Conteúdo Programático e Cronograma

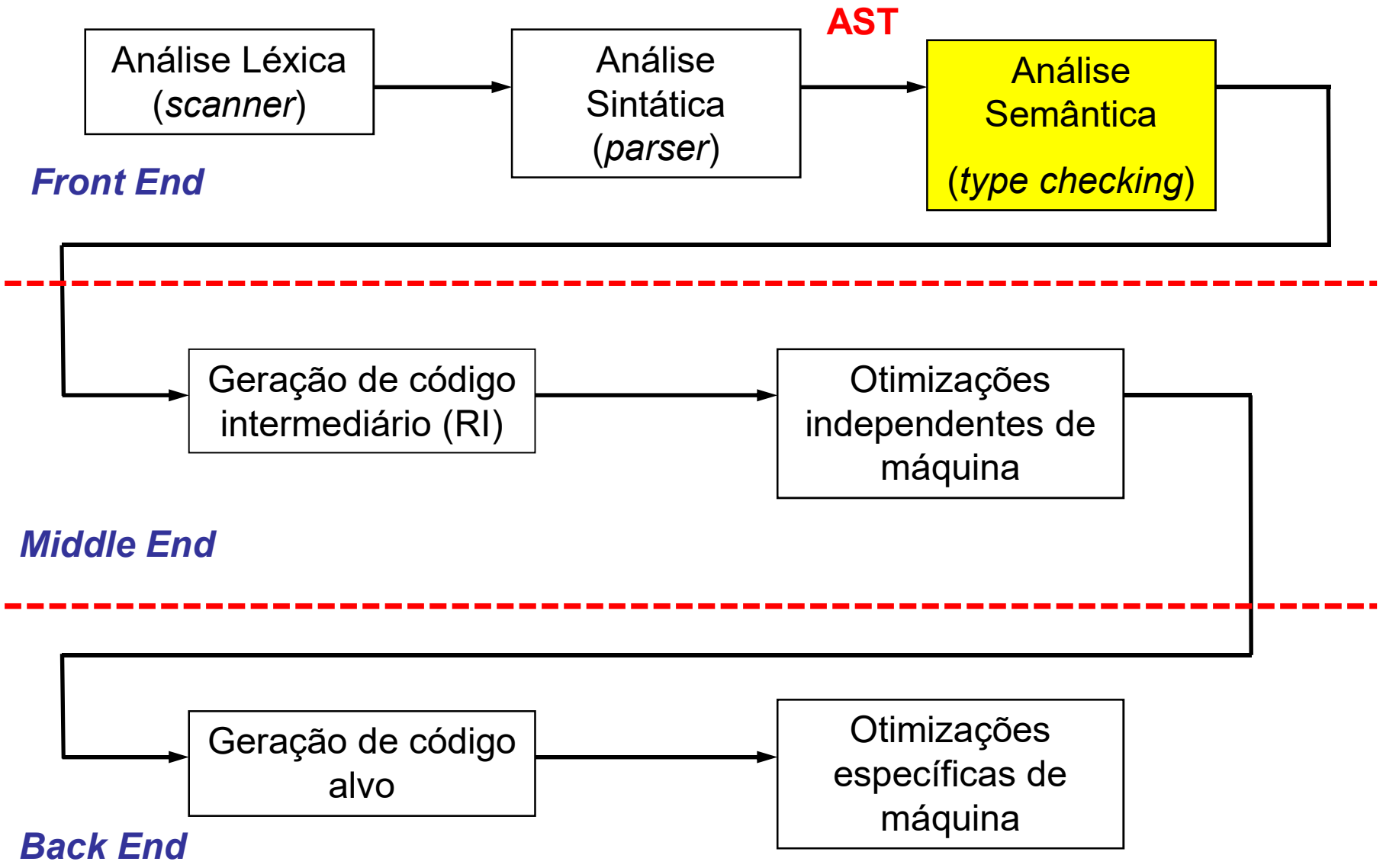
1º Semestre

- ~~Organização e estrutura de compiladores~~
- ~~Análise Léxica~~
- ~~Análise Sintática~~
- ~~Ferramentas de geração automática de compiladores~~

