Aula 10 – Análise Semântica

Curso de Ciência da Computação

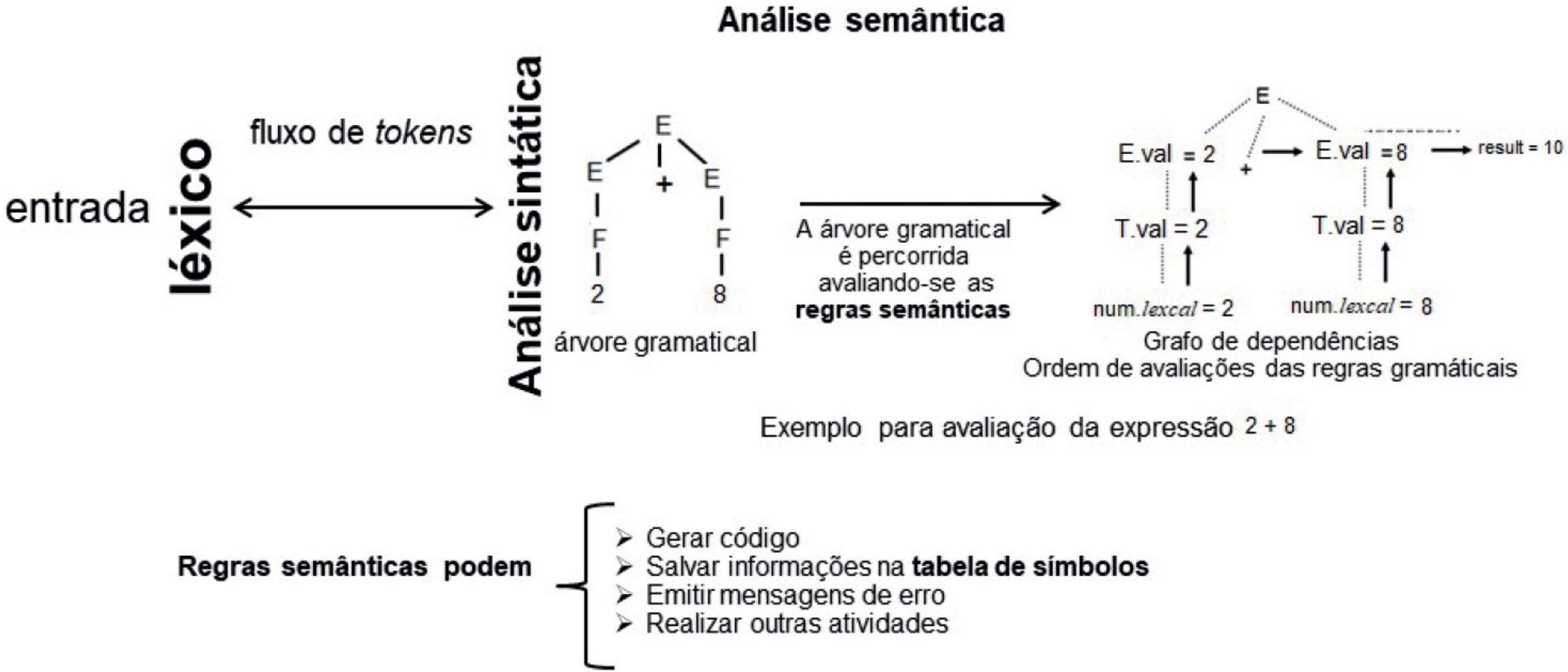
Prof. Dr. Rodrigo Xavier de Almeida Leão

Após conhecer sobre a análise sintática e as técnicas para construir um analisador sintático. Também foi possível identificar que a maioria das linguagens de programação podem ser definidas por linguagens livre de contexto (LLC). Mas, para completar a fase de análise de uma gramática, é necessário verificar se o contexto da frase, no caso das linguagens de programação os comandos, estão corretos. A esse processo de análise do contexto da frase denominamos análise semântica.

