Teoria da Computação e Compiladores

Tabela de Símbolos e Análise Semântica

Prof. Dr. Fernando Kakugawa

fernando.kakugawa@animaeducacao.com.br

Universidade Anhembi Morumbi

Tabela de Símbolos

- A Tabela de Símbolos (TS) serve para guardar informações sobre os nomes declarados em um programa;
- A TS é pesquisada cada vez que um nome é encontrado no programa fonte;
- Alterações são feitas na TS sempre que um novo nome ou nova informação sobre um nome existente é obtida;

Tabela de Símbolos

- A gerência da TS de um compilador deve ser implementada de forma a permitir inserções e consultas da forma mais eficiente possível;

- E tem que permitir o crescimento dinâmico da TS;

Entrada na Tabela de Símbolos

- Cada entrada na TS é a declaração de um nome;
- O formato das entradas pode não ser uniforme, embora seja mais fácil manipular entradas uniformes;
- Cada entrada na TS pode ser implementada como um registro contendo campos nome, tipo, classe, tamanho, escopo, entre outros dados que a qualificam.

Entrada na Tabela de Símbolos

- Se o número máximo de caracteres em um nome for limitado e pequeno (até 16 caracteres), o campo nome de uma entrada da TS pode ser um arranjo de caracteres (alocação estática);

- Se o número de caracteres de um nome for grande (maior que 16), o campo nome de uma entrada da TS deve ser um apontador para uma área alocada dinamicamente;

