SCC0217 - Linguagens de programação e compiladoers
Prof. Diego Raphael Amancio

Trabalho 2
Analisador Sintático da LALG - Relatório

Danilo Françoso Tedeschi - 8937361
Lucas de Carvalho Rodrigues da Silva - 8624511
Matheus de França Cabrini - 8937375
Rita Raad - 8061452
Rodrigo de Andrade Santos Weigert - 8937503
Universidade de São Paulo
São Carlos

11/06/2017

1 Decisões de Projeto

1.1 Implementação

O analisador sintático foi implementado utilizando GNU Bison (extensão do Yacc). Usá-lo pareceu ser o caminho mais natural, já que o analisador léxico (parte 1 do trabalho) foi implementado utilizando flex, e esses dois programas frequentemente são utilizados juntos.

1.2 Tratamento de Erros

Conforme requisitado, o tratamento de erros segue o padrão de modo de pânico. Ele é implementado utilizando o token especial *error*, cuja função é justamente facilitar a implementação desse tipo de tratamento de erros. A lógica de tratamento de erros implementada é similar a que aparece no manual do Yacc.

2 Visão Geral do Projeto até Agora

2.1 Analisador Léxico

O analisador léxico permanece como foi especificado no relatório anterior, a menos de pequenas mudanças feitas para realizar a integração entre os módulos léxico e sintático e também para adaptar apropriadamente a lógica do tratamento de erros, já que, a partir de agora, parte significativa dessa lógica residirá na (previamente ausente) análise sintática.

Por exemplo, para começar, o analisador léxico não é mais *standalone*, ou seja, não possui mais uma função *main* e não pode mais ser executado independentemente do analisador sintático.

Além disso, a maneira como os tokens são numerados foi alterada. Antes, havia um *enum* criado manualmente. Agora o mesmo é gerado automaticamente pelo Bison (com auxílio da diretiva *%token*) e exportado para o código do analisador léxico via um arquivo de cabeçalho.

Outra mudança foi na maneira como o léxico trata erros. Anteriormente, o léxico retornava um token de erro sempre que detectava um comentário não terminado. Agora, tal token não existe mais e, quando esse erro é encontrado, o léxico apenas imprime uma mensagem e prossegue com a análise, saindo do estado de leitura de comentário. Isso ocorre de modo transparente ao analisador sintático, o qual recebe o programa de entrada já com os comentários eliminados. É válido notar, porém, que o outro token de erro retornado pelo léxico - relativo à detecção de um símbolo não pertencente a gramática - permanece sendo retornado. Isso ocorre logo após a ativação de uma nova *flag* que sinaliza a ocorrência do erro para o sintático, o qual se encarrega da impressão de todos os erros que não o de comentário.

2.2 Analisador Sintático

O código do analisador sintático (*parser.y*) tem estrutura similar ao do léxico, sendo dividido em três partes. A parte 1 contém algumas declarações em C e as definições dos tokens. A parte 2 contém as definições das regras sintáticas da LALG (inclusive as regras para tratamento de erro), e a parte 3 contém a função principal do analisador sintático, além da função responsável pela impressão dos erros na tela.

O sintático se comunica com o léxico no momento em que faz chamadas à função mais importante deste, *yylex*, responsável por ler a entrada e retornar os tokens apropriados. A realização dessas chamadas faz parte do código gerado automaticamente pelo Bison e, logo, não é de responsabilidade do usuário da ferramenta. O sintático consegue acesso à função *yylex* pois é compilado junto ao léxico e apresenta o cabeçalho da função declarado na parte 1 citada anteriormente.

3 Compilação e Execução

Um *Makefile* foi fornecido com o trabalho. Assumindo ambiente Linux com GNU Make, flex e GNU Bison instalados, para gerar os códigos dos analisadores léxico e sintático e compilar o projeto basta usar:

$$make$$

E, para executar:

$$./parser$$

ou

$$./parser{<}arquivo\_de\_entrada$$

Alternativamente, para compilação, pode-se apenas compilar conjuntamente os arquivos *lex.yy.c* e *y.tab.c* fornecidos. Nesse caso, não há necessidade de ter o flex e o Bison instalados.

Ao executar, o programa reporta todos os erros que encontrar no programa de entrada. Se nenhum erro for encontrado, o programa termina silenciosamente, como fazem certos compiladores como o GCC.

4 Exemplos de Execução

Erros em comandos

```
1  program nome1;
2  {exemplo 1}
3  var a, a1, b: integer;
4  begin
5  read(a);
6  a1 := a1;
7  while (a10) do
8  begin
9  re(;
10 write(a1);
11 a1:= a1-1;
12 end;
13
14 for b:=1. to 10 do
15 begin
16 read();
17 read(;
18 write(a+a);
19 write(a,b,1);
20 b:=b+2;
21 a:=a-1;
22 end;
23
24 rea(;
25
26 if a b then read(a);
27
28 re
29 end
```

Saída do analisador sintático para o programa acima

```
[7] Error:      syntax error, unexpected RBRACKET                ())
[9] Error:      syntax error, unexpected SEMICOLON, expecting ID                (;)
[14] Error:     syntax error, unexpected PERIOD, expecting TO           (.)
[16] Error:     syntax error, unexpected RBRACKET, expecting ID         ())
[17] Error:     syntax error, unexpected SEMICOLON, expecting ID                (;)
[18] Error:     syntax error, unexpected PLUS, expecting RBRACKET               (+)
[19] Error:     syntax error, unexpected INT, expecting ID              (1)
[24] Error:     syntax error, unexpected SEMICOLON, expecting ID                (;)
[26] Error:     syntax error, unexpected ID             (b)
[29] Error:     syntax error, unexpected END, expecting SEMICOLON               (end)
Parsing completed. 10 errors found.
```

Erros diversos

```
1  program fera;
2  a=10;
3  i var x : inteer;
4  var ab,bc : ineger;
5  var for : integer;
6  var for : iteger;
7  var a;b : rel.a;
8  procedure sdf(x,h.z : integr);
9  var b : integer
10 var a : intege;
11 begin
12 read(a);
13 end;
14
15 procedure bb();
16 a=b;
17 read(a;
18
19 begin
20 a(;
21 {read(asf;
22 e@nd
```

Saída do analisador sintático para o programa acima

```
[2] Error:      syntax error, unexpected ID, expecting BEG1N or CONST or VAR or PROCEDURE              (a)
[3] Error:      syntax error, unexpected ID, expecting BEG1N or CONST or VAR or PROCEDURE              (i)
[4] Error:      syntax error, unexpected ID, expecting REAL or INTEGER          (ineger)
[5] Error:      syntax error, unexpected FOR, expecting ID              (for)
[6] Error:      syntax error, unexpected FOR, expecting ID              (for)
[7] Error:      syntax error, unexpected SEMICOLON, expecting COLON             (;)
[7] Error:      syntax error, unexpected ID, expecting BEG1N            (b)
[8] Error:      syntax error, unexpected PERIOD, expecting COLON                (.)
```

```
[10] Error:     syntax error, unexpected VAR, expecting SEMICOLON               (var)
[16] Error:     syntax error, unexpected ID, expecting BEG1N            (a)
[20] Error:     syntax error, unexpected SEMICOLON, expecting ID                (;)
[21] Error:     unclosed comment
Parsing completed. 11 errors found.
```