

Paradigmas de Linguagens de Programação

Descrevendo a Sintaxe e a Semântica

Cristiano Lehrer, M.Sc.

Introdução

- Descrição, compreensível, de uma linguagem de programação é difícil e essencial.
- Capacidade de determinar como as expressões, instruções e unidades são formadas e suas intenções de efeito quando executadas.
- Pode ser dividida em:
 - Sintaxe → forma.
 - Semântica → significado.
- Sintaxe e semântica são bastante relacionados.
- Descrever sintaxe é mais fácil do que a semântica.

Sintaxe (1/2)

- Linguagem:
 - Conjunto de cadeias de caracteres de algum alfabeto.
- As cadeias são chamadas sentenças ou instruções.
- Regras sintáticas de uma linguagem especificam quais cadeias do alfabeto pertencem à linguagem.
- Lexema:
 - Identificadores, constantes, operadores e palavras especiais.
- Token:
 - Categoria de lexemas:
 - Identificador, operação de adição,

Sintaxe (2/2)

```
index = 2 * count + 17;
```

Lexemas

index

=

2

*

count

+

17

;

Tokens

identificador

operador de atribuição

constante numérica inteira

operador de multiplicação

identificador

operador de adição

constante numérica inteira

ponto e vírgula

Reconhecedores

- Suponha:
 - Linguagem L sobre um alfabeto Σ .
- Para definir L usando o método de reconhecimento é preciso construir um mecanismo R de tal forma que quando uma cadeia for fornecida para este mecanismo, este diga se esta cadeia pertence ou não a L .
- É como um filtro que separa as sentenças corretas das erradas.
- Reconhecimento não é usado para enumerar todas as sentenças de uma linguagem.
- A análise sintática de um compilador é um reconhecedor, determinando apenas se um programa está na linguagem.

Geradores

- Dispositivo usado para gerar sentenças de uma linguagem.
- É como um botão que, quando pressionado, produz uma sentença da linguagem.
- Não é claro que sentença será produzida:
 - Parece ser pouca utilizada como descritor de linguagem.
- Para um programador, um reconhecedor (análise sintática) não é tão útil:
 - Só pode ser usado na tentativa e erro.
- Geração é preferido a reconhecimento porque é mais fácil de ler e entender.

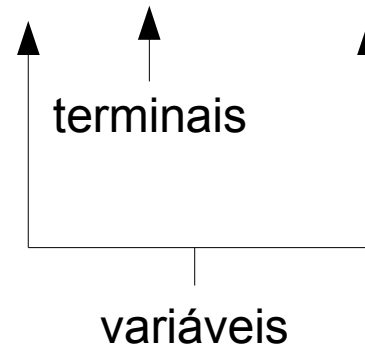
Forma de Backus-Naur (1/6)

- Popularmente conhecida como BNF.
- É um método formal para descrição da sintaxe.
- Origem:
 - [ALGOL 58](#) e [ALGOL 60](#).
- É uma notação natural de descrever a sintaxe.
- Uma metalinguagem é uma linguagem usada para descrever outra linguagem.
- BNF é uma metalinguagem para linguagens de programação.

Forma de Backus-Naur (2/6)

- BNF utiliza abstrações para representar estruturas sintáticas.
- As abstrações são geralmente chamadas de símbolos não-terminais (variáveis).
- Os lexemas e tokens são chamados símbolos terminais.
- As definições são chamadas de regras.

$\langle \text{assign} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expression} \rangle$



Forma de Backus-Naur (3/6)

- Uma gramática é uma coleção de regras.
- Símbolos não-terminais podem ter mais de um definição:

$\langle \text{if_stmt} \rangle \rightarrow \text{if} \langle \text{logic_expr} \rangle \text{then} \langle \text{stmt} \rangle$
 $\quad \quad \quad | \quad \text{if} \langle \text{logic_expr} \rangle \text{then} \langle \text{stmt} \rangle \text{else} \langle \text{stmt} \rangle$

- Listas de tamanhos variáveis:

$\langle \text{ident_list} \rangle \rightarrow \langle \text{identifier} \rangle$
 $\quad \quad \quad | \quad \langle \text{identifier} \rangle , \langle \text{ident_list} \rangle$

Forma de Backus-Naur (4/6)

- BNF é um dispositivo de geração de linguagens.
- Sentenças são geradas através da aplicação de regras.
- Inicia com um conjunto não-terminal especial:
 - start symbol.
- Esta geração de sentença é chamada de derivação.
- O start symbol representa um programa completo.

Forma de Backus-Naur (5/6)

$\langle \text{program} \rangle \rightarrow \text{begin } \langle \text{stmt_list} \rangle \text{ end}$

$\langle \text{stmt_list} \rangle \rightarrow \langle \text{stmt} \rangle$
 $\quad | \quad \langle \text{stmt} \rangle ; \langle \text{stmt_list} \rangle$

$\langle \text{stmt} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expression} \rangle$

$\langle \text{var} \rangle \rightarrow \mathbf{A} \quad | \quad \mathbf{B} \quad | \quad \mathbf{C}$

$\langle \text{expression} \rangle \rightarrow \langle \text{var} \rangle + \langle \text{var} \rangle$
 $\quad | \quad \langle \text{var} \rangle - \langle \text{var} \rangle$
 $\quad | \quad \langle \text{var} \rangle$

Forma de Backus-Naur (6/6)

- Uma derivação de um programa nesta linguagem:

```
< program >  
begin < stmt_list > end  
begin < stmt >; < stmt_list > end  
begin < var > = < expression >; < stmt_list > end  
begin A = < expression >; < stmt_list > end  
begin A = < var > + < var >; < stmt_list > end  
begin A = B + < var >; < stmt_list > end  
begin A = B + C; < stmt_list > end  
begin A = B + C; < stmt > end  
begin A = B + C; < var > = < expression > end  
begin A = B + C; B = < expression > end  
begin A = B + C; B = < var > end  
begin A = B + C; B = C end
```

Derivação (1/2)

- Um outro exemplo de gramática:

$\langle \text{assign} \rangle \rightarrow \langle \text{id} \rangle = \langle \text{expr} \rangle$