2º Semestre

- Análise Semântica
- Geração de código
- Análise de fluxo de dados
- Otimização de código
- Alocação de registradores

Fluxo do Compilador



Front-End do Compilador

Tipo de Erro	Exemplo	Detectado Por:
Léxico	...@...	Analisador Léxico
Sintático	...x*%...	Analisador Sintático
Semântico	... int x; y=x(3);	Analisador Semântico
Corretude	O seu programa	Testador/Usuário

Árvore Sintática Abstrata (AST)

Dada uma GLC, uma árvore de derivação é uma árvore rotulada cuja raiz tem sempre o não-terminal inicial como rótulo. Em uma derivação

$S \Rightarrow a_1\ a_2 \ldots\ a_n$

o nó *S* tem como filhos *n* nós rotulados de a_1 até a_n que são os símbolos terminais.

O *parser* produz ***A****bstract* ***S****yntax* ***T****rees*

- Estas árvores refletem a estrutura sintática do programa
- São a interface entre o *parser* e as próximas fases da compilação
- Já abstraem detalhes irrelevantes para as fases seguintes. Ex.: pontuação.
- À medida que um não terminal é reconhecido, um nó da árvore é criado
- De baixo para cima (em *parsers* LR)
- Cada execução de um dos métodos associados aos não-terminais cria um nó da árvore e "pendura" nele os nós filhos.

Árvore Sintática Abstrata (AST)

Considere a gramática:

$$E \rightarrow \mathit{int} \mid (E) \mid E + E$$

E a cadeia:

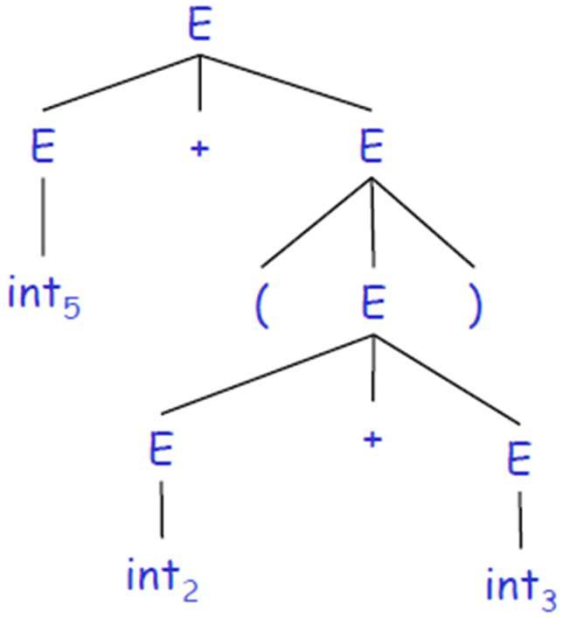
$$5 + (2 + 3)$$

O analisador léxico irá produzir a seguinte sequencia de *tokens*:

$$\mathrm{int}_5 \quad \text{‘+’} \quad \text{‘(‘} \quad \mathrm{int}_2 \quad \text{‘+’} \quad \mathrm{int}_3 \quad \text{‘)’}$$

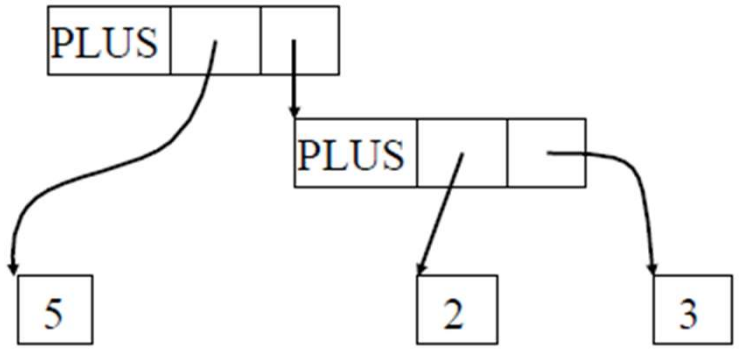
Árvore de Derivação

O analisador sintático “constrói” a árvore de derivação:



