Geração de código

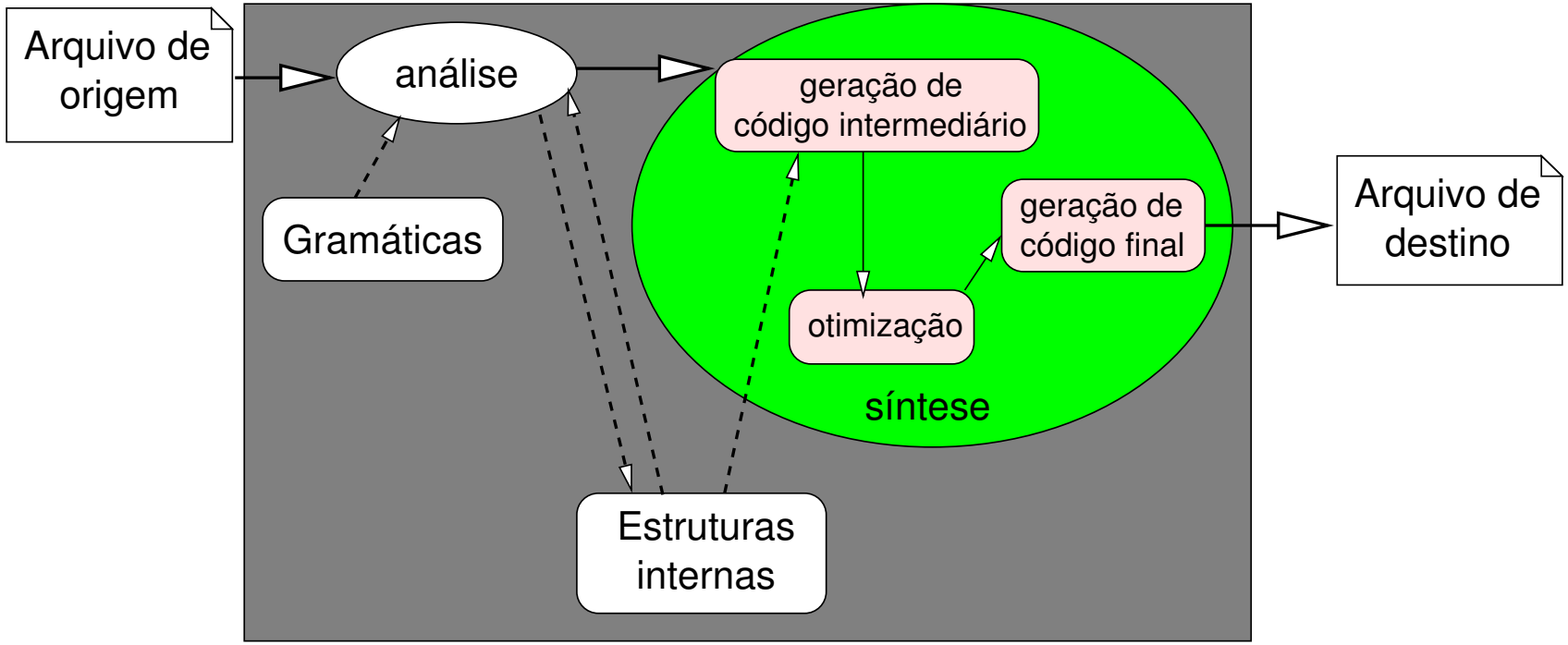
Ivan Ricarte

2008

Sumário

Estrutura geral do compilador

Atividades de síntese



Produção de código

- Etapa de síntese da compilação
- Em geral, em duas fases
  1. Tradução da estrutura construída na análise sintática para um código em linguagem intermediária independente do processador
  2. Tradução do código em linguagem intermediária para a linguagem simbólica do processador-alvo
- Produção do código binário é realizada por outro programa do sistema (montador)

Geração de código intermediário

Com o reconhecimento da estrutura de expressões básicas no código, é possível sintetizar expressões equivalentes na linguagem em formato intermediário

- Pela varredura da árvore sintática
- Durante o reconhecimento de expressões, na análise sintática
  - Tradução dirigida pela sintaxe (ações semânticas)

Formatos usuais para a linguagem em formato intermediário:

- Código de três endereços
- Notação posfixa

Código de três endereços

- Operações têm, no máximo, referências a três variáveis (dois operandos e um resultado)
- Expressões complexas devem ser decompostas em várias expressões nesse formato
- Formato independente de processador específico
- Fácil de traduzir para linguagem simbólica de qualquer processador

Tipos de instruções

- Instruções de atribuição
  - cópia
  - resultado de operações binárias
  - resultado de operações unárias
- Instruções de desvio
  - incondicional
  - condicional
  - invocação de rotinas
- Operadores de endereçamento
  - indexado
  - indireto