$\langle \text{id} \rangle \rightarrow \mathbf{A} \mid \mathbf{B} \mid \mathbf{C}$

$\langle \text{expr} \rangle \rightarrow \langle \text{id} \rangle + \langle \text{expr} \rangle$

| $\langle \text{id} \rangle * \langle \text{expr} \rangle$

| $(\langle \text{expr} \rangle)$

| $\langle \text{id} \rangle$

Derivação (2/2)

- Derivação à Esquerda (DEE):

< assign >
< id > = < expr >
A = < expr >
A = < id > * < expr >
A = B * < expr >
A = B * (< expr >)
A = B * (< id > + < expr >)
A = B * (A + < expr >)
A = B * (A + < id >)
A = B * (A + C)

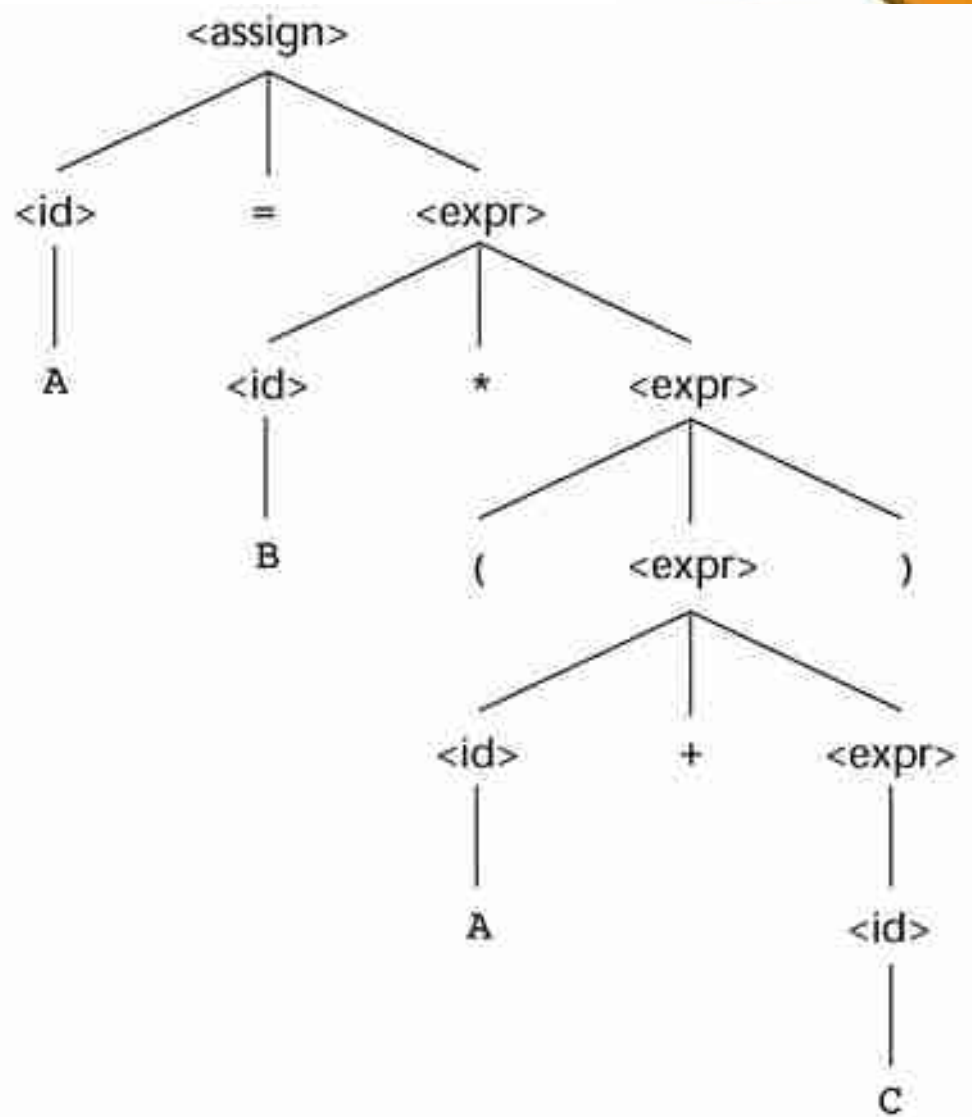
Extrema

- Derivação à Extrema Direita (DED):

< assign >
< id > = < expr >
< id > = < id > * < expr >
< id > = < id > * (< expr >)
< id > = < id > * (< id > + < expr >)
< id > = < id > * (< id > + < id >)
< id > = < id > * (< id > + C)
< id > = < id > * (A + C)
< id > = B * (A + C)
A = B * (A + C)

Árvore de Derivação

- Gramáticas descrevem uma estrutura hierárquica das sentenças.
- Essa estrutura hierárquica é chamada de *parse tree*.



