INE5331
Construção de Compiladores

AULA 5: Análise Semântica

Ricardo Azambuja Silveira
INE-CTC-UFSC
E-Mail: silveira@inf.ufsc.br
URL: www.inf.ufsc.br/~silveira

Introdução

- há um nível de correção que vai além da gramática
  - Jogo dos 6 erros:

```
foo(a,b,c,d) {
   int a, b, c, d;
     ...
}
bar() {
   int f[3],g[0], h, i, j, k;
   char *p;
   foo(h,i,“ab”,j, k);
   k = f * i + j;
   h = g[17];
   printf(“<%s,%s>.\n”,p,q);
   p = 10;
}
```

Introdução

- Jogo dos 6 erros:

```
foo(a,b,c,d) {
   int a, b, c, d;
      ...
}
bar() {
   int f[3],g[0], h, i, j, k;
   char *p;
   foo(h,i,"ab",j, k);
   k = f * i + j;
   h = g[17];
   printf("<%s,%s>.\n",p,q);
   p = 10;
}
```

O que há de errado?

(vamos contar...)

- número de argumentos de foo()
- declarou g[0], usou g[17]
- "ab" não é int
- dimensão errada no uso de f
- variável não declarada q
- 10 não é uma string

Tudo isso está além da sintaxe

Introdução

- Antes de gerar código, o compilador pode responder as questões do tipo:
  - “x” é escalar, array, ou função? “x” foi declarado?
  - Há nomes não declarados? Há nomes Declarados mas não usados?
  - Qual declaração de “x” é usada por uma referência a “x”?
  - A expressão “x * y + z” é corretamente tipada?
  - Em “a[i,j,k]”, a foi declarado com três dimensões?
  - Onde uma variável “z” pode ficar armazenada? (registrador, local, global, heap, estática)
  - Em “f ← 15”, como representar a constante15?
  - Quantos argumentos uma função “foo()” recebe? E “printf()” ?
  - “x” é definido antes de ser usado?

Introdução

- Estas questões estão além do escôpo de uma gramática livre de contexto
- Um método frequentemente utilizado é a Gramática de Atributos como mecanismo semiformal de especificação semântica e algoritmos de Semântica Dirigida por Sintaxe para computar as ações semânticas
- Um atributo é qualquer propriedade de uma construção da linguagem e podem ser fixados antes do processo de compilação (na implementação da linguagem) ou computados durante a compilação ou durante a execução do programa

Gramática de atributos

- Uma *gramática de atributos* para uma linguagem de programação é a gramática livre de contexto que representa a linguagem, com as seguintes adições:
- Para cada símbolo gramatical x há um conjunto A(x) de atributos
- Cada regra semântica tem um conjunto de funções que definem certos atributos dos símbolos não-terminais em uma regra
- Cada regra semântica tem um conjunto (possivelmente vazio) de predicados para checar a consistência dos atributos

Gramática de atributos

- Desse modo, dada uma coleção de atributos $a_1, a_2, \ldots, a_k$, o princípio da semântica dirigida pela sintaxe implica que, para cada regra gramatical, $X_0, X_1, X_2 \ldots X_n$, os valores dos atributos $X_i.a_j$ de cada símbolo X são relacionados a valores de atributos de outros símbolos na regra, da forma:

  $X_i a_j = f_{ij}(X_0.a_1, \ldots, X_0.a_k, X_1.a_1, \ldots., X_1.a_k, \ldots X_n.a_1, \ldots, X_n.a_k)$

