Geração e Otimização de Código
(continuação)

Representação de código intermediária
Código de três endereços, P-código
Técnicas para geração de código
Otimização de código

Prof. Thiago A. S. Pardo

Geração de código

- Código de 3 endereços

2*a+(b-3) → t1 = 2 * a
t2 = b - 3
t3 = t1 + t2

- P-código

2*a+(b-3) →

```
ldc 2     ; carrega constante 2
lod a     ; carrega valor da variável a
mpi       ; multiplicação de inteiros
lod b     ; carrega valor da variável b
ldc 3     ; carrega constante 3
sbi       ; subtração de inteiros
adi       ; adição de inteiros
```

Geração de código

- Como se gera o código (intermediário ou final) para um programa?

1. Gramática de atributos

2. Procedimentos/funções de geração
   - Baseados na gramática de atributos definida
   - Ou *ad hoc*

Gramática de atributos

- Exemplo de gramática de atributos para geração do P-código (atributo pcod)

exp → id = exp | aexp
aexp → aexp + fator | fator
fator → (exp) | num | id

Valor da cadeia

Concatena e não pula linha

Concatena e pula linha

Regra Gramatical	Regras Semânticas
$exp_1 \rightarrow$ `id` = exp_2	exp_1 .pcod = "`lda`" || `id`.strval ++ exp_2.pcod ++ "`stn`"
$exp \rightarrow aexp$	exp .pcod = $aexp$ .pcod
$aexp_1 \rightarrow aexp_2$ + $fator$	$aexp_1$ .pcod = $aexp_2$ .pcod ++ $fator$ .pcod ++ "`adi`"
$aexp \rightarrow fator$	$aexp$ .pcod = $fator$ .pcod
$fator \rightarrow$ (exp)	$fator$ .pcod = exp .pcod
$fator \rightarrow$ `num`	$fator$ .pcod = "`ldc`" || `num`.strval
$fator \rightarrow$ `id`	$fator$ .pcod = "`lod`" || `id`.strval

Gramática de atributos

- Exercício: analise a cadeia (x=x+3)+4

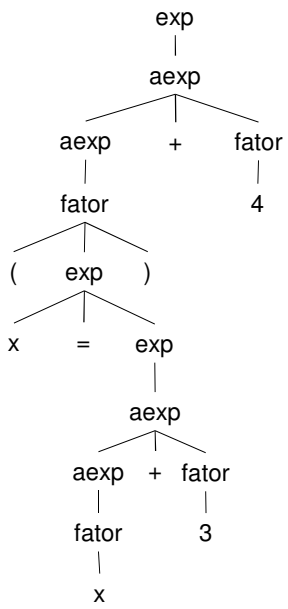
exp → id = exp | aexp
aexp → aexp + fator | fator
fator → (exp) | num | id

