Lapg
Descrição e Uso da Ferramenta

José Henrique Ricordi Cruz[1], **Rafael dos Santos Braz**[1]

[1]Centro de Ciências Tecnológicas – Universidade Estadual do Norte do Paraná (UENP)
Bandeirantes – PR – Brasil

{henriquericordi, rafaelsantosbrazpfi}@gmail.com

1. Descrição da Ferramenta

O presente trabalho intenta descrever e exemplificar o uso de uma ferramenta chamada Lapg. Esse nome nada mais é do que uma sigla para *Lexical analyzer and parser generator*. Por tradução simples de seu nome, tal ferramenta realiza a combinação de um Analisador Léxico e um gerador de Analisador Sintático (Parser) em um único mecanismo.

A ferramenta adotada começou a ser desenvolvida no ano de 2002 por Evgeny Gryaznov[1]; e esse processo ainda está ocorrendo[2]. Entretanto, atualmente o Lapg está sendo desenvolvido dentro de um projeto com outro nome: *Textmapper*[3]. O *Textmapper* tem por objetivo aprimorar o Lapg e torná-lo um *plugin* que permita a utilização do mesmo em Ambientes de Desenvolvimento Integrado (IDE). Esse *plugin* ainda não possui uma versão funcional oficial, visto que o projeto ainda está em fase pré-alfa.

É válido ressaltar que tanto o Lapg original quanto o *Textmapper* não são ferramentas famosas e, portanto, não há materiais de estudo disponíveis. As únicas fontes de informação são suas documentações e exemplos de utilização fornecidos pelo próprio autor[2]. Sabendo-se da documentação escassa e considerando-se que o *plugin* em desenvolvimento não é diretamente compatível com a versão estável do Lapg e que as novas versões ainda geram conflitos e confusões para sua utilização, decidiu-se por adotar o Lapg em sua versão estável[4] antes de ser alterada para integrar-se ao *Textmapper*.

A página oficial do Lapg[5] disponibiliza a ferramenta, sua documentação e exemplifica sua utilização. A referida documentação indica que essa ferramenta pode ser aplicada em diversas situações, desde de análises de texto até a construção de compiladores complexos como, por exemplo, o de C++.

O Lapg é uma solução de software amigável ao usuário por ser simples de utilizar-se, mesmo sendo uma aplicação acessada e controlada exclusivamente por meio de um terminal. Além disso, a mesma pode ser executada nos Sistemas Operacionais (SO) Windows, Linux e Mac OS de forma nativa ou utilizando-se a Máquina Virtual Java (JVM) – esse aspecto depende da versão utilizada.

É importante a citar alguns pontos técnicos da ferramenta adotada. O Lapg recebe uma descrição de gramática Livre de Contexto do tipo LALR(1) – em formato próprio

[1]O desenvolvedor possui uma página no GitHub, disponível em: https://github.com/inspirer
[2]Uma das páginas do projeto está disponível no GitHub: https://github.com/inspirer/textmapper
[3]Página oficial do projeto: http://textmapper.org/
[4]Versão 1.2.3 do Lapg.
[5]Lapg disponível em: http://lapg.sourceforge.net/ e https://sourceforge.net/projects/lapg/

semelhante a forma BNF, o que facilita o uso dessa solução – e gera um código-fonte capaz de de analisar sintaticamente a gramática especificada pelo usuário. Tal código-fonte pode ser escrito em, principalmente, cinco linguagens de programação diferentes: C; C++; C#; Java; e Javascript. Além das mencionadas, o Lapg aceita modelos geradores para outras linguagens de programação que podem ser inseridos por seus usuários, caso seja necessário.

Ademais, observa-se que o Analisador Sintático gerado pela ferramenta executa um algoritmo de reconhecimento que segue os padrões da análise *Bottom-Up*, ou seja, o *parser* gerado receberá um fluxo de caracteres de entrada[6] (uma *String* ou um vetor de *Char*) e aplicará regras para reduzir essa entrada ao primeiro símbolo não-terminal (variável), sendo esse o critério de aceitação. Além disso, esse mecanismo suporta verificações de ambiguidades gramaticais, precedência de operadores e recuperação de erros.

2. Instalação e Uso da Ferramenta

Após a descrição da ferramenta, as subseções seguintes expõem como deve-se instalar o Lapg e os passos necessários para utilizá-lo, respectivamente.

2.1. Instalação do Lapg

O processo de instalação da ferramenta é dependente de sua versão, mas independentemente da versão escolhida, o método pode ser considerado simples. Nas versões mais novas do Lapg e do *Textmapper*, pode-se apenas utilizar um arquivo executável Java através de um terminal, sem necessidade de instalação.