Gramática de atributos

- Seja a regra X0 -> X1 ... Xn
  - Funções na forma $S(X_0) = f(A(X_1), ... A(X_n))$ definem, *atributos sintetizados*
  - Funções na forma $I(X_j) = f(A(X_0), ... , A(X_n))$, para i <= j <= n, definem *atributos herdados*
  - Inicialmente, existem *atributos intrínsecos* nas folhas, geralmente extraídos da tabela de símbolos
- *Exemplo:*
  - seja as expressões na forma id + id e as regras:
  - - id's podem ser tipo int ou real
  - - tipos dos dois id's devem ser os mesmos
  - - tipos de expressão devem ser o mesmo que o tipo esperado
  - *BNF:*
    - <expr> -> <var> + <var>
    - <var> -> id
  - *Attributos:*
    - *tipo_efetivo* – sintetizado para <var> e <expr>
    - *tipo_esperado* – herdado para <expr>

Gramática de atributos

Regra sintática: <expr> -> <var>[1] + <var>[2]

Regra semantica: <expr>.actual_type ← <var>[1].actual_type

Predicado:

<var>[1].actual_type = <var>[2].actual_type

<expr>.expected_type = <expr>.actual_type

Regra sintática: <var> -> id

Regra semantica: <var>.actual_type ← lookup (id, <var>)

Gramática de atributos

- 1. <expr>.expected_type ← inherited from parent

- 2. <var>[1].actual_type ← lookup (A, <var>[1])
- <var>[2].actual_type ← lookup (B, <var>[2])
- <var>[1].actual_type =? <var>[2].actual_type

- 3. <expr>.actual_type ← <var>[1].actual_type
- <expr>.actual_type =? <expr>.expected_type

A função lookup busca os atributos do token na tabela de símbolos

Gramática de atributos

Como os valores dos atributos são computados?

- 1. Se todos os atributos foram herdados, a árvore é decorada em ordem top-down.
- 2. Se todos os atributos foram sintetizados, a árvore é decorada em ordem bottom-up.
- 3. Em muitos casos, os dois tipos de atributos são usados e uma combinação de top-down e bottom-up é usada.

Grafo de dependências

- Se um atributo b em um nodo da árvore de derivação depende do atributo c, então a regra semântica para b naquele nodo deve ser avaliada depois da regra semântica que define c. As interdependências entre os atributos herdados e sintetizados nos nodos da árvore de derivação pode ser apresentados por um grafo direcionado denominado grafo de dependências
- O grafo tem um nodo para cada atributo e um arco do nodo b para o nodo c, se o atributo b depende do atributo c. O seguinte algoritmo pode ser seguido para a construção do grafo de dependências

Grafo de dependências

Para cada nodo n da árvore de derivação faça

Para cada atributo a do símbolo da gramática em n faça

construa um nodo no grafo de dependências para a;

Para cada nodo n da árvore de derivação faça

Para cada regra semântica b = $f(c_1, c_2,..., c_k)$ associada a uma produção usada

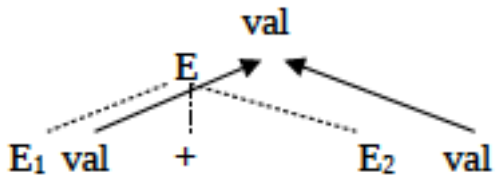
em n faça

Para i = 1 até k faça

construa um arco do nodo c_i para o nodo b.

