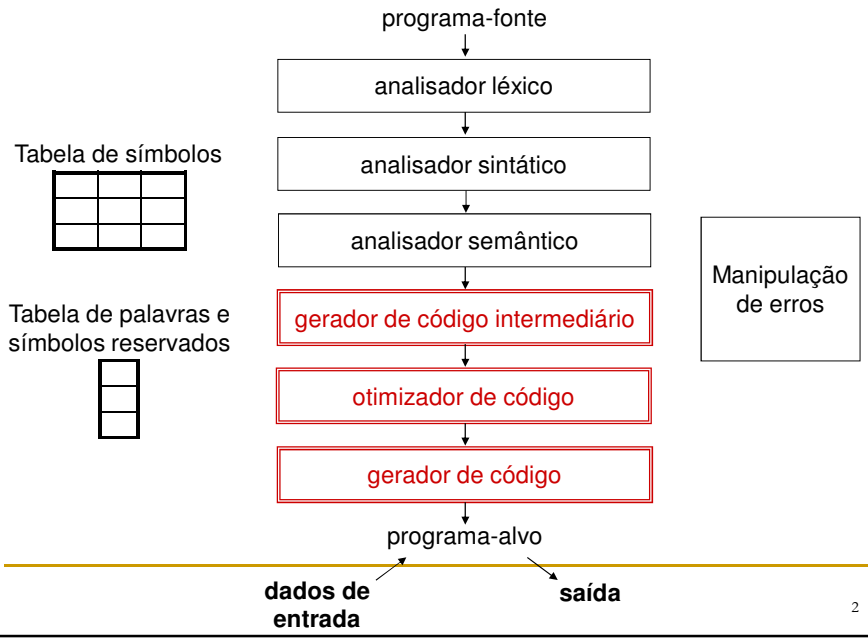
Geração e Otimização de Código

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

Prof. Thiago A. S. Pardo

1

Estrutura geral de um compilador



2

Geração de código

- A geração de código é uma das tarefas mais complexas do compilador
  - Depende da linguagem-fonte, da máquina-alvo e seu ambiente de execução e sistema operacional
- Portanto, é conveniente dividir esta etapa em etapas menores
  - Geração de código intermediário, geração de código objeto, otimização
    - Compilação de várias passagens

3

Geração de código intermediário

- Até o momento, temos utilizado a árvore sintática (mesmo que implícita) como representação interna por excelência, juntamente com as informações da tabela de símbolos
- Entretanto, ela não corresponde em nada com o código-objeto que queremos gerar, além de poder ser muito complexa
- Pode ser interessante gerar um código mais próximo do objeto antes de fazer a tradução final
  - Código intermediário

4

Geração de código intermediário

- Características
  - Pode assumir muitas formas
  - Em geral, é alguma forma de linearização da árvore sintática
  - Pode ser muito abstrato (como a árvore sintática) ou mais próximo do código-objeto
  - Pode ou não usar informações sobre a máquina-alvo e o ambiente de execução (como disponibilidade de registradores, tamanho dos tipos, etc.)
  - Pode ou não incorporar informações da tabela de símbolos (como escopo, níveis de aninhamento, etc.), dispensando ou não a tabela na geração de código-objeto

5

Geração de código intermediário

- Vantagens
  - Bom para quando se quer produzir código-objeto extremamente eficiente
  - Se for genérico o suficiente, pode ser a base para geração de código para várias máquinas
    - Aumenta a portabilidade (o tradutor do código intermediário para o objeto ainda é necessário)

6

Geração de código intermediário

- Veremos aqui as características gerais de 2 formas populares de código intermediário
  - Código de três endereços
  - P-código
    - Também possuem diversas formas distintas na literatura e na prática de compiladores

7

Código de três endereços

8

Código de três endereços

- Instrução mais básica: x = y op z
  - O operador op é aplicado a y e z e o resultado é armazenado em x
- Origem do nome desse código: três endereços de memória envolvidos no cômputo
  - y e z (mas não x) podem ser constantes ou literais, na realidade
- Exemplo

2*a+(b-3) ⟹

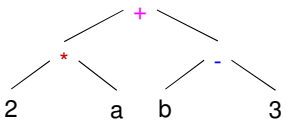
```
t1 = 2 * a
t2 = b - 3
t3 = t1 + t2
```


Código de três endereços

- Atenção
  - É necessário que se gerem variáveis temporárias (t1, t2 e t3 no exemplo) com nomes diferentes dos possíveis identificadores no programa
  - As variáveis podem ser mantidas na memória ou em registradores
  - As variáveis temporárias correspondem aos nós internos da árvore sintática subjacente à expressão

2*a+(b-3) ⟹

```
t1 = 2 * a
t2 = b - 3
t3 = t1 + t2
```

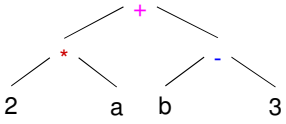


Código de três endereços

- No exemplo, o código intermediário corresponde à linearização da árvore quando ela é percorrida em pós-ordem
  - Qualquer alternativa é possível, dependendo somente da semântica da linguagem
    - Dependendo da alternativa, o código intermediário pode mudar (por exemplo, faz-se primeiro a subtração e depois a soma)