Valor da cadeia

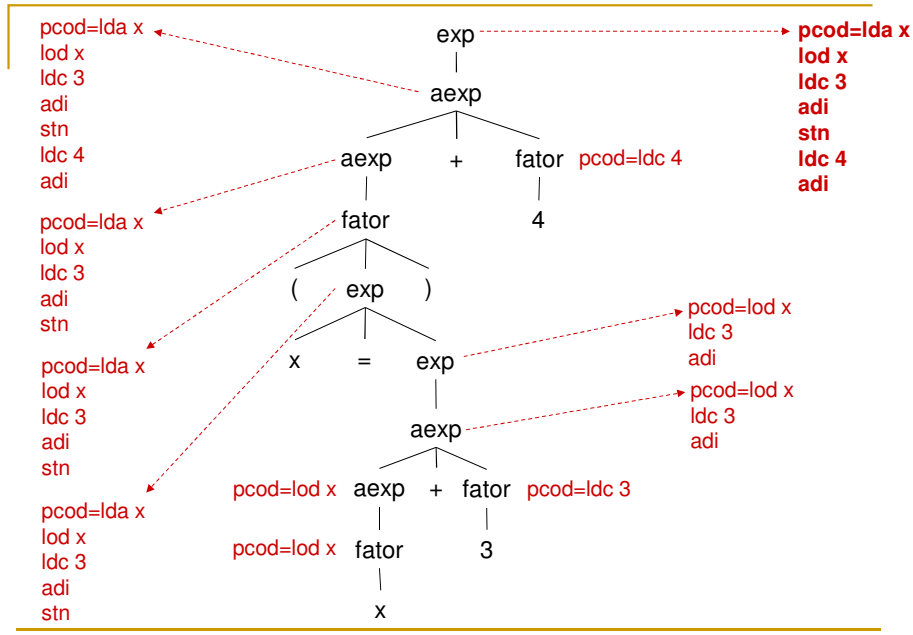
Concatena e não pula linha

Concatena e pula linha

Regra Gramatical	Regras Semânticas
$exp_1 \rightarrow$ `id` = exp_2	exp_1 .pcod = "`lda`" || `id`.strval ++ exp_2.pcod ++ "`stn`"
$exp \rightarrow aexp$	exp .pcod = $aexp$ .pcod
$aexp_1 \rightarrow aexp_2$ + $fator$	$aexp_1$ .pcod = $aexp_2$ .pcod ++ $fator$ .pcod ++ "`adi`"
$aexp \rightarrow fator$	$aexp$ .pcod = $fator$ .pcod
$fator \rightarrow$ (exp)	$fator$ .pcod = exp .pcod
$fator \rightarrow$ `num`	$fator$ .pcod = "`ldc`" || `num`.strval
$fator \rightarrow$ `id`	$fator$ .pcod = "`lod`" || `id`.strval



6



Gramática de atributos

- Exemplo de gramática de atributos para geração do código de 3 endereços (atributo tacode)

exp → id = exp | aexp
aexp → aexp + fator | fator
fator → (exp) | num | id

Regra Gramatical	Regras Semânticas
$exp_1 \rightarrow$ `id` = exp_2	exp_1 .name = exp_2 .name exp_1.tacode = exp_2 .tacode ++ `id.`strval || "=" || exp_2 .name
$exp \rightarrow aexp$	exp .name = aexp .name exp .tacode = aexp .tacode
$aexp_1 \rightarrow aexp_2$ + fator	$aexp_1$ .name = newtemp( ) $aexp_1$ .tacode = $aexp_2$ .tacode ++ fator .tacode ++ $aexp_1$ .name || "=" || $aexp_2$ .name || "+" || fator .name
$aexp \rightarrow fator$	aexp .name = fator .name aexp .tacode = fator .tacode
$fator \rightarrow$ (exp)	fator .name = exp .name fator .tacode = exp .tacode
$fator \rightarrow$ `num`	fator .name = `num.`strval fator .tacode = ""
$fator \rightarrow$ `id`	fator .name = `id.`strval fator .tacode = ""

Gera novo nome temporário, que é guardado no atributo "nome"

Cadeia vazia

Gramática de atributos

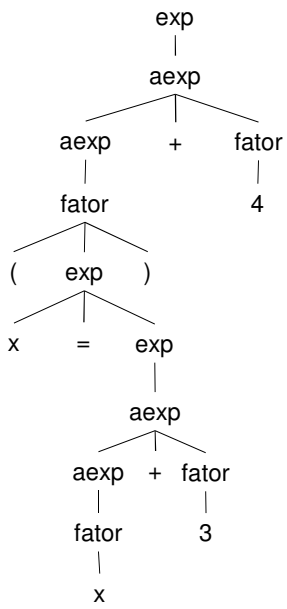
- Exercício: analise a cadeia (x=x+3)+4

exp → id = exp | aexp
aexp → aexp + fator | fator
fator → (exp) | num | id

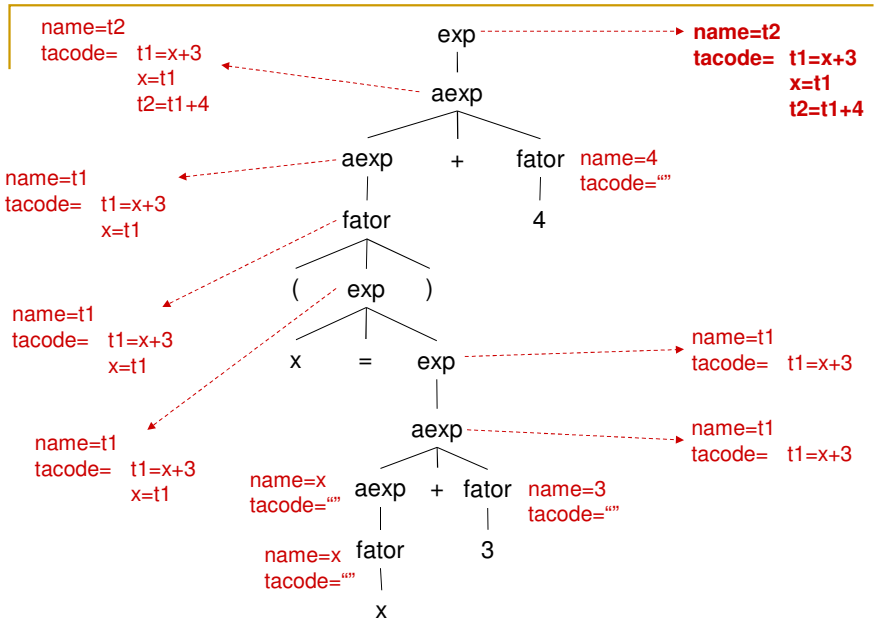
Regra Gramatical	Regras Semânticas
$exp_1 \rightarrow$ `id` = exp_2	exp_1 .name = exp_2 .name exp_1.tacode = exp_2 .tacode ++ `id.`strval || "=" || exp_2 .name
$exp \rightarrow aexp$	exp .name = aexp .name exp .tacode = aexp .tacode
$aexp_1 \rightarrow aexp_2$ + fator	$aexp_1$ .name = newtemp( ) $aexp_1$ .tacode = $aexp_2$ .tacode ++ fator .tacode ++ $aexp_1$ .name || "=" || $aexp_2$ .name || "+" || fator .name
$aexp \rightarrow fator$	aexp .name = fator .name aexp .tacode = fator .tacode
$fator \rightarrow$ (exp)	fator .name = exp .name fator .tacode = exp .tacode
$fator \rightarrow$ `num`	fator .name = `num`.strval fator .tacode = ""
$fator \rightarrow$ `id`	fator .name = `id`.strval fator .tacode = ""