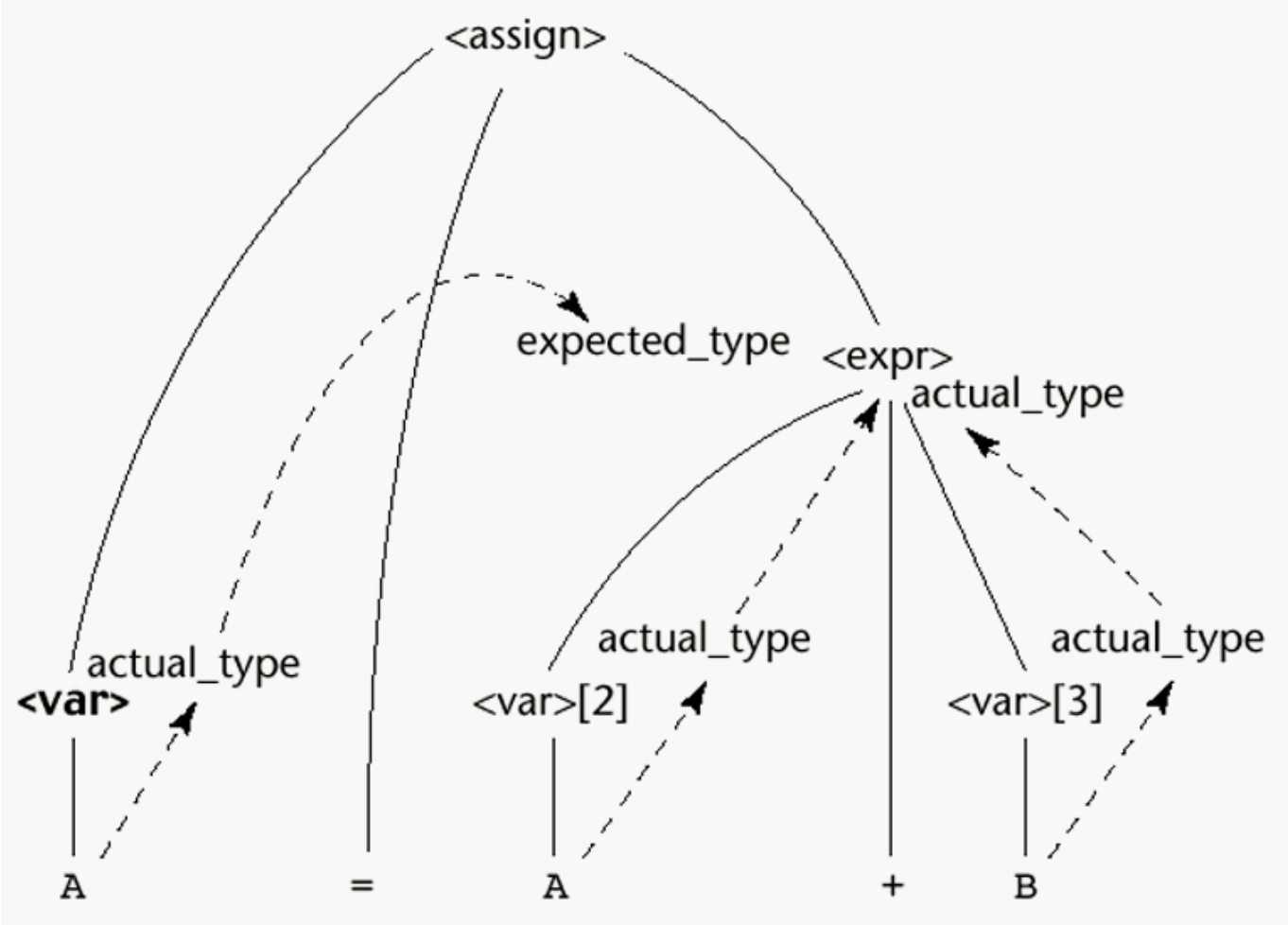
Grafo de dependências

- Por exemplo: considerando que a regra semântica A.a = f(X.x,Y.y) esteja associada à produção A 🎭 XY, de forma que o atributo sintetizado A.a dependa dos atributos X.x e Y.y. Se esta produção é usada em uma árvore de derivação, haverá três nodos A.a, X.x e Y.y no grafo de dependências com um arco de X.x para A.a e de Y.y para A.a.
- Se a produção A 🎭 XY tivesse uma regra semântica X.i = g(A.a,Y.y) associada, haveria um arco de A.a para X.i e de Y.y para X.i.
- A seguir é apresentado um exemplo de grafo de dependência para a produção E 🎭 E1 + E2, que tem a seguinte regra semântica associada: E.val = E1.val + E2.val.
- O grafo de dependência resultante tem três nodos, representando os atributos sintetizados E.val, E1.val e E2.val. O arco de E.val para E1.val mostra que E.val depende de E1.val.



