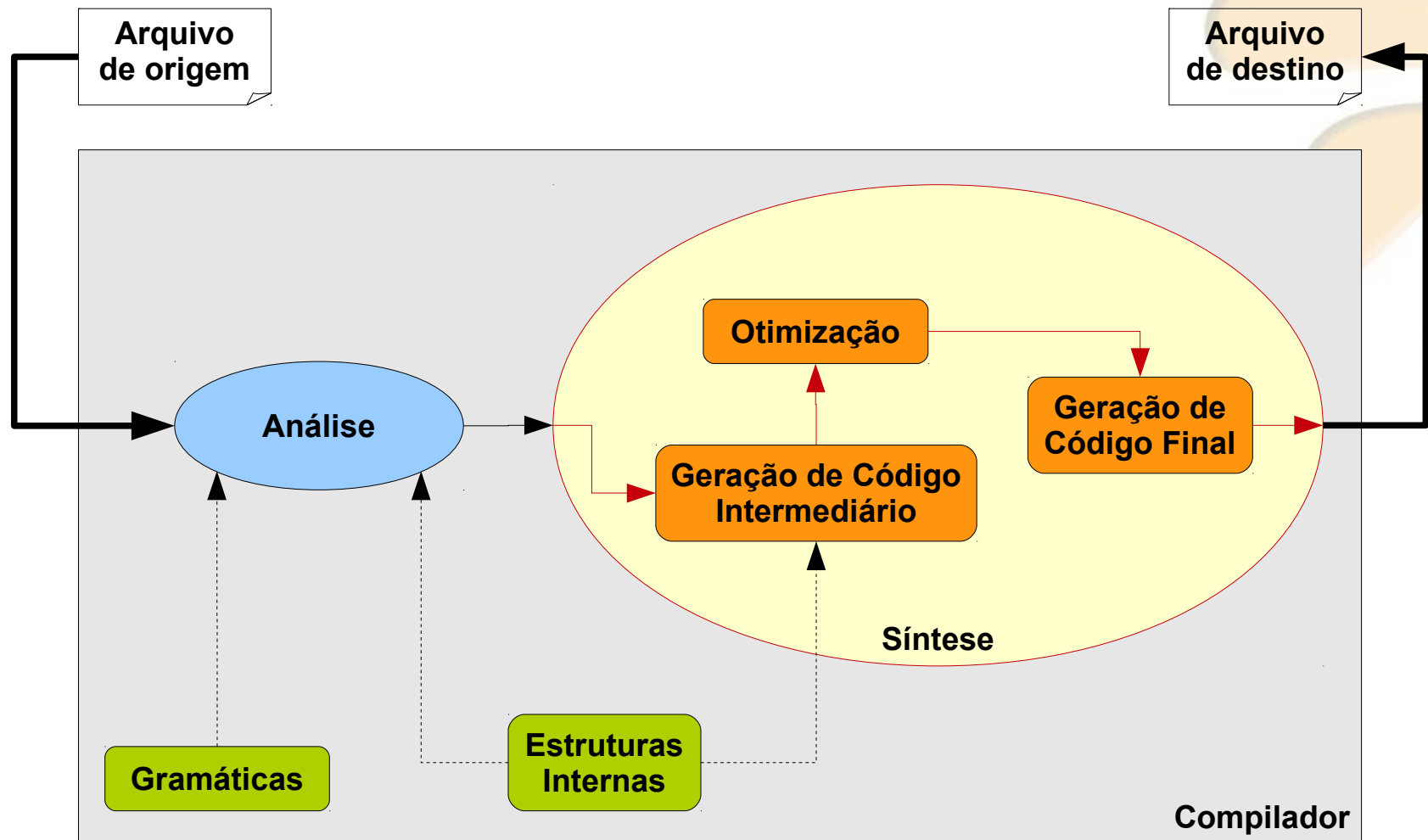


Compiladores

Geração de Código Intermediário

Cristiano Lehrer, M.Sc.

Atividades do Compilador



Introdução

- A geração de código intermediário é a transformação da árvore de derivação em um segmento de código.
- Esse código pode, eventualmente, ser o código objeto final, mas, na maioria das vezes, constitui-se num código intermediário.
- A grande diferença entre o código intermediário e o código objeto final é que o intermediário não especifica detalhes da máquina alvo, tais como quais registradores serão usados, quais endereços de memória serão referenciados, entre outros detalhes físicos da máquina alvo.

