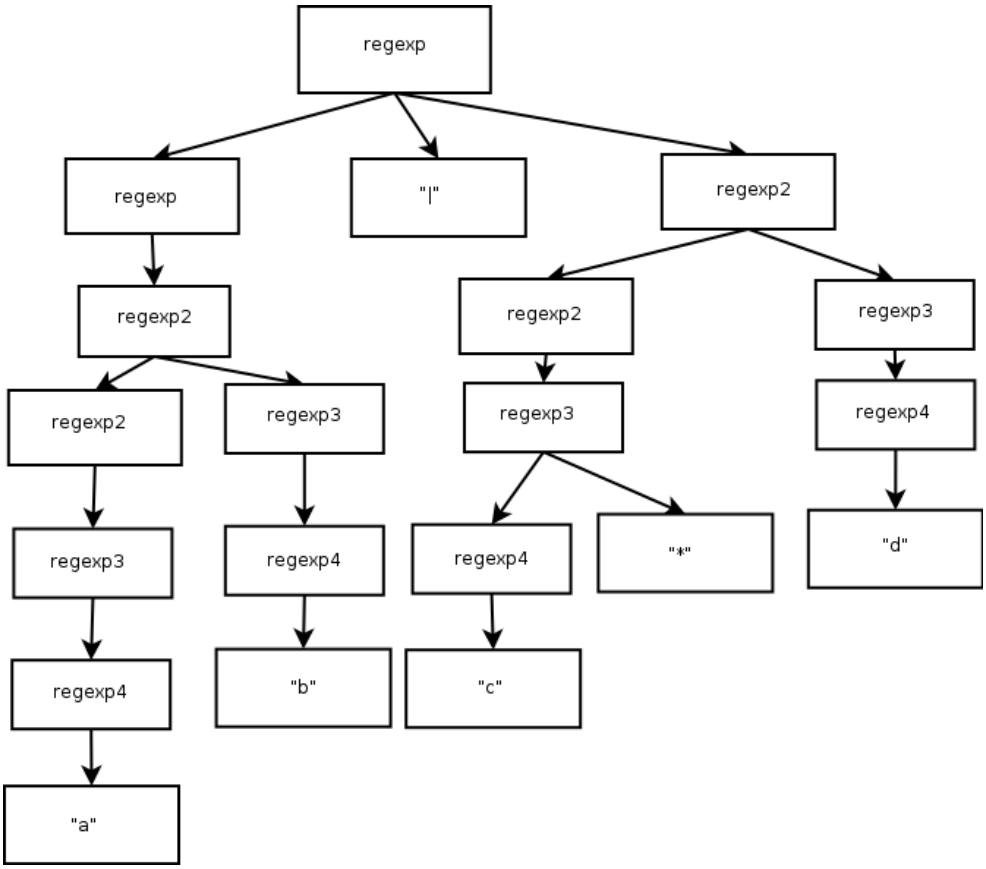
Portanto, o programador tem uma interface que lhe permite representar uma gramática e dela pode extrair uma árvore do texto fornecido através de um buffer.

3.1 Exemplo

Abaixo, segue um exemplo de entrada e saída da biblioteca. Segue uma gramática representando uma classe de expressões regulares, seguido de uma expressão. Esses itens são utilizados como entrada para a biblioteca. Em seguida, há a imagem de uma árvore, sendo esta a saída gerada pela biblioteca de forma a permitir sua manipulação pelo programador para que execute as ações desejadas.

```
regexp: regexp "|" regexp2
regexp: regexp2
regexp:
regexp2: regexp2 regexp3
regexp2: regexp3
regexp3: regexp4 "*"
regexp3: regexp4
regexp4: "symbol"
regexp4: "empty"
```

```
ab | c*d
```



4 Conclusão

A tarefa de analisar sintaticamente textos para as mais diversas aplicações é feita hoje em dia gerando analisadores sintáticos em tempo de compilação. A geração em tempo de execução dá maior flexibilidade ao permitir a escolha do algoritmo e a manipulação da gramática.

Implementar os algoritmos sob uma interface genérica é viável e há ainda a possibilidade de que outros algoritmos sejam implementados e utilizados mesmo que não tenham sido compilados em conjunto com a biblioteca.

O desenvolvimento de aplicações que se beneficiam dessa biblioteca pode demonstrar a necessidade de sua criação. Portanto, o licenciamento da biblioteca como software livre para estimular seu uso por outros desenvolvedores é o passo imediato a ser tomado.

5 Bibliografia

[1] http://www.gnu.org/software/bison/
[2] http://www.speculate.de/styx/
[3] https://javacc.dev.java.net/
[4] http://www.cs.princeton.edu/~appel/modern/java/CUP/
[5] http://www.antlr.org/
[6] http://www.garshol.priv.no/download/text/bnf.html
[7] AHO, A. V. E ULLMAN, J. D. The Theory of Parsing, Translation and Compiling

[8] FORD, B. Parsing Expression Grammars: A Recognition-Based Syntactic Foundation. Symposium on Principles of Programming Languages (2004).

[9] BURGE, W. H. Recursive Programming Techniques

[10] AHO, A. V., SETHI, R. E ULLMAN, J. D. Compilers: Principles, Techniques and Tools.

[11] FORD, B. Packrat Parsing: Simple, Powerful, Lazy, Linear Time. International Conference on Functional Programming (2002).

[12] EARLEY, J. An efficient context-free parsing algorithm. Communications of the Association for Computing Machinery (1970).

[13] John Cocke and Jacob T. Schwartz (1970). Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University.

[14] T. Kasami (1965). An efficient recognition and syntax-analysis algorithm for context-free languages. Scientific report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA.

[15] Daniel H. Younger (1967). Recognition and parsing of context-free languages in time n3. Information and Control 10(2): 189-208.`