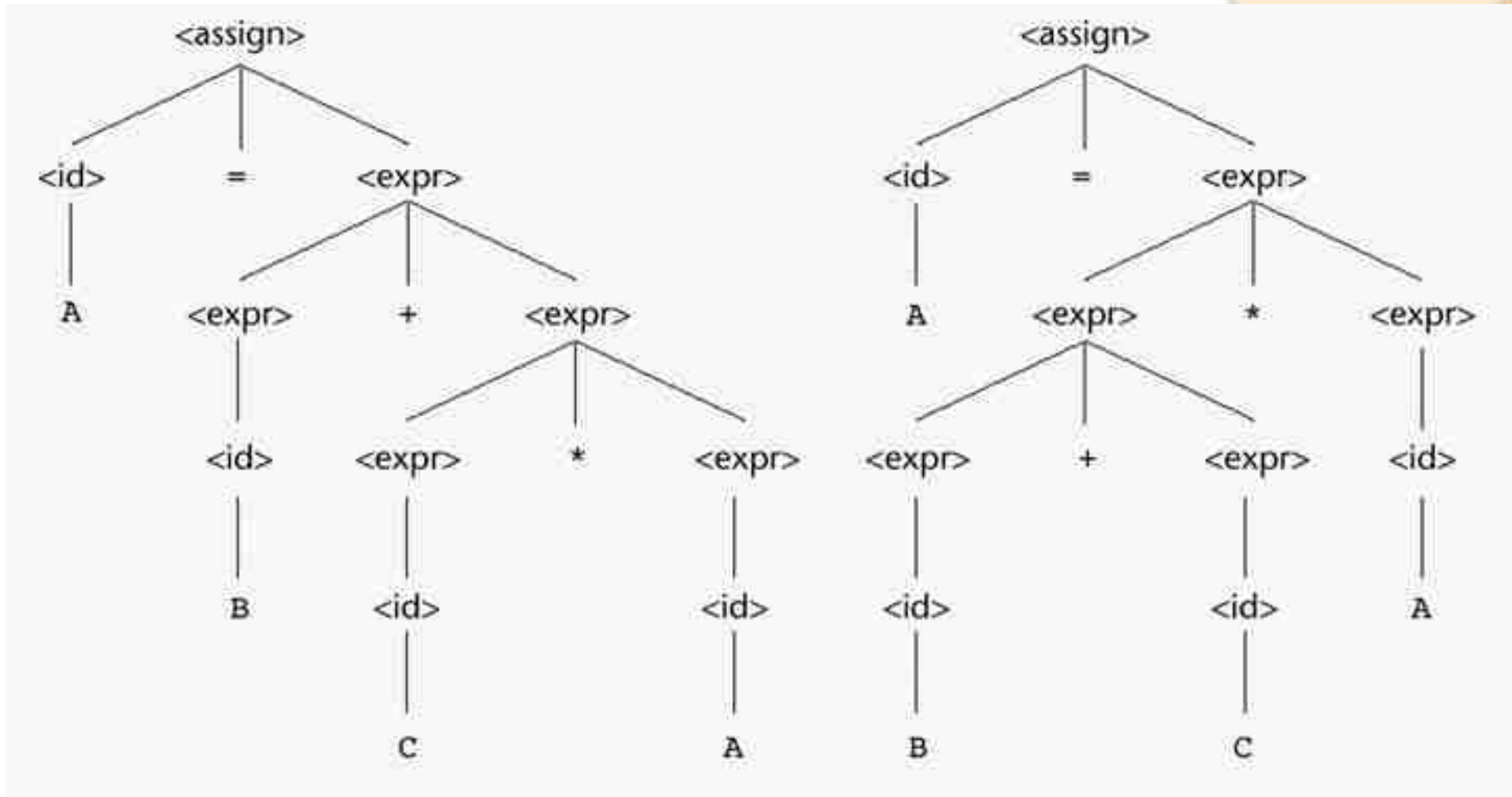
Ambiguidade (1/2)

- Uma gramática que gera uma sentença para o qual existem duas ou mais árvores de derivação.

$\langle \text{assign} \rangle \rightarrow \langle \text{id} \rangle = \langle \text{expr} \rangle$
 $\langle \text{id} \rangle \rightarrow \mathbf{A} \mid \mathbf{B} \mid \mathbf{C}$
 $\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle + \langle \text{expr} \rangle$
 $\mid \langle \text{expr} \rangle * \langle \text{expr} \rangle$
 $\mid (\langle \text{expr} \rangle)$
 $\mid \langle \text{id} \rangle$

Ambiguidade (2/2)

A := B + C * A



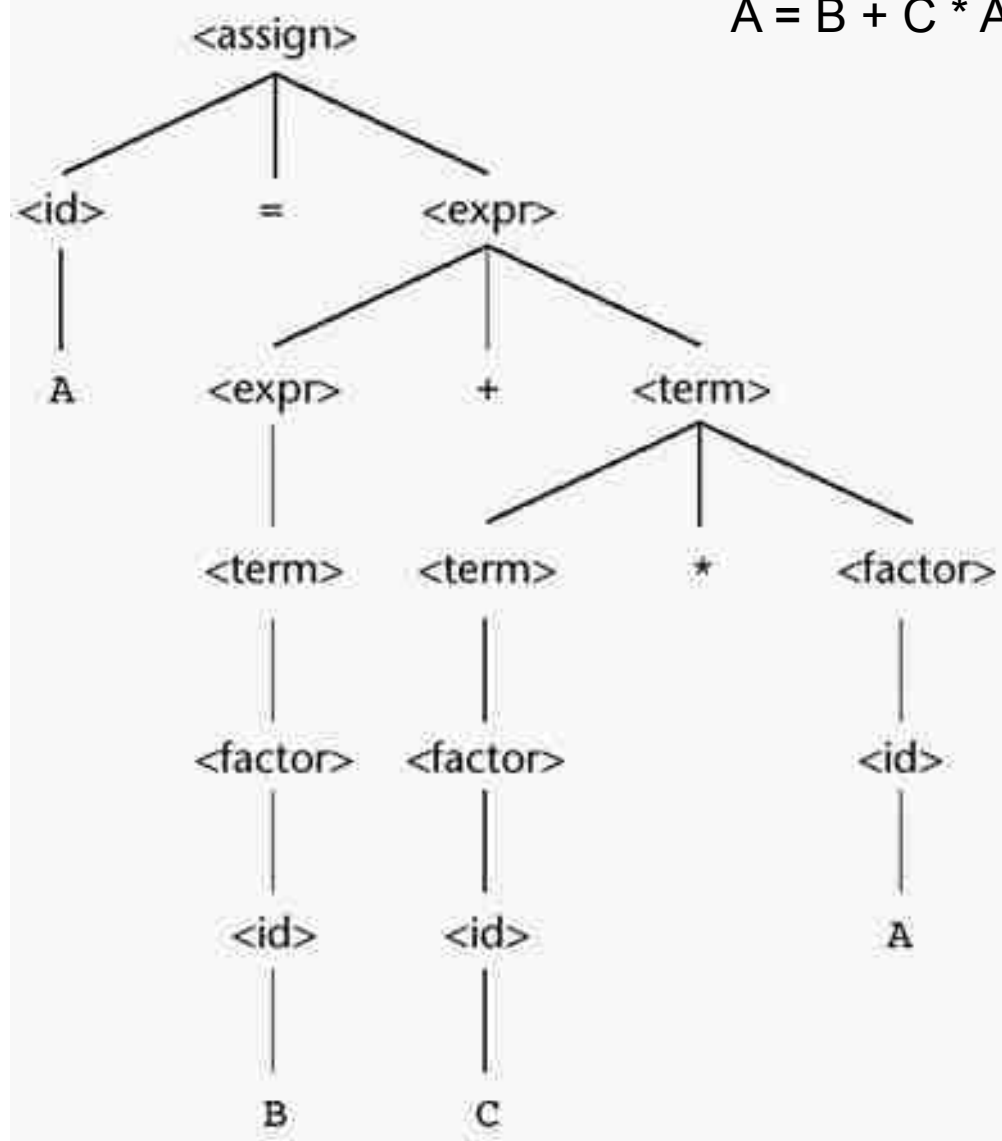
Precedência de Operadores (1/3)

- Uma árvore de derivação pode ser usada para indicar precedência de operadores.