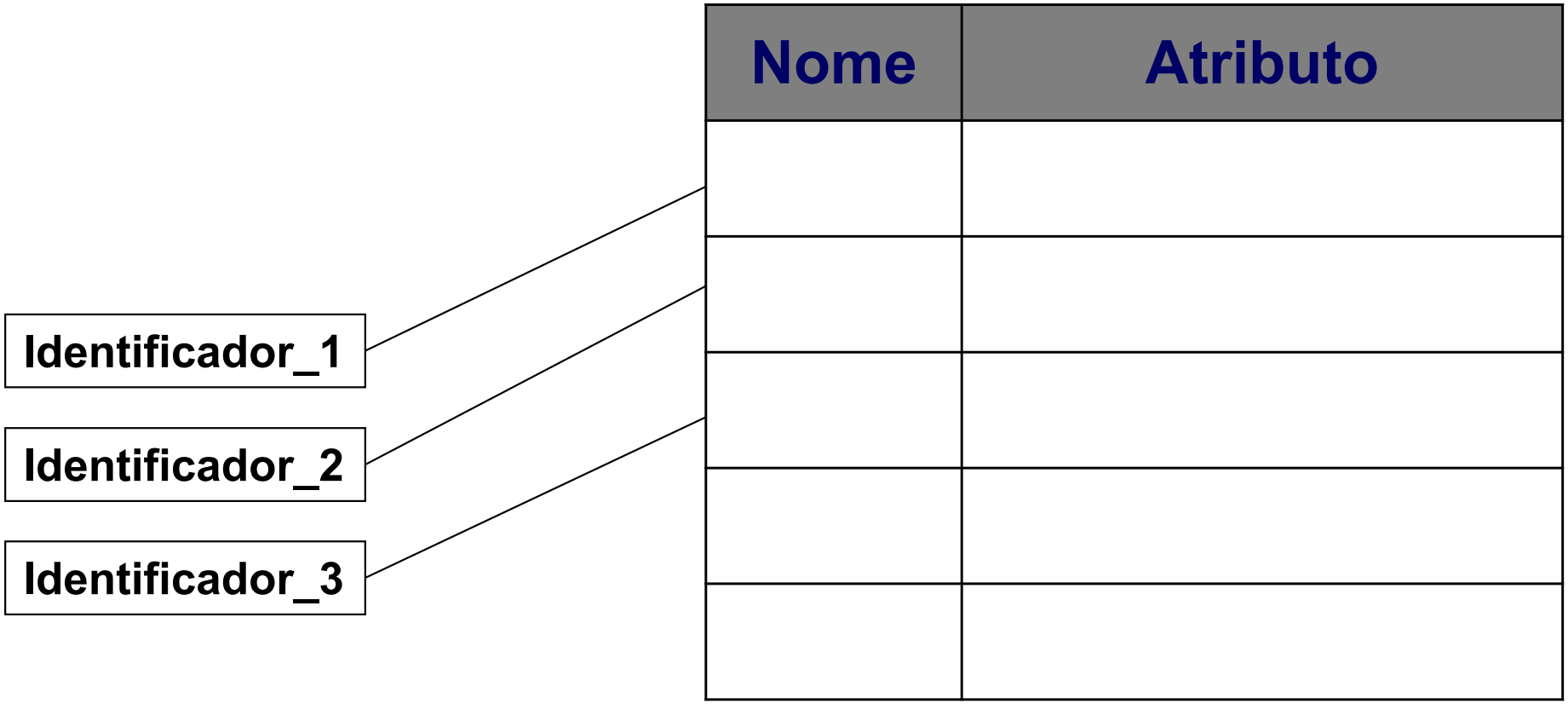
Entrada na Tabela de Símbolos

- Alocação Estática

Nome	Atributo
Identificador_01	
Identificador_02	
Identificador_03	

Entrada na Tabela de Símbolos

Alocação Dinâmica



Alocação de Posições de Memória

- Informação sobre quais endereços de memória serão associados aos nomes devem ser guardadas na TS;

- Nomes Globais a estratégia é:

  1. nome global, associa-se o endereço relativo 0 (zero);
  2. o endereço 0 + ‘espaço ocupado pelo 1º nome global’;
  3. e assim sucessivamente.

Alocação de Posições de Memória

- Nomes Locais e parâmetros de subprogramas:
  - A alocação é feita em uma pilha e o endereçamento pode ser feito de forma semelhante ao usado com os nomes globais;
- Vale lembrar que o endereço relativo 0 (zero) é, na realidade, 0 + valor do apontador de pilha na hora de execução do programa;

Implementação da Tabela de Símbolos

- A TS pode ser implementada como uma lista linear de registros onde os nomes novos são inseridos no fim e a pesquisa sempre se processa do fim para o início;

- Sempre que encontra um nome de identificador em um programa fonte, ele pode estar sendo declarado ou sendo usado pelo programador;

Implementação da Tabela de Símbolos

- Caso ele esteja sendo declarado:
  - Em uma linguagem com estrutura de blocos, devemos pesquisar a TS do fim até o ponto onde começaram a ser inseridos os símbolos do bloco mais interno;
  - Em uma linguagem sem estrutura de blocos, deve-se pesquisar a TS do fim até o início;

Implementação da Tabela de Símbolos

- Nos casos anteriores, o nome não pode estar na TS, e devendo ser inserido na sequência;
- Se o nome estiver sendo usado pelo programador, pesquisa-se a TS do fim até o início;
  - Se não for encontrado é um símbolo indefinido.

