CEFET-MG
Centro Federal de Educação Tecnológica de Minas Gerais

Compilador

Analisador Semântico

Erick H. D. de Souza

Lucas C. Dornelas

Professora Orientadora: Kecia Aline Marques Ferreira

Maio de 2023

Sumário

Resumo

O analisador semântico é responsável por analisar o significado e a coerência de uma sentença, verificando se as expressões e construções utilizadas estão de acordo com as regras semânticas da linguagem. Ele vai além da análise sintática, que verifica apenas a estrutura gramatical, e busca compreender o sentido e a correta interpretação dos elementos presentes na sentença. No desenvolvimento do analisador semântico, foram utilizadas técnicas que permitem identificar e verificar informações semânticas, como a compatibilidade de tipos, a consistência de expressões e a unicidade de declaração de identificadores.

Palavras-chave: analisador semântico, regras semânticas, verificação de tipos, tabela de símbolos, compiladores, Java.

Introdução

A implementação de um analisador semântico é de grande importância no campo da ciência da computação e das linguagens de programação. Assim como o analisador sintático, o analisador semântico desempenha um papel fundamental na compreensão e análise de sentenças em uma determinada linguagem, porém seu foco está na interpretação do significado e na verificação da coerência semântica.

Este relatório descreve o desenvolvimento e a execução de testes utilizando a perspectiva de um analisador semântico. O objetivo principal dessa etapa foi criar um analisador capaz de analisar o significado e a coerência de sentenças em uma linguagem específica.

No desenvolvimento do analisador semântico, foram empregadas estruturas de dados como tabelas de símbolos e métodos de checagem de tipos para garantir a correção e a coerência das sentenças analisadas.

Espera-se que a implementação do analisador semântico resulte em um sistema eficiente e preciso, capaz de identificar e reportar erros semânticos em tempo hábil. O uso desse analisador pode contribuir para uma melhor compreensão do significado das sentenças em uma linguagem específica e encontrar aplicação em diversas áreas, como processamento de linguagem natural, tradução automática e verificação de conformidade de código.

Desenvolvimento

Forma de uso do compilador

Para utilizar o compilador, siga os passos abaixo:

1. Extraia o arquivo **`Compilers.zip`**.
2. Abra o terminal e navegue até o diretório onde foi criada a pasta extraída.
3. Execute o compilador utilizando o comando:

   `java -jar Compilers.jar <file-path>`.

   3.1. Para executar um dos programas presentes dentro da pasta **`programs`**, onde **`<file>`** deve ser substituído pelo nome do arquivo fonte:

   `java -jar Compilers.jar ./programs/<file>.`

   3.2. Para executar programas que não estão dentro da pasta **`programs`** é necessário que o caminho para o arquivo fonte seja informado corretamente em relação a pasta extraída.

   3.3. O compilador espera receber como parâmetro, opcionalmente, o parâmetro `-d` que ira ativar o modo de debug, que exibe informações sobre o processo de compilação.

   `java -jar Compilers.jar ./programs/<file> -d.`

4. Para a execução do compilador, é necessário que a versão java instalada na máquina seja a **19.0.1** ou superior.

Descrição da abordagem utilizada na implementação

A seguir, temos as principais classes criadas dentro do compilador que se referem a execução do analisador sintático:

Class Semantic

A classe Semantic é responsável por realizar a análise semântica do programa. Ela desempenha um papel fundamental na compreensão e verificação do significado das expressões e construções presentes no código fonte.

Essa classe é utilizada n parser para fazer as validações que as regras semânticas da linguagem. Dessa forma, durante a execução da análise sintática executamos as validações.

Class Parser

A implementação de um Parser LL(1) com recuperação de erro na inserção, que é responsável por receber como entrada os tokens identificados pelo analisador e criar uma representação da estrutura gramatical da sequência de tokens (árvore sintática). Além disso, também verifica se a ordem dos tokens respeita a sintaxe da linguagem.

Programas e Resultados

A seguir, temos os códigos fontes utilizados para a realização do analisador sintático, e podemos ver a sequência de expressões identificadas:

program teste1

```
 1   program teste1
 2       a, b is int;
 3       result is int;
 4       a, x is float;
 5   begin
 6       a = 12;
 7       x = 12.1;
 8       read (a);
 9       read (b);
10       read (c);
11       result = (a*b + 1) / (c+2);
12       write ({Resultado: });
13       write (result);
14
15   end.
```

O programa acima foi executado pelo analisador sintático e suas saídas foram a seguinte:

Teste 1:

```
utils.CompilerException: Linha: 4
Erro:
(EAT) Erro na leitura do token: Lexeme: a | Tag:ID
```

Conforme identificado pela saída, foi encontrado um semântico de unicidade na linha 4, onde foi declarado novamente um identificador “a”. Alterando o identificador de “a” para “c”, temos :

Teste 1 (Correção 1):

```
Linha: 11
Erro:
(FACTOR) Erro na atribuição de constantes, tipos incompatíveis
```

Conforme identificado pela saída, foi encontrado durante a atribuição que os tipos do identificador “result” não é o mesmo da expressão. Mudando a declaração do identificador para o tipo “int”, temos:

Teste 1 (Correção 2):

```
Analise concluída com sucesso.
```

Conforme identificado pela saída, após essa alteração, o código foi analisado com sucesso.

program teste2

```
 1   program teste2
 2       a, b, c is int;
 3       d, var_ is float;
 4   begin
 5       teste2 = 1;
 6       read (a);
 7       b = a * a;
 8       c = b + a/2 * (35/b);
 9       write (c);
10       val = 34.2;
11       c = val + 2.2 + a;
12       write (val);
13   end.
```

