

# Seleção de Instruções

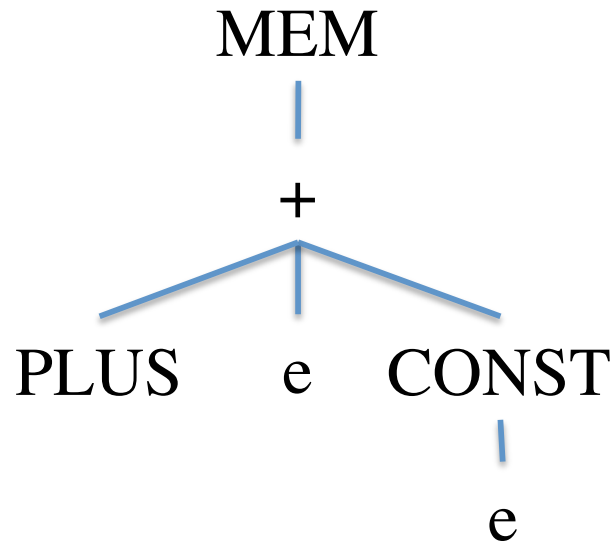
Guido Araújo

[guido@ic.unicamp.br](mailto:guido@ic.unicamp.br)

# Introdução

- Vamos supor que a IR é representada com árvores
- Uma árvore da IR expressa uma operação “simples” em cada nó
  - Acesso à memória
  - Operador Binário
  - Salto condicional
- Instruções da máquina podem realizar uma ou mais dessas operações

# Introdução



- Que instrução seria essa?
- Encontrar o conjunto de instruções de máquinas que implementa uma dada árvore da IR é o objetivo da ***Seleção de Instruções***

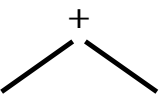
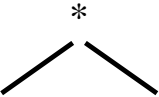
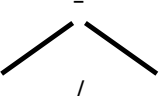
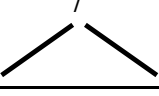

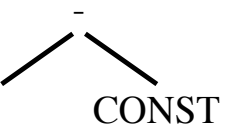
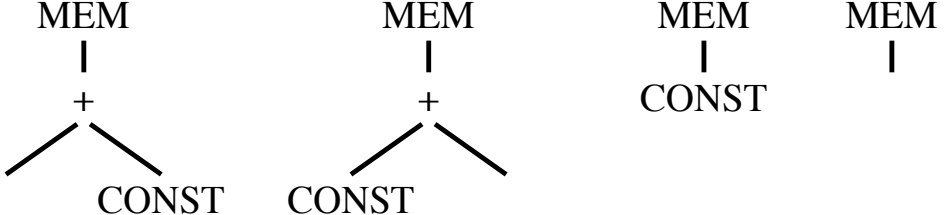
# Padrões de Árvores

- Podemos expressar instruções da máquina como um fragmento da árvore da IR.
  - Padrões de árvore.
- Seleção de instruções:
  - Cubra a árvore da IR com o “menor” número de padrões existentes para a máquina alvo.
- Exemplo:
  - Arquitetura *Jouette*

# Arquitetura *Joulette*

- Arquitetura Load/Store
- Qualquer instrução pode acessar qualquer registrador
- Registradores podem armazenar dado ou endereço
- $r_0$  sempre contém zero.
- Cada instrução gasta 1 ciclo
- Executa apenas uma instrução por ciclo

# Padrões de Árvore - *Jouette*

<i>Name</i>	<i>Effect</i>	<i>Trees</i>
—	$r_i$	TEMP
ADD	$r_i \leftarrow r_j + r_k$	
MUL	$r_i \leftarrow r_j \times r_k$	
SUB	$r_i \leftarrow r_j - r_k$	
DIV	$r_i \leftarrow r_j / r_k$	
ADDI	$r_i \leftarrow r_j + c$	
SUBI	$r_i \leftarrow r_j - c$	
LOAD	$r_i \leftarrow M[r_j + c]$	

# Padrões de Árvore- *Jouette*

<i>Name</i>	<i>Effect</i>	<i>Trees</i>
STORE	$M[r_j + c] \leftarrow r_i$	
MOVEM	$M[r_j] \leftarrow M[r_i]$	

# Seleção de Instruções

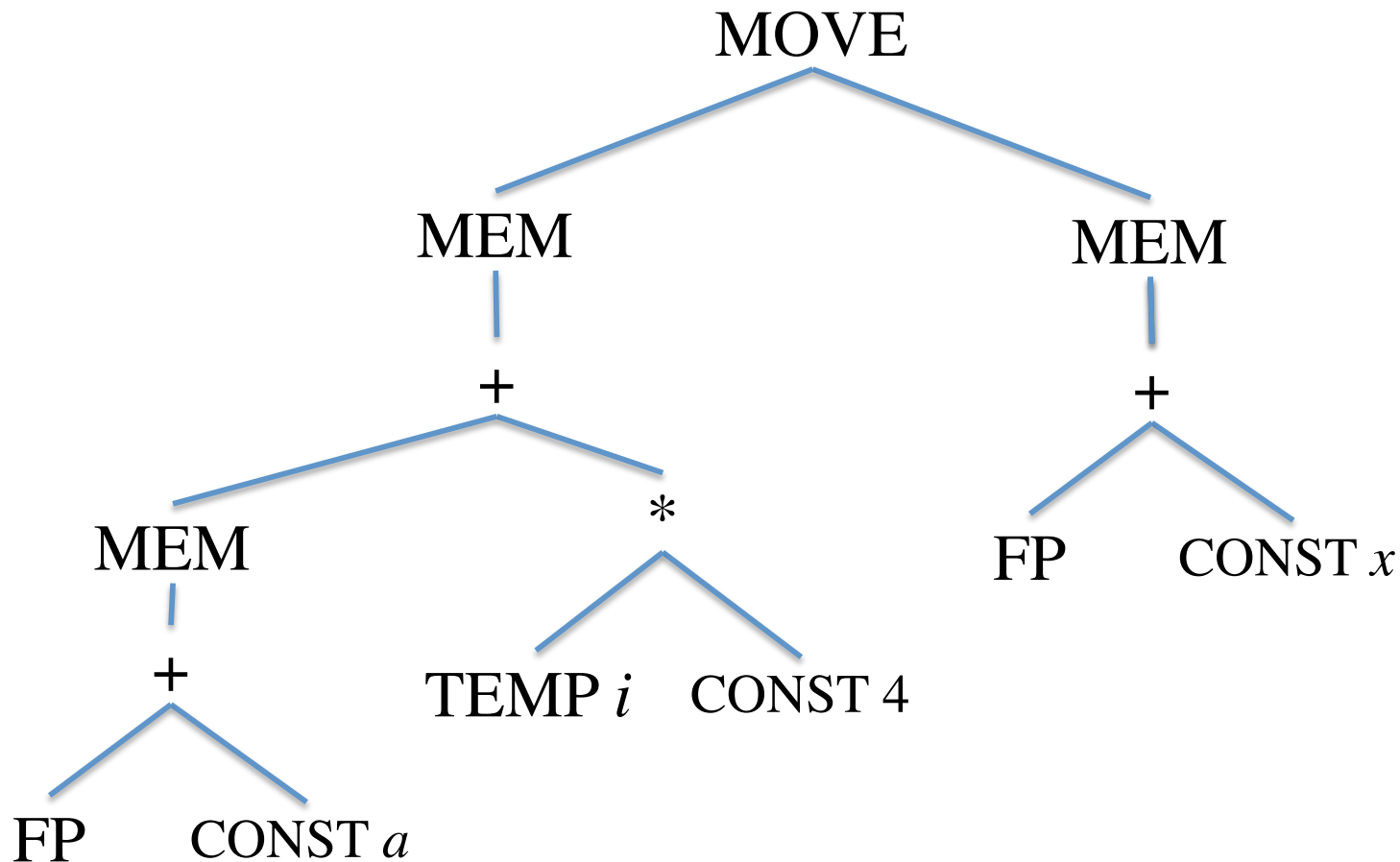
- Objetivo é cobrir a árvore com padrões, sem sobreposição entre padrões.
- Exemplo:
  - $a[i] := x$
  - Supondo que  $i$  está em um registrador e as variáveis  $a$  e  $x$  estão na pilha (indexada por FP)



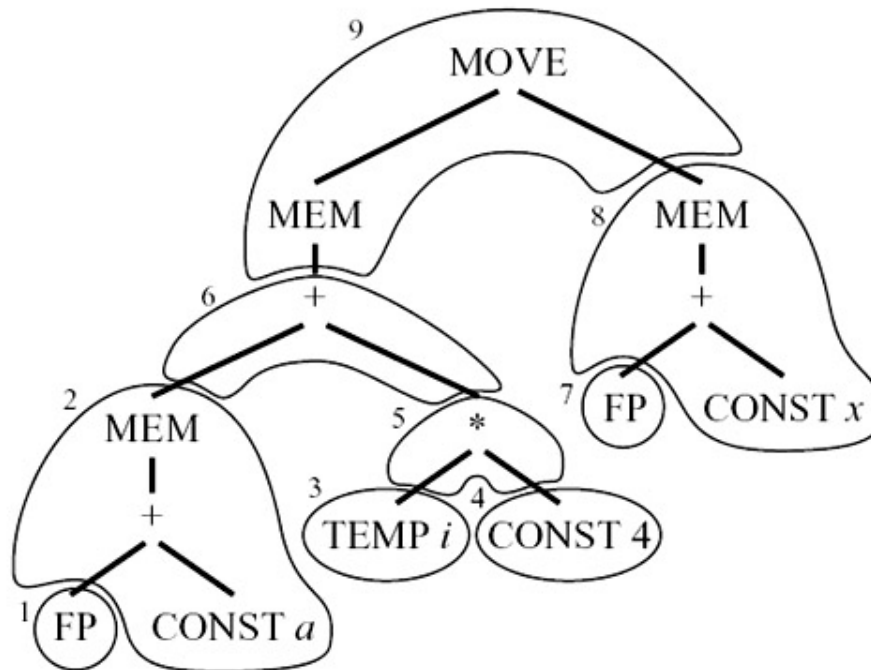
# Seleção de Instruções

$a[i] := x$

- Supondo que  $i$  está em um registrador e as variáveis  $a$  e  $x$  estão na pilha (indexada por FP)