Vantagens e Desvantagens

- Vantagens da geração de código intermediário:
 - Possibilita a otimização do código intermediário, de modo a obter-se o código objeto final mais eficiente.
 - Simplifica a implementação do compilador, resolvendo, gradativamente, as dificuldades da passagem de código fonte para objeto (alto nível para baixo nível), já que o código fonte pode ser visto como um texto condensado que explode em inúmeras instruções elementares de baixo nível.
 - Possibilita a tradução do código intermediário para diversas máquinas.
- Desvantagens da geração de código intermediário:
 - O compilador requer um passo a mais, pois a tradução direta do código fonte para objeto leva a uma compilação mais rápida.

Linguagens Intermediárias

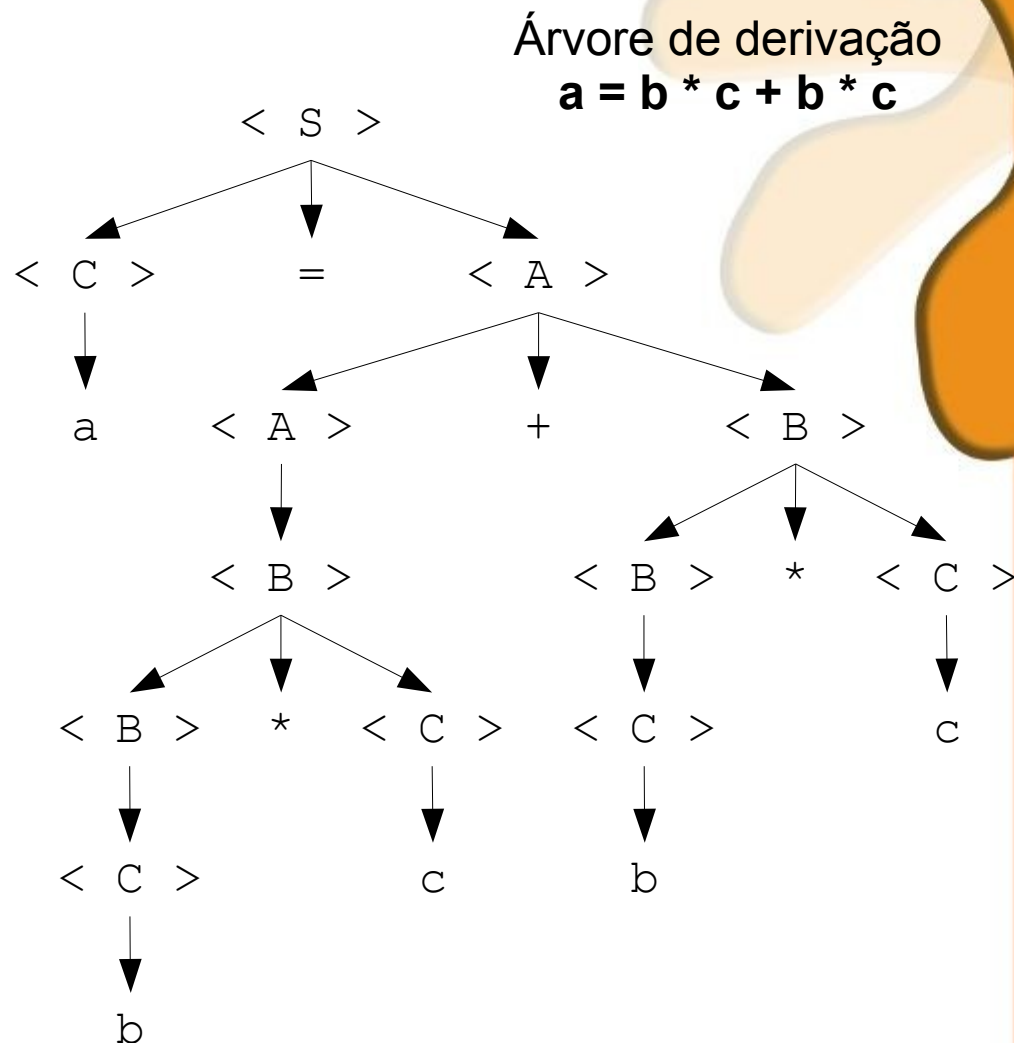
- Os vários tipos de código intermediário fazem parte de uma das seguintes categorias:
 - Representações gráficas:
 - Árvore e grafo de sintaxe.
 - Notação pós-fixada ou pré-fixada.
 - Código de três endereços:
 - Quádruplas e triplas.