Código de três endereços

Instruções de atribuição

Atribuição simples

```
le := ld
```

Exemplo

Código C/C++:

```
x = a;
```

Tradução:

```
x := a
```

Código de três endereços

Instruções de atribuição

Operação binária com atribuição

```
le := ld1 op ld2
```

Exemplo

Código C/C++:

```
x = a + b;
```

Tradução:

```
x := a + b
```

Código de três endereços

Instruções de atribuição

Operação unária com atribuição

```
le := op ld
```

Exemplo

Código C/C++:

```
x = -a;
```

Tradução:

```
x := -a
```

Código de três endereços

Instruções de atribuição

Expressões mais complexas são traduzidas para uma série de expressões nesse formato

- Variáveis temporárias são criadas para armazenar resultados intermediários, conforme necessário

Exemplo

Expressão C/C++:

```
a = b + c * d;
```

Tradução:

```
_t1 := c * d
a := b + _t1
```

Código de três endereços

Instruções de desvio

`L` é um rótulo de uma linha do código intermediário

Desvio incondicional

```
goto L
```

Desvio condicional

```
if x opr y goto L
```

onde `opr` é um operador relacional

Código de três endereços

Exemplo de tradução para instrução de desvio

C++

```
while (i++ <= k)
    x[i] = 0;
x[0] = 0;
```

Tradução

```
_L1: if i > k goto _L2
     i := i + 1
     x[i] := 0
     goto _L1
_L2: x[0] := 0
```

Código de três endereços

Invocação de subrotinas

Padrão de invocação:

- Parâmetros da função, se presentes, são registrados com instrução `param`
- Subrotina é invocada com instrução `call`, com indicação do número de parâmetros
- Retorno, se presente, pode ser obtido a partir de atribuição do resultado de `call`

Código de três endereços

Exemplo de tradução de invocação de subrotina

C++

```
x = f (a, g (b));
```

Tradução

```
param a
param b
_t1 := call g, 1
param _t1
x := call f, 2
```

Código de três endereços

Modos de endereçamento

Modo direto

```
x := a
```

Modo indexado

```
x := y[i]
x[i] := y
```

com `i` em bytes!

Modo indireto

```
 x := &y
 w := *x
*x := z
```

Código de três endereços

Estruturas de representação interna

Quádruplas

Tabela com quatro colunas:

1. operador
2. primeiro argumento
3. segundo argumento
4. resultado

Código de três endereços

Estruturas de representação interna

Triplas

Coluna com resultado é omitida

- Referência à linha da tabela é utilizada para indicar resultados intermediários
- Operador de atribuição simples deve ser utilizado para explicitar armazenamento de resultados não temporários

Estruturas de representação interna

Exemplo

C++

```
a = b + c * d;
```

Quádrupla

	operador	arg 1	arg 2	resultado
1	*	c	d	_t1
2	+	b	_t1	a

Estruturas de representação interna

Exemplo

C++

```
a = b + c * d;
```

Tripla

	operador	arg 1	arg 2
1	*	c	d
2	+	b	(1)
3	:=	a	(2)

Notação pós-fixa

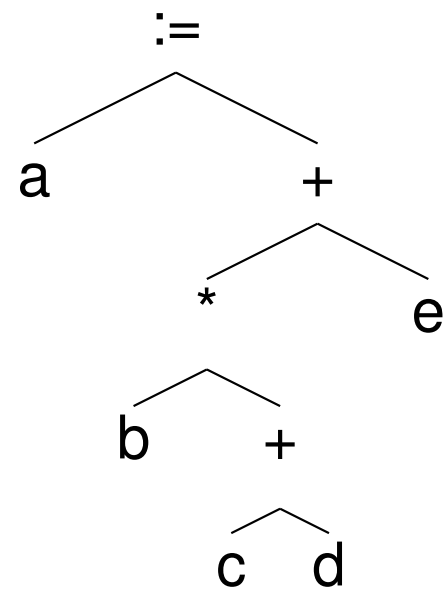
- Alternativa para representação de expressões
- Operações são indicadas após operandos
- Representação interna de cada expressão na forma de uma lista
- Não há necessidade de parênteses

Notação pós-fixa

C++

```
a = b*(c+d)+e;
```

Árvore sintática abstrata



Tradução (varredura pós-ordem)

```
a b c d + * e + :=
```

Otimização de código

- Código gerado automaticamente pode ser ineficiente
  - Redundâncias
  - Instruções desnecessárias
- Ineficiência pode vir também da codificação original
- Heurísticas de otimização podem ser aplicadas no código em formato intermediário antes da produção do código em linguagem simbólica

