= Roteiro 05 =

Compiladores – Análise Sintática - Semântica

Prof. José Rui
jose.castro@ifsudestemg.edu.br

Acompanhamento 1

JavaCUP possui muitos recursos que não foram explorados na especificação `ex1.cup`. Um deles é a inserção de ações semânticas.

Ações semânticas são trechos de código associados a cada produção da gramática, que são executados no momento em que a produção é utilizada.

Em JavaCUP, são definidas dentro de blocos **{: :}**. Portanto dentro desses delimitadores podemos inserir qualquer código java.

Para que o seu código java criado dentro do bloco {: :} consiga acessar o símbolo da sua regra de produção é necessário especificar `identificadores` para cada símbolo(variável ou terminal).
Ex: `expr:e {: System.out.println("= " + e); :};`
Onde:
- `expr`: é o simbolo
- `e`: é o identificador (uma espécie de varíavel no java)

Insira as ações semânticas sugeridas abaixo, na especificação expr.cup:

```
expr_ptv ::=
      expr:e {: System.out.println("= " + e); :} PTVIRG;

expr ::=
      INTEIRO:n MAIS expr:e
      {: RESULT = new Integer(n.intValue() + e.intValue()); :}
  | INTEIRO:n MENOS expr:e
      {: RESULT = new Integer(n.intValue() - e.intValue()); :}
  | INTEIRO:n
     {: RESULT = new Integer(n.intValue()); :}
  ;
```

Essas ações semânticas retornam os o valor de suas expressões, criando um objeto quando necessário.

É importante ressaltar, que ordem em que inserimos os códigos associados, fará toda diferença. Isto significa que, se modificarmos o local onde colocamos as ações associadas o resultado será outro.

Outro fator importante é não perder de vista a árvore de derivação, isto é, você deve sempre estar com um papel ao lado fazendo a derivação para enteder perfeitamente o que está acontecendo.

Por fim, não se pode esquecer que isso tudo é feito de forma recursiva!

EXEMPLO 1: Vamos mandar o nosso parser imprimir a palavra "Ola mundo" imediatamente antes de casar(match) com o simbolo MAIS, e a frase "mundo dos numeros" imediatamente após expr virar INTEIRO.

```
expr_ptv ::=
      expr:e {: System.out.println("= " + e); :} PTVIRG;

expr ::=
      INTEIRO:n {:System.out.print("Ola mundo");:} MAIS expr:e
      {: RESULT = new Integer(n.intValue() + e.intValue()); :}
  | INTEIRO:n MENOS expr:e
      {: RESULT = new Integer(n.intValue() - e.intValue()); :}
  | INTEIRO:n
    {: System.out.print("mundo dos numeros");
        RESULT = new Integer(n.intValue()); :}
  ;
```

Quando entrarmos com a seguinte expressão: 1+2; teremos a seguinte tela:

```
1+2;
Ola mundo
mundo dos numeros
=3.0
```

Agora vamos alterar o local da ação associada da mensagem "Ola mundo" para após a derivação expr → INTEIRO MAIS expr.

```
expr_ptv ::=
      expr:e {: System.out.println("= " + e); :} PTVIRG;

expr ::=
      INTEIRO:n MAIS expr:e
      {: System.out.print("Ola mundo");
         RESULT = new Integer(n.intValue() + e.intValue()); :}
  | INTEIRO:n MENOS expr:e
      {: RESULT = new Integer(n.intValue() - e.intValue()); :}
  | INTEIRO:n
    {: System.out.print("mundo dos numeros");
        RESULT = new Integer(n.intValue()); :}
  ;
```

Teremos a seguinte saída:

```
1+2;
mundo dos numeros
Ola mundo
=3.0
```

Por que isso acontece?
- Isso ocorre por causa da recursividade
- Para exergar isso, você deve desenhar no papel a arvore de derivação. Não tente enteder sem fazer a árvore(o slide da aula pode te ajudar na parsetree);

Acompanhamento 2

Ao invés de usar “Ola mundo” e “mundo dos numeros” vamos mandar imprimir a regra de produção que foi usada. Isso facilitará nosso entendimento, e conseguiremos ver nossa gramática em ação.

```
expr ::=
      INTEIRO:n MAIS expr:e
      {: System.out.print(““expr -> INTEIRO MAIS expr””);
         RESULT = new Integer(n.intValue() + e.intValue()); :}
  | INTEIRO:n MENOS expr:e
      {: RESULT = new Integer(n.intValue() - e.intValue()); :}
  | INTEIRO:n
    {: System.out.print(“expr -> INTEIRO”);
        RESULT = new Integer(n.intValue()); :}
  ;
```

Teremos a seguinte saída:

```
1+2;
expr -> INTEIRO
expr -> INTEIRO MAIS expr
=3.0
```

- Note que foi impresso a àrvore de derivação... Porém na ordem inversa! Por que?

- Faça isso para todas as regras de produções. Ao fazer isso, quando você inserir uma expressão ponto e virgula(`1+2+8;`) deve aparecer toda a arvore de derivação deste `expr_list` até `INTEIRO`;

Acompanhamento 3

Uma forma de apresentar melhor os dados, é mostrar o valor do numero inteiro que foi encontrado. Para isso deve-se usar o `identificador` que você definiu para `INTEIRO`;

```
expr ::=
      INTEIRO:n MAIS expr:e
      {: System.out.print(“expr -> INTEIRO(”+n+”) MAIS expr”);
         RESULT = new Integer(n.intValue() + e.intValue()); :}
  | INTEIRO:n MENOS expr:e
      {: RESULT = new Integer(n.intValue() - e.intValue()); :}
  | INTEIRO:n
    {: System.out.print(“expr -> INTEIRO”);
        RESULT = new Integer(n.intValue()); :}
  ;
```

A saída agora fica um pouco mais entendível:

```
1+2;
expr -> INTEIRO
expr -> INTEIRO(1.0) MAIS expr
=3.0
```

Atividade 1

Faça as devidas alterações para obter a seguinte saída:

```
1+2+3;
expr -> INTEIRO(3.0)
expr -> INTEIRO(2.0) MAIS expr
expr -> INTEIRO(1.0) MAIS expr
=6.0
```

Atividade 2

Faça as devidas alterações para obter a saída da arvore de derivação na ordem correta, conforme mostrada abaixo:

```
1+2+3;
expr -> INTEIRO(1.0) MAIS expr
expr -> INTEIRO(2.0) MAIS expr
expr -> INTEIRO(3.0)
=6.0
```

Até aqui nós só utilizamos o arquivo `.cup` para fazer a impressão. Mas isso também é totalmente possível de ser feito no arquivo `.lex`. Afinal lá também é permitido colocar código java a vontade. Para você entender e mostrar que entendeu, você deverá mandar imprimir o valor do `INTEIRO` lá no arquivo `.lex`. Para diferenciar, coloque a palavra `scanner` antes do numero. Ex: *scanner: 4.0*

Faça as alterações necessárias, teste várias vezes e perceba que, por causa da recursividade ele normalmente é o primeiro a ser jogado na tela. Para finalizar, discorra sobre o que você fez e percebeu.

Instruções para entrega

Entregue os arquivos expr.cup e scanner.flex para o professor. Compacte-os com o nome da dupla.