A versão adotada[4] não possui o arquivo Java para execução. Todavia, o processo de instalação permanece simples. Se o usuário utilizar um SO Windows, o Lapg já está disponível como um arquivo executável ("*.exe") único que pode ser manipulado via console. Se o usuário utilizar um SO Linux, o Lapg está disponível como uma pasta de código-fonte que possui um arquivo *Makefile*, logo, apenas é preciso um comando do tipo *make* para que a ferramente seja compilada automaticamente.

2.2. Utilização do Lapg

Conforme mencionado anteriormente, o processo de instalação da ferramenta adotada é relativamente simples; esse mesmo pensamento pode ser atribuído ao processo de utilização do Lapg, uma vez que necessita-se apenas de poucos parâmetros. Por padrão, a ferramenta espera um arquivo chamado *syntax* contendo a descrição da gramática, que será transformado em um arquivo chamado *parse* correspondente. Portando, se o usuário não modificar tais nomes, o Lapg poderá ser executado no terminal sem parâmetros adicionais. Em outros casos, deve-se seguir o seguinte padrão:

lapg [opções] [entrada] [saída]

É interessante informar que não há opções que influenciem o processo de geração da saída do programa, visto que o objetivo da ferramenta é fornecer uma utilização amigável e simplificada. Portanto, as configurações são realizadas totalmente no arquivo de entrada. Assim sendo, os seguintes parâmetros são válidos:

[6]Isso é possível devido a presença de um analisador léxico nativo no *parser* gerado.

1. Opções:
    (a) **-d**: gera arquivos de *debug* com informações adicionais sobre a gramática e o código gerado.
    (b) **-e**: realiza o mesmo que o item anterior, porém fornece uma quantidade maior de detalhes.
    (c) **-vb**: exibe todo o código-fonte gerado pela ferramenta.
    (d) **-ts**, **-tf** e **-l**: utilizados para manipulação de *template* para outra linguagem de programação.

Para a **[entrada]** e **[saída]**, não há restrição para os nome e formato de ambos os arquivos informados. Assim sendo, o ponto central da ferramenta encontra-se no conteúdo em si do arquivo de entrada, o qual será descrito na subseção seguinte.

2.3. Arquivo de Entrada

O Lapg opera a partir de um arquivo normalmente nomeado *syntax*, o qual é subdividido em quatro seções que desempenham funções distintas entre si: Diretivas; Vocabulário; Atributos; e Gramática LALR(1). Cada seção encontra-se descrita a seguir.

2.3.1. Diretivas

Essa primeira seção do arquivo de entrada tem por objetivo especificar e controlar o processo de geração de código. Nesse momento, pode-se definir aspectos que são independentes da linguagem de programação escolhida para a construção do *parser*. Os principais aspectos que podem ser modificados são os seguintes:

1. ***maxtoken***: tamanho máximo de cada *token*.
2. ***stack***: tamanho máximo da pilha de análise.
3. ***class***: nome do *parser* a ser gerado.
4. ***getsym***: código para receber o próximo símbolo e colocá-lo na variável *chr*.
5. ***errorprefix***: mensagem inicial para erros.
6. ***namespace***: nome do ambiente que conterá o item *class*.
7. ***lang***: indica qual será a linguagem de programação utilizada para gerar o *parser*.

2.3.2. Vocabulário

Essa seção do arquivo de entrada é responsável por declarar os itens léxicos que irão compor a gramática. Para tanto, segue-se o seguinte padrão:

identificador [tipo] : padrão [prioridade] [ação]

No padrão indicado, os itens que não possuem **[]** são obrigatórios. Cada um dos elementos indicam algo diferente:

1. **Identificador**: nome do terminal descrito.
2. **Tipo**: indica qual o tipo de dado associado ao terminal na linguagem de programação alvo.
3. **Padrão**: padrão para casamento do terminal, especificado a partir de Expressão Regular.

4. **Prioridade**: valor inteiro para resolução de conflitos entre os terminais, indicando a ordem de consideração dos mesmos.
5. **Ação**: especifica comandos dependentes da linguagem de programação alvo para manipular os dados associados ao *token* – quando esse for identificado.

2.3.3. Atributos

Essa parte do arquivo de entrada define atributos, isto é, dados que são associados e herdados pelos símbolos. É válido ressaltar que essa é uma seção apenas declarativa com fins de análise estática e controle de tipos. O atributo deve ser sempre inserido entre **[]** e, caso exista mais de um elemento, deve ser separado dos demais por **vírgulas**.

2.3.4. Gramática LALR(1)

Nessa seção, define-se a gramática que o *parser* a ser gerado deverá aplicar para analisar suas entradas. Para que essa gramática possa ser descrita, utiliza-se um padrão semelhante ao BNF. O padrão mais básico aceito é o seguinte:

identificador ::= termo ;

No padrão mencionado, os elementos são:

1. **Identificador**: elemento que não pertence ao vocabulário especificado; serve para indicar uma variável, isto é, indicar uma outra regra da gramática.
2. **Termo**: pode ser um Identificador ou um item do Vocabulário especificado.

É possível inserir mais de um termo por regra e, para tanto, é necessário separá-los por $|$. Além disso, pode-se adicionar ações semânticas em qualquer parte da derivação por meio de comandos dependentes da linguagem de programação alvo. Essa ações semânticas devem ser inseridas entre $\{\}$.

Ademais, é obrigatório criar-se uma derivação especial com o identificador *input*. Essa ação corresponde a estender a gramática para que *input* seja a primeira variável da gramática especificada. Essa regra especial tem por objetivo definir qual é o início da gramática, pois a análise será feita no formato *Bottom-UP*, conforme supracitado.

Ao final dessa seção, é preciso criar-se ao menos uma função principal que irá inicializar o *parser* e o processo de análise em si na linguagem de programação alvo. Para tanto, insere-se o código após os símbolos %%.

3. Exemplo de Aplicação de Gramática

Para exemplificar o uso do Lapg, selecionou-se uma gramática para ser aplicada na ferramenta:

```
<prog>  ::= <cmd> eol | <prog> <cmd> eol
<cmd>   ::= <io> | <if>
<io>    ::= read id | write id | id = id
<if>    ::= if <cond> then <block>
            | if <cond> then <block> else <block>
<cond>  ::= (id) | (id==id)
<block> ::= begin <cmd> end
```

Utilizando-se essa gramática e seguindo-se os padrões apresentados nas seções anteriores, construiu-se um arquivo de exemplo[7] que especifica a geração de um *parser* em Javascript. O Arquivo de entrada completo é o seguinte:

```
#   Automatically generated grammar

.lang        "js"
.getsym      "chr = this.buff.charAt(this.l++)"
.positioning "full"
.class       "myparser"

# Vocabulary

 _skip:         /\/\/.*/
 _skip:         /[\t\r\n ]+/                        { continue; }
 'eol':         /;/
 'read':        /read/
 'write':       /write/
 '=':           /=/
 'if':          /if/
 'then':        /then/
 'else':        /else/
 '(':   /\(/
 ')':   /\)/
 '==':          /==/
 'begin':       /begin/
 'end':         /end/
 'id':          /[a-zA-Z_][a-zA-Z_0-9]*/  -1

 # Attributes

 []

 # Grammar

 input ::=
     prog { dump("[good]"); } ;

 prog ::=
       cmd 'eol'
     | prog cmd 'eol' ;

 cmd ::=
   io
     | if ;

 io ::=
       'read' 'id'
```

[7]Disponível em: https://github.com/RafaelSantosBraz/Compiladores/tree/master/Ferramenta%20Lapg

```
45      | 'write' 'id'
46      | 'id' '=' 'id' ;
47
48  if ::=
49        'if' cond 'then' block
50      | 'if' cond 'then' block 'else' block ;
51
52  cond ::=
53        '(' 'id' ')'
54      | '(' 'id' '==' 'id' ')' ;
55
56  block ::=
57      'begin' cmd 'end' ;
58
59  %%
60
61  dump = alert;
62
63  function error(s) {
64  dump(s);
65  }
66
67  function parse(string) {
68  var p = new myparser();
69  p.buff = string;
70  p.l = 0;
71  // p.DEBUG_syntax = 1;
72  p.parse();
73  }
```

4. Considerações Finais

Após o teste do Lapg, pode-se afirmar que essa ferramenta atinge seus objetivos quanto a simplicidade e a facilidade de utilização, visto que não há processo de instalação e há uma pequena quantidade de parâmetros que podem ser aplicados durante seu uso. Além disso, toda a especificação da gramática a ser empregada encontra-se em um único arquivo de texto bem definido, subdividido e de fácil entendimento. Esse arquivo pode ser aplicado na criação de *parsers* em diversas linguagens de programação realizando-se apenas pequenos ajustes. O único aspecto negativo encontrado é a falta de materiais que auxiliem no processo de aprendizagem.