Gera novo nome temporário, que é guardado no atributo "nome"

Cadeia vazia



10



11

Geração de código

- Gramática de atributos
  - Útil para apresentar com clareza as relações entre as sequências de código de diferentes partes da árvore sintática e para comparar diferentes métodos de geração de código...
  - mas não é prática, pois
    - Grande quantidade de cópia de cadeias e desperdício de memória
    - É desejável que se gerem pequenos trechos de código a medida que se analisa a cadeia e que se gravem esses trechos em um arquivo ou em outra estrutura de dados
    - A gramática de atributos pode ficar muito complicada

12

Procedimentos/funções para geração de código

- Com base na gramática de atributos e na árvore sintática
  - Similar à forma que se fazia a análise semântica
  - Aparência geral de um procedimento nessa linha (que deve ser instanciado de acordo com a geração de código pretendida)

```
procedure genCod(T: nó-árvore);
begin
    if T não é nulo then
            gere código de preparação para T;
            genCod(filho à esquerda de T);
            gere código de preparação para T;
            genCod(filho à direita de T);
            gere código para implementação de T;
end;
```

13

Procedimentos/funções para geração de código

- Com base na gramática de atributos e na árvore sintática
  - Similar à forma que se fazia a análise semântica
  - Exemplo instanciado básico para a gramática anterior

```
procedure genCod(T: nó-árvore);
begin
    if T não é nulo then
            if ('+') then
                    genCode(t->leftchild);
                    genCode(t->rightchild);
                    write("adi");
            else if ('=') then
                    write("lda"+id.strval);
                    genCode(t->rightchild);
                    write("stn");
            else if ('num') then write("ldc"+num.strval);
            else if ('id') then write("lod"+id.strval);
end;
```

exp → id = exp | aexp
aexp → aexp + fator | fator
fator → (exp) | num | id

Procedimentos/funções para geração de código

- Solução *ad hoc*
  - Geração de código amarrada aos procedimentos sintáticos

```
função fator(Seg): string;
Início
    declare cod: string;
    se (simbolo='(') então   inicio
                             obtem_simbolo(cadeia,simbolo);
                             cod=exp(Seg+{')'});
                             se (simbolo=')')
                                        então obtem_simbolo(cadeia,simbolo);
                                        senão ERRO(Seg);
                             fim
    senão se (simbolo='num') então   inicio
                                     cod="ldc "+cadeia;
                                     obtem_simbolo(cadeia,simbolo);
                                     fim
    senão se (simbolo='id') então    inicio
                                     cod="lod "+cadeia;
                                     obtem_simbolo(cadeia,simbolo);
                                     fim
    senão ERRO(Seg);
    retorne cod;
fim
```

fator → (exp) | num | id

Procedimentos/funções para geração de código

- Solução *ad hoc*
  - Geração de código amarrada nos procedimentos sintáticos
    - Alternativamente, podem-se chamar outras funções e procedimentos (com ou sem parâmetros que indiquem que código gerar) que gerem o código, em vez de embutir a geração diretamente no próprio procedimento sintático

16

Geração de código

- Ponto importante
  - O código-alvo (objeto) desejado pode ser similar ao P-código ou outra linguagem de montagem qualquer (caso da LALG)
    - Nesse caso, um montador também é necessário