# Seleção de Instruções

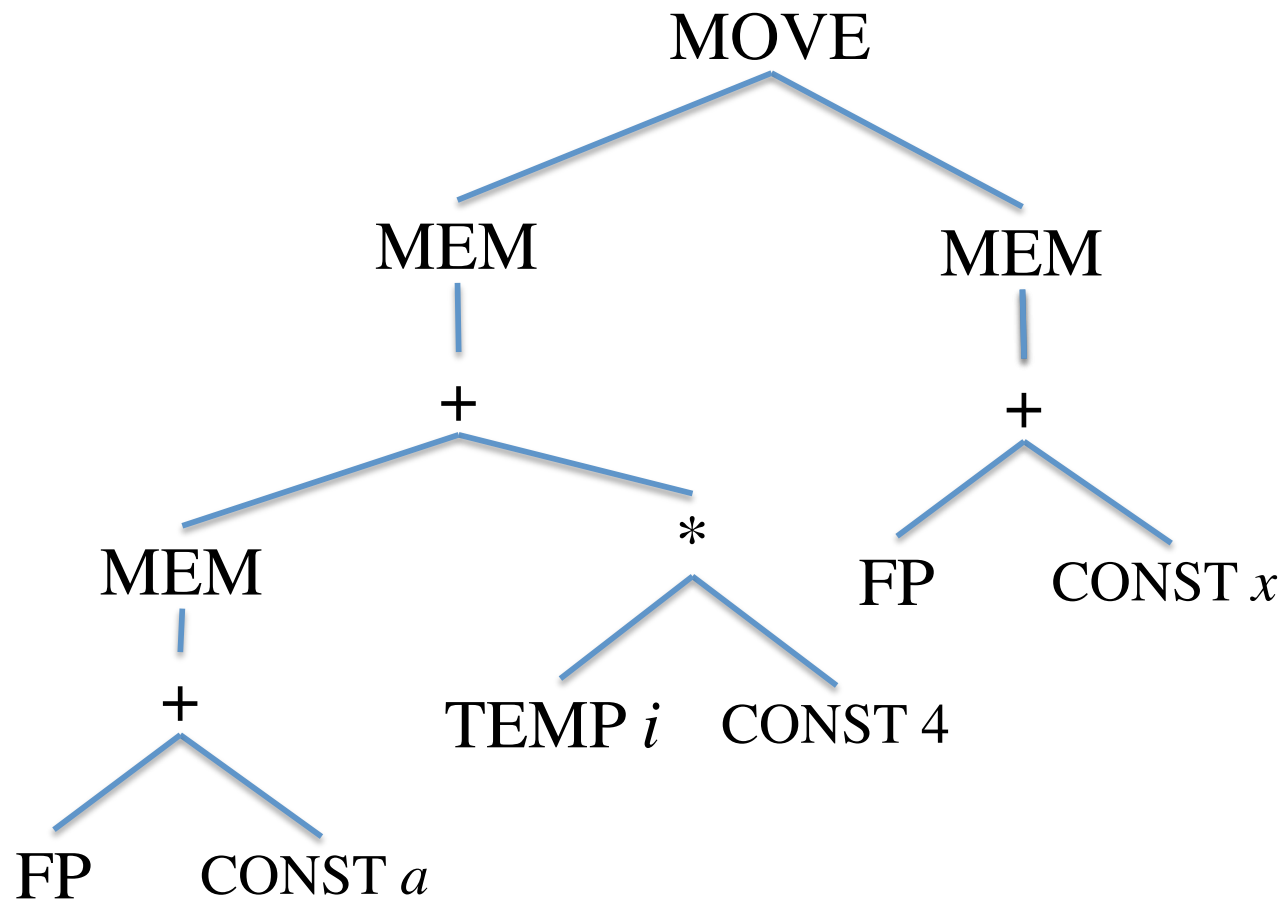


2	LOAD	$r_1 \leftarrow M[\mathbf{fp} + a]$
4	ADDI	$r_2 \leftarrow r_0 + 4$
5	MUL	$r_2 \leftarrow r_i \times r_2$
6	ADD	$r_1 \leftarrow r_1 + r_2$
8	LOAD	$r_2 \leftarrow M[\mathbf{fp} + x]$
9	STORE	$M[r_1 + 0] \leftarrow r_2$

- Uma possível solução
- A solução não é única?

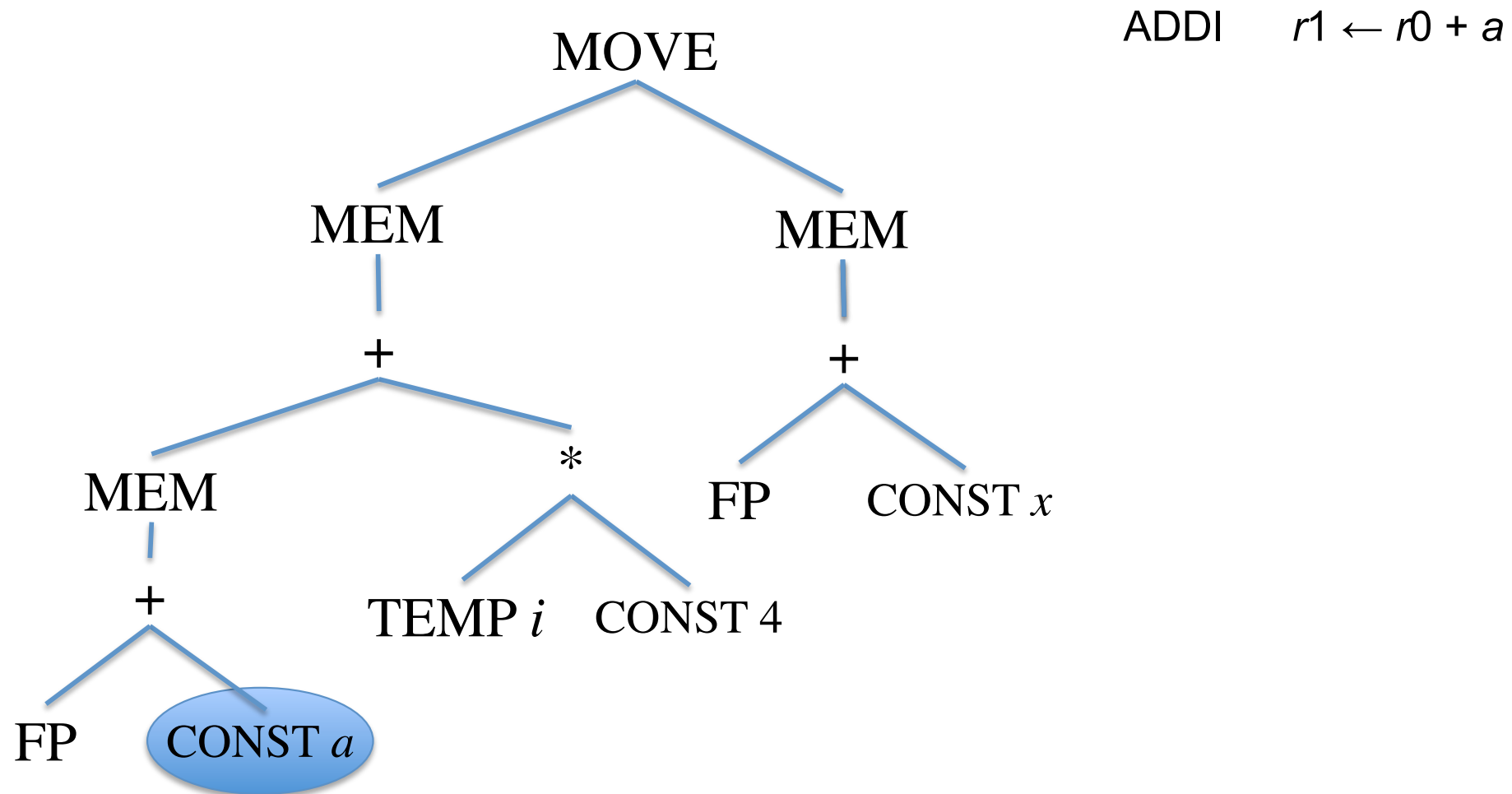
# Seleção de Instruções

- E se cobrirmos a árvore com padrões simples (de um nó apenas)?