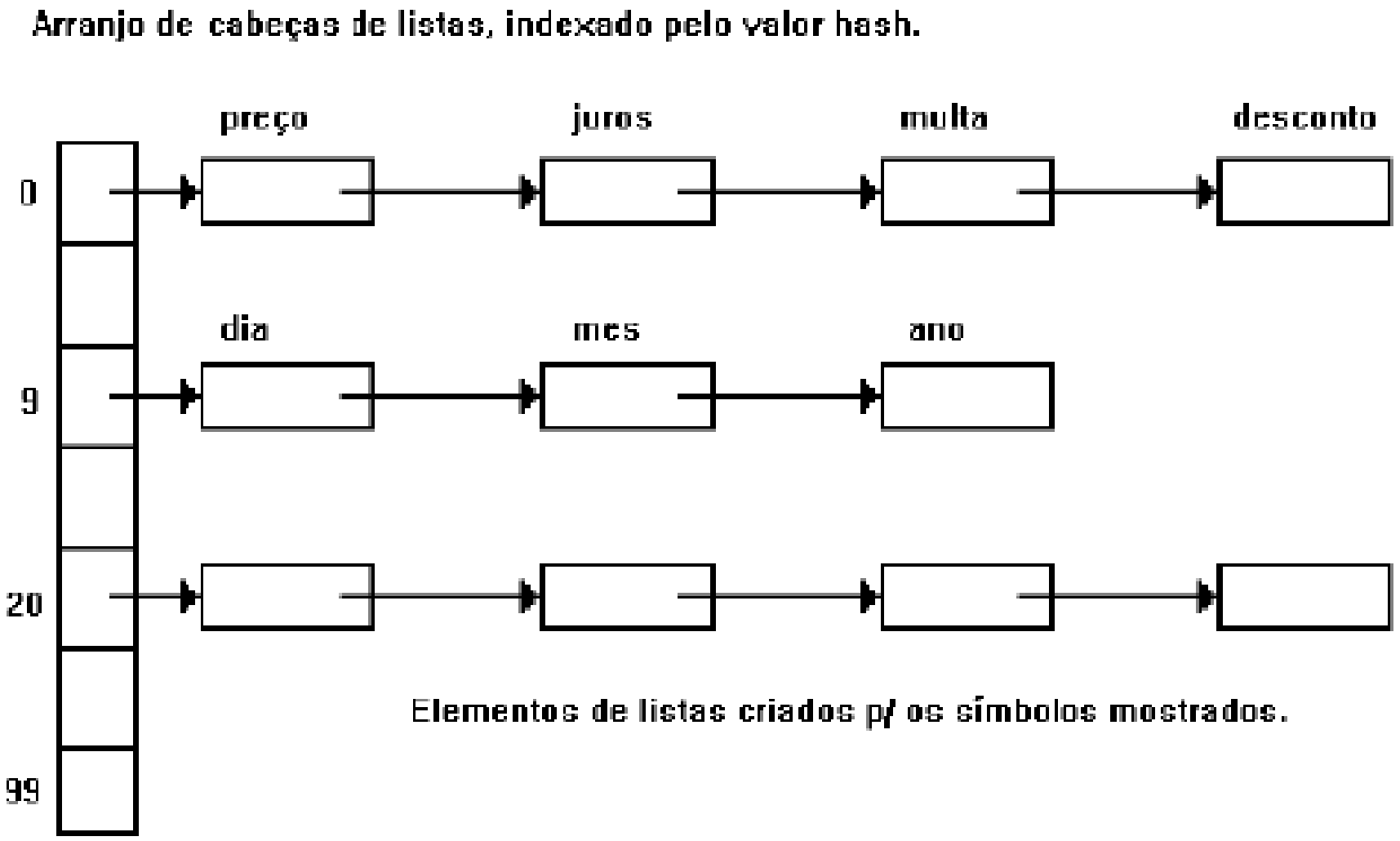
Implementação da Tabela de Símbolos

- Em uma tabela com *n* entradas, considera que, na média, é pesquisado *n/2* nomes para verificar se um dado nome está ou não na TS;
- O tempo gasto com inserção e consultas a esta tabela é muito alto;
- Uma saída para este problema é a implementação utilizando uma Tabela Hash (tabela de acesso direto).

Implementação da Tabela de Símbolos

- Uma tabela hash consiste basicamente de:
  - Um arranjo de tamanho fixo com *m* apontadores para entradas de tabela (idealmente um número primo);
  - Entrada de tabela organizadas em *m* listas encadeadas separadas;
    - Cada entrada da tabela de símbolos aparece em somente uma dessas listas.

Implementação da Tabela de Símbolos



Implementação da Tabela de Símbolos

- Na tabela hash, o *m* pode ser tão grande quanto necessário;
- Este método é mais eficiente que listas lineares;
- A maioria dos compiladores utilizam tabela hash para a implementação da TS.

Análise Semântica

- É uma técnica que permite realizar tradução concomitantemente com a análise sintática;
- Ações semânticas são associadas às regras de produção da gramática;
- A execução dessas ações podem:
  - Gerar ou interpretar código;
  - Armazenar informações na tabela de símbolos;
  - Emitir mensagens de erro;
  - Etc.

Análise Semântica

- Pode-se associar variáveis aos símbolos (terminais e não-terminais) da gramática;
- Com isso, os símbolos gramaticais passam a conter atributos capazes de armazenar valores durante o processo de reconhecimento;

Exemplo

1) D → var :T {adTabSimb(var.nome, T.tipo)}
2) T → real {T.tipo := “r”}
3) T → integer {T.tipo := “i”}

- O atributo nome de var é inicializado quando esse terminal é reconhecido durante o processo de análise léxica;

Esquema de Tradução

- Um esquema de tradução é uma extensão de uma GLC:
  - Através da associação de atributos aos símbolos gramaticais;
    - Um atributo de um símbolo pode conter um valor numérico, uma cadeia de caracteres, um tipo de dado, um endereço de memória, etc;
  - Ações semânticas às regras de produção;
    - Ações semânticas podem ser avaliações de atributos ou chamadas a procedimentos e funções.

Esquemas de Tradução

- Os atributos podem ser sintetizados ou herdados;
- Sendo S um símbolo, o valor de um atributo de S é dito:
  - Sintetizado, se ele é computado a partir, exclusivamente, dos valores dos atributos dos filhos de S.
  - Herdado, se ele é computado a partir dos valores dos atributos dos irmãos ou do pai de S.

Esquemas de Tradução

- Um token é uma tupla que contém todas as informações referentes a um símbolo terminal;
  - Constante numérica:
    - código ‘`cte`’ e o valor da constante.
  - Identificador:
    - código ‘id’, e o valor que pode ser um apontador para o índice da tabela de símbolos.