17

Geração de código

- Geração de código-alvo a partir do código intermediário
  - Passo necessário se código intermediário é utilizado e é diferente do código-alvo desejado
  - Pode ser um processo complicado se o código intermediário não contém informações da máquina-alvo e de seu ambiente de execução

18

Geração de código

- Geração de código-alvo a partir do código intermediário
  - Em geral, dependendo do código com que se está lidando, se requer uma ou ambas destas técnicas: expansão de macros e simulação estática
    - **Expansão de macros**: encara cada linha de código como uma macro e a substitui por uma porção de código correspondente do código-alvo
      - Pode gerar código ineficiente ou redundante
    - **Simulação estática**: requer uma simulação direta dos efeitos do código intermediário e a geração do código-alvo correspondente a estes efeitos
      - Pode ser muito simples ou muito sofisticada

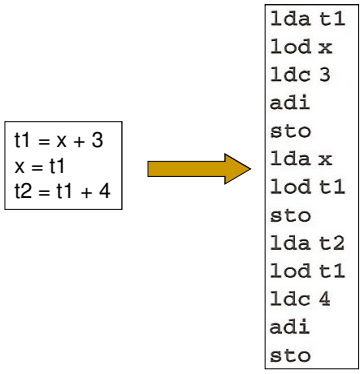
Geração de código

- Exemplo de expansão de macro
  - Supondo que temos código de três endereços como código intermediário e queremos o P-código como código-alvo
    - A expressão a=b+c pode ser substituída pela sequência abaixo

```
lda a
lod b (ou ldc b, se b é constante)
lod c (ou ldc c, se c é constante)
adi
sto
```

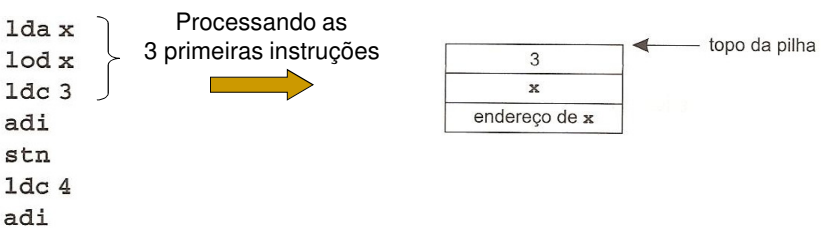
Geração de código

- Exemplo de expansão de macro
  - Supondo que temos código de três endereços como código intermediário e queremos o P-código como código-alvo



Geração de código

- Exemplo de simulação estática
  - Supondo que temos P-código como código intermediário e queremos o código de três endereços como código-alvo
    - Precisamos gerar os temporários e, por isso, é necessário que se simule a pilha e seu funcionamento



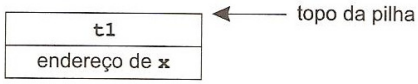
Geração de código

- Exemplo de simulação estática
  - Supondo que temos P-código como código intermediário e queremos o código de três endereços como código-alvo
    - Precisamos gerar os temporários e, por isso, é necessário que se simule a pilha e seu funcionamento

```
lda x
lod x
ldc 3
adi
stn
ldc 4
adi
```

Processando adi

Código de três endereços:
t1 = x + 3



23

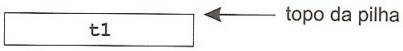
Geração de código

- Exemplo de simulação estática
  - Supondo que temos P-código como código intermediário e queremos o código de três endereços como código-alvo
    - Precisamos gerar os temporários e, por isso, é necessário que se simule a pilha e seu funcionamento

```
lda x
lod x
ldc 3
adi
stn
ldc 4
adi
```

Processando stn

Código de três endereços:
t1 = x + 3
x = t1



24

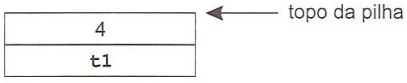
Geração de código

- Exemplo de simulação estática
  - Supondo que temos P-código como código intermediário e queremos o código de três endereços como código-alvo
    - Precisamos gerar os temporários e, por isso, é necessário que se simule a pilha e seu funcionamento

```
lda x
lod x
ldc 3
adi
stn
ldc 4   }
adi
```

Processando ldc 4

Código de três endereços:
t1 = x + 3
x = t1



25

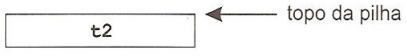
Geração de código

- Exemplo de simulação estática
  - Supondo que temos P-código como código intermediário e queremos o código de três endereços como código-alvo
    - Precisamos gerar os temporários e, por isso, é necessário que se simule a pilha e seu funcionamento

```
lda x
lod x
ldc 3
adi
stn
ldc 4
adi     }
```

Processando adi

Código de três endereços:
t1 = x + 3
x = t1
t2 = t1 + 4



26

Otimizando código

Otimização

- Definição: processo de se melhorar a qualidade do código gerado pelo compilador
  - Nome enganoso, pois é raro se gerar o código ótimo
- Qualidade de código pode ser medida por vários ângulos
  - Quais?