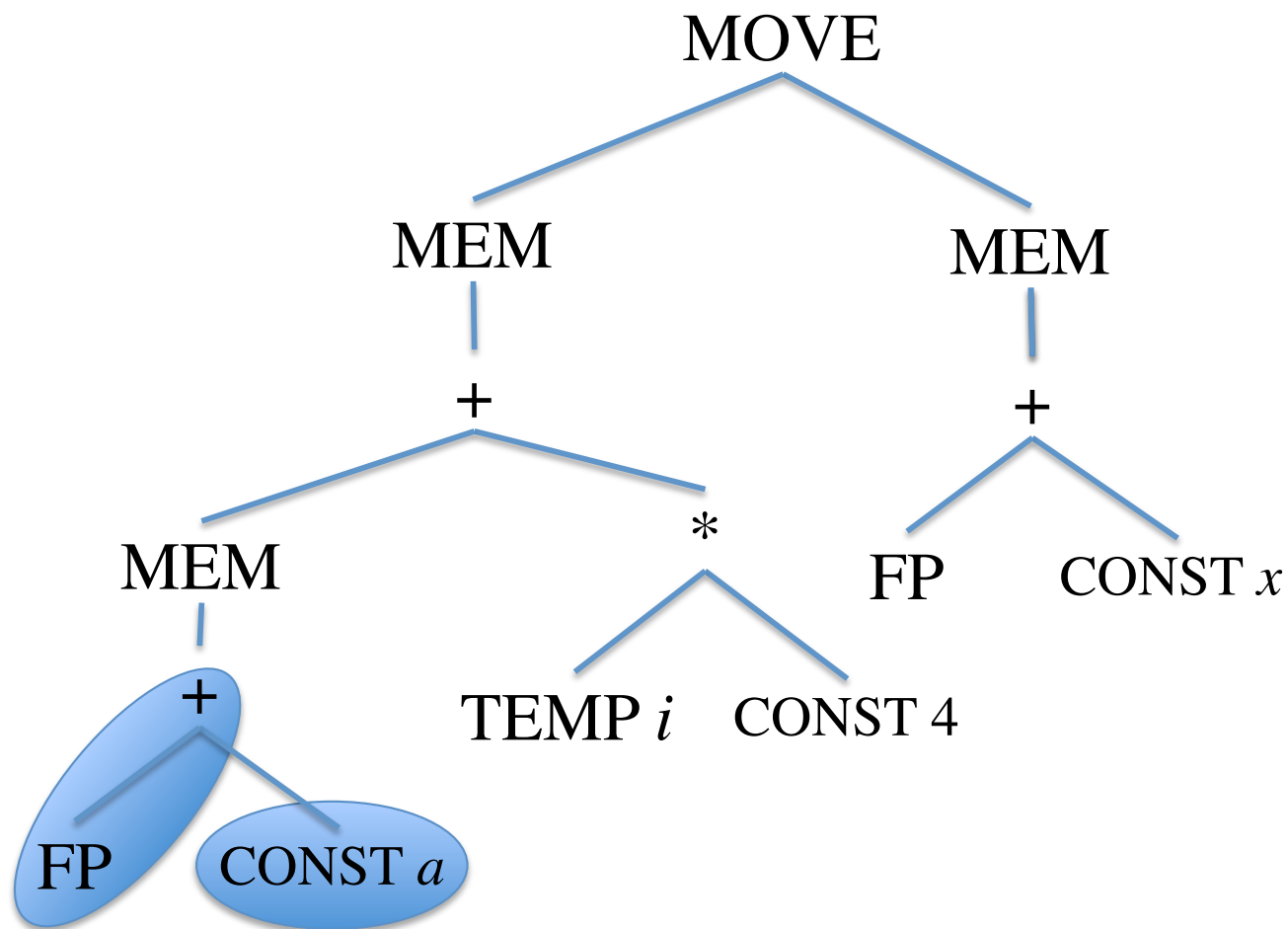
# Seleção de Instruções

- E se cobrirmos a árvore com padrões simples (de um nó apenas)?



# Seleção de Instruções

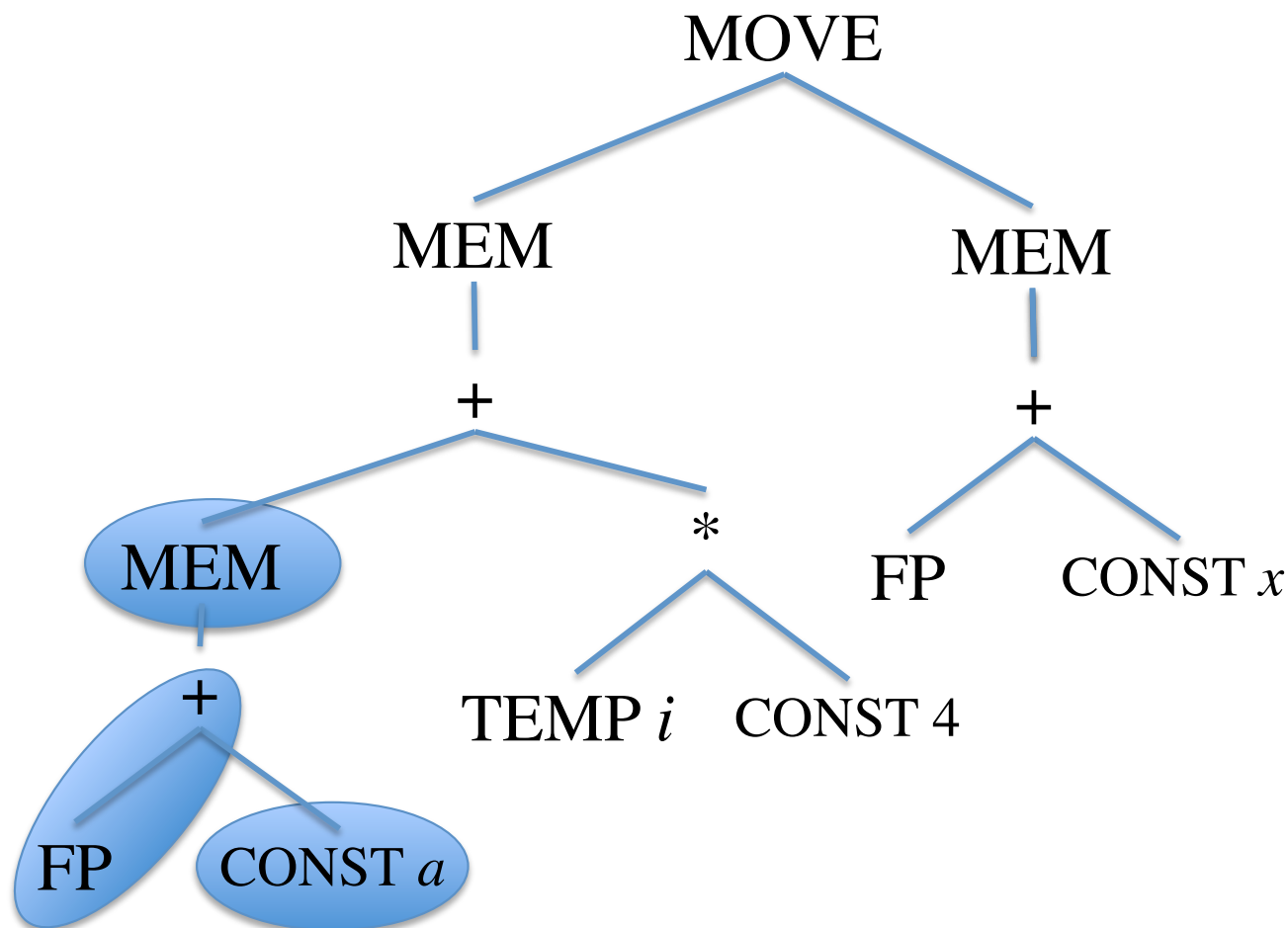
- E se cobrirmos a árvore com padrões simples (de um nó apenas)?



ADDI  $r1 \leftarrow r0 + a$   
ADD  $r1 \leftarrow \mathbf{fp} + r1$

# Seleção de Instruções

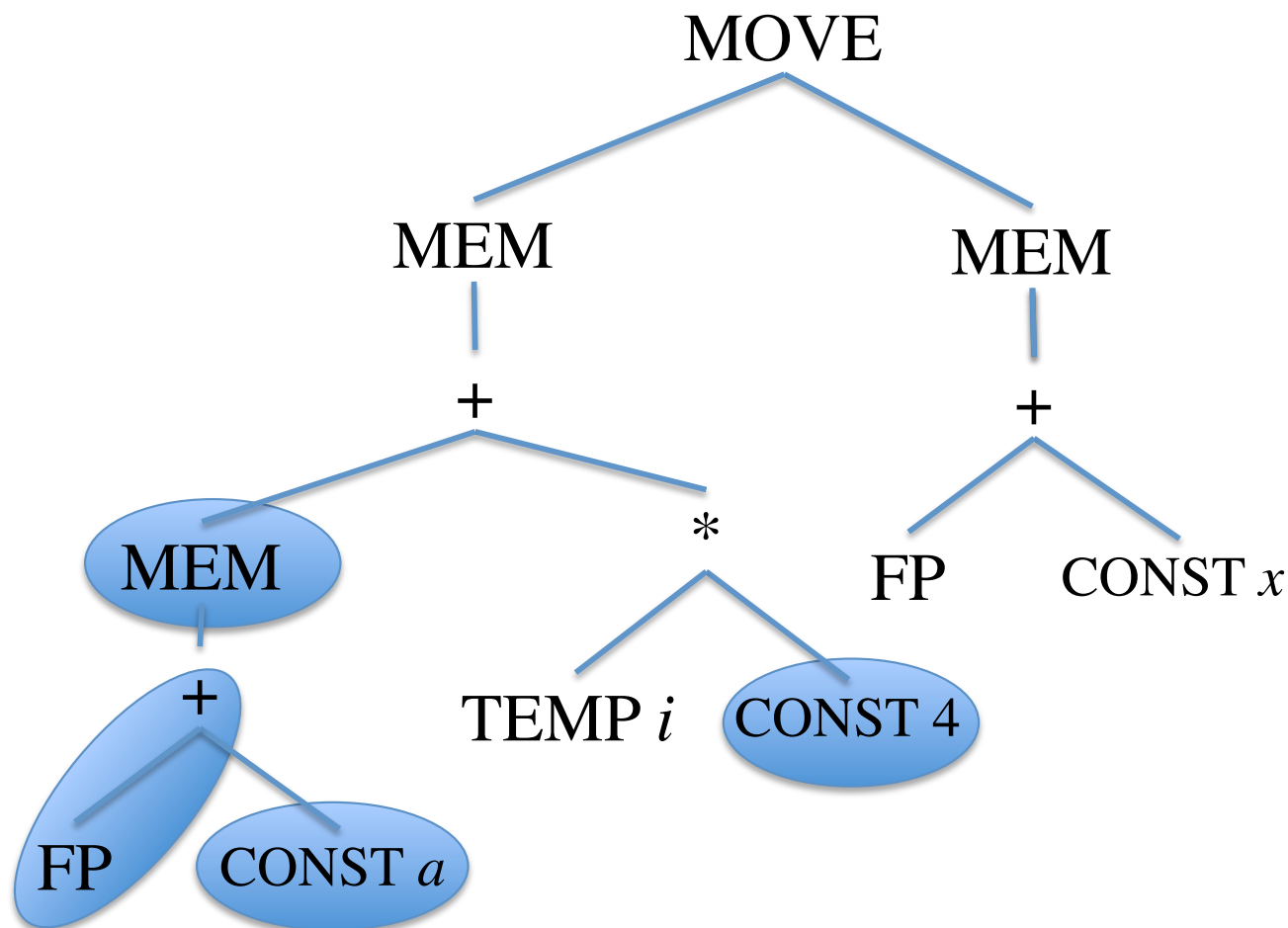
- E se cobrirmos a árvore com padrões simples (de um nó apenas)?