Eliminação de subexpressões comuns

Operações que se repetem sem que seus argumentos sejam alterados podem ser realizadas uma única vez

Exemplo

Sem atribuição a `a` ou `b` entre as duas instruções:

```
x = a + b + c;
...
y = a + b + d;
```

Sem otimização:

```
_t1 := a + b
x := _t1 + c
...
_t2 := a + b
y := _t2 + d
```

Com otimização:

```
_t1 := a + b
x := _t1 + c
...
y := _t1 + d
```

Eliminação de código redundante

Instruções sem efeito podem ser eliminadas

Exemplo

Sem nenhuma atribuição a `x` ou a `y` entre as duas instruções,

```
x := y
...
x := y
```

```
x := y
...
y := x
```

a segunda instrução pode ser seguramente eliminada

Propagação de cópias

Variáveis que só mantêm cópia de um valor, sem outros usos, podem ser eliminadas

Exemplo

Sem outras atribuições a `y` e sem outros usos de `x`

```
x := y
...
z := x
```

pode ser reduzido a

```
...
z := y
```

Eliminação de desvios desnecessários

Desvio incondicional para a próxima instrução pode ser eliminado

Exemplo

```
        a := _t2
        goto _L6
 _L6:   c := a + b
```

equivale a

```
        a := _t2
        c := a + b
```

Eliminação de código não-alcançável

Instruções que nunca serão executadas podem ser eliminadas

Exemplo

```
      goto _L3
      _t1 := x
_L4: _t2 := b + c
_L3: d := a + _t2
```

equivale a

```
      goto _L3
_L4: _t2 := b + c
_L3: d := a + _t2
```

Movimentação de código

Retirar do corpo de comandos iterativos (laços) cálculo de expressões invariáveis

Exemplo

```
while ( i < 2*max )
    a[i] = i + max/4;
```

Como valor de `max` não varia, equivale a

```
_t1 = 2*max;
_t2 = max/4;
while (i < _t1)
    a[i] = i + _t2;
```

Uso de propriedades algébricas

Substituição de expressões aritméticas por formas equivalentes

Original	Equivalente
x + y	y + x
x + 0	x
x - 0	x
x * y	y * x
x * 1	x
x / 1	x
2 * x	x + x
x^2	x * x

Limites e objetivos de otimização

Em geral, usuário pode indicar ao compilador quanto de esforço deve ser dispendido em otimização

Compilador gcc/g++

-O0 nenhuma otimização deve ser feita, compilação fica mais rápida mas programa gerado é menos eficiente

- Mas pode ficar mais fácil de seguir execução num processo de depuração, por estar mais próximo do código original

-O3 máxima otimização, programa gerado tem menor tempo de execução mas tempo de compilação é mais longo

-O1, -O2 graus de otimização intermediários

-Os Se objetivo da otimização não for tempo de execução, mas a minimização do espaço ocupado em memória

Geração de código em linguagem simbólica

Objetivo Obter, a partir das instruções elementares usadas no código em formato intermediário, código equivalente na linguagem simbólica do processador-alvo

- Diferentes processadores podem ter distintos formatos de instruções
- Classificação pelo número de endereços na instrução:
  - 3 dois operandos e o resultado
  - 2 dois operandos (resultado sobrescreve primeiro operando)
  - 1 só segundo operando, primeiro operando implícito (registrador acumulador), resultado sobrescreve primeiro operando
  - 0 operandos e resultado numa pilha, sem endereço explícitos

Tradução para linguagem simbólica

Tradução ocorre segundo gabaritos definidos de acordo com o tipo de máquina

Exemplo: x := y + z

3-end

```
ADD y, z, x
```

2-end

```
MOVE Ri, y
ADD  Ri, z
MOVE x, Ri
```

1-end

```
LOAD  y
ADD   z
STORE x
```

0-end

```
PUSH y
PUSH z
ADD
POP  x
```

Otimização no código em linguagem simbólica

Ao combinar seqüências de instruções traduzidas pelos gabaritos, estratégias de otimização podem ser também aplicadas ao código em linguagem simbólica

- Eliminação de código redundante
- Eliminação de instruções desnecessárias

Além disso, há uma oportunidade de realizar otimizações que são específicas para um determinado processador

- Otimizações dependentes de máquina
- Permitem o aproveitamento de instruções pouco usuais, de uso restrito
- Muitas vezes, é difícil para o compilador reconhecer a possibilidade de uso dessas instruções

Sugestões de leitura

Artigos na Wikipedia sobre

- Geração de código: `http://en.wikipedia.org/wiki/Code_generation_(compiler)`
- Otimização de código: `http://en.wikipedia.org/wiki/Optimization_(computer_science)`