Gramática de atributos

Fluxo de computação dos atributos para a expressão A = A + B



TRADUÇÃO DIRIGIDA POR SINTAXE

- Cada símbolo da gramática tem, portanto, um conjunto de atributos associado, particionados em dois: atributos sintetizados e herdados.
- Um atributo pode representar uma palavra, um número, um tipo ou uma posição na memória
- O valor de um atributo é definido por uma regra semântica associada à produção:
  - um atributo sintetizado é computado a partir dos atributos dos filhos daquele nodo.
  - um atributo herdado é computado a partir dos atributos dos irmãos ou dos pais daquele nodo.

TRADUÇÃO DIRIGIDA POR SINTAXE

- Regras semânticas estabelecem dependências entre atributos que são representadas pelo grafo de dependências, a partir do qual é derivada uma ordem de avaliação para as regras semânticas.
- Uma árvore de derivação que apresenta os valores dos atributos de cada nodo é denominada árvore de derivação "decorada".

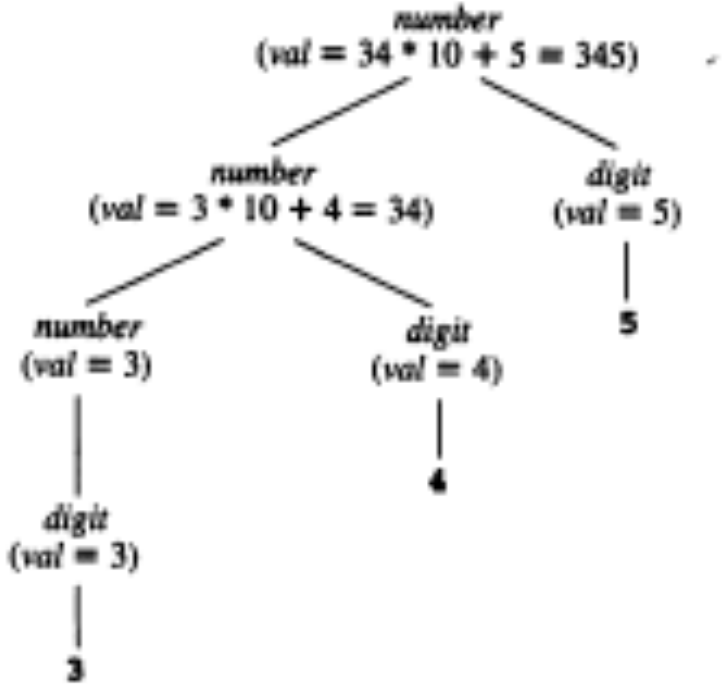
TRADUÇÃO DIRIGIDA POR SINTAXE

- Uma gramática de atributos é uma definição dirigida por sintaxe na qual as funções nas regras semânticas não têm efeitos colaterais.
- Mas estas funções poder ser usadas para diversas finalidades, inclusive a geração do código objeto
- Assume-se que terminais tenham apenas atributos sintetizados, assim como o símbolo inicial da gramática. Um exemplo de definição dirigida por sintaxe é apresentado na Tabela a seguir

Exemplo 1

$$number \rightarrow number\ digit \mid digit$$
$$digit \rightarrow \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9}$$

Grammar Rule	Semantic Rules
$number_1 \rightarrow number_2\ digit$	$number_1.val = number_2.val * 10 + digit.val$
$number \rightarrow digit$	$number.val = digit.val$
$digit \rightarrow \mathbf{0}$	$digit.val = 0$
$digit \rightarrow \mathbf{1}$	$digit.val = 1$
$digit \rightarrow \mathbf{2}$	$digit.val = 2$
$digit \rightarrow \mathbf{3}$	$digit.val = 3$
$digit \rightarrow \mathbf{4}$	$digit.val = 4$
$digit \rightarrow \mathbf{5}$	$digit.val = 5$
$digit \rightarrow \mathbf{6}$	$digit.val = 6$
$digit \rightarrow \mathbf{7}$	$digit.val = 7$
$digit \rightarrow \mathbf{8}$	$digit.val = 8$
$digit \rightarrow \mathbf{9}$	$digit.val = 9$



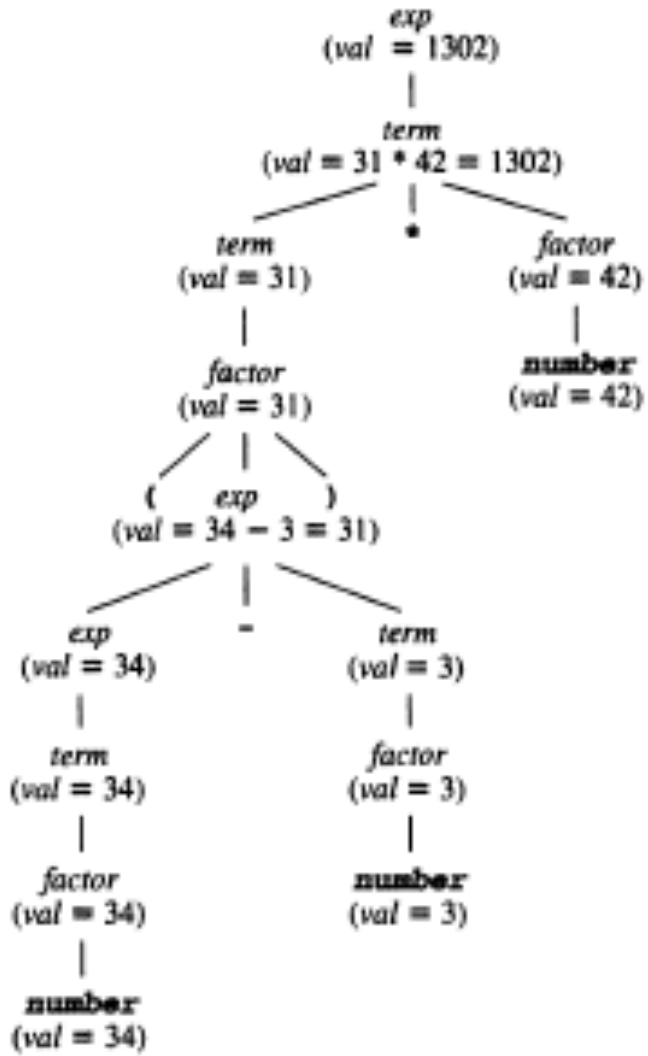
Exemplo 2

$$exp \rightarrow exp + term \mid exp - term \mid term$$
$$term \rightarrow term * factor \mid factor$$
$$factor \rightarrow (exp) \mid \mathbf{number}$$

Grammar Rule	Semantic Rules
$exp_1 \rightarrow exp_2 + term$	$exp_1.val = exp_2.val + term.val$
$exp_1 \rightarrow exp_2 - term$	$exp_1.val = exp_2.val - term.val$
$exp \rightarrow term$	$exp.val = term.val$
$term_1 \rightarrow term_2 * factor$	$term_1.val = term_2.val * factor.val$
$term \rightarrow factor$	$term.val = factor.val$
$factor \rightarrow (exp)$	$factor.val = exp.val$
$factor \rightarrow \mathbf{number}$	$factor.val = \mathbf{number}.val$