ADDI     $r1 \leftarrow r0 + a$   
ADD     $r1 \leftarrow \mathbf{fp} + r1$   
LOAD    $r1 \leftarrow M[r1 + 0]$

# Seleção de Instruções

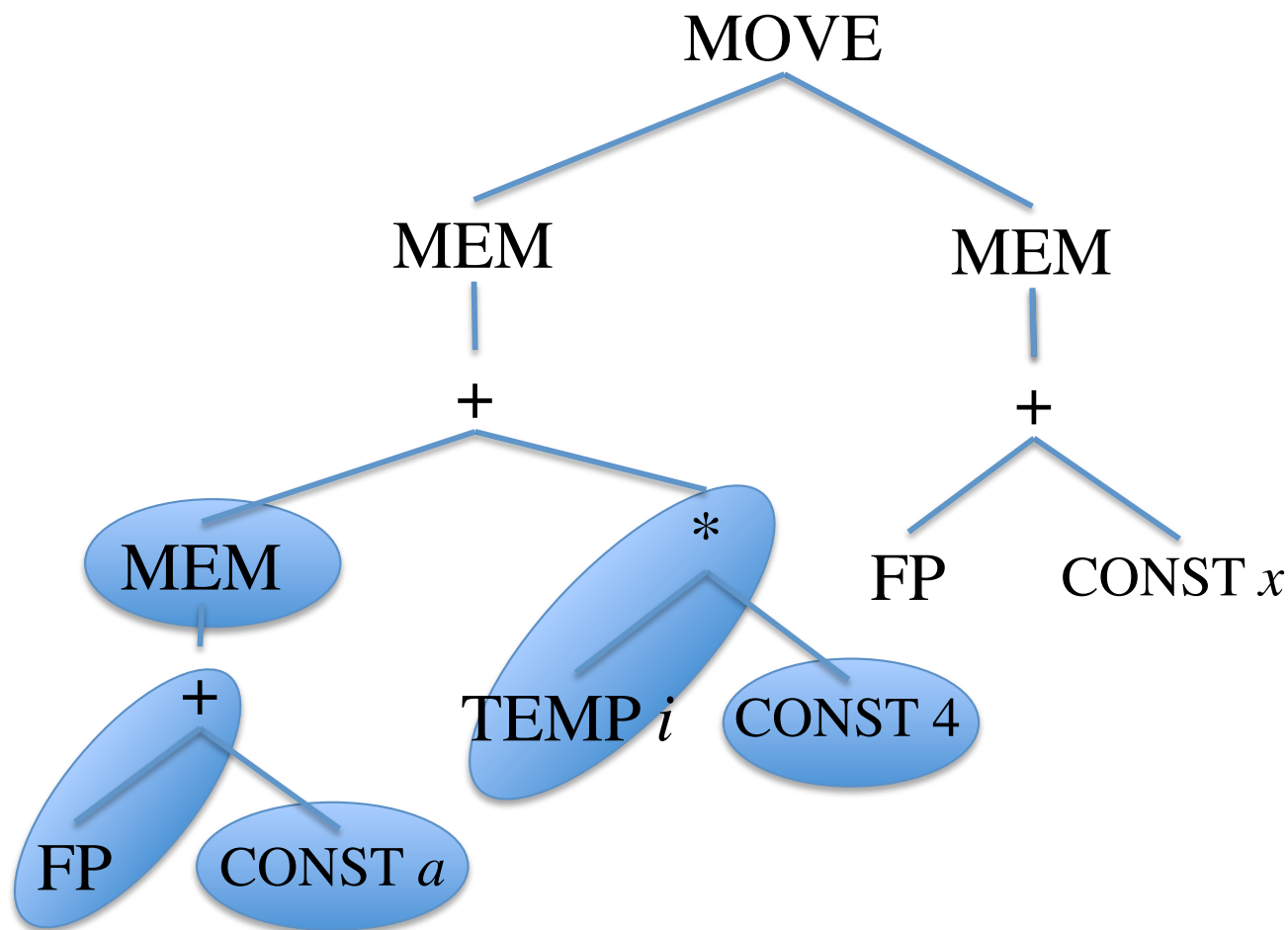
- E se cobrirmos a árvore com padrões simples (de um nó apenas)?



ADDI  $r1 \leftarrow r0 + a$   
ADD  $r1 \leftarrow \mathbf{fp} + r1$   
LOAD  $r1 \leftarrow M[r1 + 0]$   
ADDI  $r2 \leftarrow r0 + 4$

# Seleção de Instruções

- E se cobrirmos a árvore com padrões simples (de um nó apenas)?

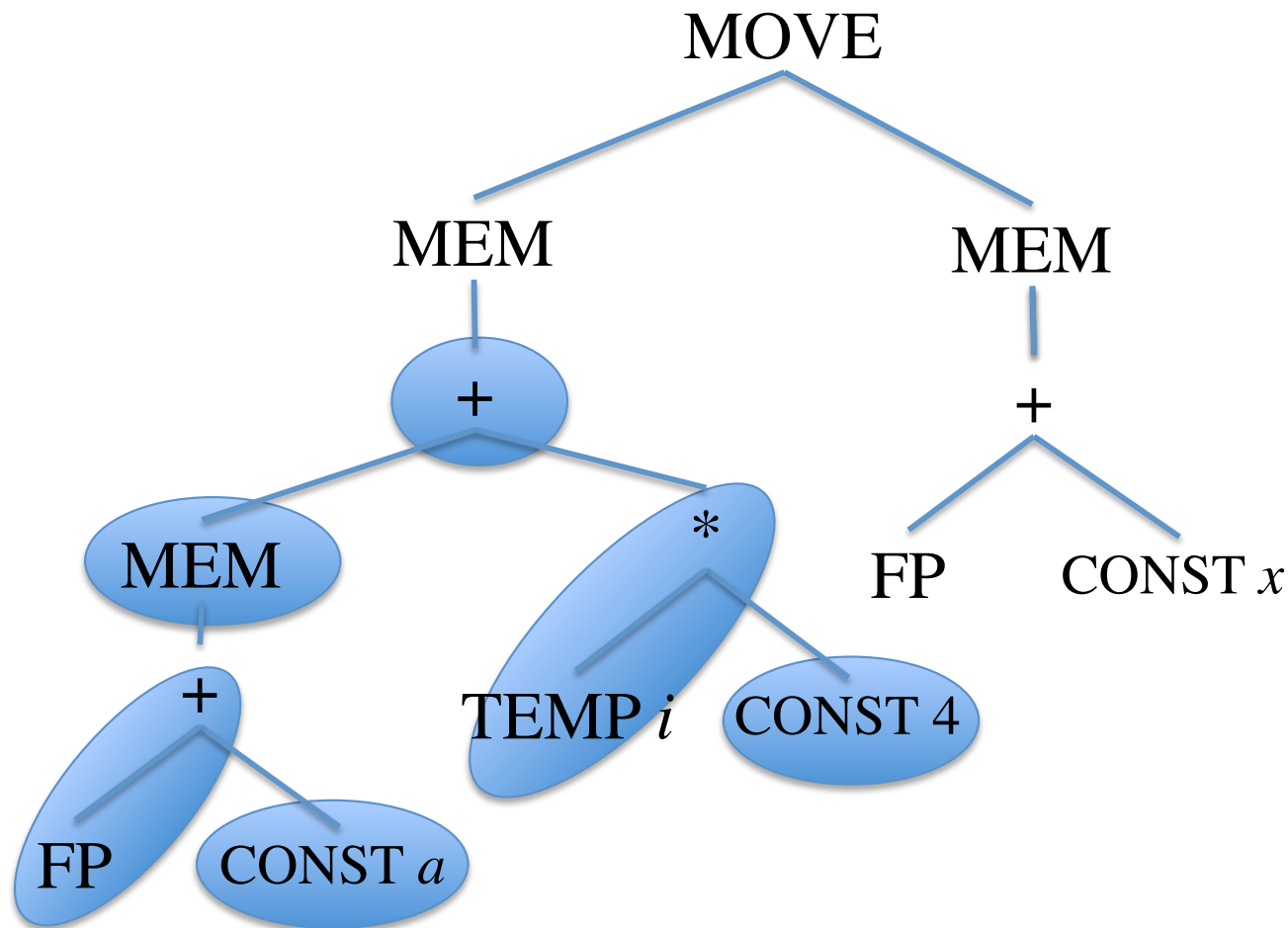


ADDI	$r1 \leftarrow r0 + a$
ADD	$r1 \leftarrow \mathbf{fp} + r1$
LOAD	$r1 \leftarrow M[r1 + 0]$
ADDI	$r2 \leftarrow r0 + 4$
MUL	$r2 \leftarrow r1 \times r2$