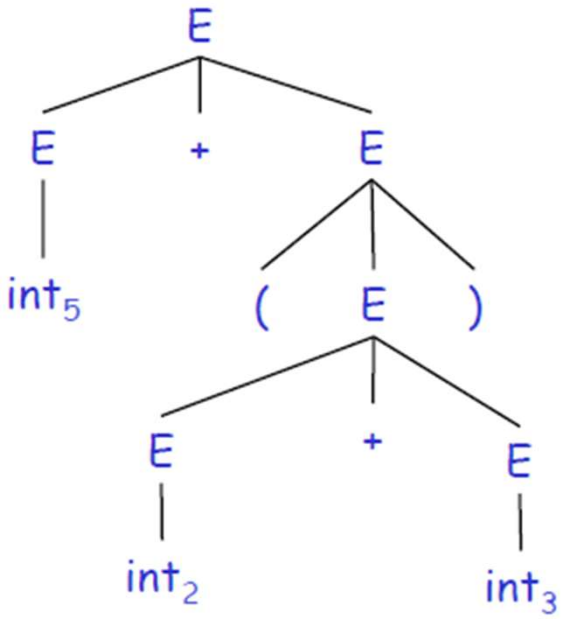
- Detalha o processo de funcionamento do *parser*
- Captura e estrutura de aninhamento do programa
- Mas contém informação desnecessária
  - Parênteses
  - Nós não-terminais

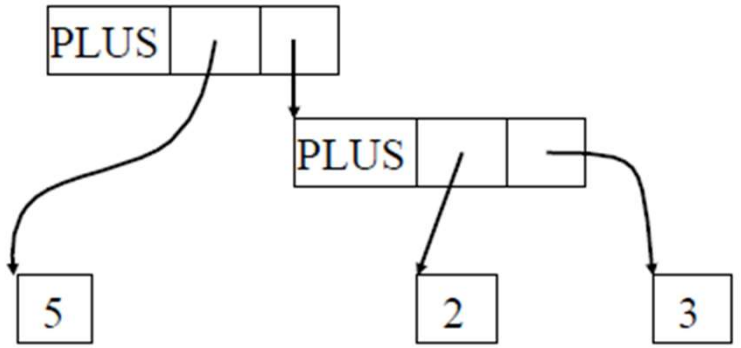
Árvore Sintática Abstrata (AST)



- Também captura e estrutura de aninhamento do programa
- É uma abstração da sintaxe da gramática, mais compacta e simples de usar
- É uma importante estrutura de dados do compilador

5 + (2 + 3)





Tradução Dirigida Pela Sintaxe: Ações Semânticas

A **AST** é construída utilizando **Tradução Dirigida pela Sintaxe**, através de ações semânticas.

Nas ações semânticas tem-se:

- Cada símbolo da gramática pode possuir vários atributos associados a ele.
- No caso de símbolos terminais, o valor de tais atributos é determinado pelo analisador léxico.
- Cada produção da gramática possui uma ação associada:

$$X \rightarrow Y_1\ Y_2 \ldots Y_n \quad \{ \text{action} \}$$

- As ações podem se referir ou computar atributos de símbolos da gramática.

Ações Semânticas: Exemplo

Considere a gramática:

$$E \rightarrow int \mid (E) \mid E + E$$

Para cada símbolo ***X*** da gramática, define-se um atributo ***val***, que armazena valores inteiros, sendo o conjunto representado como ***X.val***

- Para símbolos terminais, ***val*** corresponde ao valor léxico associado
- Para símbolos não-terminais, ***val*** é o valor de uma expressão (podendo ser computada através de valores de sub-expressões).

Reescrevendo a gramática com as devidas ações semânticas tem-se:

$E \rightarrow int$ { E.val = int.val }
$\mid E_1 + E_2$ { E.val = E_1.val + E_2.val }
$\mid (E_1)$ { E.val = E_1.val }