< assign > → < id > = < expr >
< id > → **A** | **B** | **C**
< expr > → < expr > + < term >
 | < term >
< term > → < term > * < factor >
 | < factor >
< factor > → (< expr >)
 | < id >

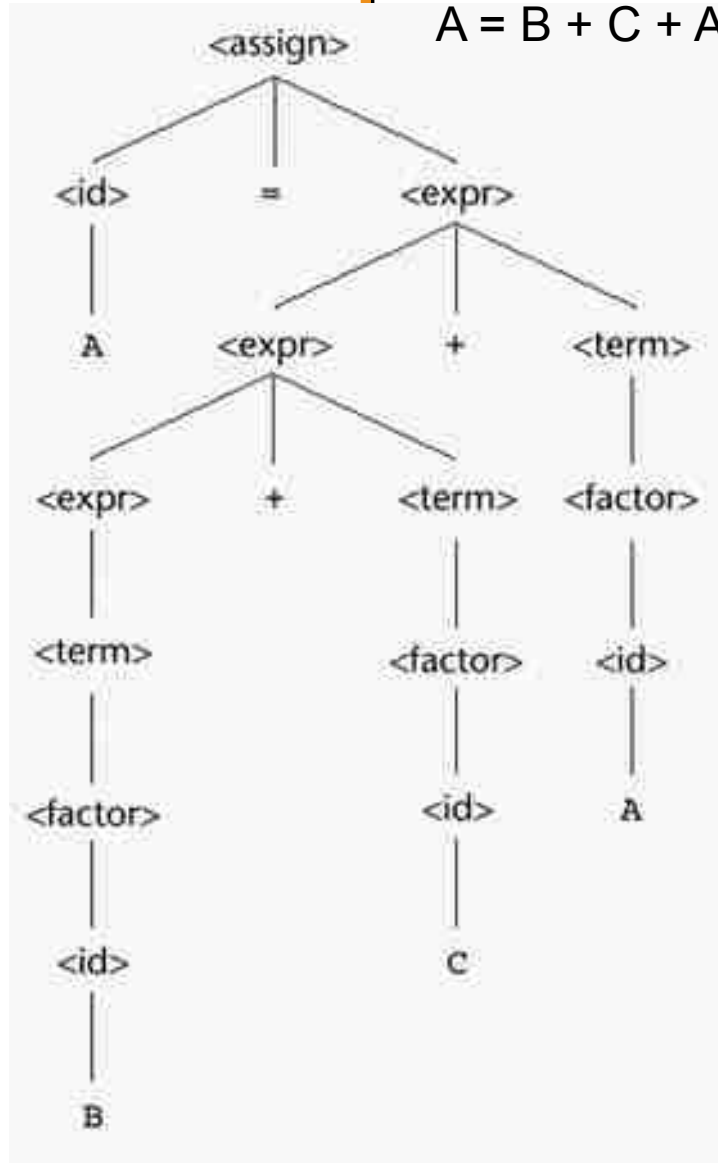
Precedência de Operadores (2/3)

$$A = B + C * A$$



Precedência de Operadores (3/3)

A = B + C + A



BNF Estendida (1/3)

- Três extensões foram propostas:
 - Não melhoram o poder descritivo, mas aumenta a legibilidade e a redigibilidade.
- Parte opcional:
 - `< selection > → if (< expression >) < statment > [else < statement >] ;`
- Repetição:
 - `< ident_list > → < identifier > { , < identifier > }`
- Múltipla escolha:
 - `< for_stmt > → for < var > := < expr > (to | downto) < expr > do < stmt >`

BNF Estendida (2/3)

- BNF:

$$\begin{aligned} \langle \text{expr} \rangle &\rightarrow \langle \text{expr} \rangle + \langle \text{term} \rangle \\ &| \langle \text{expr} \rangle - \langle \text{term} \rangle \\ &| \langle \text{term} \rangle \\ \langle \text{term} \rangle &\rightarrow \langle \text{term} \rangle * \langle \text{factor} \rangle \\ &| \langle \text{term} \rangle / \langle \text{factor} \rangle \\ &| \langle \text{factor} \rangle \end{aligned}$$

- EBNF:

$$\begin{aligned} \langle \text{expr} \rangle &\rightarrow \langle \text{term} \rangle \{ (+ | -) \langle \text{term} \rangle \} \\ \langle \text{term} \rangle &\rightarrow \langle \text{factor} \rangle \{ (* | /) \langle \text{factor} \rangle \} \end{aligned}$$

BNF Estendida (3/3)

- Algumas versões de EBNF permitem que se coloque um número sobrescrito indicando a quantidade de repetições;
- Ou que se coloque um sinal de mais (+) para indicar uma ou mais vezes.