Teste 2:

```
Linha: 5
Erro:
(FACTOR) Erro na atribuição de constantes, tipos incompatíveis
```

Conforme identificado pela saída, foi encontrado um erro na atribuição na linha 5, onde o tipo teste2 não é o mesmo da constante 1, que é um inteiro. Alterando na linha 5 de "teste2" para "a", temos:

Teste 2 (Correção 3):

```
Analise concluída com sucesso.
```

program teste3

```
 1   program teste3
 2       a, aux is int;
 3       b is float;
 4   begin
 5       b = 0;
 6       read(a);
 7       read(b);
 8       if (a>b) then //troca variaveis
 9           aux = b;
```

```
 10           b = a;
 11           a = aux;
 12       end
 13       write(a);
 14       write(b);
 15   end.
```

Teste 3:

```
Linha: 5
Erro:
(FACTOR) Erro na atribuição de constantes, tipos incompatíveis
```

Conforme identificado pela saída, foi encontrado um erro na atribuição de b na linha 5, onde a constante 0 do tipo int não é igual à b, corrigindo essa atribuição. Sendo assim, agora temos a seguinte saída:

Teste 3 (Correção 1):

```
Analise concluída com sucesso.
```

Conforme identificado pela saída, foi analisado o código com sucesso.

program teste4

```
  1   program teste4
  2       pontuacao, pontuacaoMaxina, disponibilidade is int;
  3       pontuacaoMinima is char;
  4   begin
  5       pontuacaoMinima = 50;
  6       pontuacaoMaxima = 100;
  7       write({Pontuacao do candidato: });
  8       read(pontuacao);
  9       write({Disponibilidade do candidato: });
 10       read(disponibilidade);
 11
 12       while (pontuacao>0 && (pontuacao<=pontuacaoMaxima)) do
 13           if ((pontuação > pontuacaoMinima) && (disponibilidade==1))
      then
 14               write({Candidato aprovado.});
 15           else
 16               write({Candidato reprovado.});
 17           end
 18
 19           write({Pontuacao do candidato: });
 20           read(pontuacao);
 21           write({Disponibilidade do candidato: });
 22           read(disponibilidade);
 23       end
 24   end.
```

Teste 4:

```
Linha: 5
Erro:
(FACTOR) Erro na atribuição de constantes, tipos incompatíveis
```

Conforme identificado pela saída, foi encontrado um erro de atribuição na linha 8, onde “pontuacaoMinima” possui um tipo diferente da constante 50. Sendo assim, agora temos a seguinte saída:

Teste 4 (Correção 1):

```
Analise concluída com sucesso.
```

Conforme identificado pela saída, foi analisado o código com sucesso.

program teste5

```
 1   /* Teste do meu compilador */
 2
 3   program teste5
 4       a, b, c, maior is int;
 5       outro is char;
 6   begin
 7           repeat
 8               write({A});
 9               read(a);
10               write({B});
11               read(b);
12               write({C});
13               read(c);
14
15
16
17               if ( (a>b) && (a>c) ) then
18                   maior = a;
19
20               else
21                   if (b>c) then
22                       maior = b;
23
24                   else
25                       maior = c;
26                   end
27               end
28               write({Maior valor:});
29               write (maior);
30               write ({Outro? (S/N)});
```

```
 31              read(outro);
 32          until (outro == 'N' || outro == 'n')
 33   end.
```

Teste 5:

```
Analise concluída com sucesso.
```

Conforme identificado pela saída, não foi encontrado nenhum erro.

program order

```
 1   program order
 2     num1, num2, temp is int;
 3   begin
 4
 5     read(num1);
 6     read(num2);
 7
 8     if num1 > num2 then
 9       temp = num1;
10       num1 = num2;
11       num2 = temp;
12     end
13
14     write({Menor: });
15     write(num1);
16     write({Maior: });
17     write(num2);
18   end.
```

order:

```
Compilação concluída com sucesso.
```

Conforme identificado pela saída, não foi encontrado nenhum erro também.

program factorial

```
 1   program factorial
 2     num, fact is int;
 3   begin
 4     read(num);
 5     fact = 1;
 6     i = 1;
 7     while i <= num do
 8       fact = fact * i;
 9       i = i + 1;
10     end
11     write(fact);
12   end.
```

factorial:

```
Linha: 6
Erro:
Elemento não encontrado
```

Conforme identificado pela saída, não foi encontrado o identificador “i”, dessa forma devemos defini-lo como um inteiro. Sendo assim, agora temos a seguinte saída:

factorial (Correção 1):

```
Compilação concluída com sucesso.
```

Conforme identificado pela saída, não foi encontrado um erro. Sendo assim, foi analisado o código com sucesso.

Conclusão

Em conclusão, a implementação do analisador semântico como parte deste trabalho foi bem-sucedida, atingindo os objetivos propostos. O analisador demonstrou ser capaz de analisar o significado e a coerência das sentenças, identificando erros semânticos e garantindo a consistência das operações realizadas no programa. A utilização de técnicas como a construção de tabelas de símbolos e a aplicação de regras e algoritmos adequados foi fundamental para a eficiência da análise semântica.

A partir deste trabalho, abre-se espaço para possíveis melhorias e aprimoramentos no analisador semântico. Pode-se explorar a incorporação de regras semânticas mais complexas e o uso de técnicas avançadas de análise contextual, visando aprimorar a precisão e a abrangência da análise realizada. Essas melhorias contribuíram para uma análise semântica mais completa e refinada, auxiliando no desenvolvimento de sistemas mais confiáveis e livres de erros semânticos.