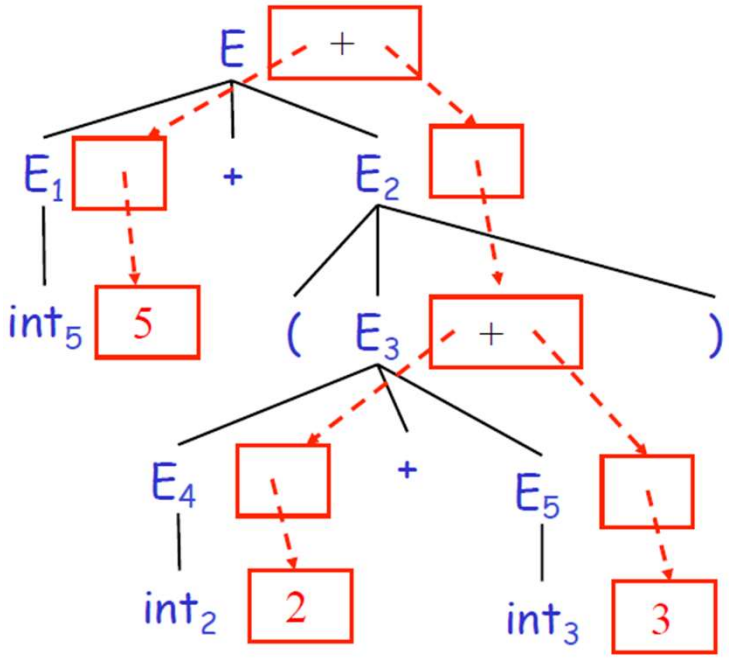
Com essas ações, tem-se um simples calculadora.

Ações Semânticas: Exemplo

Cadeia: **5 + (2 + 3)**
Tokens: **int_5 '+' '(' int_2 '+' int_3 ')'**

Derivações	**Ações Semânticas**
$E \rightarrow E_1 + E_2$	$E.val = E_1.val + E_2.val$
$E_1 \rightarrow int_5$	$E_1.val = int_5.val = 5$
$E_2 \rightarrow (E_3)$	$E_2.val = E_3.val$
$E_3 \rightarrow E_4 + E_5$	$E_3.val = E_4.val + E_5.val$
$E_4 \rightarrow int_2$	$E_4.val = int_2.val = 2$
$E_5 \rightarrow int_3$	$E_5.val = int_3.val = 3$

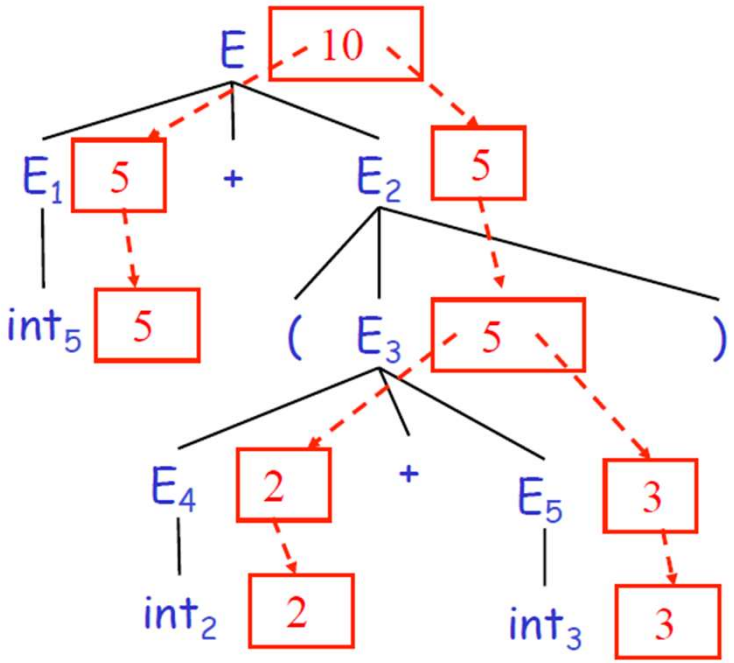
Ações Semânticas: Exemplo



Derivações	Ações Semânticas
$E \rightarrow E_1 + E_2$	$E.val = E_1.val + E_2.val$
$E_1 \rightarrow int_5$	$E_1.val = int_5.val = 5$
$E_2 \rightarrow (E_3)$	$E_2.val = E_3.val$
$E_3 \rightarrow E_4 + E_5$	$E_3.val = E_4.val + E_5.val$
$E_4 \rightarrow int_2$	$E_4.val = int_2.val = 2$
$E_5 \rightarrow int_3$	$E_5.val = int_3.val = 3$

- Na árvore de derivação, cada nó rotulado **E** possui um campo para o atributo val.
- Os nós em vermelho formam um grafo de dependência, onde cada atributo deve ser computado somente após os seus sucessores serem computados.
- No caso apresentado, os atributos podem ser computados em um percurso em pós-ordem no grafo de dependência.