< program > → **begin** < stmt > { < stmt > } **end**

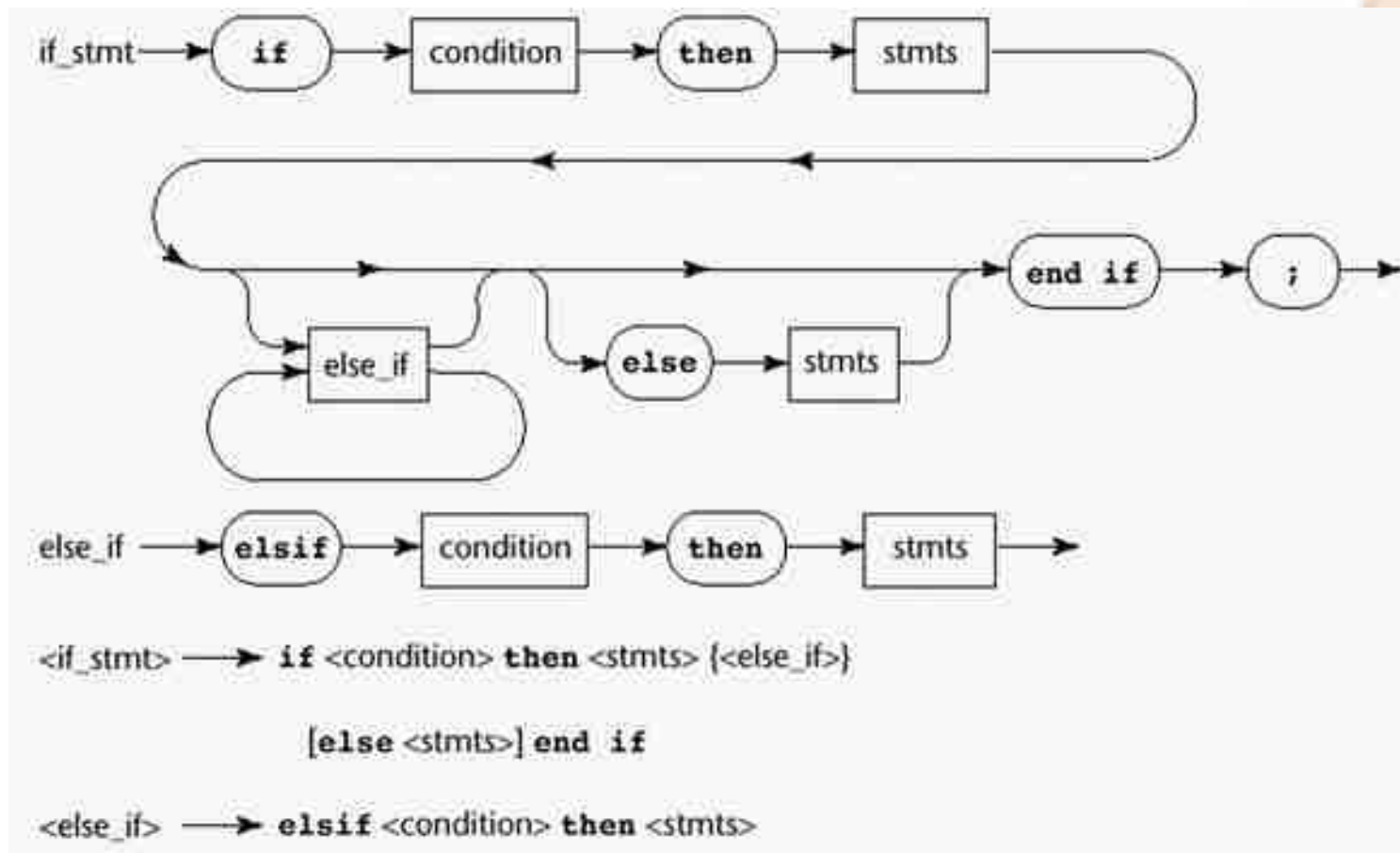
< program > → **begin** { < stmt > }⁺ **end**

Grafos de Sintaxe (1/2)

- Um grafo é uma coleção de nodos, alguns conectados através de linhas chamadas arestas.
- Um grafo direcionado possui arestas direcionais.
- As informações de regras BNF podem ser expressas em termos de grafos direcionais:
 - Chamados de grafos sintáticos.
- Usam diferentes tipos de nodos para representar os terminais e os não-terminais.
- Usar gráficos melhora a legibilidade.

Grafos de Sintaxe (2/2)

- Grafo sintático e descrição EBNF da instrução **if** na linguagem **ADA**.



Análise Sintática Descendente Recursiva (1/5)

- *Recursive Descent Parsing*.
- As gramáticas servem como base para o analisador sintático, ou *parser*, de um compilador.
- *Parsing* é o processo de construir uma árvore de derivação para uma dada cadeia de caracteres de entrada.
- A idéia é que existe um subprograma para cada não-terminal.
- Quando é dado um texto de entrada, é produzido uma árvore de derivação que inicia neste não-terminal e cujas folhas casam com esta.

Análise Sintática Descendente Recursiva (2/5)

- Este subprograma é um *parser* para esta linguagem que pode ser gerado por este não-terminal.
- As linguagens têm unidades sintáticas aninhadas, por isso os subprogramas são recursivos.
- Considere a seguinte descrição EBNF:

$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \{ (+ | -) \langle \text{term} \rangle \}$
 $\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \{ (* | /) \langle \text{factor} \rangle \}$
 $\langle \text{factor} \rangle \rightarrow \langle \text{id} \rangle \mid (\langle \text{expr} \rangle)$

Análise Sintática Descendente Recursiva (3/5)

- O subprograma descendente recursivo para < expr >:

```
void expr ()
{
    term(); /*analise o primeiro termo*/
    while (prox_token == sinal_mais || prox_token ==
sinal_menos)
    {
        lexical(); /*peque o próximo token da entrada*/
        term(); /*analise o próximo termo*/
    }
}
```

Análise Sintática Descendente Recursiva (4/5)

- A parte da linguagem que `<expr>` analisa consiste de um ou mais `<term>` separados pelos operadores de adição ou subtração.
- Esta é a linguagem gerada por `<expr>`.
- Ela é simples e não detecta erros sintáticos.
- O processo geral de escrita de subprograma para um dado não-terminal é feito da seguinte forma:
 - Para cada terminal, este é comparado com o próximo token. Se não casarem houve um erro de sintaxe. Se casarem, o analisador léxico é chamado para pegar o próximo token.