# Seleção de Instruções

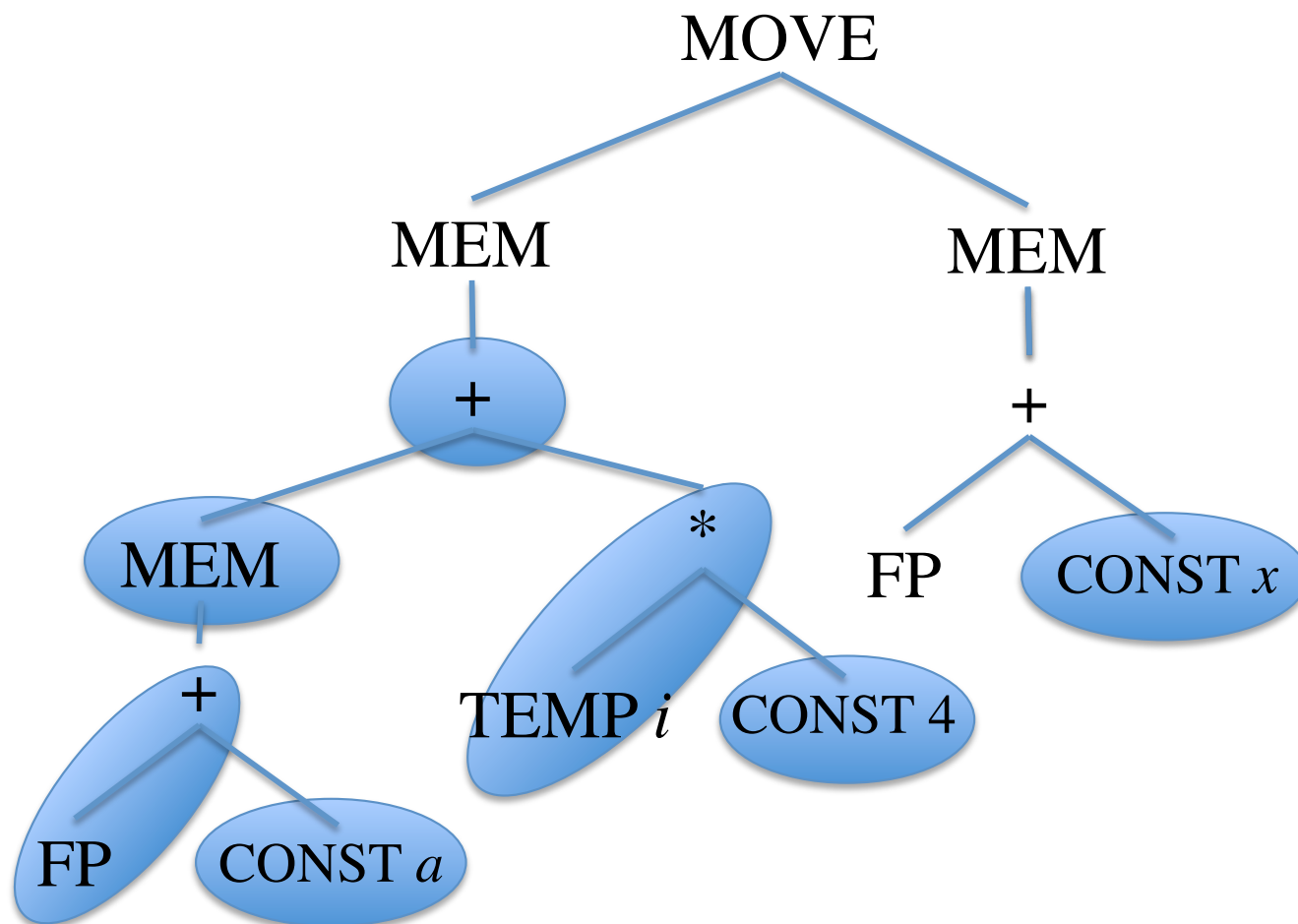
- E se cobrirmos a árvore com padrões simples (de um nó apenas)?



ADDI	$r1 \leftarrow r0 + a$
ADD	$r1 \leftarrow \mathbf{fp} + r1$
LOAD	$r1 \leftarrow M[r1 + 0]$
ADDI	$r2 \leftarrow r0 + 4$
MUL	$r2 \leftarrow r1 \times r2$
ADD	$r1 \leftarrow r1 + r2$

# Seleção de Instruções

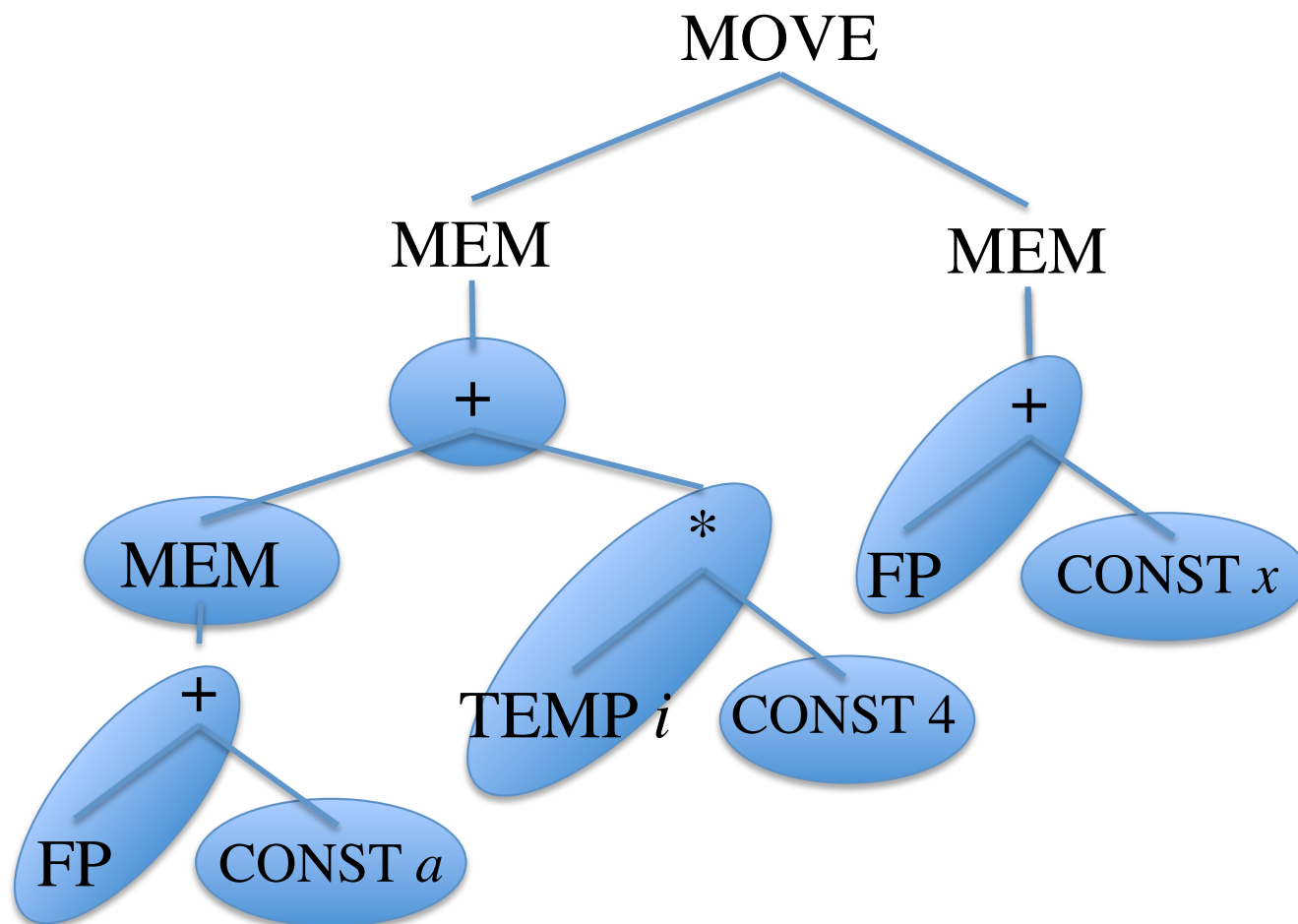
- E se cobrirmos a árvore com padrões simples (de um nó apenas)?



ADDI	$r1 \leftarrow r0 + a$
ADD	$r1 \leftarrow \mathbf{fp} + r1$
LOAD	$r1 \leftarrow M[r1 + 0]$
ADDI	$r2 \leftarrow r0 + 4$
MUL	$r2 \leftarrow r1 \times r2$
ADD	$r1 \leftarrow r1 + r2$
ADDI	$r2 \leftarrow r0 + x$

# Seleção de Instruções

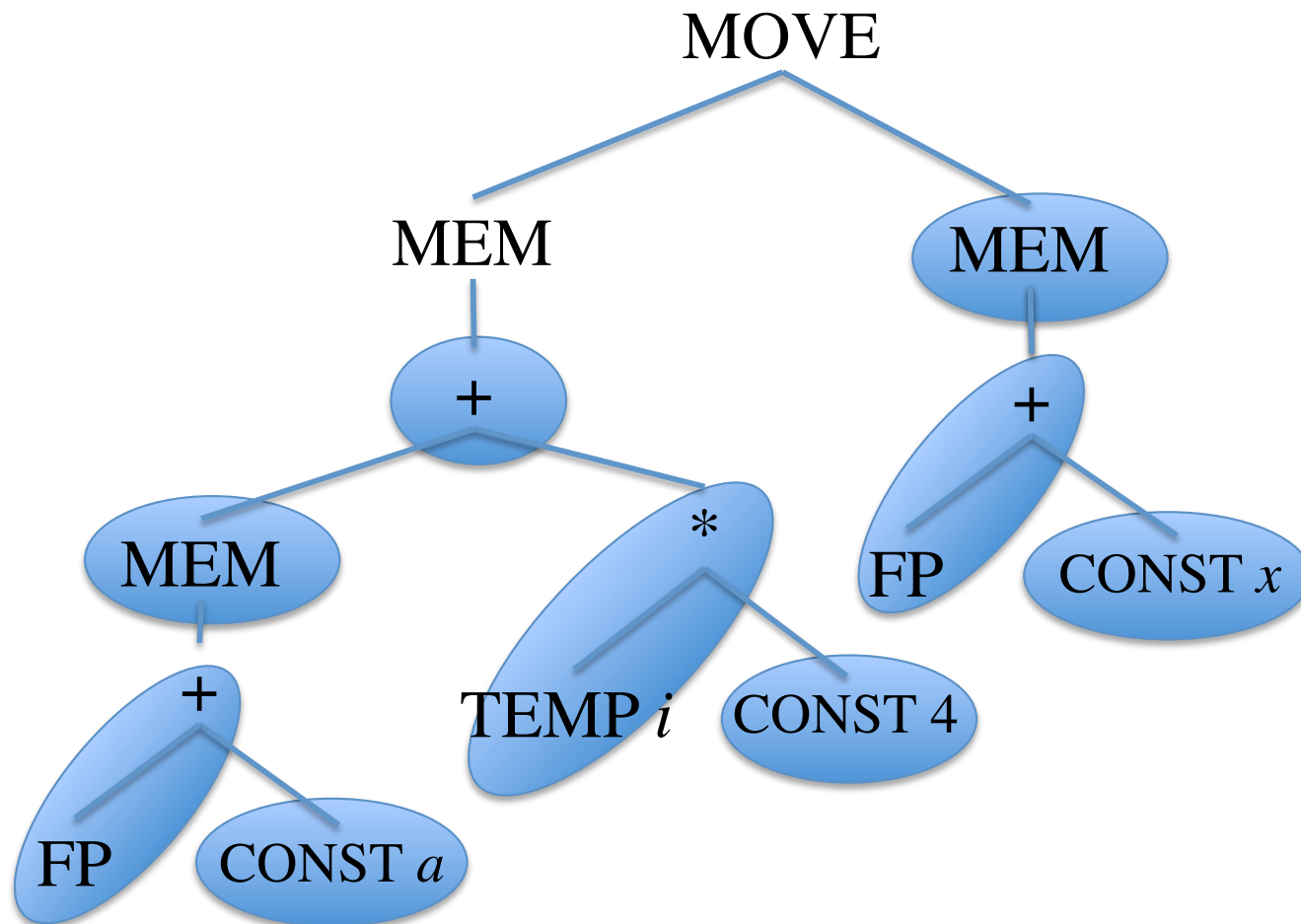
- E se cobrirmos a árvore com padrões simples (de um nó apenas)?



ADDI	$r1 \leftarrow r0 + a$
ADD	$r1 \leftarrow \mathbf{fp} + r1$
LOAD	$r1 \leftarrow M[r1 + 0]$
ADDI	$r2 \leftarrow r0 + 4$
MUL	$r2 \leftarrow r1 \times r2$
ADD	$r1 \leftarrow r1 + r2$
ADDI	$r2 \leftarrow r0 + x$
ADD	$r2 \leftarrow \mathbf{fp} + r2$

# Seleção de Instruções

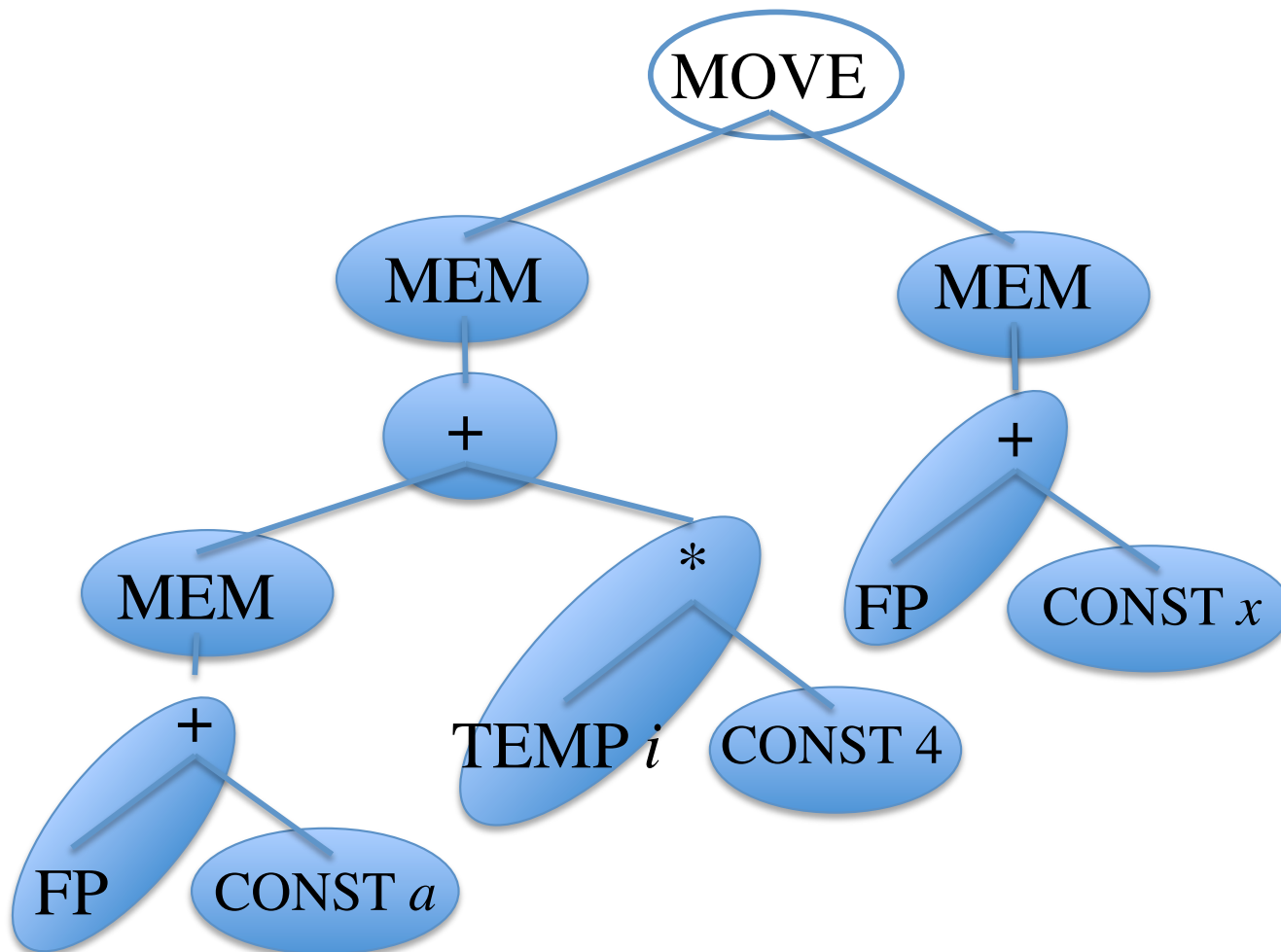
- E se cobrirmos a árvore com padrões simples (de um nó apenas)?



ADDI	$r1 \leftarrow r0 + a$
ADD	$r1 \leftarrow \mathbf{fp} + r1$
LOAD	$r1 \leftarrow M[r1 + 0]$
ADDI	$r2 \leftarrow r0 + 4$
MUL	$r2 \leftarrow r1 \times r2$
ADD	$r1 \leftarrow r1 + r2$
ADDI	$r2 \leftarrow r0 + x$
ADD	$r2 \leftarrow \mathbf{fp} + r2$
LOAD	$r2 \leftarrow M[r2 + 0]$

# Seleção de Instruções

- E se cobrirmos a árvore com padrões simples (de um nó apenas)?



ADDI	$r1 \leftarrow r0 + a$
ADD	$r1 \leftarrow \mathbf{fp} + r1$
LOAD	$r1 \leftarrow M[r1 + 0]$
ADDI	$r2 \leftarrow r0 + 4$
MUL	$r2 \leftarrow r1 \times r2$
ADD	$r1 \leftarrow r1 + r2$
ADDI	$r2 \leftarrow r0 + x$
ADD	$r2 \leftarrow \mathbf{fp} + r2$
LOAD	$r2 \leftarrow M[r2 + 0]$
STORE	$M[r1 + 0] \leftarrow r2$

# *Optimal e Optimum*

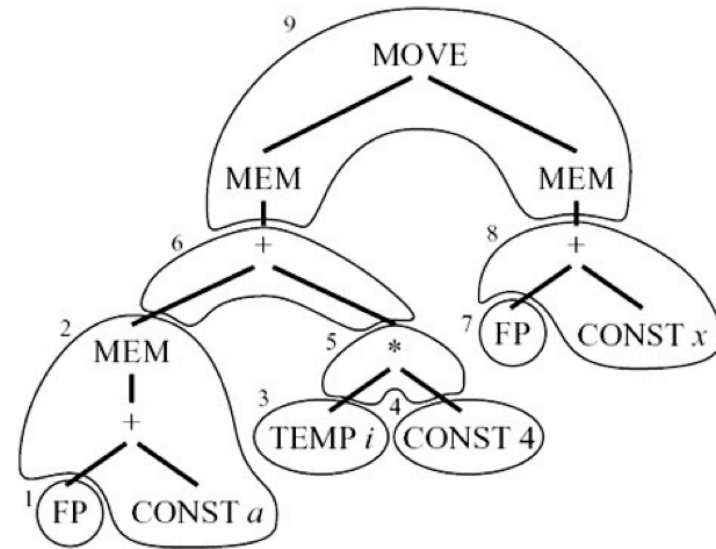
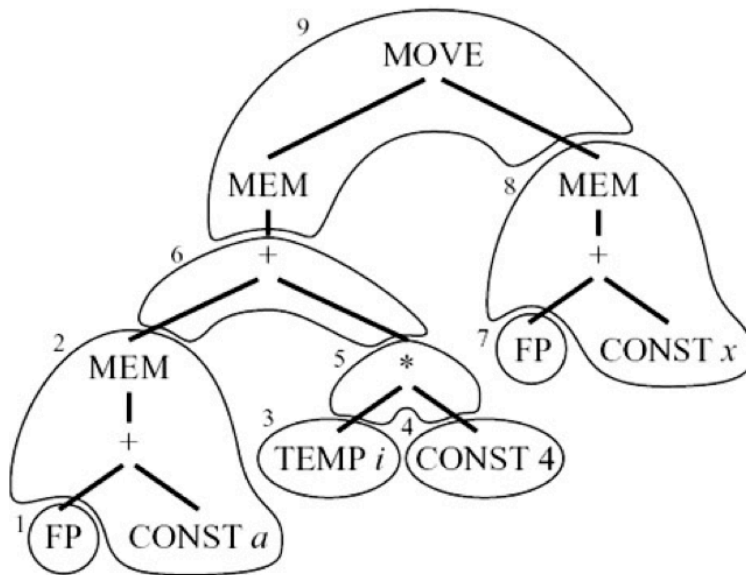
- Queremos a cobertura que nos traga o menor custo
  - Normalmente a menor
  - Caso as instruções tenham latências diferentes
    - A de menor tempo total
- Cada instrução recebe um custo
  - A melhor cobertura da árvore é a que a soma dos custos dos padrões utilizados é a menor possível
  - Este é o *optimum*

# *Optimal e Optimum*

- Uma cobertura onde nenhum par de padrões adjacentes possa ser combinado em um par de menor custo é *optimal*
- Caso haja um padrão que possa ser quebrado e diminua o custo total, ele deve ser descartado
- *Optimum* é sempre *optimal*
- *Optimal* nem sempre é *optimum*

# *Optimal e Optimum*

- No exemplo anterior assumamos:
  - MOVEM tem custo  $m$
  - Todas as outras têm custo 1
  - O que acontece com as duas coberturas se
    - $m = 0, 1$  ou  $2$ ?





# Algoritmos

- Achar coberturas “optimais” é mais fácil
- CISC
  - Dada a complexidade das instruções, os padrões costumam ser grandes
  - A diferença entre *optimal* e *optimum* se torna mais considerável
- RISC
  - Instruções simples levam a padrões pequenos
  - Custo costuma ser mais uniforme
  - A diferença entre *optimal* e *optimum* praticamente desaparece

# *Maximal Munch*

- Encontra cobertura *optimal*
- Bastante simples
  - Inicie na raiz
  - Encontre o maior padrão que possa ser encaixado nesse nó
    - Cubra o raiz e provavelmente outros nós
  - Repita o processo para cada sub-árvore a ser coberta
- A cada padrão selecionado, uma instrução é gerada
- Ordem inversa da execução! A raiz é a última a ser executada

# *Maximal Munch*

- Algumas observações
  - O maior padrão é aquele com maior número de nós
  - Se dois padrões do mesmo tamanho encaixam, a escolha é arbitrária
- Facilmente implementado através de funções recursivas
  - Ordene as cláusulas com a prioridade de tamanho dos padrões
  - Se para cada tipo de nó da árvore existir um padrão de cobertura de um nó, nunca pode ficar travado.

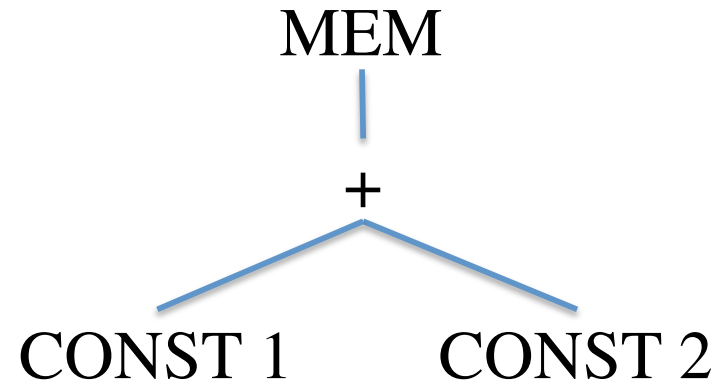
# *Maximal Munch*

- Faça a cobertura Maximal Munch das seguintes árvores:
  - MOVE(MEM(+(+ (CONST 1000, MEM(TEMP<sub>x</sub>)), TEMP<sub>fp</sub>)), CONST 0)
  - BINOP(MUL, CONST 5, MEM(CONST 100))

# Programação Dinâmica

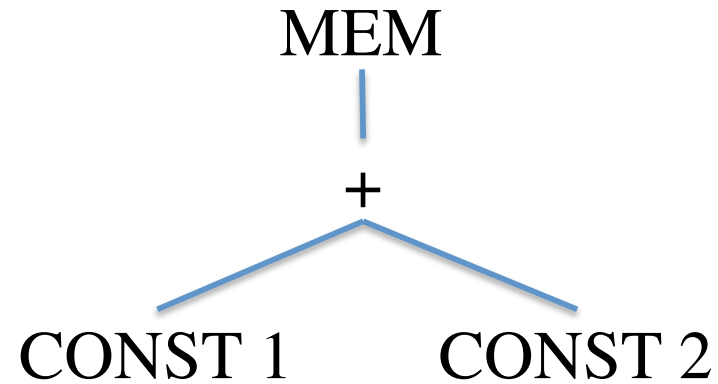
- Encontra uma cobertura ótima (*optimum*)
- PD monta uma solução ótima baseada em soluções ótimas de sub-problemas
- O algoritmo atribui um custo a cada nó da árvore
  - A soma do custo de todas as instruções da melhor cobertura da sub-árvore com raiz no respectivo nó
  - Para um dado nó  $n$ 
    - Encontra o melhor custo para suas sub-árvores
    - Analisa os padrões que podem cobrir  $n$
    - Algoritmo bottom-up

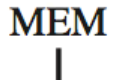
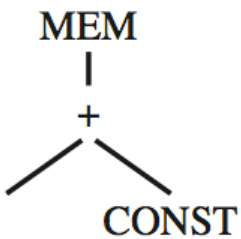
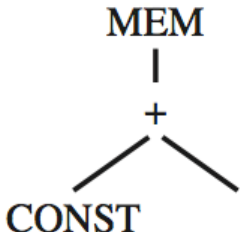
# Programação Dinâmica



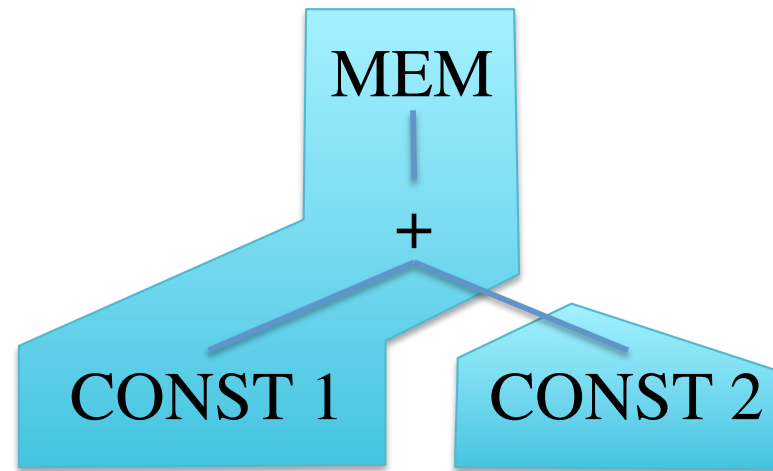
Tile	Instruction	Tile Cost	Leaves Cost	Total Cost
	ADD	1	1+1	3
	ADDI	1	1	2
	ADDI	1	1	2

# Programação Dinâmica



Tile	Instruction	Tile Cost	Leaves Cost	Total Cost
	LOAD	1	2	3
	LOAD	1	1	2
	LOAD	1	1	2

# Programação Dinâmica



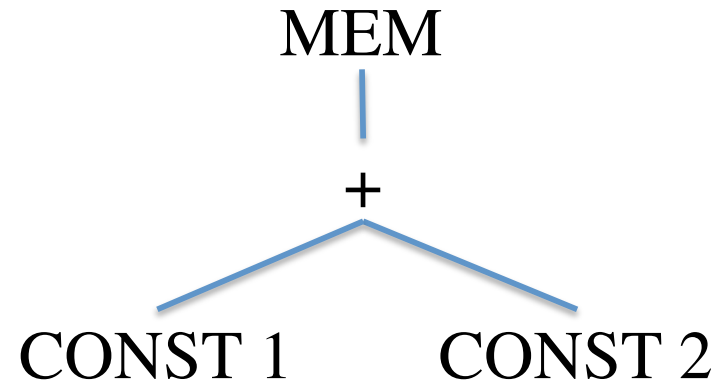
Tile	Instruction	Tile Cost	Leaves Cost	Total Cost
<pre> MEM   + /  \       </pre>	LOAD	1	2	3
<pre> MEM   + /  \ CONST       </pre>	LOAD	1	1	2
<pre> MEM   + /  \ CONST       </pre>	LOAD	1	1	2



# Programação Dinâmica

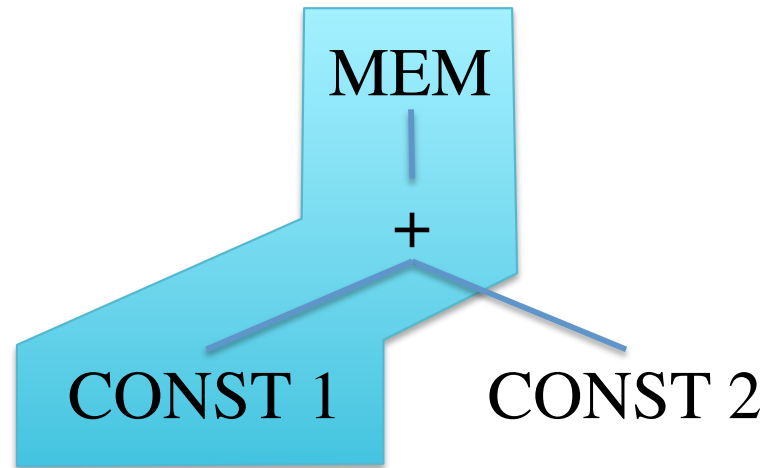
- Após computar o custo da raiz, emitir as instruções
- Emissão( $n$ )
  - Para cada folha  $f$  do padrão de árvore selecionado para o nó  $n$ , execute emissão( $f$ ) recursivamente.
  - Emita a instrução do padrão de  $n$
- A emissão de código é feita através da chamada
  - Emissão(raiz)

# Programação Dinâmica



Emissão(MEM)

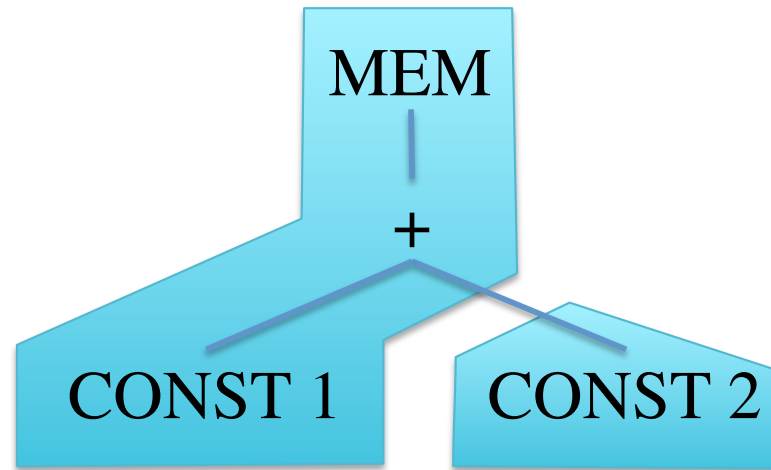
# Programação Dinâmica



Emissão(MEM)

Emissão(CONST 2)

# Programação Dinâmica

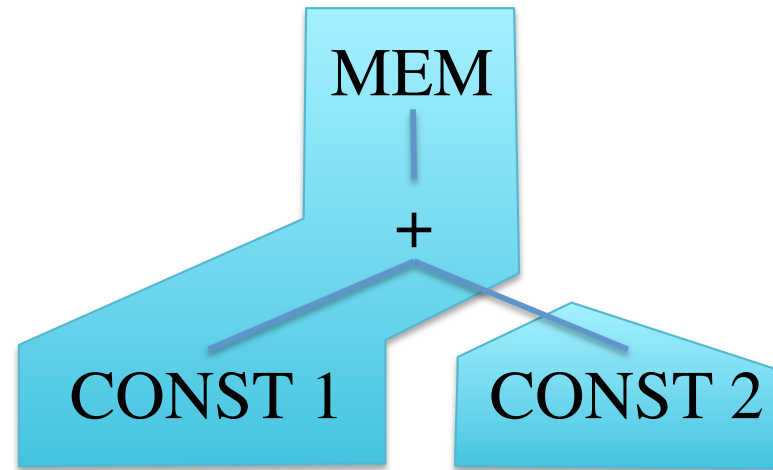


Emissão(MEM)