Ações Semânticas: Exemplo



Derivações	Ações Semânticas
$E \rightarrow E_1 + E_2$	$E.val = E_1.val + E_2.val$
$E_1 \rightarrow int_5$	$E_1.val = int_5.val = 5$
$E_2 \rightarrow (E_3)$	$E_2.val = E_3.val$
$E_3 \rightarrow E_4 + E_5$	$E_3.val = E_4.val + E_5.val$
$E_4 \rightarrow int_2$	$E_4.val = int_2.val = 2$
$E_5 \rightarrow int_3$	$E_5.val = int_3.val = 3$

Os atributos **E.val** que aparecem para os não-terminais na árvore de derivação, são chamados de **atributos sintetizados**, pois seus valores são computados com base nos atributos de seus nós filhos.

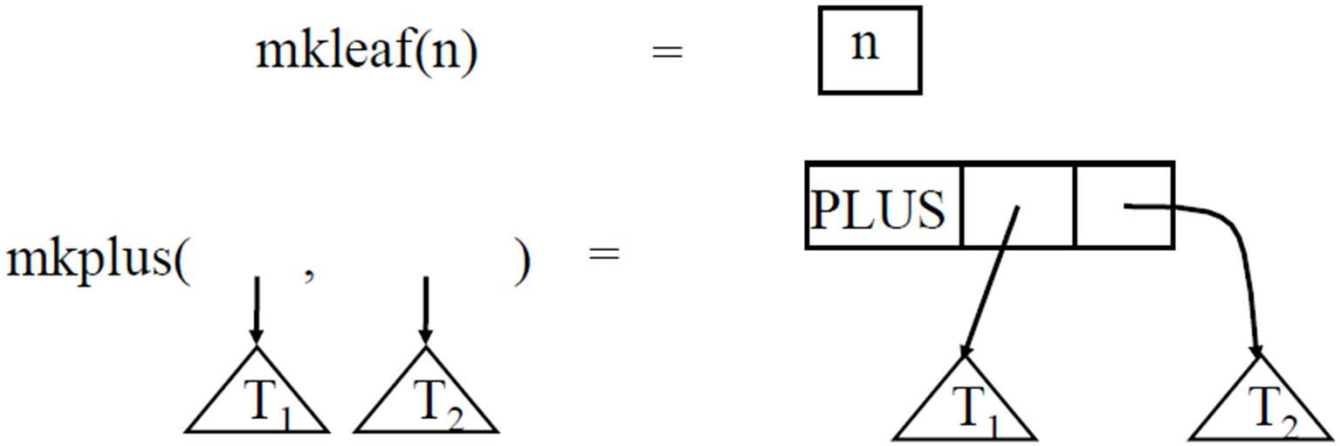
Atributos de não-terminais que são computados com base em valores de nós pai são chamados de **atributos herdados**.

Tradução Dirigida Pela Sintaxe: Construção da AST

As ações semânticas podem ser utilizadas para se construir a **AST**, bem como realizar a verificação de tipos.

Para a construção da AST, devem ser definidas estruturas de dados para representar o programa.

Ex.: considere uma árvore abstrata com dois tipos de básicos de estruturas e construtores:



Tradução Dirigida Pela Sintaxe: Construção da AST

Define-se um atributo sintetizado chamado de **ast**
Os valores de **ast** são ASTs (ponteiros para estruturas de árvore)
O atributo **int.lexval** is o valor inteiro do *token* (o valor léxico do *token*)

Gramática com as ações semânticas:

$E \rightarrow int$	{ E.ast = mkleaf(int.lexval) }
$\mid E_1 + E_2$	{ E.ast = mkplus(E_1.ast + E_2.ast) }
$\mid (E_1)$	{ E.ast = E_1.ast }