2*a+(b-3) ⟹

t1 = 2 * a
t2 = b - 3
t3 = t1 + t2



11

Código de três endereços

- Para acomodar todas as possibilidades de uma linguagem de programação, o código precisa ser mais flexível
  - Nem sempre se têm três elementos
    - Essa é uma das razões da diversidade de formas de código intermediário encontradas
  - Por exemplo, para operadores unários, pode-se ter o código t2 = -t1
    - Outras possibilidades: cópia, salto incondicional e condicional, índices, endereços

12

Código de três endereços

- Um exemplo de uma linguagem fictícia e seu possível código intermediário

```
{  Programa exemplo
   -- computa o fatorial
}
read x; { inteiro de entrada }
if 0 < x then { não computa se x <= 0 }
   fact := 1;
   repeat
      fact := fact * x;
      x := x - 1
   until x = 0;
   write fact { fatorial de x como saída }
end
```

```
read x
t1 = x > 0
if_false t1 goto L1
fact = 1
label L2
t2 = fact * x
fact = t2
t3 = x - 1
x = t3
t4 = x == 0
if_false t4 goto L2
write fact
label L1
halt
```

13

Código de três endereços

- Estrutura de dados
  - Nem sempre os códigos são armazenados textualmente como no exemplo
  - Listas encadeadas e arranjos podem ser usadas
  - Em vez de armazenar nomes de identificadores, pode ser interessante guardar ponteiros para os identificadores na tabela de símbolos (caso ela esteja sendo usada)
  - Muitas vezes, a noção de tuplas é utilizada (implementadas via registros, por exemplo)
    - Haverá, no máximo, quádruplas, sendo que algumas posições podem ficar vazias

14

Código de três endereços

- Exemplo de representação em tuplas

```
read x
t1 = x > 0
if_false t1 goto L1
fact = 1
label L2
t2 = fact * x
fact = t2
t3 = x - 1
x = t3
t4 = x == 0
if_false t4 goto L2
write fact
label L1
halt
```

```
(rd,x,_,_)
(gt,x,0,t1)
(if_f,t1,L1,_)
(asn,1,fact,_)
(lab,L2,_,_)
(mul,fact,x,t2)
(asn,t2,fact,_)
(sub,x,1,t3)
(asn,t3,x,_)
(eq,x,0,t4)
(if_f,t4,L2,_)
(wri,fact,_,_)
(lab,L1,_,_)
(halt,_,_,_)
```

15

P-código

16

P-código

- Um pouco da história
  - Criado como código-alvo de montagem padrão para os compiladores de PASCAL dos anos 70 e 80
  - Criado como código para uma máquina hipotética chamada P-máquina, sendo que diversos montadores para máquinas reais foram disponibilizados
    - Aumento da portabilidade
  - Foi redescoberto como um bom código intermediário

17

P-código

- Assume que
  - O ambiente de execução é baseado em pilhas
  - Dados do ambiente de execução são conhecidos (como tamanho de tipos)
- Exemplo de P-código para 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
```

18

P-código

- Outro exemplo: x:=y+1

```
lda x     ; carrega endereço de x
lod y     ; carrega valor de y
ldc 1     ; carrega constante 1
adi       ; adição
sto       ; armazena topo no endereço
          ; abaixo do topo & retira os dois
```

19

P-código

- Exemplo para o programa fictício completo

```
{ Programa exemplo
  -- computa o fatorial
}
read x; { inteiro de entrada }
if 0 < x then { não computa se x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact { fatorial de x como saída }
end
```

20

P-código

- Exemplo para o programa fictício completo

```
{  Programa exemplo
   -- computa o fatorial
}
read x; { inteiro de entrada }
if 0 < x then { não computa se x
   fact := 1;
   repeat
      fact := fact * x;
      x := x - 1
   until x = 0;
   write fact { fatorial de x co
end
```

```
lda x       ; carrega endereço de x
rdi         ; lê um inteiro, armazena no
            ; endereço no topo da pilha (& o retira)
lod x       ; carrega o valor de x
ldc 0       ; carrega a constante 0
grt         ; retira da pilha e compara os dois valores do topo
            ; coloca na pilha o resultado booleano
fjp L1      ; retira o valor booleano, salta para L1 se falso
lda fact    ; carrega endereço de fact
ldc 1       ; carrega constante 1
sto         ; retira dois valores, armazena primeiro
            ; em endereço representado pelo segundo
lab L2      ; definição do rótulo L2
lda fact    ; carrega endereço de fact
lod fact    ; carrega valor de fact
lod x       ; carrega valor de x
mpi         ; multiplica
sto         ; armazena topo em endereço do segundo & retira
lda x       ; carrega endereço de x
lod x       ; carrega valor de x
ldc 1       ; carrega constante 1
sbi         ; subtrai
sto         ; armazena (como no caso anterior)
lod x       ; carrega valor de x
ldc 0       ; carrega constante de 0
equ         ; teste de igualdade
fjp L2      ; salta para L2 se falso
lod fact    ; carrega valor de fact
wri         ; escreve topo da pilha & retira
lab L1      ; definição do rótulo L1
stp
```

P-código

- Em relação ao código de três endereços
  - Mais próximo do código de máquina
  - Exigem menos endereços (quase todas as instruções vistas até agora tinham 1 ou nenhum endereço)
  - Menos compacto (isto é, tem mais instruções)
  - Não é "auto-suficiente", pois assume a existência de uma pilha
  - Pode ser representado e implementado da mesma forma que o código de três endereços

22