Exemplo 2

Árvore resultante para (34-3)*42

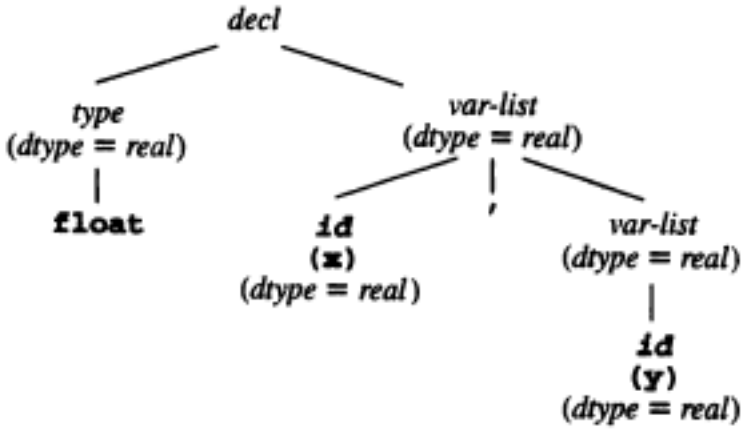


Exemplo 3

Declaração de variáveis:

decl → *type var-list*
type → **int** | **float**
var-list → **id,** *var-list* | **id**

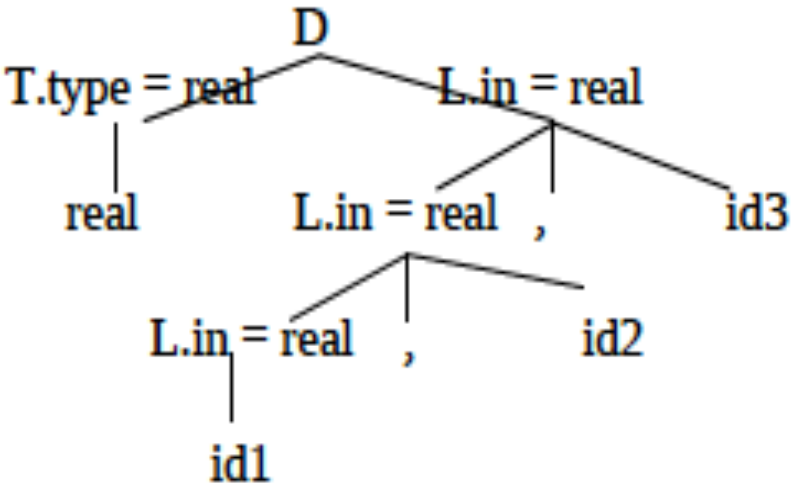
Grammar Rule	Semantic Rules
decl → *type var-list*	*var-list.dtype* = *type.dtype*
type → **int**	*type.dtype* = *integer*
type → **float**	*type.dtype* = *real*
$var\text{-}list_1$ → **id,** $var\text{-}list_2$	**id** .*dtype* = $var\text{-}list_1$.*dtype*
	$var\text{-}list_2$.*dtype* = $var\text{-}list_1$.*dtype*
var-list → **id**	**id** .*dtype* = *var-list.dtype*



Exemplo 3.1

Declaração de tipo com função addType para inclusão na tabela de símbolos

PRODUÇÃO	REGRAS SEMÂNTICAS
D → T L	L.in = T.type;
T → int	T.type = integer;
T → real	T.type = real;
L → L_1,id	L_1.in = L.in; addType(id.entry,L.in);
L → id	addType(id.entry,L.in);



Tipos de esquemas de tradução

- S-atribuídos: contém apenas atributos sintetizados (*bottom-up*)
- ações semânticas são dispostas à direita das produções

T→ int T.tipo := inteiro

- L-atribuídos: restringem o uso de atributos herdados

D → T {L.in := T.tipo} L

Avaliação Bottom-up de Esquemas de Tradução S-atribuídos

- Uma gramática de atributos em que todos os atributos são sintetizados é denominada gramática S-atribuída
- Atributos sintetizados podem ser avaliados por um analisador redutivo a medida em que a cadeia de entrada é reconhecida.
- Os valores dos atributos sintetizados são conservados, associados aos símbolos da gramática, em sua própria pilha.
- Sempre que ocorre uma redução, são computados os valores dos novos atributos sintetizados, a partir dos atributos que estão na pilha, associados aos símbolos do lado direito da produção que está sendo reduzida
- São usados campos adicionais na pilha do analisador para armazenar os valores dos atributos sintetizados

Avaliação Bottom-up de Esquemas de Tradução S-atribuídos

- Por exemplo: considerando o seguinte esquema de tradução:
- A ☹ X Y Z A.a = f(X.x,Y.y,Z.z);
- tem-se o seguinte conteúdo na pilha

PILHA	
ANÁLISE	ATRIBUTOS
Z	Z.z
Y	Y.y
X	X.x
...	...