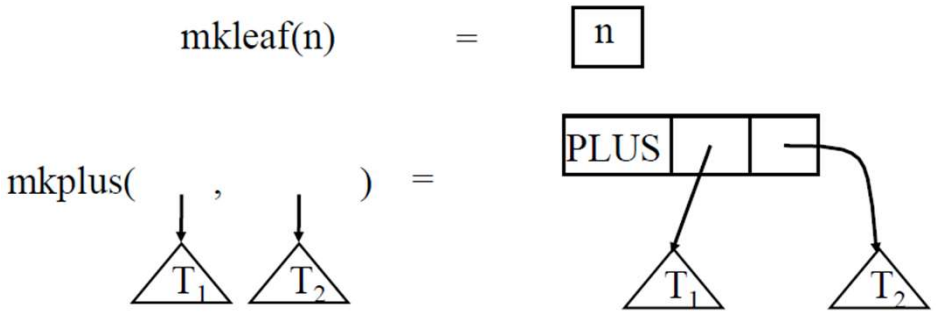
Tradução Dirigida Pela Sintaxe: Construção da AST

Cadeia: **5 + (2 + 3)**
Tokens: **int_5 '+' '(' int_2 '+' int_3 ')'**

$E \rightarrow int$	**{ E.ast = mkleaf(int.lexval) }**
$\mid E_1 + E_2$	**{ E.ast = mkplus(E_1.ast + E_2.ast) }**
$\mid (E_1)$	**{ E.ast = E_1.ast }**

Como seria a AST considerando a seguinte sequência de derivações?

$E_1 \rightarrow int_5$
$E_4 \rightarrow int_2$
$E_5 \rightarrow int_3$
$E_3 \rightarrow E_4 + E_5$
$E_2 \rightarrow (E_3)$
$E \rightarrow E_1 + E_2$



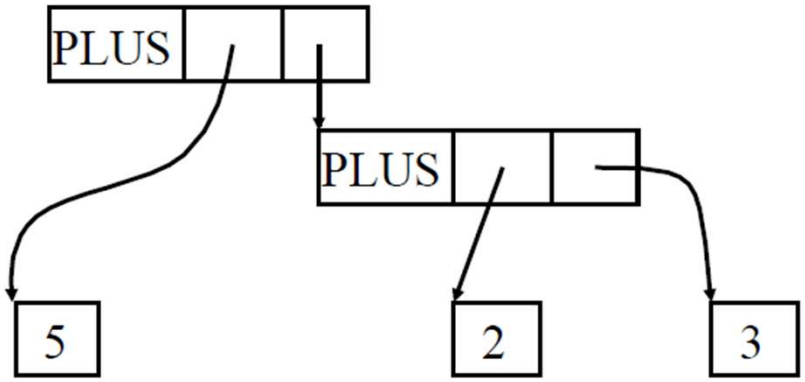
Tradução Dirigida Pela Sintaxe: Construção da AST

Cadeia: **5 + (2 + 3)**
Tokens: **int_5 '+' '(' int_2 '+' int_3 ')'**

E → *int*	**{ E.ast = mkleaf(int.lexval) }**
| E_1 + E_2	**{ E.ast = mkplus(E_1.ast + E_2.ast) }**
| (E_1)	**{ E.ast = E_1.ast }**

Construção da **AST**:

E.ast = mkplus(mkleaf(5),
mkplus(mkleaf(2), mkleaf(3))



Tradução Dirigida Pela Sintaxe: Construção da AST

Quais seriam as estruturas de dados e ações semânticas para a construção de uma **AST** para a gramática a seguir ?

$S \rightarrow \textit{if}\ E\ \textit{then}\ S\ \textit{else}\ S$
$S \rightarrow \textit{begin}\ S\ L$
$S \rightarrow \textit{print}\ E$
$L \rightarrow \textit{end}$
$L \rightarrow ;\ S\ L$
$E \rightarrow E + E$
$E \rightarrow (\ E\ )$
$E \rightarrow \textit{num}$

Tradução Dirigida pela Sintaxe

Derivações	Ações Semânticas
$D \rightarrow T\ L$	L.type = T.type
$T \rightarrow int$	T.type = integer
$T \rightarrow float$	T.type = float
$L \rightarrow L_1\ ,\ id$	L_1.type = L.type
$L \rightarrow id$	id.type = L.type

Cadeia: **float a, b, c**
Tokens: **float id , id, id**

Como é a aplicação das ações semânticas **durante** a construção da árvore de derivação para a seguinte sequencia de ***reduces***, baseados em um *parser LR* ?