Emissão(CONST 2)

Emitte: ADDI r1 := r0 + 2

# Programação Dinâmica



Emissão(MEM)

Emissão(CONST 2)

Emite: ADDI r1 := r0 + 2

Emite: LOAD r1 := M[r1 + 1]

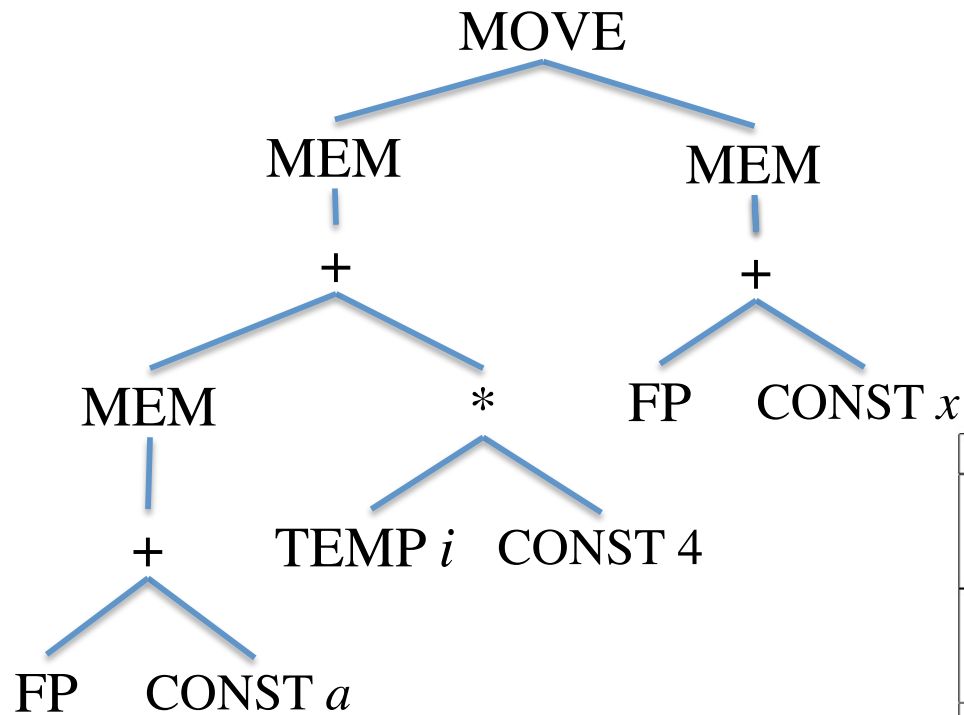
# Comentários sobre Eficiência

- Seja:
  - T: números de padrões diferentes
  - K: no. médio de nós não-folhas dos padrões casados
  - K': maior # de nós a serem olhados para identificar quais padrões casam a uma dada sub-árvore. Aprox. o tamanho do maior padrão
  - T': média de padrões diferentes que casam em cada nó
  - N: # nós da árvore
  - RISC típico:
    - T=50, K=2, K'=4, T'=5
- *Maximal Munch*:  $N/K * (K' + T')$
- Programação Dinâmica:  $N * (K' + T')$ 
  - Requer duas passadas na árvore

# Comentários sobre Eficiência

- Ambos são lineares
- Seleção de instruções é bastante rápida comparada com outras fases da compilação
- Até análise léxica pode ser mais demorada

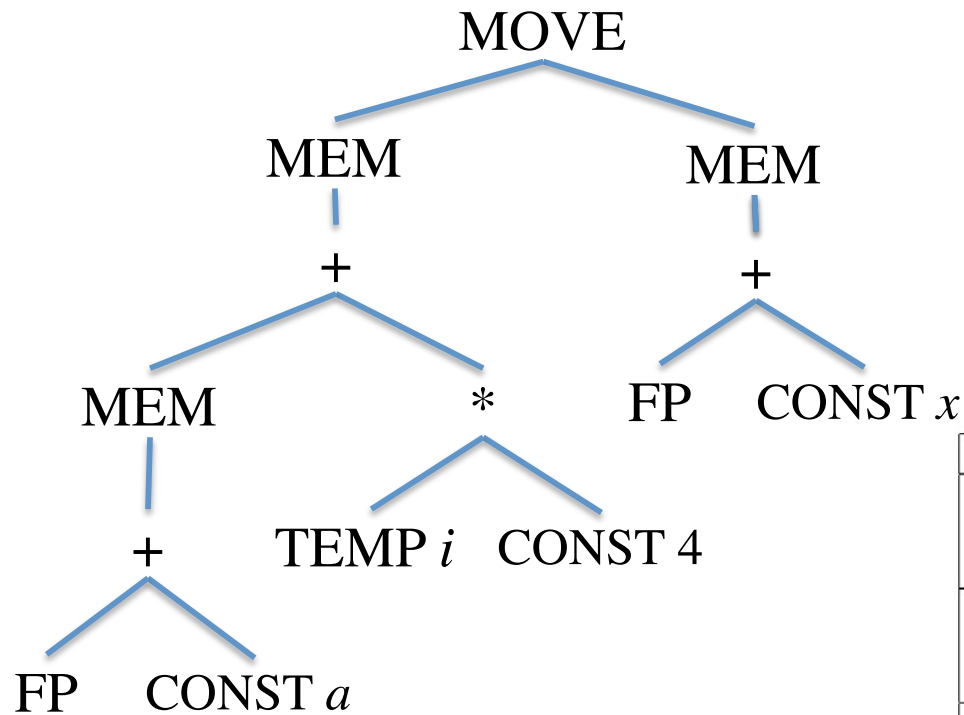
# Exemplo: *Maximal Munch*



Name	Effect	Trees
—	$r_i$	TEMP
ADD	$r_i \leftarrow r_j + r_k$	$\begin{array}{c} + \\ \swarrow \quad \searrow \end{array}$
MUL	$r_i \leftarrow r_j \times r_k$	$\begin{array}{c} * \\ \swarrow \quad \searrow \end{array}$
SUB	$r_i \leftarrow r_j - r_k$	$\begin{array}{c} - \\ \swarrow \quad \searrow \end{array}$
DIV	$r_i \leftarrow r_j / r_k$	$\begin{array}{c} / \\ \swarrow \quad \searrow \end{array}$
ADDI	$r_i \leftarrow r_j + c$	$\begin{array}{c} + \\ \swarrow \quad \searrow \\ \text{CONST} \quad \text{CONST} \end{array} \quad \text{CONST}$
SUBI	$r_i \leftarrow r_j - c$	$\begin{array}{c} - \\ \swarrow \quad \searrow \\ \text{CONST} \quad \text{CONST} \end{array}$
LOAD	$r_i \leftarrow M[r_j + c]$	$\begin{array}{c} \text{MEM} \quad \text{MEM} \quad \text{MEM} \quad \text{MEM} \\   \quad   \quad   \quad   \\ + \quad + \quad \text{CONST} \quad   \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ \text{CONST} \quad \text{CONST} \quad \text{CONST} \quad \text{CONST} \end{array}$
STORE	$M[r_j + c] \leftarrow r_i$	$\begin{array}{c} \text{MOVE} \quad \text{MOVE} \quad \text{MOVE} \quad \text{MOVE} \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ \text{MEM} \quad \text{MEM} \quad \text{MEM} \quad \text{MEM} \\   \quad   \quad   \quad   \\ + \quad + \quad \text{CONST} \quad   \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ \text{CONST} \quad \text{CONST} \quad \text{CONST} \quad \text{CONST} \end{array}$
MOVEM	$M[r_j] \leftarrow M[r_i]$	$\begin{array}{c} \text{MOVE} \\ \swarrow \quad \searrow \\ \text{MEM} \quad \text{MEM} \\   \quad   \end{array}$



# Exemplo: PD



Name	Effect	Trees
—	$r_i$	TEMP
ADD	$r_i \leftarrow r_j + r_k$	$\begin{array}{c} + \\ \swarrow \quad \searrow \end{array}$
MUL	$r_i \leftarrow r_j \times r_k$	$\begin{array}{c} * \\ \swarrow \quad \searrow \end{array}$
SUB	$r_i \leftarrow r_j - r_k$	$\begin{array}{c} - \\ \swarrow \quad \searrow \end{array}$
DIV	$r_i \leftarrow r_j / r_k$	$\begin{array}{c} / \\ \swarrow \quad \searrow \end{array}$
ADDI	$r_i \leftarrow r_j + c$	$\begin{array}{c} + \\ \swarrow \quad \searrow \\ \text{CONST} \quad \text{CONST} \end{array}$
SUBI	$r_i \leftarrow r_j - c$	$\begin{array}{c} - \\ \swarrow \quad \searrow \\ \text{CONST} \quad \text{CONST} \end{array}$
LOAD	$r_i \leftarrow M[r_j + c]$	$\begin{array}{c} \text{MEM} \quad \text{MEM} \quad \text{MEM} \quad \text{MEM} \\   \quad   \quad   \quad   \\ + \quad + \quad \text{CONST} \quad \text{CONST} \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ \text{CONST} \quad \text{CONST} \quad \text{CONST} \quad \text{CONST} \end{array}$
STORE	$M[r_j + c] \leftarrow r_i$	$\begin{array}{c} \text{MOVE} \quad \text{MOVE} \quad \text{MOVE} \quad \text{MOVE} \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ \text{MEM} \quad \text{MEM} \quad \text{MEM} \quad \text{MEM} \\   \quad   \quad   \quad   \\ + \quad + \quad \text{CONST} \quad \text{CONST} \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ \text{CONST} \quad \text{CONST} \quad \text{CONST} \quad \text{CONST} \end{array}$
MOVEM	$M[r_j] \leftarrow M[r_i]$	$\begin{array}{c} \text{MOVE} \\ \swarrow \quad \searrow \\ \text{MEM} \quad \text{MEM} \\   \quad   \end{array}$