Compiladores
Trabalho 3 - Analisador Semântico

Jackson Willian Brito; Talles Tatagiba Martins de Souza

Fevereiro, 2014

Contents

Abstract

Neste relatório explica-se como é o funcionamento do analisador semântio para a disciplina Compiladores assim como as considerações feitas pelos autores do trabalho a fim de garantir integridade na linguagem.

1 Introdução

Da literatura, temos a definição de um compilador como sendo um programa que recebe um programa escrito em uma linguagem fonte e o transforma em um programa em uma linguagem destino. Os programas fonte e alvo têm o mesmo significado. Uma característica imprescindível de um compilador é a capacidade de detectar e reportar erros.

O processo de compilação pode ser subdividido nas etapas de análise e síntese. A etapa de análise pode ser subdividida em 3 fases, são elas: léxica, sintática e semântica. Neste trabalho abordamos a fase semântica da etapa de análise.

2 Analisador Semântico

A análise semântica consiste na varredura do programa de entrada representa de fato um programa válido na linguagem fonte atentando-se as regras de escrita da linguagem e as restrições de tipos e operadores. Nesta etapa são construídas as estruturas de controles de variaáveis e funções nas quais constarão dados semânticos tais como escopo, tipo, aridade, entre outros que serão analisados. Caso o arquivo fonte não esteja escrito de maneira semanticamente correta na linguagem fonte deve-se reportar um erro indicando a linha na qual está a divergência entre o escrito e o que se esperava para que estivesse, o programa, corretamente redigido na linguagem esperada.

É de suma importância frizar que ao contrário da etapa de análise lÃ©xica e sintática, nas quais fizemos uso de ferramentas computacionais para realização das mesmas, nesta etapa, apesar do uso das ferramentas como suporte, toda a lógica de anÃ¡lise foi desenvolvida por nós.

As particularidades da linguagem aceita pelo nosso compilador serão pormenorizadas ao decorrer deste relatório.

3 Ajustes no Analisador Léxico

Para desenvolvermos a etapa de análise semântica foi feita a a inserção, no analisador léxico, de trechos de código onde flags e conteúdos de tokens são repassados em variáveis globais para que possamos utilizar os valores nos trechos de código inseridos no corpo do arquivo .y que Â´onde se encontram definidas os procedimentos de análise aplicáveis a cada sequência reconhecida.

4 Ajustes no Analisador Sintático

A gramática que desenvolvemos na parte 2 do trabalho, apesar de consistente não possuia todo o poder necessário para a reconhecimento de uma linguagem com as principais funcionalidades apresentadas pelas linguagens modernas como se propõe a ser a nossa. Portanto alguns pequenos ajustes foram feitos.

4.1 Declarações de funções

As declarações de funções não estavam aceitando mais de um comando seu corpo. Limitação muito simples de resolver alterando para receber blocos de comandos.

4.2 Atribuição a posição em matrizes

Entre as diversas considerações feitas haviamos nos esquecido de aceitar atribuição de matrizes por posição. Para resolver esse conflito, fez-se necessário adicionar no analisador sintático a expressão de atribuição a posição de matrizes como um possível comando.

4.3 Posição de matrizes como termos de funções

Outro objeto de esquecimento foi aceitar referencia a posições de matrizes como variáveis. Para resolver esse conflito, fez-se necessário adicionar no analisador sintático a expressão de referência a posição de matrizes como uma expressão válida.

5 A linguagem reconhecida

Ao fim da fase de análise semântica temos por fim uma visÃ£o integral e consisa do que pode ser escrito na linguagem e consequentemente interpretado pelo nosso compilador. Abaixo seguem as especificidades da nossa linguagem e como o analisador trata cada caso específico.

5.1 Expressões

Conforme especificação expressões não incluem, exceto em atribuições, literais ou caracteres. Mas incluem variáveis, acessos a posições de matrizes, chamadas de funções e números inteiros e reais, todos precedidos ou nÃ£o de sinais indicativo de número negativo. Ainda reconhece expressões lógicas com operadoes de comparaÃ§Ã£o e operadores lógicos.

É válido ressaltar que todos os operadores definidos ($+$,$-$,$*$,$/$,$<$,$>$,*e*, *ou* e outros) só podem ser executados sobre operandos de mesmo tipo e para os tipos que nÃ£o sejam caracter ou literal, ou matrizes de qualquer tipo sem seleção posição.

5.2 Estruturas de seleção e repetição

Estruturas de seleção aceitam expressções que retornem valores lógicos, seja apenas uma variável, as palavras reservadas $verdadeiro$ e $falso$ ou expressões complexas cujo resultado seja um lógico.

Comandos de laço avaliam expressões ógicas similarmente à estutura de seleção na parte condicional.

É de bom tom frizarmos que para o comando *selecione* os tipos inseridos dentro de um *caso* devem ser o mesmo da varivél usada para comparação, caso contrário, pela própria restrição feita em expressões, a comparação não poderá ser feita, sendo assim um erro semântico que será acusado pelo nosso compilador. Ainda nessa caminho, as iterações

automáticas do laço *faça* devem ser definidas sobre os mesmo tipos, também respeitando as restrições impostas ao operador de adição/subtração.

5.3 Declaração de variáveis

Na parte de declaração de variáveis o analisador semântico gera a tabela com as variáveis criadas e retorna erro em caso de encontrar duas declarações da mesma variável, independentemente de haver mudança de tipo ou não.

