Tradução Dirigida pela Sintaxe

(Extraído do livro “Compiladores – Aho et al” e adaptado de www.ici.unifei.edu.br/ramos/download/Compiladores/CO-03.ppt)

Tradução Dirigida pela Sintaxe

Corresponde a tradução de linguagens guiadas por gramáticas livres de contexto, onde se associa informações a uma construção de linguagem de programação, atrelando atributos aos símbolos gramaticais que representam a construção.

Existem duas notações básicas:

1. Definições dirigidas pela sintaxe, que servem para especificar traduções em termos dos atributos associados a seus componentes sintáticos; e

2. Esquemas de tradução, que especificam traduções a partir de notações mais procedimentais.

Definições Dirigidas pela Sintaxe

São definições que usam uma gramática livre de contexto para especificar a estrutura sintática da entrada. A cada símbolo da gramática associa-se um conjunto de atributos, e a cada produção associa-se um conjunto de regras semânticas para computar os valores dos atributos associados aos símbolos que figuram naquela produção.

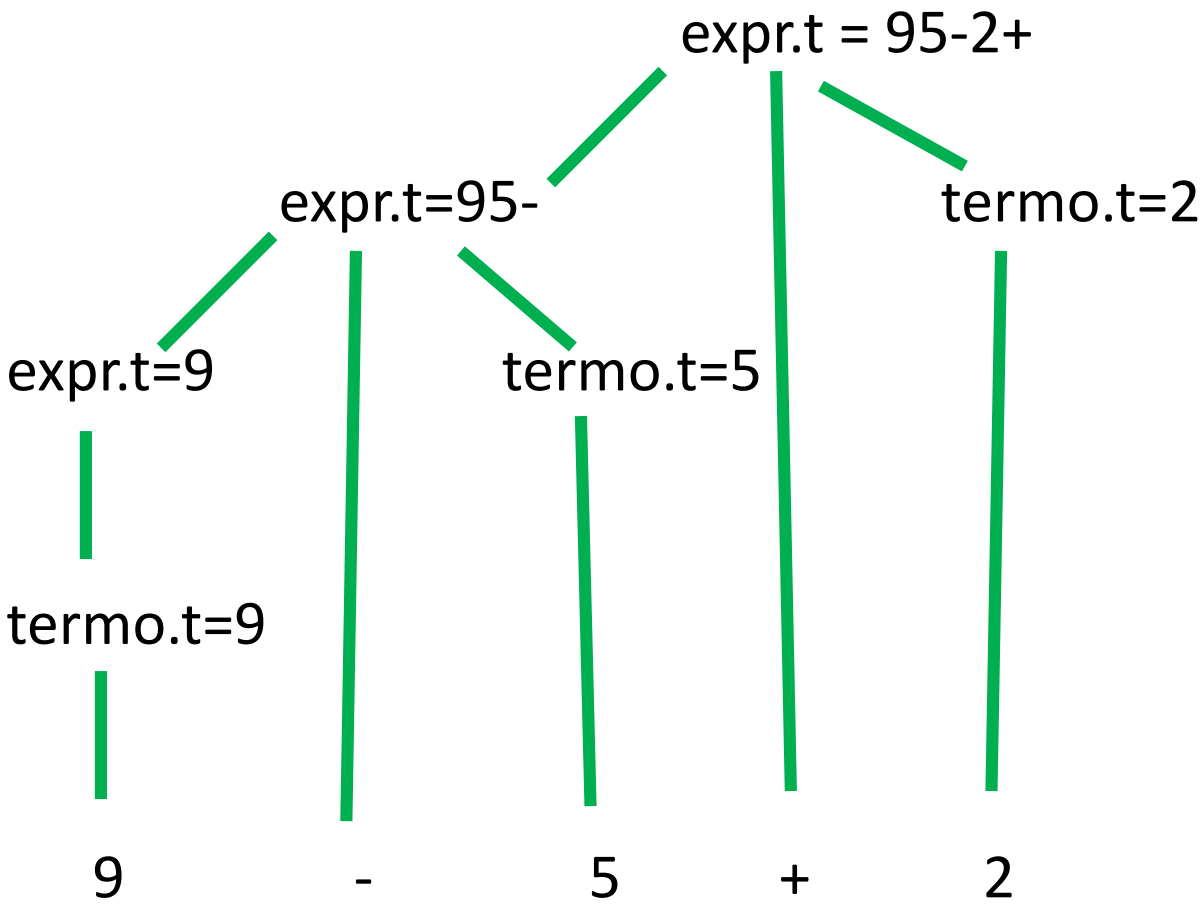
A gramática e o conjunto de regras semânticas constituem a definição dirigida pela sintaxe.