Esquemas de Tradução

- A árvore de derivação que mostra os valores dos atributos associados a cada nó é chamada de *árvore de derivação anotada*;

- Cada nó com exceção das folhas, corresponde a zero ou mais ações;

Exemplo

- Traduzir expressões infixadas para pós-fixadas

$E \rightarrow T\ R$

$R \rightarrow Op\ T\ \{print(Op.simbol)\}\ R \mid \varepsilon$

$T \rightarrow num\ \{print(num.lexval)\}$

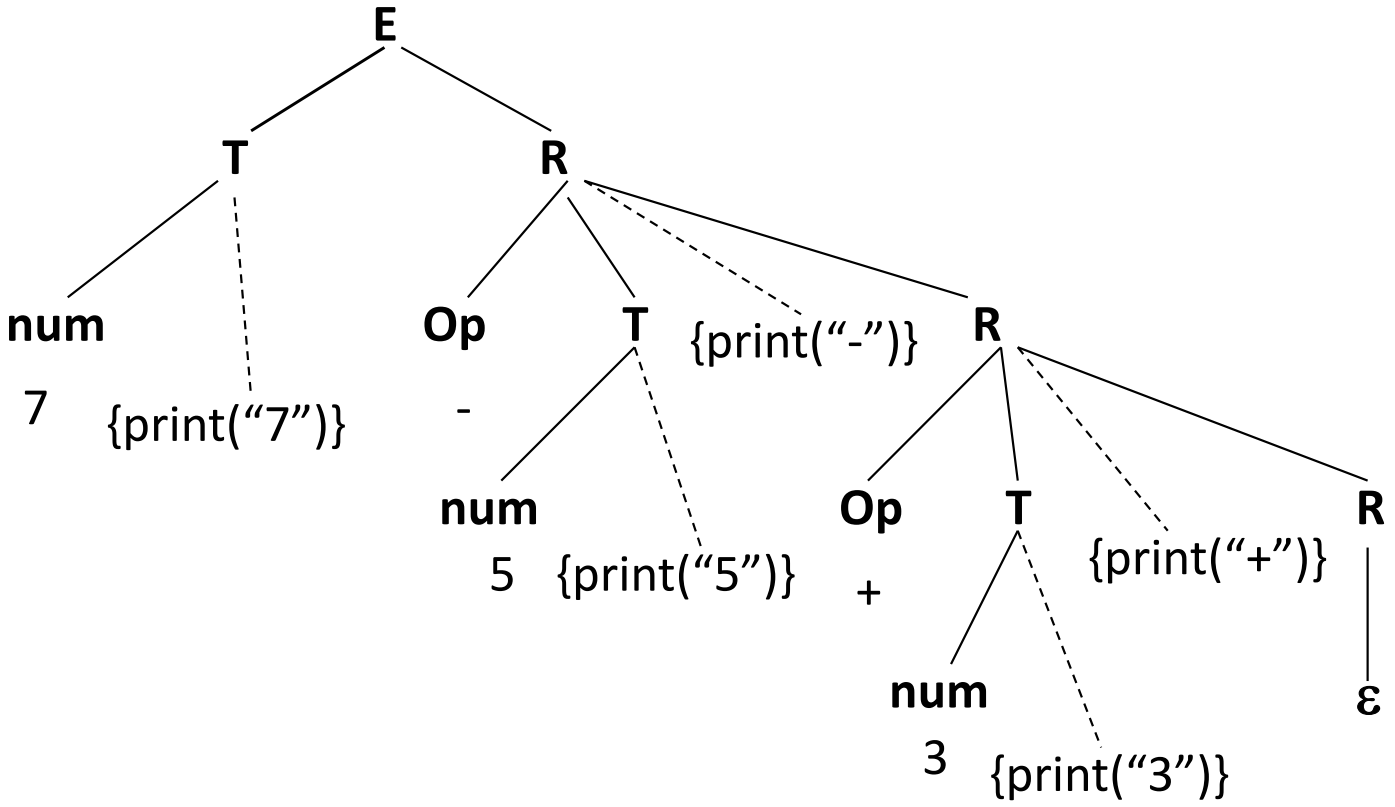
Ordem de Execução

- Pode-se visualizar a ordem das ações semânticas desenhando a árvore de derivação e agregando-se as ações como se fossem folhas da árvore.

- As ações são executadas quando as folhas correspondentes são visitadas usando a estratégia `depth-first`.

Exemplo

- 7 - 5 + 3
- A impressão fica 7 5 - 3 +



Estratégia depth-first

- Esta estratégia define uma forma de percorrer (visitar) os nós de uma árvore;

```
DEPTH_FIRST(N: node);
```

Para cada filho m de n, da esquerda para a direita, faça:

```
    DEPTH_FIRST(m);
    VISITE(n);
```

Tipos de Esquemas

- S-atribuído:
  - Este esquema opera somente com atributos sintetizados.
- L-atribuído:
  - Este esquema opera com atributos herdados, porém pode haver atributos sintetizados.