$T \rightarrow float$
$L_2 \rightarrow id$
$L_1 \rightarrow L_2\ ,\ id$
$L \rightarrow L_1\ ,\ id$
$D \rightarrow T\ L$

Tradução Dirigida pela Sintaxe

- **Atributo Sintetizado** para um não-terminal A em um nó N da árvore de derivação é definido por uma regra semântica associada à produção naquele nó, sendo definido apenas em termos dos valores dos atributos dos nós filhos de N e do próprio N.

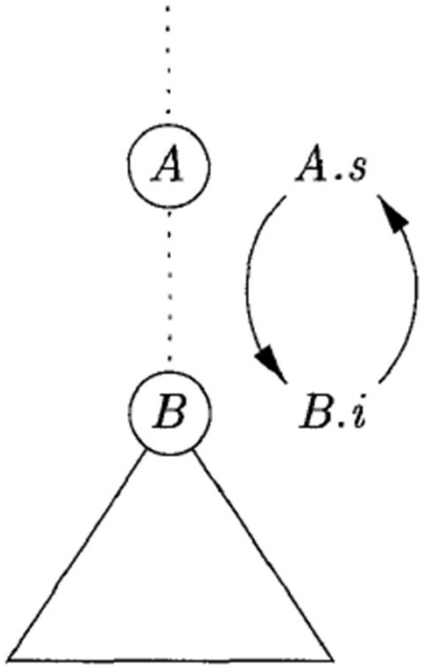
- **Atributo Herdado** para um não-terminal B em um nó N da árvore de derivação é definido por uma regra semântica associada à produção nó pai de N, sendo definido apenas em termos dos valores dos atributos do pai de N, do próprio N e dos irmãos de N.

Tradução Dirigida pela Sintaxe

PRODUCTION	SEMANTIC RULES
$A \rightarrow B$	$A.s = B.i;$ $B.i = A.s + 1$

Dependência circular entre ***A.s*** e ***B.i***

Não é possível determinar a ordem em que os atributos devem ser avaliados.



Tradução Dirigida Pela Sintaxe: Ordem de Avaliação

- **Definições S-Atribuídas**: um conjunto de definições de tradução dirigida por sintaxe é S-atribuída se todos os atributos são sintetizados.

- **Definições L-Atribuídas**: um conjunto de definições de tradução dirigida por sintaxe é L-atribuída se cada atributo é:

  1. Sintetizado ou,
  2. Herdado, onde para $A \rightarrow X_1 X_2 \ldots X_n$ e que exista um atributo herdado $\mathbf{X_i.a}$, calculado por uma regra associada a essa produção, onde a regra pode usar apenas:

     - Os atributos herdados associados a A
     - Os atributos herdados ou sintetizados associados à ocorrências dos símbolos $X_1 X_2 \ldots X_{i-1}$ localizados à esquerda de $\mathbf{X_i}$
     - Os atributos herdados ou sintetizados associados a $\mathbf{X_i}$ desde que não existam ciclos em um grafo de dependência formado pelos atributos desse $\mathbf{X_i}$

Tradução Dirigida Pela Sintaxe: Ordem de Avaliação

Definições **S-atribuídas** podem ser avaliadas construindo-se a árvore de derivação e depois realizando-se um percurso em **Pós-Ordem** na árvore de derivação.

Definições **L-atribuídas** podem ser avaliadas construindo-se a árvore de derivação e depois um grafo de dependência com os atributos e ordenando-se topologicamente o mesmo e executando as ações semânticas na ordem topológica obtida.

Para definições de tradução dirigida pela sintaxe que não sejam S ou L atribuídas, pode não ser possível determinar uma ordem de avaliação dos atributos.