Otimização

- Definição: processo de se melhorar a qualidade do código gerado pelo compilador
  - Nome enganoso, pois é raro se gerar o código ótimo
- Qualidade de código pode ser medida por vários ângulos
  - Velocidade
  - Tamanho do código
  - Memória utilizada para temporários

29

Otimização

- Análise do código e de sua execução permitem determinar o que otimizar
  - Algumas técnicas de otimização são simples e impactantes
  - Algumas são caras (em termos de tempo), complexas e com pouco ganho
    - Há muitas técnicas de otimização, algumas boas para algumas linguagens, outras não
      - O projetista do compilador deve fazer esse julgamento!

30

Otimização

- Principais fontes de otimização de código
  - Alocação de registradores: quanto mais e melhor usados, maior a velocidade
  - Operações desnecessárias
    - Expressões de valores invariáveis, mas avaliadas várias vezes no programa
    - Código inatingível ou morto (por exemplo, if com condição falsa sempre ou procedimento nunca ativado)
    - Saltos para outros saltos (poderia ir direto)

Otimização

- Principais fontes de otimização de código
  - Operações caras
    - Redução de força: trocam-se expressões caras por mais baratas (por exemplo, multiplicação por 2 pode ser implementada como uma transposição)
    - Empacotamento e propagação de constantes: reconhecimento e troca de expressões constantes pelo valor calculado (por exemplo, troca-se 2+5 por 7)
    - Procedimentos: pode-se melhorar fazendo-se alinhamento de procedimentos (inserção de seus códigos no corpo do programa), identificação e remoção de recursão de cauda
    - Uso de dialetos de máquina: instruções mais baratas oferecidas por máquinas específicas
  - Previsão do comportamento do programa
    - Conhecimento do comportamento do programa para otimizar saltos, laços e procedimentos ativados mais frequentemente

Otimização

- Tipos de otimização
  - Quanto ao instante em que são realizadas
  - Quanto à área do programa em que se aplicam
- Quanto ao instante
  - Empacotamento de constantes pode ser feito durante análise sintática
  - Otimização de saltos poderia ser feita após a geração do código-alvo
    - Otimizações de "buraco de fechadura" (veem-se apenas pequenas porções de código)

33

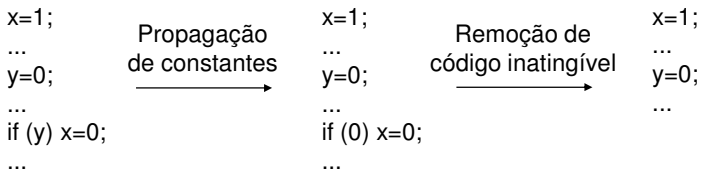
Otimização

- Quanto ao instante
  - A maioria das otimizações é feita sobre o código intermediário ou durante a geração do código-alvo
    - Otimizações de nível de fonte: depende do programa (por exemplo, quantas vezes as variáveis são acessadas determina quais variáveis ficarão em registradores)
    - Otimizações de nível de alvo: depende da máquina-alvo e do ambiente de execução (por exemplo, número de registradores disponíveis)

34

Otimização

- Quanto ao instante
  - Análise do efeito de uma otimização sobre outras
    - Faz sentido propagar constantes antes de tratar código inatingível (pode ser mais fácil identificá-los)



Otimização

- Quanto à área do programa
  - Otimizações locais: que se aplicam a segmentos de código de linha reta, ou seja, que não contêm saltos para dentro ou fora da sequência; sequência maximal de código de linha reta é chamada “bloco básico”
    - Relativamente fáceis de efetuar
  - Otimizações globais: que se estendem para além dos blocos básicos, mas que sejam confinadas a um procedimento individual
    - Exigem análise de fluxo de dados
  - Otimizações interprocedimentos: que se estendem para além dos limites dos procedimentos, podendo atingir o programa todo
    - As mais complexas, exigindo diversos tipos de informações e rastreamentos do programa

Para estudar em casa

- Pesquisar sobre bytecode, o código intermediário de java
  - Fazer/responder
    - Como é o código? Características do código, que estilo segue (P-código ou 3 endereços)?
    - Exemplos
    - É independente ou assume a existência de alguma organização de memória (pilha)?

37