Exemplo – 2 Esquemas

Produções	Regras Semânticas
(1) D → T L	{L.t := T.tipo}
(2) T → int	{T.tipo := inteiro}
(3) T → real	{T.tipo := real}
(4) L → id , L1	{adtipo(id.indice,L.t); L1.t := L.t}
(5) L → id	{adtipo(id.indice,L.t)}

Esquema L-atribuído

- O esquema L-atribuído restringe o uso de atributos herdados, para permitir que ações semânticas possam ser executadas durante a análise sintática, em um único passo.
- Restrição:
  - Para um símbolo X no lado direito de uma regra de produção, a ação que calcula um atributo herdado de X deve aparecer à esquerda de X.

Esquema L-atribuído

- As regras 1 e 4 do esquema de tradução anterior fazem com que ele deixe de ser L-atribuído.
- Para satisfazer a restrição, a regra 1 teria que ser:

$$D \rightarrow T \; \{L.t := T.tipo\} \; L$$

Esquemas de Tradução L-atribuídos

- Um esquema é dito L-atribuído se cada atributo X_j, $1 \leq j \leq n$, no lado direito de $A \rightarrow X_1X_2...X_n$ depende somente:
  - dos atributos dos símbolos $X_1,X_2,...,X_{j-1}$ à esquerda na produção;
  - dos atributos herdados de A.
- Todo esquema S-atribuído é, L-atribuído, pois as restrições acima aplica-se somente a atributos herdados.

Algoritmo para avaliação de atributos

DEPTH_FIRST (n:nodo);

Para cada filho m de n, da esquerda para a direita, faça:

Calcule os atributos herdados de m;

DEPTH_FIRST(m);

Calcule os atributos sintetizados de n.

Cuidados na Ordenação das ações semânticas

- Na definição de um esquema de tradução, deve-se assegurar que os valores dos atributos já tenham sido calculados quando as ações referirem-se a eles.

- Se existem ambos, atributos herdados e sintetizados, deve-se proceder da seguinte forma:

Cuidados na Ordenação das ações semânticas

1) Atributo herdado associado a símbolo do lado direito da produção deve ser computado em ação semântica especificada antes desse símbolo;
2) Uma ação semântica só pode se referir a atributo sintetizado de símbolo que apareça a esquerda dessa ação;
3) Atributo sintetizado associado ao não-terminal do lado esquerdo da produção deve ser computado somente depois de serem computados todos os atributos que ele referencia.

Exemplo

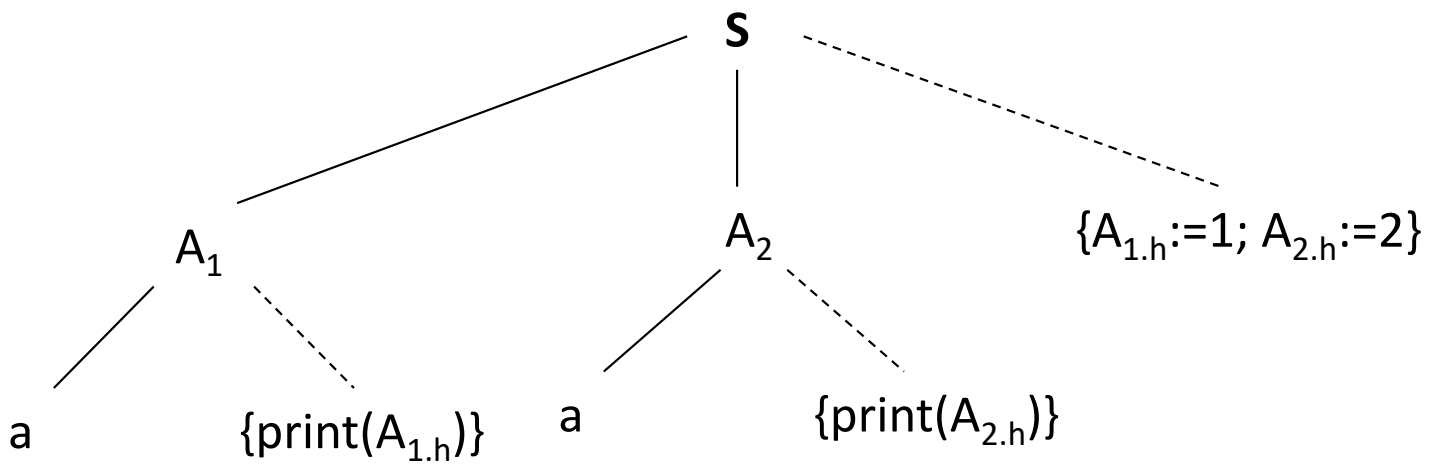
- Esquema de tradução não implementável em um único passo:

$S \rightarrow A1\ A2$	`{A1.h :=1; A2.h :=2}`
$A \rightarrow a$	`{print(A.h)}`

- Este esquema é não implementável porque não satisfaz o primeiro dos requisitos.