Como estudamos, o coração da análise é o analisador sintático, que recebe dados do léxico e verifica se existe uma árvore de derivação para o fluxo de dados gerado pelo léxico, seguindo as regras da gramática da linguagem. Já os aspectos semânticos são tratados por sub-rotinas específicas acionadas pelo analisador sintático. Durante

> A fase de análise semântica de um compilador conecta as definições das variáveis com o seu uso, verifica se cada expressão tem um tipo correto e traduz a sintaxe abstrata em uma representação mais simples e adequada para a geração do código de máquina (Appel, 2002, p.103, tradução nossa).

Figura 3.7 | Processo da tradução dirigida pela sintaxe



Lembrando: lexema é o valor representado pelo padrão do token.

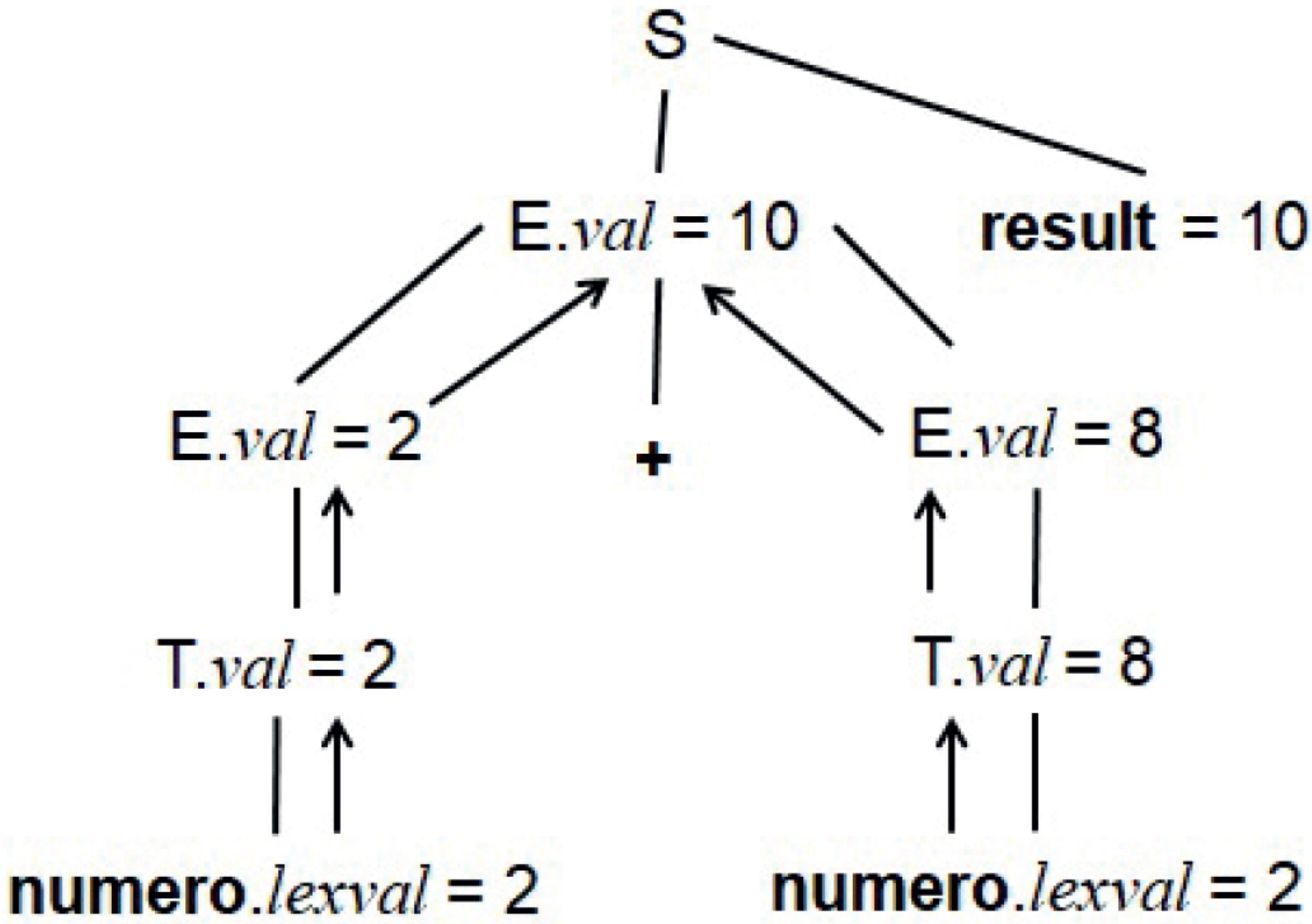
Tomemos como exemplo "valor = 10". Na análise léxica desse comando, identificamos que:

- "valor" é o lexema e o *token* é ID
- "10" é o lexema e o *token* é NUMERO

O analisador léxico para integrar-se aos analisadores sintático e semântico deverá, além de identificar o tipo do token, **adicionar na tabela de símbolos (TS) os atributos do *token***, por exemplo: nome do token, o lexema e o valor.