Análise Sintática Descendente Recursiva (5/5)

```
void factor()
{
  if (prox_token_ == id_code)
  {
    lexical();
    return;
  }
  else if (prox_token == cod_parent_esq)
  {
    lexical();
    expr();
    if (prox_token == cod_parent_dir)
    {
      lexical();
      return();
    }
    else error(); /*à espera do parêntese direito*/
  }
  else error(); /*não era nem um id, nem um parêntese esquerdo*/
}
```

Gramática com Atributos (1/6)

- Serve para descrever tanto a sintaxe quanto a semântica estática.
- É um dispositivo usado para descrever melhor a estrutura de uma linguagem de programação do que o permitido pelas BNF.
- Existem algumas características de linguagens que são difíceis de descrever usando a notação BNF:
 - Compatibilidade de tipos:
 - A gramática ficaria muito grande.
 - Variáveis que devem ser declaradas antes de serem utilizadas:
 - Impossível.
 - O **end** de um subprograma em **ADA** deve casar com o nome do subprograma.

Gramática com Atributos (2/6)

- Estes problemas exemplificam a categoria de regras de linguagem chamada de regras de semântica estática.
- Tem pouca relação com significado do programa:
 - Sintaxe ao invés de semântica.
- A análise requerida para checar estas especificações podem ser feitas em tempo de compilação.
- Gramática com atributos são gramáticas para as quais foram adicionadas atributos.
- Atributos intrínsecos possuem valores determinados fora da árvore de derivação. O tipo de uma variável pode vir de uma tabela de símbolos.

Gramática com Atributos (3/6)

- O nome no **end** de um procedimento **ADA** deve combinar com o nome do procedimento.

- Regra sintática:

$\langle \text{proc_def} \rangle \rightarrow \text{procedure } \langle \text{proc_name} \rangle[1]$
 $\langle \text{proc_body} \rangle \text{end } \langle \text{proc_name} \rangle[2];$

- Regra semântica:

$\langle \text{proc_name} \rangle[1].\text{string} = \langle \text{proc_name} \rangle[2].\text{string}$

- Quando existe mais de uma ocorrência de um não-terminal, este é indexado com colchetes para diferenciá-los.

Gramática com Atributos (4/6)

1. Sintaxe: $\langle \text{assign} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expr} \rangle$

Semântica: $\langle \text{expr} \rangle.\text{expected_type} \leftarrow \langle \text{var} \rangle.\text{actual_type}$

2. Sintaxe: $\langle \text{expr} \rangle \rightarrow \langle \text{var} \rangle[2] + \langle \text{var} \rangle[3]$

Semântica: $\langle \text{expr} \rangle.\text{actual_type} \leftarrow$ if ($\langle \text{var} \rangle[2].\text{actual_type} = \text{int}$) &
($\langle \text{var} \rangle[3].\text{actual_type} = \text{int}$)
then int
else real

Predicado: $\langle \text{expr} \rangle.\text{actual_type} = \langle \text{expr} \rangle.\text{expected_type}$

3. Sintaxe: $\langle \text{expr} \rangle \rightarrow \langle \text{var} \rangle$

Semântica: $\langle \text{expr} \rangle.\text{actual_type} \leftarrow \langle \text{var} \rangle.\text{actual_type}$

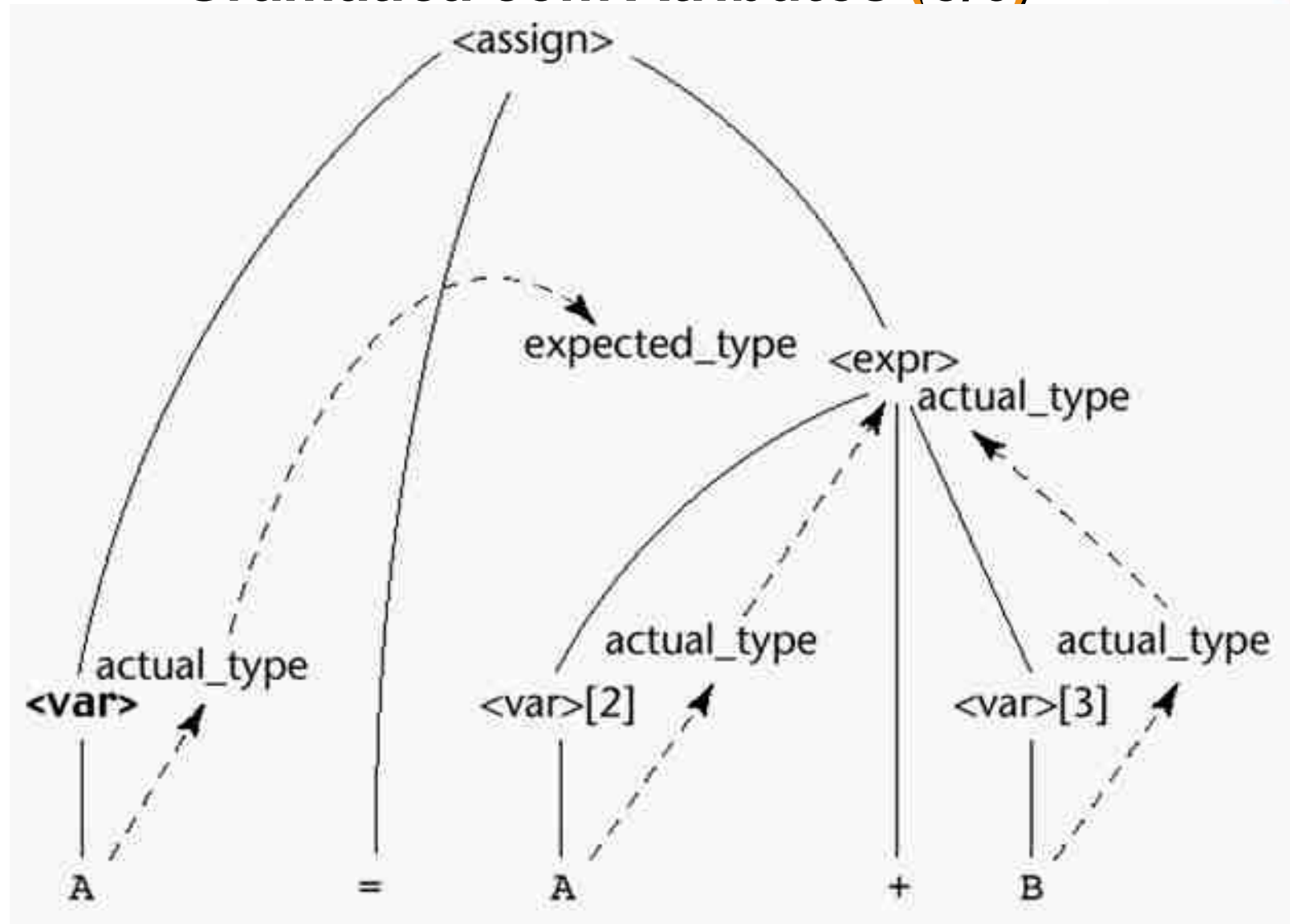
Predicado: $\langle \text{expr} \rangle.\text{actual_type} = \langle \text{expr} \rangle.\text{expected_type}$