5.4 Inicialização de variáveis

Nosso analisador semântico faz ainda a verificação de variáveis que foram declaradas mas não chegaram a ser inicializadas, gerando assim um erro quando da sua utilização em alguma expressão (evitando lixos de memória) e emitindo um warning quando, excluindo o bloco de declaração, não aparece em parte alguma dentro do seu escopo.

5.5 Declaração de função

Funções devem ser declaradas no topo do programa e sempre antes de serem efetivamente utilizadas. Funções que tenham retorno devem apresentar o comando retorne como último comando da funçã, após o fechamento de todos os blocos intermediários. Sabemos que é uma limitação bastante expressiva, mas resultante de uma escolha infeliz realizada nas partes anteriores que, quando percebida, não dispunha mais de tempo hábil para que os ajustes necessários fossem realizados. Assim, nesta entrega o deixamos com essa limitação e nos propomos a corrigí-lo para a próxima, e definitiva, parte como um ajuste necessário à etapa de síntese.

5.6 Aridades de função

Ao verificar a declaração de uma função é feito ainda a contagem dos parâmetros que a mesma recebe definido assim sua aridade que é conferida a cada invocação da mesma acusando erro em caso de falta ou excesso de parâmer

5.7 Atribuição à variaveis

No momento da atribuição de uma expressão a uma variável faz-se a checagem dos tipo da variável e da expressão para saber se são compatíveis o que só pode ocorrer se a variável em questão tiver sido previamente declarada no bloco em que está sendo utilizada, que é o alvo da primeira verificação feita.

5.8 Matrizes

Matrizes, nesta etapa, são tratadas como um tipo especial de variável, tendo na sua estrutura uma flag indicando que se trata de uma matriz e as informações referente a dimensão e número de linhas e colunas. Conforme restição já documentada na entrega referente ao analisador sintático

Inicialização

A inicialização de um vetor (matriz de dimensão 1) deve ser realizada da seguinte forma:

```
//Exemplo para um vetor de 5 inteiros
mat = [1,2,3,4,5];
```

Já para uma matriz de dimensão 2 deve-se proceder da seguinte maneira:

```
//Exemplo para uma matriz[5][2] de inteiros
mat = {[1,2,3,4,5],[6,7,8,9,10]};
```

Atribuição a posição específica

Pode-se atribuir uma expressão, cujo resultado seja do mesmo tipo da matriz, a uma posição específica da matriz da formar mostrada a seguir:

```
//Exemplo para uma matriz de inteiros
mat[0][0] := 10+7;
```

Ressaltando que neste momento é feita verificação do tamanho de cada dimensão da matriz, evitando assim atribuições a espaços de memória errôneos.

Uso em expressões

Pode-se utilizar, como uma variável comum, numa expressão uma posição específica da matriz como segue:

```
//Exemplo para uma matriz de inteiros
    //variavel a  inteira
a := 10 + mat[0][0];
```

6 Casos de teste

Conforme solicitado na especificação do trabalho enviamos no diretório exemplos 15 exemplos de programas escrito na linguagem portugol aceita pelo nosso compilador. Destes, 5 escritos de forma correta, 5 apresentando erros sintáticos e 5 com erros semânticos.

6.1 Testes corretos

in1.gpt

Neste arquivo testa-se o bloco de declaração de variáveis, bem como atribuições.

in2.gpt

Neste arquivo testa-se as funções primitivas leia() e imprima(ARGUMENTOS) e as estrutura de seleção se-então-senão aninhado e um seleciona.

in3.gpt

Aqui aparecem as estruturas de repetição enquanto, faça-enquanto e para.

in4.gpt

Exemplo de um programa vazio.

in5.gpt

Exemplo com declaração e uso de funções e aninhamento de comandos de seleção e repetição.

6.2 Testes com erro: sintaxe

Os testes com erros são essencialmente os testes corretos com algum erro introduzido propositalmente e abaixo descritos.

inerr1.gpt

Introduzido erro na linha 10. Falta um : entre a variavel e o tipo.

inerr2.gpt

Introduzido erro na linha 28. Falta fechar aspas.

inerr3.gpt

Introduzido erro na linha 15. Foi trocado o simbolo de diferente ¡¿ por !=.

inerr4.gpt

Introduzido erro na linha 1. Foi removido o nome do algoritmo.

inerr5.gpt

Introduzido erro na linha 102. Foi acrescentado o : mas nao foi especificado o tipo de retorno.

6.3 Testes com erro: semântica

Os testes com erros são essencialmente os testes corretos com algum erro introduzido propositalmente e abaixo descritos.

inerr6.gpt

inerr7.gpt

inerr8.gpt

inerr9.gpt

inerr10.gpt

Falta fazer os exemplos errados desses e verificar compatibilidade dos corretos e errados anteriores..

É importante lembrar que em certos casos como a falta de ponto-e-vírgula ou delimitadores de repetição tais como fim-se, não necessariamente o erro ocorrerá na linha em que esta falta ocorre. Isto se deve ao fato de que o compilador irá procurar o delimitador até o final do programa por exemplo. Isto é comum em qualquer compilador moderno e portanto não é um erro.

7 Conclusão

Ao fim deste trabalho solidificou-se o conhecimento em relação a fase de análise semântica do processo de compilação através da programaçõ do comportamento de cada sentença. Sedimentando assim os conhecimentos abordados na teoria da aula de Compiladores e habilitando-nos para o passo seguinte na elaboração de um compilador: a etapa de síntese.