Árvore e Grafo de Sintaxe (1/4)

- Uma **árvore de sintaxe** é uma forma condensada de árvore de derivação na qual somente os operandos da linguagem aparecem como folhas:
 - Os operandos constituem nós interiores da árvore.
 - Outra simplificação da árvore de sintaxe é que cadeias de produções simples (por exemplo, $A \rightarrow B$, $B \rightarrow C$) são eliminadas.
- Um **grafo de sintaxe**, além de incluir as simplificações da árvore de sintaxe, faz a fatoração das subexpressões comuns, eliminando-as.

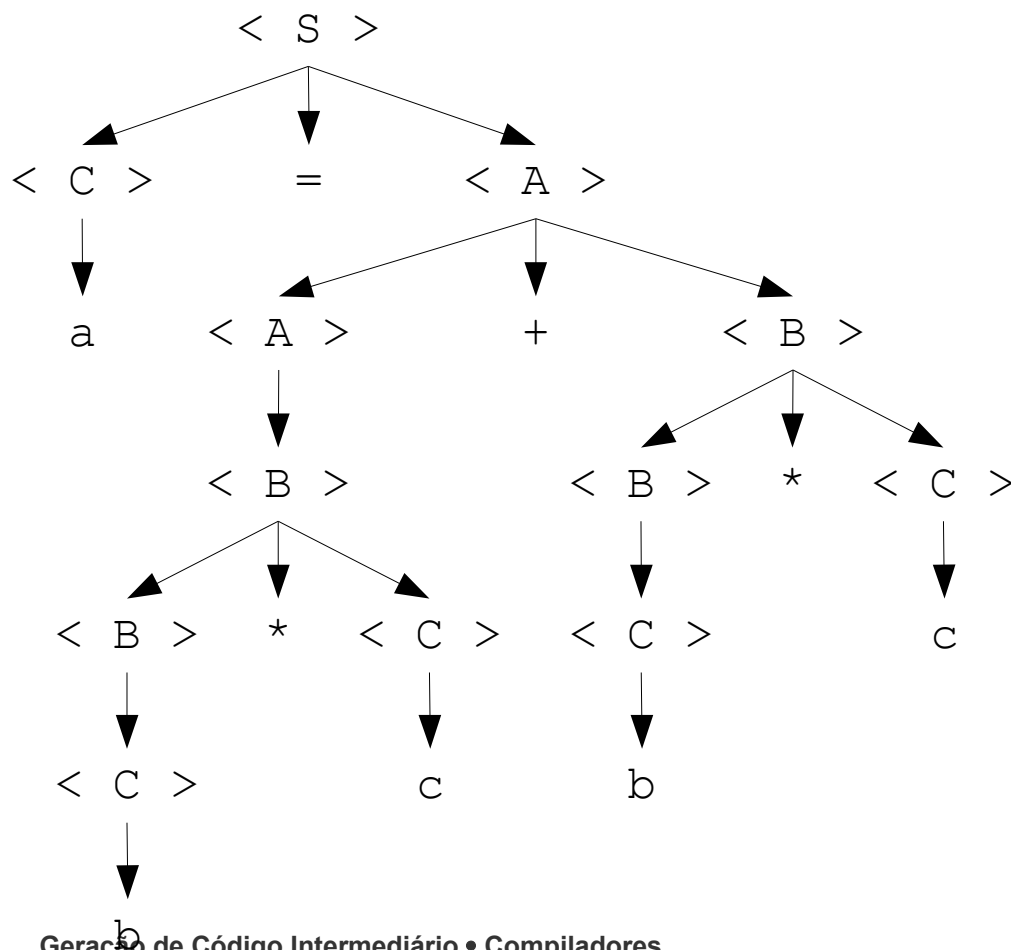
Árvore e Grafo de Sintaxe (2/4)

$G = (\{S, A, B, C\},$
 $\{a, b, c, +, *, =\}, P, S)$
 $P = \{ \langle S \rangle \rightarrow \langle C \rangle = \langle A \rangle$
 $\langle A \rangle \rightarrow \langle A \rangle + \langle B \rangle$
 $\quad \quad \quad | \langle B \rangle$
 $\langle B \rangle \rightarrow \langle B \rangle * \langle C \rangle$
 $\quad \quad \quad | \langle C \rangle$
 $\langle C \rangle \rightarrow a$
 $\quad \quad \quad | b$
 $\quad \quad \quad | c \quad \}$

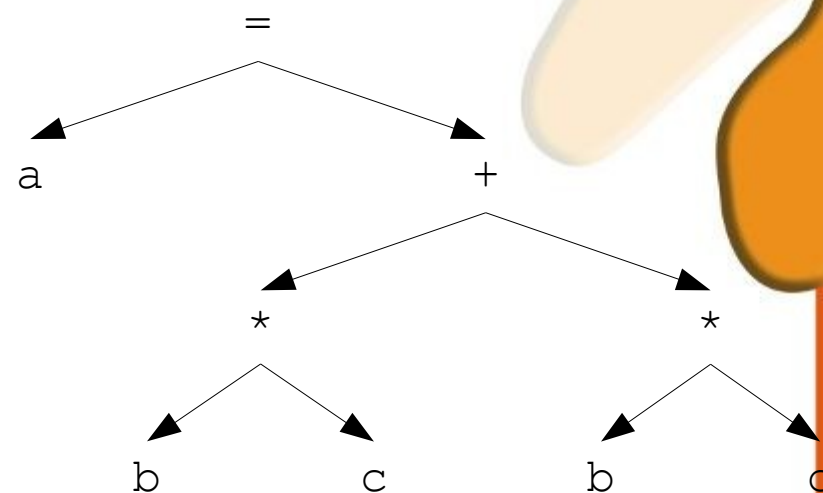


Árvore e Grafo de Sintaxe (3/4)

Árvore de derivação
 $a = b * c + b * c$

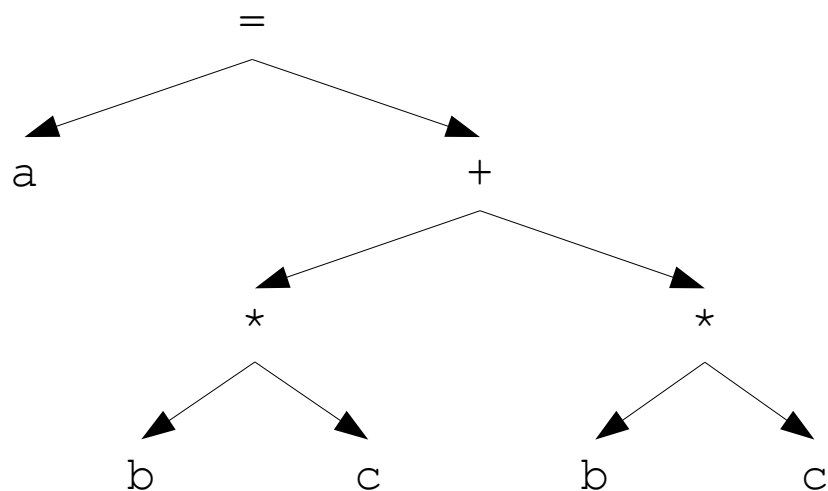


Árvore de sintaxe
 $a = b * c + b * c$

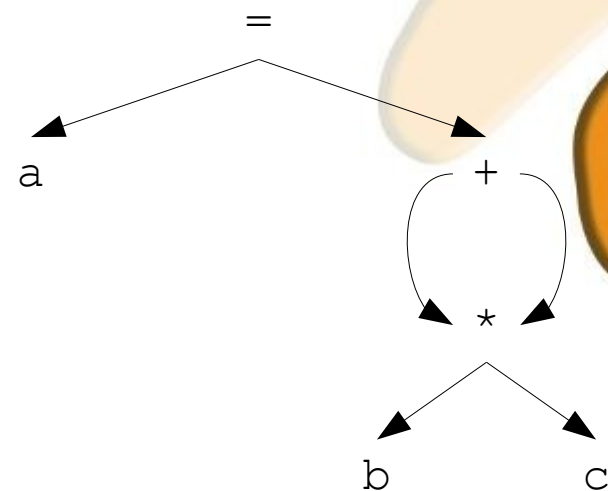


Árvore e Grafo de Sintaxe (4/4)

Árvore de sintaxe
 $a = b * c + b * c$



Grafo de sintaxe
 $a = b * c + b * c$



Notações Pós-fixada e Pré-fixada (1/4)

- Se E_1 e E_2 são expressões pós-fixadas e q é um operador binário, então, $E_1 E_2 q$ é a representação pós-fixada para a expressão $E_1 q E_2$.
- Se E_1 e E_2 são expressões pré-fixadas e q é um operador binário, então, $q E_1 E_2$ é a representação pré-fixada para a expressão $E_1 q E_2$.
- Notações pós e pré-fixadas podem ser generalizadas para operadores n-ários.