4. Sintaxe: $\langle \text{var} \rangle \rightarrow \mathbf{A} \mid \mathbf{B} \mid \mathbf{C}$

Semântica: $\langle \text{var} \rangle.\text{actual_type} \leftarrow \text{look-up}(\langle \text{var} \rangle.\text{string})$

Obs.: look-up pesquisa determinado nome de variável na tabela de símbolos e retorna o tipo desta.

Gramática com Atributos (5/6)



Gramática com Atributos (6/6)

1. $\langle \text{var} \rangle.\text{actual_type} \leftarrow \text{look-up}(A)$ (regra 4)
2. $\langle \text{expr} \rangle.\text{expected_type} \leftarrow \langle \text{var} \rangle.\text{actual_type}$ (regra 1)
3. $\langle \text{var} \rangle[2].\text{actual_type} \leftarrow \text{look-up}(A)$ (regra 4)
4. $\langle \text{var} \rangle[3].\text{actual_type} \leftarrow \text{look-up}(B)$ (regra 4)
5. $\langle \text{expr} \rangle.\text{actual_type} \leftarrow \text{int ou real}$ (regra 2)
6. $\langle \text{expr} \rangle.\text{expected_type} = \langle \text{expr} \rangle.\text{actual_type}$ é TRUE ou FALSE (regra 2)

Semântica Dinâmica (1/2)

- Descrever a semântica dinâmica, ou significado, de expressões, instruções e unidades, não é uma tarefa fácil.
- Não existe uma notação universalmente aceita para descrevê-la.
- Por que descrever a semântica dinâmica:
 - Programadores precisam saber o que os elementos da linguagem fazem.
 - Desenvolvedores de compiladores baseiam a semântica em descrições da linguagem natural.

Semântica Dinâmica (2/2)

- É objeto de pesquisa encontrar um formalismo semântico que possa ser usado tanto por programadores quanto por desenvolvedores de compiladores.
- A prova de corretude de programas confiam em alguma descrição formal da semântica da linguagem:
 - Semântica operacional.
 - Semântica axiomática.
 - Semântica denotacional.

Semântica Operacional (1/5)

- A ideia por trás da semântica operacional é a descrição do significado de um programa pela execução de suas instruções em uma máquina real ou simulada.
- As alterações que ocorrem no estado da máquina quando esta executa uma instrução define o significado desta instrução.
- Exemplo:
 - Instrução em linguagem de máquina.

Semântica Operacional (2/5)

- Descrever a semântica operacional de uma linguagem de alto nível requer a construção de um computador real ou simulado.
- A semântica para uma linguagem de alto nível pode ser descrita usando um interpretador puro para esta linguagem.
- Dois problemas:
 - Complexidade do hardware pode fazer com que as ações sejam difíceis de entender.
 - Só serviriam para um computador igualmente configurado.

Semântica Operacional (3/5)

- Estes problemas são evitados pela troca do computador real por um computador virtual, implementado via simulação por software.
- O conjunto de instruções poderiam ser projetado de tal forma que a semântica de cada instrução fosse fácil de entender.
- Requer a construção de dois componentes:
 - Tradutor para converter as instruções da linguagem de alto nível na linguagem intermediária.
 - A máquina virtual para esta linguagem intermediária.

Semântica Operacional (4/5)

- Este conceito é frequentemente usado em manuais de linguagens.
- O leitor das descrições de semântica operacional é o computador virtual e é assumido ser capaz de executar corretamente as instruções e reconhecer os efeitos da execução.

Semântica Operacional (5/5)

Instrução em C

```
for (expr1; expr2; expr3){  
...  
}
```

Semântica Operacional

```
expr1;  
loop: if expr2 == 0 goto out;  
...  
expr3;  
goto loop;  
out: .....
```

Semântica Axiomática (1/7)

- Semântica axiomática foi definida em conjunto com o desenvolvimento de um método para provar a corretude de programas.
- Tais provas de corretude, mostram que um programa executa a computação descrita por sua especificação.
- Cada instrução é precedida e seguida por uma expressão lógica que especifica restrições sobre variáveis de programa.
- Estas expressões, ao invés de representar o estado de uma máquina abstrata, são usadas para representar o significado de instruções.

Semântica Axiomática (2/7)

- Notação usada para descrever restrições é o cálculo de predicados.
- Semântica axiomática é baseada em lógica matemática.
- As expressões lógicas são chamadas predicados ou assertivas.
- Uma assertiva imediatamente precedente de uma instrução descreve as restrições sobre as variáveis naquele ponto:
 - Chamadas de pré-condições.

Semântica Axiomática (3/7)

- Uma assertiva imediatamente seguinte a uma instrução descreve uma nova restrição sobre estas, e outras, variáveis após a execução:
 - Chamadas de pós-condições.
- Toda a instrução em um programa tem que ter as duas condições.
- As pré-condições são computadas a partir das pós-condições.

Semântica Axiomática (4/7)

- Exemplo:
 - $\text{sum} = 2 * x + 1 \{ \text{sum} > 1 \}$
- A pré-condição mais fraca é a pré-condição menos restritiva que garantirá a validade da pós-condição.
- $\{x > 10\}$ e $\{x > 100\}$ são exemplos de pré-condições válidas.
- Entretanto:
 - $\{x > 0\}$ é a pré-condição mais fraca.

Semântica Axiomática (5/7)

- Para algumas instruções de programas, a computação da pré-condição mais fraca é simples e pode ser especificada por um axioma.
- Na maioria dos casos, entretanto, só via regra de inferência.
- Um axioma é uma instrução lógica que é assumida ser verdade.
- Uma regra de inferência é um método de inferir a verdade de uma assertiva baseada em outras assertivas verdadeiras.

Semântica Axiomática (6/7)

- Instrução de Atribuição:
 - Seja $x := e$ uma instrução de atribuição geral e seja Q a pós-condição.
 - A pré-condição é definida pelo seguinte axioma:
 - $P = Qx \rightarrow e$, o que significa que P é computado como Q com todas as instâncias de x trocadas por e .
 - Exemplo:
 - $a := b / 2 - 1 \{a > 10\}$
 - Calculando a pré-condição mais fraca:
 - $b / 2 - 1 > 10$
 - $b > 22$

Semântica Axiomática (7/7)

- Instrução de Atribuição:
 - A notação usual para especificar a semântica axiomática é:
 - $\{P\} S \{Q\}$
 - No caso de uma instrução de atribuição:
 - $\{Qx \rightarrow e\} S \{Q\}$

Semântica Denotacional (1/6)

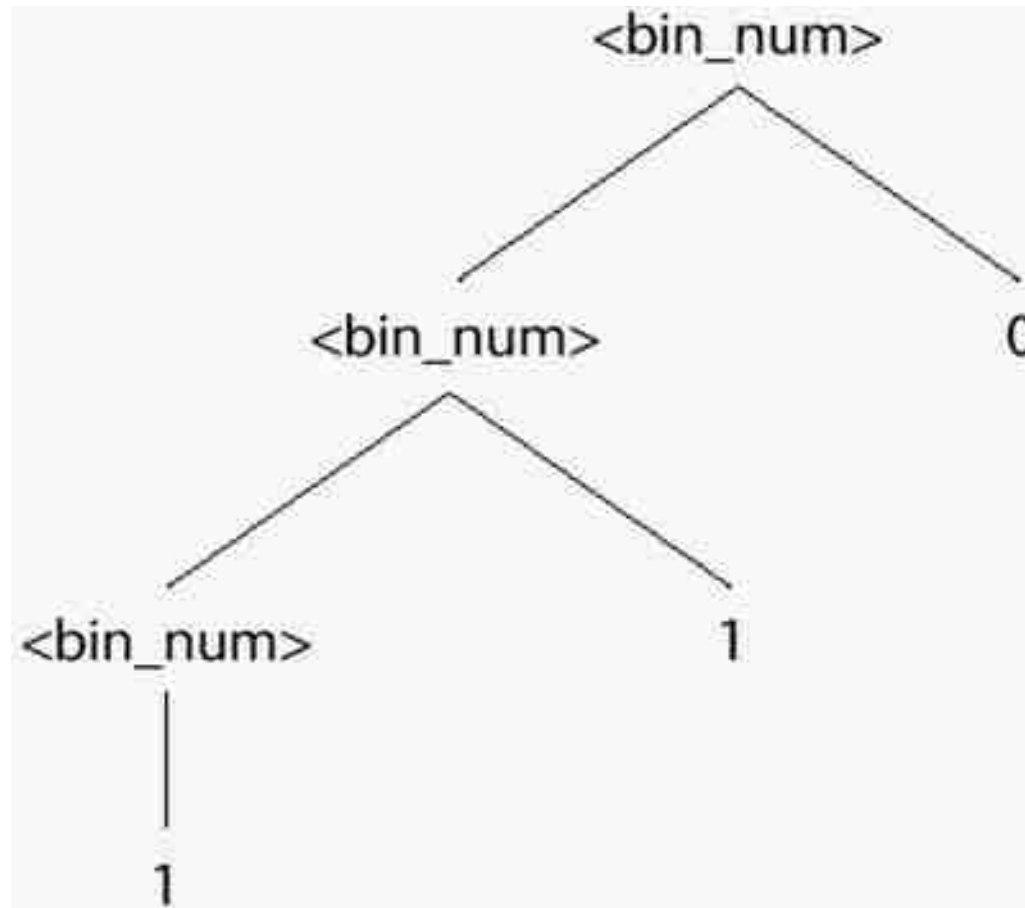
- Semântica denotacional é o método mais rigoroso para descrever o significado de programas.
- É baseado na teoria das funções recursivas.
- O conceito fundamental é a definição para cada entidade da linguagem tanto um objeto matemático quanto uma função que mapeia instâncias desta entidade em instâncias do objeto matemático.
- Os objetos são tão rigorosamente definidos que eles representam o significado exato das entidades correspondentes.

Semântica Denotacional (2/6)

- A dificuldade deste método está na criação dos objetos e na função de mapeamento.
- O método é chamado denotacional porque os objetos denotam o significado de suas correspondentes entidades sintáticas.
- Exemplo:
 - A sintaxe de números binários podem ser descritas pelas seguintes regras gramaticais:
 - $\langle \text{bin_num} \rangle \rightarrow 0 \mid 1 \mid \langle \text{bin_num} \rangle 0 \mid \langle \text{bin_num} \rangle 1$

Semântica Denotacional (3/6)

- Uma parse tree para o número binário 110 é:



Semântica Denotacional (4/6)

- Para descrever o significado de números binários usando a semântica denotacional e as regras gramaticais acima, o significado real é associado com cada regra que tem um simples símbolo terminal no lado direito. Objetos serão simples números decimais.
- Faça o domínio de valores semânticos dos objetos ser \mathbb{N} . Estes serão os objetos que queremos associar com números binários.
- A função semântica, chamada M_{bin} mapeia a sintaxe abstrata, descrito pelas regras gramaticas, nos objetos de \mathbb{N} .

Semântica Denotacional (5/6)

- A função M_{bin} foi definida como:
 - $M_{bin}(0) = 0$
 - $M_{bin}(1) = 1$
 - $M_{bin}(\langle bin_num \rangle 0) = 2 * M_{bin}(\langle bin_num \rangle)$
 - $M_{bin}(\langle bin_num \rangle 1) = 2 * M_{bin}(\langle bin_num \rangle) + 1$
- O significado, ou objetos denotados (que no caso são números decimais), podem ser colocados junto aos nodos da *parse tree*.
- Entidades sintáticas abstratas são mapeadas em objetos matemáticos com significado concreto.

Semântica Denotacional (6/6)