Exemplo

- Árvore de derivação para a sentença “aa”



Exemplo corrigido

- Para corrigir o problema do exemplo anterior, basta fazer as seguintes alterações:

$S \rightarrow$ `{A1.h:=1}` A1 `{A2.h:=2}` A2

$A \rightarrow$ `a` `{print(A.h)}`

Implementação de esquemas L-atribuídos

- Como os analisadores top-down não admitem recursividade a esquerda;
  - Há a necessidade de se fazer essa eliminação;
- E a sua transformação/eliminação nos obriga a fazer transformações nas ações semânticas associadas às regras.

Exemplo

- O exemplo abaixo reconhece a sequência de dígitos e obtém o seu somatório no atributo `val` do símbolo raiz

$A \rightarrow A1\ digit$ `{A.val :=A1.val+digit.lexval}`

$A \rightarrow digit$ `{A.val :=digit.lexval}`

- Porém é necessário eliminar a sua recursividade a esquerda

Exemplo Transformado

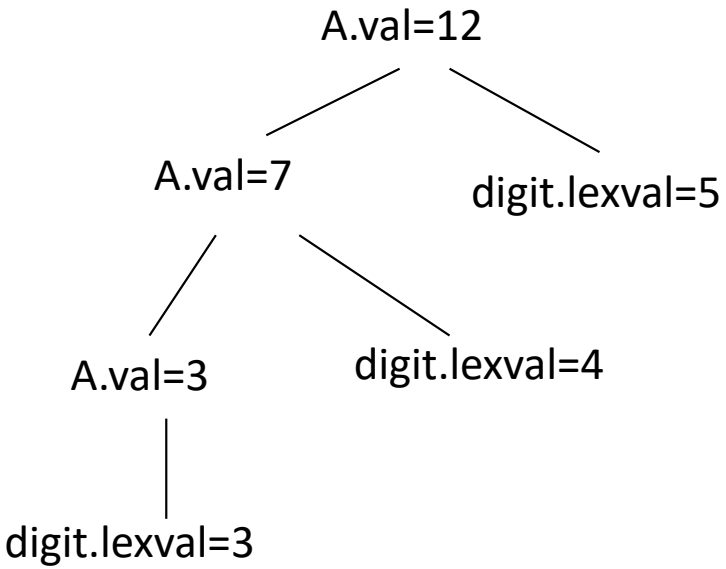
A → digit {X.h:=digit.lexval} X {A.val :=X.s}

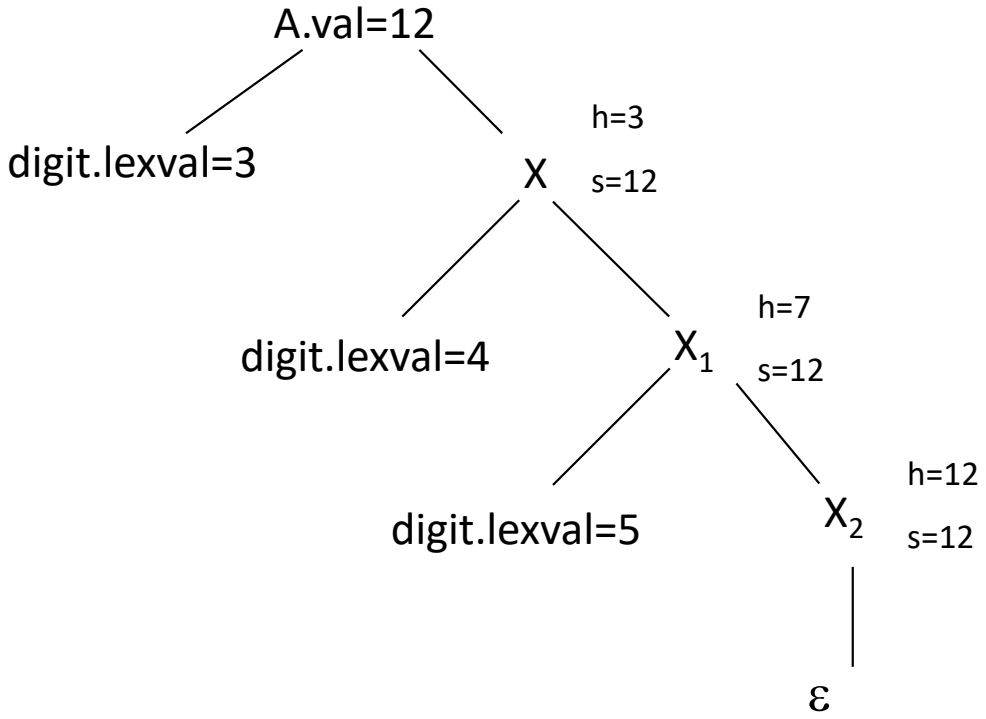
X → digit {X1.h:=X.h +digit.lexval} X1 {X.s :=X1.s}

X → ε {X.s :=X.h}

Exemplo Transformado

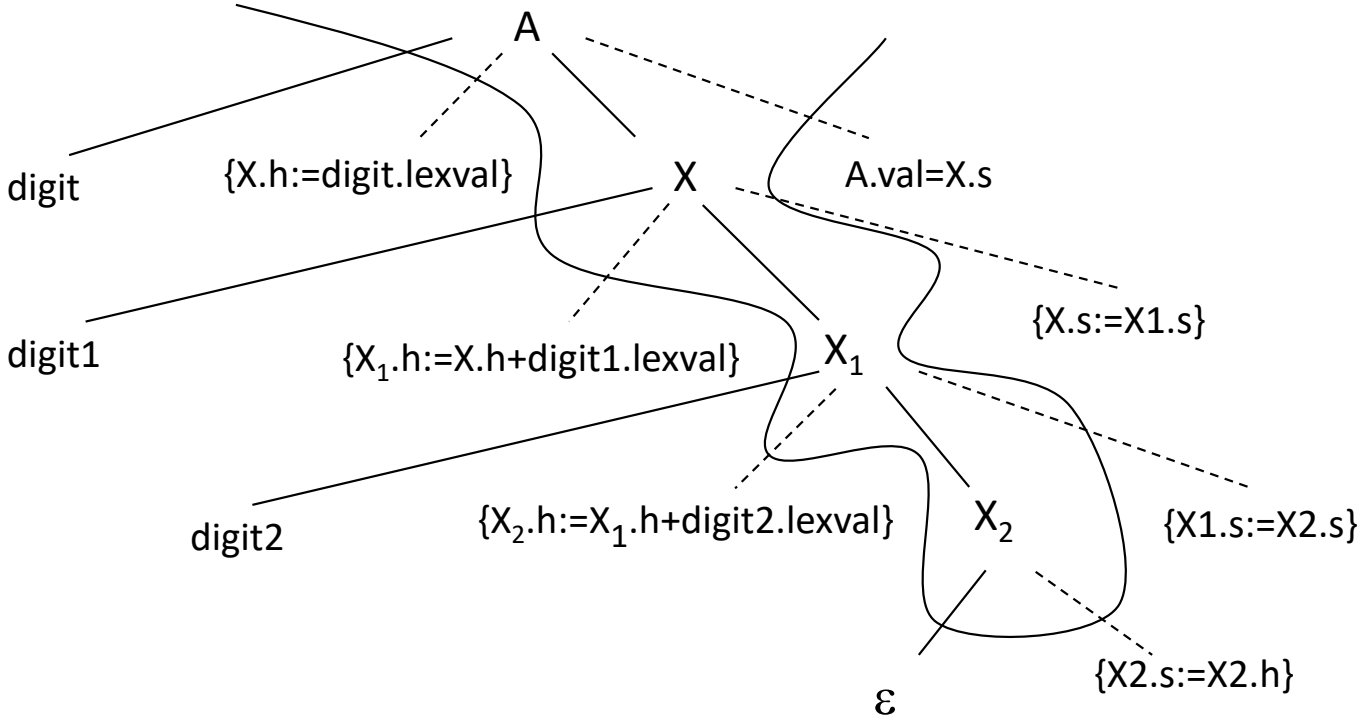
- Árvore de derivação para sequência de tokens 3 4 5





Ordem de avaliação dos atributos

- A primeira linha corresponde a descida na árvore e a segunda a subida.
- Na descida são computados os atributos herdados e na subida os sintetizados.



Generalização da transformação para L-atribuído

- Seja o seguinte esquema de tradução de S-atribuído.