No exemplo apresentado, uma ação semântica será addTS(ID,"valor",tipo.inteiro), que indica inclusão na tabela de símbolo do ID, o lexema e o valor, e a outra ação será addTS(NUMERO,10,tipo.inteiro).

Figura 3.8 | Árvore gramatical anotada



Dada a gramática:

<*decl*> ::= <*tipo*> <*listaVariaveis*>

<*tipo*> ::= **int** | **double**

<*listaVariaveis*> ::= **id ,** <*listaVariaveis*> | **id**

Vamos incluir as regras semânticas usando a definição dirigida pela sintaxe, em que cada símbolo tem um conjunto de atributos associados.

1. Vamos usar a notação formal para deixar a definição dos atributos mais evidente:

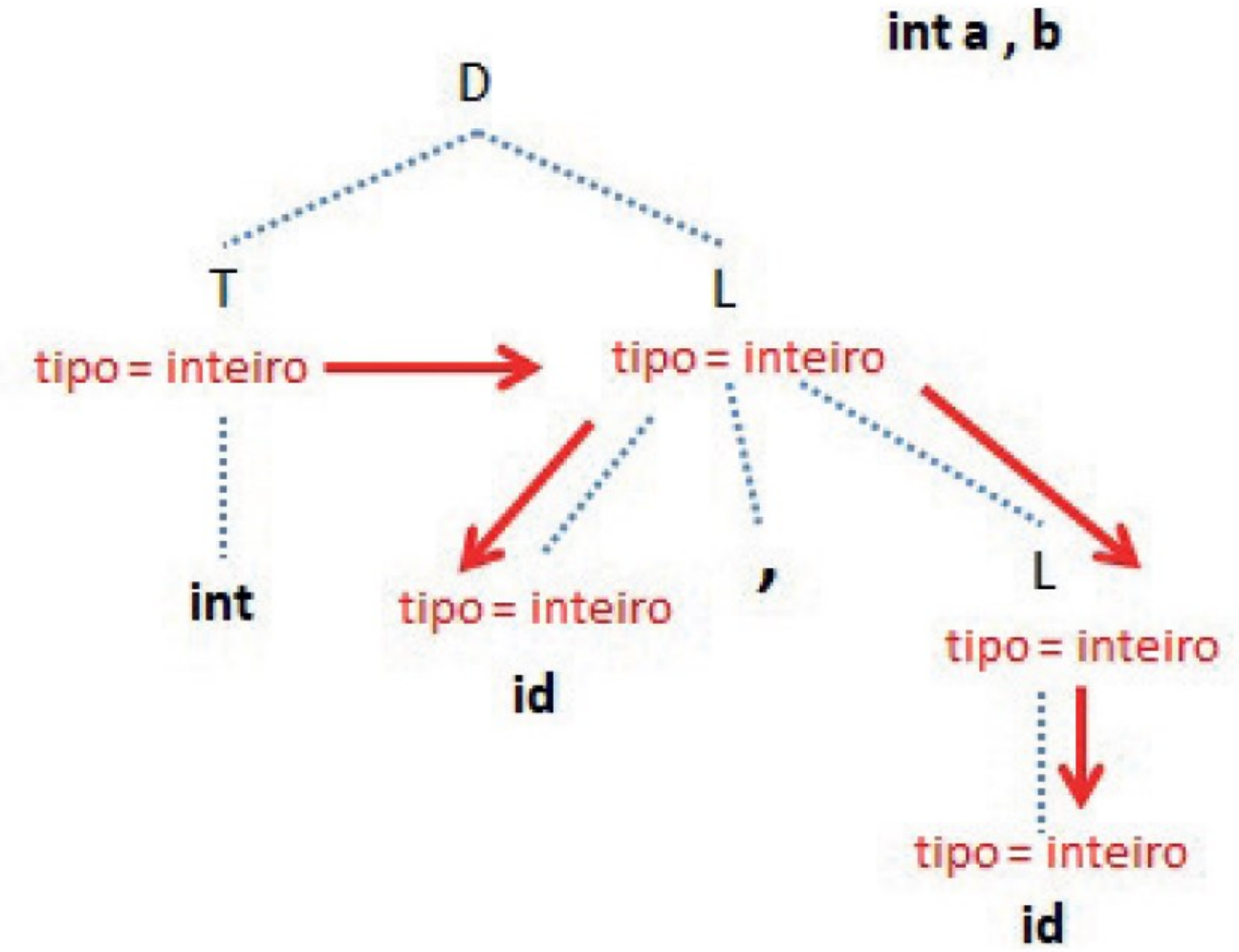
D := T L	D é o não terminal para <*decl*> T é o não terminal para <*tipo*> L é o não terminal para <*listaVariaveis*>
T := **int** | **double**	A produção T nos leva aos símbolos terminais **int** ou **double**
L := **id ,** L | **id**	id e , são símbolos terminais