Tradução Dirigida pela Sintaxe

Definição L-atribuída:

Derivações	Ações Semânticas
$T \rightarrow F\ T'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \rightarrow * F\ T'_1$	$T'_1.inh = T'.inh * F.val$ $T'.syn = T_1'.syn$
$T' \rightarrow$	$T'.syn = T'.inh$
$F \rightarrow digit$	$F.val = digit.lexval$

Cadeia: **3*5**

Tokens: $digit_3$ * $digit_5$

Qual seria a sequência de avaliação das ações semânticas?

$T \rightarrow F\ T'$

$F \rightarrow digit_3$

$T' \rightarrow * F\ T'$

$F \rightarrow digit_5$

$T' \rightarrow$

Tradução Dirigida pela Sintaxe

Definição L-atribuída:

Derivações	Ações Semânticas
T → F T’	T’.inh = F.val T.val = T’.syn
T’ → * F T'_1	T'_1.inh = T’.inh * F.val T’.syn = T_1’.syn
T’ →	T’.syn = T’.inh
F → *digit*	F.val = *digit*.lexval

Cadeia: **3*5**

Tokens: $digit_3$ * $digit_5$

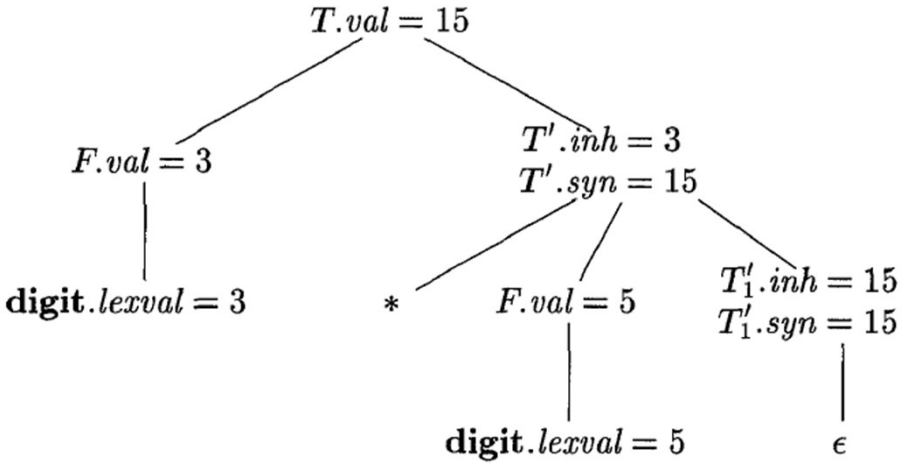
T → F T’

F → $digit_3$

T’ → * F T’

F → $digit_5$

T’ →



Tradução Dirigida Pela Sintaxe: Ordem de Avaliação

- **Definições S-Atribuídas**: são adequadas para *parsers bottom-up*, já que os atributos são todos sintetizados e a árvore de derivação é construída das folhas até a raiz, como em *parsers LR.*

- **Definições L-Atribuídas**: são adequadas para *parsers top-down*, já que os atributos herdados já terão sido computados uma vez que a árvore de derivação é construída da raiz até as folhas, como em *parsers LL*.

Tradução Dirigida pela Sintaxe

Derivações	Ações Semânticas
D → T L	L.inh = T.type
T → *int*	T.type = integer
T → *float*	T.type = float
L → L_1 , *id*	L_1.inh = L.inh *addType*(id.entry,L.inh)
L → *id*	*addType*(id.entry,L.inh)

Cadeia: **float a, b, c**
Tokens: **float id , id, id**

A definição é S-Atribuída ou L-Atribuída?

D → T L
T → *float*
L → L_1 , *id*
L → L_1 , *id*
L → *id*

Tradução Dirigida pela Sintaxe

Derivações	Ações Semânticas
D → T L	L.inh = T.type
T → *int*	T.type = integer
T → *float*	T.type = float
L → L_1 , *id*	L_1.inh = L.inh *addType*(id.entry,L.inh)
L → *id*	*addType*(id.entry,L.inh)

Cadeia: **float a, b, c**
Tokens: **float id , id, id**

D → T L
T → *float*
L → L_1 , *id*
L → L_1 , *id*
L → *id*