Exemplo

Definição Dirigida pela Sintaxe para a tradução da notação infixa para a pósfixa :

Produção	Regra Semântica
expr->$expr_1$+termo	expr.t:= $expr_1$.t|| termo.t|| ‘ + ‘
expr-> $expr_1$-termo	expr.t:= $expr_1$.t||termo.t|| ‘ - ‘
expr-> termo	expr.t:= termo.t
expr-> termo	expr.t:= termo.t
termo-> 0	termo.t:= ‘ 0 ‘
.....	.....
termo-> 9	termo.t:= ‘ 9 ‘

Árvore Gramatical da Expressão 9-5+2

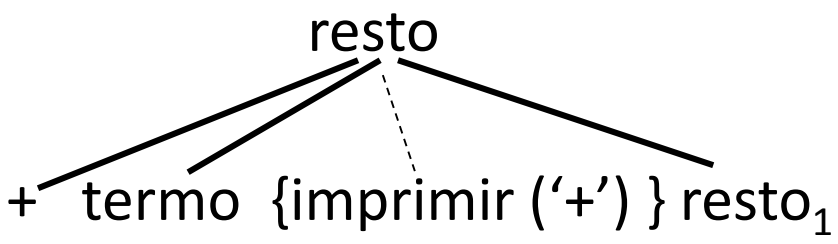


Esquemas de Tradução

Outra notação para associar regras semânticas às produções é um Esquema de Tradução,

- Trata-se de uma gramática livre de contexto na qual fragmentos de programas, chamados "ações semânticas", são inseridos nos lados direitos das produções.
- A posição à qual uma ação deve ser executada é mostrada envolvendo-a entre chaves e escrevendo-a no lado direito da produção. Exemplo:

resto -> + termo {imprimir ('+')} $resto_1$



A ação semântica será realizada depois que a sub-árvore para termo seja percorrida, mas antes que o filho para $resto_1$ seja visitado

Emitindo uma Tradução

As ações semânticas nos esquemas de tradução irão escrever a saída de uma tradução num arquivo, uma cadeia ou um caractere de cada vez.

Uma Definição Dirigida pela Sintaxe possui a cadeia representando a tradução do não terminal ao lado esquerdo de cada produção sendo a concatenação das traduções dos não terminais à direita, na mesma ordem que na produção, com algumas cadeias adicionais (ou nenhuma) entremeadas.

Exemplo:

Produção	Regra Semântica
expr->expr$_1$ + termo	expr.t:= expr.t || termo.t || '+'

A tradução de **expr.t** é a concatenação das traduções de **expr$_1$** e **termo**, seguida pelo símbolo **+**. Note que **expr$_1$** figura antes de **termo** no lado direito da produção.

Emitindo uma Tradução

Definições simples dirigidas pela sintaxe podem ser implementadas através de esquemas de tradução onde as ações imprimam as cadeias adicionais na ordem em que elas aparecem na definição. Exemplos:

expr -> $expr_1$ + termo {imprimir ('+')}

resto -> + termo {imprimir ('+')} resto

Análise Sintática

É um processo de se determinar se uma cadeia de tokens pode ser gerada por uma gramática.

Na Análise Sintática, pode-se imaginar que “uma árvore gramatical está sendo construída, ainda que o compilador não a construa efetivamente“.

Entretanto, um analisador sintático precisa ser capaz de constituir uma arvore ou então a compilação não poderá ser garantida correta.

De acordo com a forma de construção dos nós da árvore sintática, existem duas classes de Analisadores Sintáticos:

1. **Top-down**, a construção inicia na raiz e prossegue em direção às folhas; e
2. **Bottom-up**, a construção inicia nas folhas e continua até a raiz.

Análise Sintática Top-down

A seguinte gramática gera um subconjunto dos tipos de Pascal:

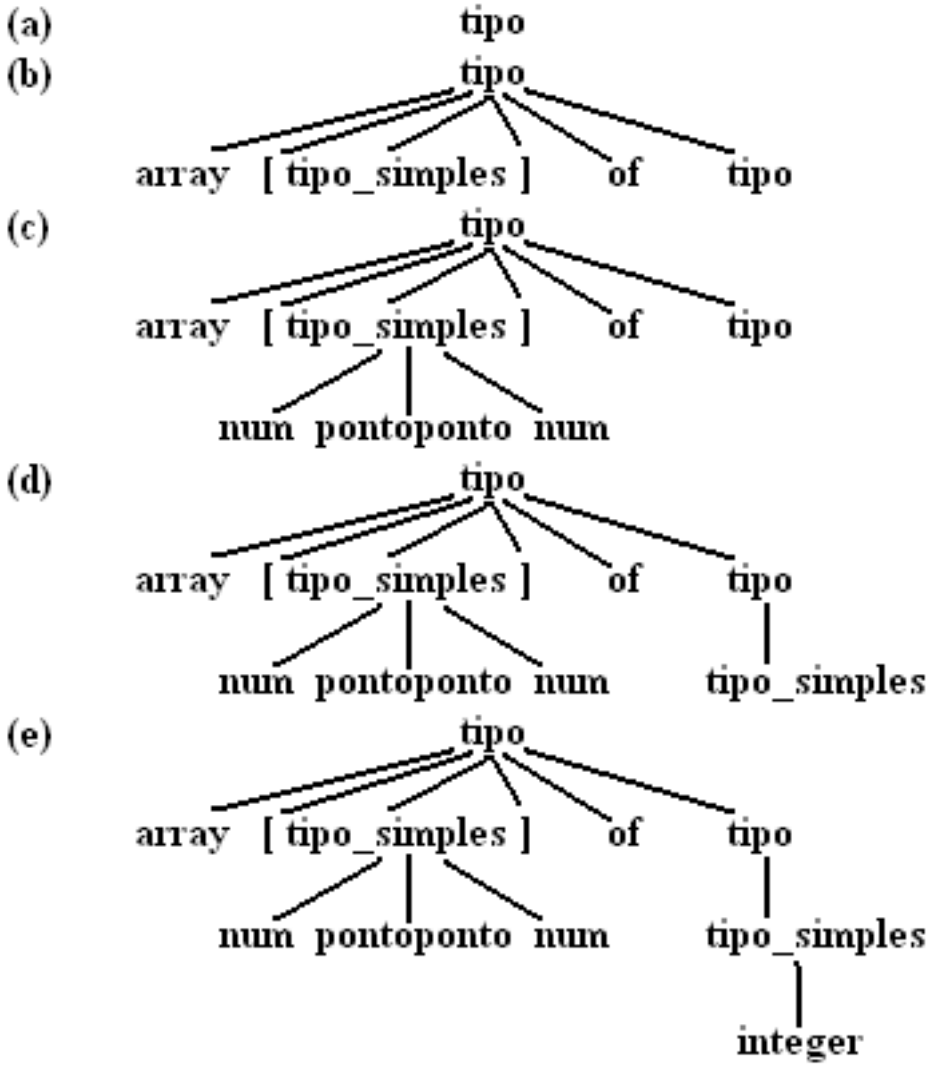
tipo $\rightarrow$ tipo_simples | $\uparrow$**id** | **array** [tipo_simples] **of** tipo

tipo_simples $\rightarrow$ **integer** | **char** | num .. num

A construção de uma árvore sintática top-down é feita iniciando-se pela raiz, rotulada pelo não-terminal de partida e realizando-se repetidamente os seguintes passos:

1. Ao nó "**n**" rotulado por um não terminal "**A**", selecione uma das produções de **A** e construa os filhos de "**n**" com os símbolos do lado direito da produção.
2. Encontre o próximo nó no qual uma sub-árvore deva ser construída.