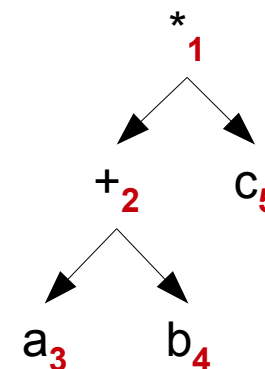
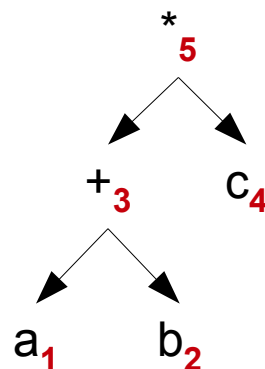
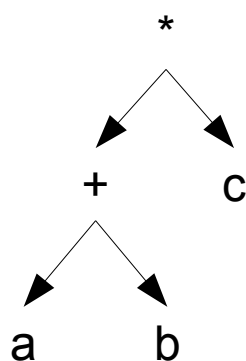
Notações Pós-fixada e Pré-fixada (2/4)

Notação		
Infixada	Pós-fixada	Pré-fixada
$(a + b) * c$	$ab+c^*$	$*+abc$
$a * (b + c)$	$abc+^*$	$*a+bc$
$a + b * c$	abc^*+	$+a^*bc$

$(a + b) * c$

Pós-ordem

Pré-ordem



Notações Pós-fixada e Pré-fixada (3/4)

- Para a avaliação de expressões pós-fixadas, pode-se utilizar uma pilha e um processo que age do seguinte modo:
 - Lê a expressão da esquerda para a direita, empilhando cada operando até encontrar um operador.
 - Encontrando um operador n-ário, aplica o operador aos n operandos do topo da pilha.
- Processamento semelhante pode ser aplicado para a avaliação de expressões pré-fixadas:
 - Nesse caso, a expressão é lida da direita para a esquerda.

Notações Pós-fixada e Pré-fixada (4/4)

Execução de expressões pós-fixadas

$ab+c^*$

$b+c^*$

$+c^*$

c^*

$*$

ϵ

ϵ

a

b
a

x

c
x

y

Execução de expressões pré-fixadas

$*+abc$

$*+ab$

$*+a$

$*+$

$*$

ϵ

ϵ

c

b
c

a
b
c

x
c

y

Código de Três Endereços (1/4)

- No código intermediário de três endereços, cada instrução faz referência, no máximo, a três variáveis (endereços de memória).
- As instruções dessa linguagem são as seguintes:

`A := B op C`

`A := op B`

`A := B`

`goto L`

`if A oprel B goto L`

- Onde `A`, `B` e `C` representam endereços de variáveis.
- `op` representa operador (binário ou unário).
- `oprel` representa operador relacional.
- `L` representa o rótulo de uma instrução intermediária.

Código de Três Endereços (2/4)

- Exemplo $A := X + Y * Z$

$T1 := Y * Z$

$T2 := X + T1$

$A := T2$

- onde $T1$ e $T2$ são variáveis temporárias.
- Um código de três endereços pode ser implementado através de quádruplas ou triplas.

Código de Três Endereços (3/4)

- As quádruplas são constituídas de quatro campos:
 - Um operador, dois operandos e um resultado.
- Exemplo $A := B * (-C + D)$

	oper	arg1	arg2	result
(0)	-u	C		T1
(1)	+	T1	D	T2
(2)	*	B	T2	T3
(3)	:=	T3		A

Código de Três Endereços (4/4)

- As triplas são formadas por:
 - Um operador e dois operandos.
- Essa representação utiliza ponteiros para a própria estrutura, evitando a nomeação explícita de temporários.
- Exemplo $A := B * (-C + D)$

	oper	arg1	arg2
(0)	-u	C	
(1)	+	(0)	D
(2)	*	B	(1)
(3)	:=	A	(2)