Universidade Federal do Rio Grande do Norte
Departamento de Informática e Matemática Aplicada

Disciplina: DIM0611 — Compiladores
Docente: Martin Alejandro Musicante
Discente: Felipe Cortez de Sá

Parsers preditivos

O cálculo dos conjuntos **FIRST**, **FOLLOW** e da tabela de predição foram feitos através das ferramentas [1] e [2], usando como entrada a gramática desenvolvida na primeira atividade.

Os tokens são acessados através da função **yylex()**, inclusa no arquivo gerado pelo lexer desenvolvido anteriormente.

Para simplificar o entendimento do código, cada terminal e não terminal tem um macro correspondente (prefixo **T_** para terminais e **NT_** para não terminais) substituído por um inteiro ao compilar. Os macros estão definidos no arquivo **lex**. Há uma função **get_token(char tok)** que recebe o inteiro associado ao token e devolve a string correspondente para facilitar a depuração.

1 Parser por tabela

Foi implementada uma pilha que armazena símbolos (codificados por **chars**). O resultado do processo de parsing é salvo num arquivo **.dot** e pode ser visualizado através do programa **graphviz** [3] com o comando **graphviz graph.dot -Tpdf -o graph.pdf**.

As produções estão definidas na matriz **r** como cadeias de caracteres, em que cada caractere corresponde a um símbolo terminal ou não-terminal. A matriz **rules2**, gerada pela ferramenta [2], contém os índices das regras definidas em **r** (com as regras de código 89 e 90 indicando erros), onde cada coluna corresponde a um terminal e cada linha corresponde a um não-terminal. A ordem desses símbolos nas linhas e colunas seguem a ordem em que aparecem os terminais na tabela do conjunto **FIRST** e os não terminais que aparecem no conjunto **FOLLOW** na ferramenta. Tendo isso em vista, os macros definidos seguem essa ordem para a numeração.

2 Parser recursivo

Foi utilizado o algoritmo apresentado por Appel [4] na seção 3.2.

Para cada não-terminal da gramática foi implementada uma função recursiva que, com base no conjunto **FIRST**, determina a regra a ser seguida, chamando as funcões recursivas associadas aos não-terminais da produção e a função **eat()** para os símbolos terminais. Para produções que geram ϵ, a função retorna sem chamadas adicionais. Para cada **switch**, há um caso **default** que chama uma função **error()** caso o token sendo lido não esteja no conjunto **FIRST**. O token atual é armazenado numa variável global **current_tok** para não haver necessidade de passar ponteiros para todas as funções.

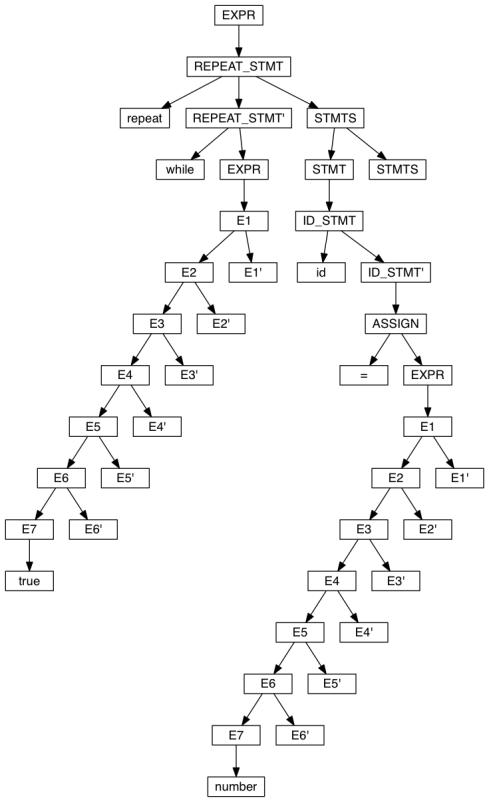

Figura 1: Árvore gerada pelo **graphviz**

Referências

[1] *JS Machines — LL(1) parser generator.*
http://jsmachines.sourceforge.net/machines/ll1.html

[2] *HackingOff — LL(1) parser generator.*
http://hackingoff.com/compilers/ll-1-parser-generator#ll-1-parsing-table

[3] *Graphviz — Graph Visualization Software.*
http://www.graphviz.org/

[4] Andrew W. Appel. *Modern compiler implementation in ML.* Cambridge University Press, 1998.