Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
SCC0605 – Teoria da Computação e Compiladores
Docente: Prof. Dr. Thiago A. S. Pardo

Analisador Sintático

Leonardo Alves Paiva 10276911
Rafael Pastre 9783675

São Carlos, 18 de junho de 2020

Sumário

Revisões no Analisador Léxico

Antes de iniciar o desenvolvimento do analisador sintático, foi realizada uma revisão no analisador léxico. A fim de simplificar seu código e melhorar sua legibilidade o código foi reimplementado em C++, visto que, dessa forma, seria possível utilizar as tabelas hash nativas da linguagem. O código final, antes implementados por 3 arquivos: “hash.c”, “hash2.c” e “lexico.c” agora está contido no arquivo “lexico.cpp”.

Decisões de projeto

Reconhecimento da gramática

Para implementar um analisador sintático descendente preditivo recursivo, o primeiro passo é garantir que a linguagem que será reconhecida é LL(1). Entretanto, analisando gramática da linguagem P--, percebemos que ela já está praticamente nesta forma, sendo assim, as alterações necessárias eram poucas e foram feitas espontaneamente durante a construção do gráfico. Um exemplo de alteração feita foi a retirada de ambiguidade das regras “<cmd> ::= ident := <expressão>” e “<cmd> ::= ident <lista_arg>”.

Em seguida foram desenvolvidos os grafos sintáticos da linguagem P-- (o arquivo contendo os grafos se encontra em anexo). Devido a extensão gramatical da linguagem, o grafo foi montado tendo como finalidade possuir alto reuso de regras gramaticais (funções) e baixo número de regras a serem implementadas, visto que isso facilitaria a implementação, legibilidade, deixaria o código mais compacto e com menos chamadas de funções, o que pode ser um problema devido ao uso da recursão.

Dessa forma, todas as regras gramaticais que eram “chamadas” por mais de uma regra foram mantidas separadas, enquanto as que eram “chamadas” por apenas uma regra foram incorporadas nesta regra, a menos que isso causasse dificuldades de implementação ou deixasse o grafo muito longo. Por exemplo as regras de <corpo> e <dc> foram incorporadas na regra de <programa>, entretanto as regras de <dc_c>, <dc_v> e <dc_p> assim como a de <comandos> não foram incorporadas a <programa>, respectivamente, para não deixar o grafo longo, e para não haver problemas de desvio de código. As regras gramaticais que de fato foram implementadas como funções são as seguintes:

programa	argumentos
dc_c	comandos
dc_v	cmd
tipo_var	condicao
variaveis	expressao
dc_p	termo
corpo_p	

Tabela 1 – Funções gramaticais implementadas

As regras que não foram transformadas em funções estão incorporadas dentro das que foram implementadas.

Tratamento de erros

O tratamento de erros foi feito pelo modo pânico. Quando era detectado algum erro, isto é, o símbolo lido não era esperado pela gramática, uma mensagem de erro era exibida. Na mensagem, é relatada em que linha ocorreu o erro, o token lido e o símbolo esperado. Quando o token se tratava de um identificador, era exibido o que havia sido entendido como identificador no lugar do token, a fim de facilitar a identificação do problema.

O modo pânico era ativado logo após a exibição da mensagem de erro. Basicamente, lia os símbolos do arquivo até encontrar algum de sincronização para poder continuar a análise sintática. Os símbolos de sincronização podiam ser locais ou do pai (função que invocou a atual). No caso de ser um símbolo do pai, era necessário sair da função atual.

Instruções de compilação e execução

Os códigos desenvolvidos (analisador léxico e analisador sintático) foram implementados em C++, e a compilação dos códigos pode ser feita, utilizando o compilador G++, com o seguinte comando:

g++ lexico.cpp sintático.cpp -o compilador.exe

A execução do programa depende dos arquivos “reservados.txt” e “transicoes.txt”, portanto é necessário manter estes arquivos no mesmo diretório do compilador, visto que estes arquivos serão carregados no início da execução do compilador. Além disso, o compilador tem como parâmetros de execução um arquivo texto contendo o programa fonte em P--, e um arquivo texto onde será impressa a saída do compilador. Ou seja, a execução do compilador pode ser feita da seguinte forma:

compilador.exe <programa_fonte_pascal> <nome_arquivo_saida>

No caso específico do programa testado “p.txt” a execução é feita por:

compilador.exe p.txt saida.txt

O programa foi testado em duas plataformas: Um Windows 10 x64 e um Windows 7 Home Basic SP1 x64. Além disso, os códigos completos podem ser encontrados no repositório do GitHub: https://github.com/rafael-pastre/SCC0605-2020-Trabalho_02-Analisador_Sintatico

Testes

Para testar o compilador foi feito um código em P--, contido no arquivo “p.txt” em anexo. O programa visa testar o maior número de regras gramaticais possíveis de ser interpretadas pelo compilador. Dessa forma, o programa contém testes para declaração de constantes, variáveis, procedimentos, e utilização dos comandos de read, write, while, if, atribuição, for, chamadas de procedimento, e blocos de códigos. Este programa obteve uma compilação bem sucedida, com seu arquivo de saída vazio.

Para testar o tratamento de erros, o arquivo “p.txt” foi modificado para o arquivo “p_com_erros.txt”, que também se encontra em anexo, no qual foram inseridos propositalmente erros gramaticais como falta de “;”, comandos incorretos e falta de caracteres. As diferentes saídas e a atuação do modo pânico do compilador podem ser visualizadas através do arquivo de saída do compilador (saida.txt) que está ilustrado abaixo.

Além disso, é possível visualizar no terminal o caminho gramatical percorrido pela recursão. Está funcionalidade pode ser ativada ou desativada de acordo com a remoção de um macro (#define) definido no início do arquivo “sintatico.cpp”

saida.txt

Erro na linha 5 : NUM_INT. Esperado : '='

Erro na linha 11 : simb_real. Esperado : ':'

Erro na linha 25 : y. Esperado : ';'

Erro na linha 30 : a. Esperado : ';'

Erro na linha 30 : simb_mais. Esperado : ';'

Erro na linha 30 : simb_mais. Esperado : end