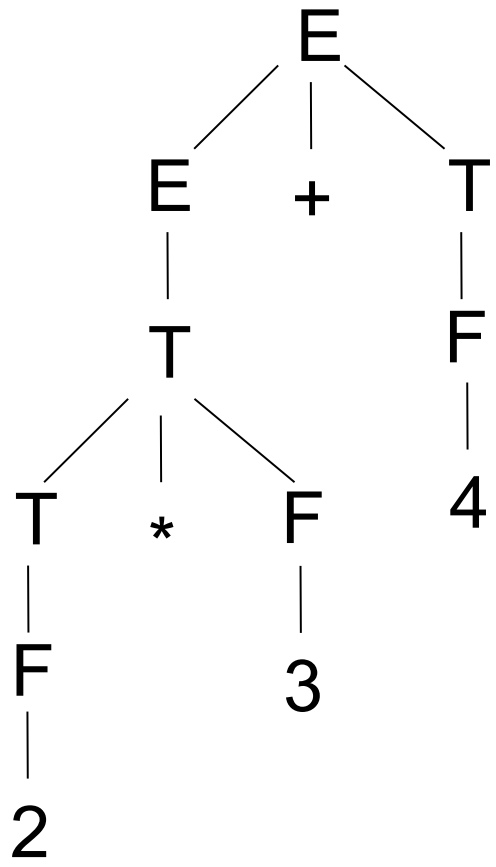
Esquemas de Tradução L-atribuídos

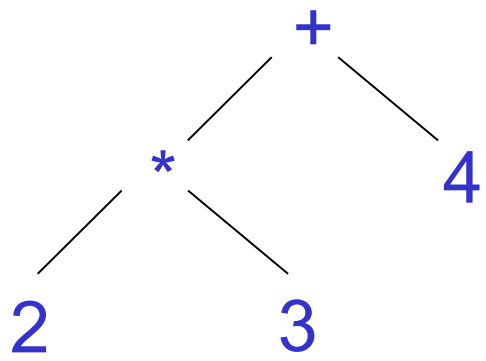
- Os atributos dos esquemas L-atribuídos devem ser avaliados sempre na ordem depth-first. Este tipo de esquema baseia-se nas gramáticas LL(1).
- Um esquema é L-atribuído se cada atributo herdado de Xj, 1<j<n, no lado direito de A ⇝ X1
- X2 ... Xn, depende somente:
  - dos atributos dos símbolos X1, X2, ..., Xj-1 à esquerda de Xj na produção
  - dos atributos herados de A.

Árvores de Sintaxe

- Forma condensada da árvore de derivação, na qual somente os operandos aparecem nas folhas, enquanto que os operadores aparecem em nodos interiores da árvore

Árvore de derivação e árvore de sintaxe para a sentença 2*3+4





Exemplo

Produções	Regras semânticas
$E \rightarrow E_1 + T$	E.ptr := geranodo("+", E_1.ptr, T.ptr)
$E \rightarrow E_1 - T$	E.ptr := geranodo("-", E_1.ptr, T.ptr)
$E \rightarrow T$	E.ptr := T.ptr
$T \rightarrow (E)$	T.ptr := E.ptr
$T \rightarrow id$	T.ptr := gerafolha("id", id.nome)
$T \rightarrow num$	T.ptr := gerafolha("num", num.lexval)

Árvore de sintaxe da expressão *x + 5 - y*