$A \rightarrow A1\ Y$ `{A.a :=g(A1.a,Y.y)}`

$A \rightarrow X$ `{A.a :=f(X.x)}`

- O algoritmo elimina a recursividade a esquerda e altera as regras para:

$A \rightarrow X\ R$

$R \rightarrow Y\ R \mid \varepsilon$

Generalização da transformação para L-atribuído

- A transformação completa resulta em:

$A \rightarrow X$ {R.h :=f(X.x)} R {A.a :=R.s}

$R \rightarrow Y$ {R1.h :=g(R.h, Y.y)} R1 {R.s :=R1.s}

$R \rightarrow \varepsilon$ {R.s :=R.h}

Exemplo

$E \rightarrow E1+T$	{E.val :=E1.val + T.val}
$E \rightarrow E1-T$	{E.val :=E1.val - T.val}
$E \rightarrow T$	{E.val :=T.val}
$T \rightarrow (E)$	{T.val :=E.val}
$T \rightarrow num$	{T.val :=num.val}

Exemplo

- Eliminada a recursividade a esquerda teríamos:

$E \rightarrow T$ {R.h :=T.val} R {E.val :=R.s}

$R \rightarrow +T$ {R1.h :=R.h + T.val} R1 {R.s :=R1.s}

$R \rightarrow -T$ {R1.h :=R.h - T.val} R1 {R.s :=R1.s}

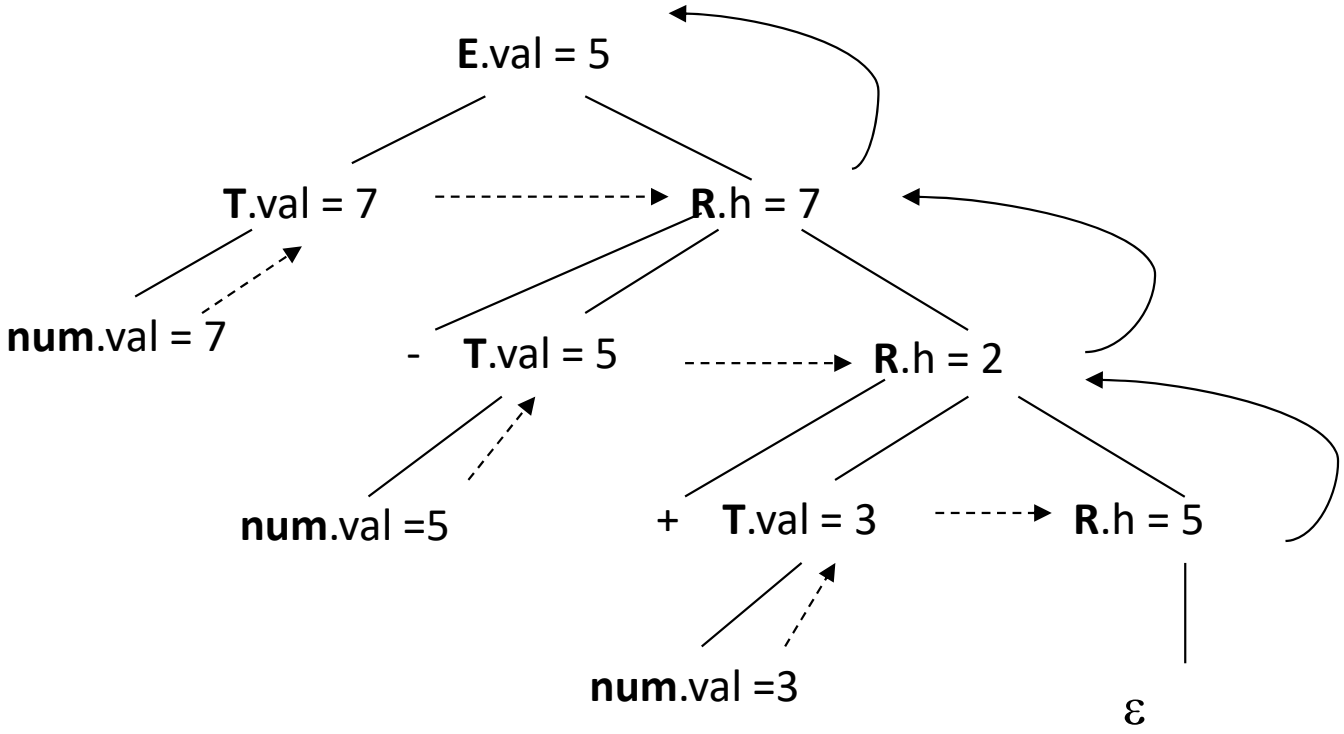
$R \rightarrow \varepsilon$ {R.s :=R.h}

$T \rightarrow (E)$ {T.val :=E.val}

$T \rightarrow num$ {T.val :=num.val}

Exemplo

- Para a expressão 7-5+3, a seguinte árvore de derivação anotada é produzida



Material elaborado por:

Prof. Dr. Augusto Mendes Gomes Jr.

augusto.gomes@animaeducacao.com.br

Prof. Dr. Fernando Kakugawa

fernando.kakugawa@animaeducacao.com.br