Construção top-down da árvore sintática



Árvore sintática preditiva

O método de análise sintática top-down é um método de análise descendente recursiva onde a entrada é processada através de um conjunto de procedimentos recursivos que são associados a cada não terminal de uma gramática.

A árvore sintática preditiva é um método especial de análise gramatical descendente recursiva na qual o símbolo de entrada determina, de forma não ambígua, o procedimento selecionado para cada não terminal.

Projeto de um Analisador Sintático Preditivo

Um analisador sintático preditivo é um programa consistindo em um procedimento para cada não terminal.

Cada procedimento realiza 2 ações:

1. Decide que produção usar, através do exame do símbolo de entrada
2. Empilha o símbolo de entrada e o desempilha de acordo com a prioridade das operações a ele referidas

Um Tradutor para Expressões Simples

Podemos agora construir um tradutor dirigido pela sintaxe sob a forma de um programa Pascal que traduz expressões aritméticas na forma infixa para a forma posfixa. Para a especificação para o tradutor utilizaremos o seguinte esquema:

expr -> expr + termo	{imprimir (‘+’)}
expr -> expr - termo	{imprimir (‘-’)}
expr -> expr * termo	{imprimir (‘*’)}
expr -> expr / termo	{imprimir (‘/’)}
expr -> termo	
termo -> 0...9	{imprimir (‘0’,...,’9’)}

Algoritmo para Conversão de Infixa para Posfixa

Para chegar a esse algoritmo, observamos que a ordem dos operandos não é alterada quando a expressão é convertida: eles são copiados para a saída logo que encontrados. Por outro lado os operadores devem mudar de ordem, já que na posfixa ele aparecem logo depois dos seus operandos.

Numa expressão posfixa, as operações são efetuadas na ordem em que aparecem. Logo, o que determina a posição de um operador na forma posfixa é justamente a prioridade que ele tem na forma infixa. Aqueles operadores de maior precedência aparecem primeiro na expressão de saída e seu escopo será definido por parênteses.

Algoritmo para Definir Prioridades dos Operadores

```
Função Prio (S: caracter): inteiro;
Inicio
  Caso S seja
        '(': Prio <- 1;
        '+',
        '-': Prio <- 2;
        '*'
        '/': Prio <- 3;
   fim;
Fim;
```

Conversão de Infixa para Posfixa

A conversão de uma expressão infixa qualquer para posfixa, utilizando pilhas, pode ser realizada assim:

1. Inicie com a pilha vazia;
2. Realize uma varredura na expressão infixa, copiando todos os identificadores encontrados diretamente na expressão de saída:
    - 2.1. Ao encontrar o operador
        - 2.1.1. Enquanto a pilha não estiver vazia e houver no seu topo um operador com prioridade maior ou igual ao encontrado, desempilhe o operador e copie-o na saída;
        - 2.1.2. Empilhe o operador encontrado
    - 2.2. Ao encontrar um parêntese de abertura, empilhe-o;
    - 2.3 Ao encontrar uma parêntese de fechamento, remova o símbolo da pilha e copie-o na saída, até que seja desempilhado o parêntese de abertura correspondente
3. Ao final da varredura, esvazie a pilha, movendo os símbolos desempilhados para a saída.

Exercício

Veja o funcionamento do algoritmo na conversão da expressão

A * (B + C) / D

Símbolo	Ação	Pilha	Saída
A	copia para a saída	P:[ ]	A

Algoritmo de Conversão

1. Inicie com a pilha vazia;
2. Realize uma varredura na expressão infixa, copiando todos os identificadores encontrados diretamente na expressão de saída:
   2.1. Ao encontrar o operador
      2.1.1. Enquanto a pilha não estiver vazia e houver no seu topo um operador com prioridade maior ou igual ao encontrado, desempilhe o operador e copie-o na saída;
         2.1.2. Empilhe o operador encontrado
   2.2. Ao encontrar um parêntese de abertura, empilhe-o;
   2.3 Ao encontrar uma parêntese de fechamento, remova o símbolo da pilha e copie-o na saída, até que seja desempilhado o parêntese de abertura correspondente
3. Ao final da varredura, esvazie a pilha, movendo os símbolos desempilhados para a saída.

Solução

Símbolo	Ação	Pilha	Saída
A	copia para a saída	P:[ ]	A
*	pilha vazia, empilha	P:[*]	A
(	sempre deve ser empilhado	P:[(,*]	A
B	copia para a saída	P:[(,*]	AB
+	prioridade maior, empilha	P:[+,(,*]	AB
C	copia para a saída	P:[+,(,*]	ABC
)	desempilha até achar ‘(’	P:[*]	ABC+
/	prioridade igual, desempilha	P:[/]	ABC+*
D	copia para a saída	P:[/]	ABC+*D
	final, esvazia a pilha	P:[ ]	ABC+*D/

Algoritmo para a Geração da Forma Posfixa

```
Função Posfixa (E: string): string;
Var     P: pilha;   S: string;   i: inteiro;   x: caracter;
Início
    Init (P);    {inicializa a pilha}
    S← ‘ ‘;
    Para i ← 1 até length(E) Faça  Caso E[i] seja
      ‘A’,...,’Z’: S ← E[i];
      ‘+’, ’-’, ‘*’, ’/’:  Início Enquanto não IsEmpty(P) e (Prio(Top(P)) >= Prio(E[i]))
                                  Faça    S ← S + Pop (P);
                                  Push (P,E[i]);
                           Fim;
       ‘(‘ :  Push (P,E[i]);
       ‘)’ :  Início   Enquanto Top(P) <> ‘(‘ Faça  S ← S + Pop(P);
                    Posfixa  ← S;
              Fim;
        Fim;
       Enquanto não IsEmpty (P) Faça S ← S + Pop(P);
       Posfixa ← S;
Fim.
```

Implementação

Para implementar o algoritmo anterior, usa-se um vetor com capacidade para armazenar 26 números reais. Os elementos deste vetor poderiam ser indexados por números inteiros variando de 1 até 26. Entretanto neste caso usar-se-hão caracteres ao invés de inteiros pois a linguagem Pascal assim o permite, exemplo:

```
Tipo
      Valores = vetor [‘A’..’Z’] de real;
  Var
      V: Valores;
```

Algoritmo para a Definição de Variáveis do Tradutor

```
Procedimento Atribui (var V: Valores);
Var N: caracter;
Início
   Escreva (‘Digite . Para terminar! ‘);
   Repita
      Escreva (‘Nome: ‘);
      Leia (N);
      Se N in [ ‘A’..’Z’ ] Então
         Início
               Escreva (‘Valor: ‘);
               Leia (V[N]);
         Fim;
    até que N = ‘ . ‘;
Fim;
```

Observações

Observar que V é passado por referência a Atribui( ), isto significa que qualquer alteração realizada na variável permanecerá mesmo após a execução da rotina. Um laço (Repita) é executado até que o usuário digite um ponto. A cada caractere lido, o condicional Se testa se é um nome válido para a variável; caso seja, o valor correspondente é solicitado.

Somente as variáveis cujos nomes forem indicados pelo usuário serão inicializadas, as demais permanecerão com valor indefinido.

Algoritmo para Avaliar Expressões

```
Função Avalia (E:string, V: Valores): real;
Var     P:Pilha;   x, y: real;   i: inteiro;
Início
    Init (P);
    Para i ← 1 até length (E) Faça
        SeE[i] in ['A'..'Z']  Então Push(P,V[E[i]])
          Senão Se E[i] in ['+','-','*','/'] Então
             Início     Y ← Pop(P);
                        X ← Pop(P);
                        Caso E[i] seja '+': Push (P,x+y);
                                       '-': Push (P,x-y);
                                       '*': Push (P,x*y);
                                       '/': Push (P,x/y);
                        Fim;
              Fim;
     Avalia ← Pop(P);
Fim;
```

Exercícios

1. O que você entende por: árvore de derivação e análise gramatical? Dê exemplos.

2. Desenvolver um programa em linguagem C onde, a partir dos algoritmos apresentados, seja realizado um tradutor, sendo que as expressões fornecidas na entrada estarão na forma infixa e na saída deste seja apresentada a expressão correspondente na forma posfixa bem como o respectivo resultado do cálculo.