2. Agora vamos inserir as regras de atributos. Para que você possa entender melhor, leia os comentários na sequência 1 e 2, assim será fácil observar que L tem uma dependência e D uma dependência herdada:

Regras de Produção	Regras Semânticas		Comentário
D := T L	L.tipo = T.tipo	1	Separamos a produção, pois **cada tipo de dado tem valor específico**, logo ações diferentes.
T := **int**	T.tipo = inteiro		
T := **double**	T.tipo = real		
L := **id**	id.tipo = L.tipo	2	O identificador (variável) deve ter o mesmo tipo da lista de variáveis(L)
L := **id** , L	id.tipo = L.tipo		

3. Para concluirmos nosso exemplo, vamos construir a árvore de derivação anotada para a sentença **int a , b**:

Figura 3.9 | Árvore anotada com o grafo de dependência



As ações semânticas muitas vezes são implementadas por meio de de sub-rotinas e, no processo de verificação da semântica, é necessário apontar os erros: a) de declaração de identificadores; b) compatibilidade de tipos; e c) compatibilidade entre parâmetros e argumentos. Vejamos o exemplo apresentado a seguir do trecho de um programa em C e o tratamento para cada erro semântico existente no código:

	Código em C	Identificação dos erros - mensagem
1	`void funcao1(int a, char`	
2	`b){`	
3	`// comandos`	
4	`}`	
5	`char funcao2(double c){`	
6	`// comandos`	(E1)-Há um erro semântico de incompatibilidade no retorno da função - linhas 4 e 6
7	`return 10.5;`	
8	`}`	
9	`int main(){`	
10	`int x, y;`	(E2)-Há um erro de incompatibilidade de tipo entres as linhas 9 e 10
	`x = '1';`	

```
11
    k = 10;                          (E3)-[Error] 'k' was not declared
                                     in this scope ( erro de declaração
                                     da variável ) - linha 11
12
    x = funcao2(y);                  (E4)-Há um erro de incompatibilidade
                                     entre as linhas 4 e 12
13
    funcao1(x);
14  }                                (E5)- [Error] too few
                                     arguments to function 'void
                                     funcao1(int, char)
                                     [Note] declared here (linha 1)
                                     Erro de compatibilidade
                                     entre argumentos - linha 4
                                     e 13. Foram passados menos
                                     argumentos na chamada em
                                     relação a declaração)
```

a definição das regras semânticas possuem complexidade e nem sempre todos os erros semânticos são implementados pelo projetista do compilador. A implicação disso é que o programador poderá ter dificuldade para encontrar erros em seus programas, pois, no caso apresentado, não ocorre erro de compilação nem erro de execução, mas ocorrerá um erro de saída não esperado, o chamado erro lógico, e erros lógicos são difíceis de serem corrigidos. Outros detalhes importantes com relação ao tratamento de erro são acuracidade e clareza das mensagens do tratamento dos erros. Se a linha 10 fosse escrita x = "1", o compilador C apontaria o seguinte erro: