Geração de código intermediário: Geração de código de fluxo de controle Backpatching

Geração de código intermediário

- Em código que manipula o fluxo de controle de um programa (ifs, while, for, do..while, etc.), operadores booleanos &&, II e ! são traduzidos em comandos de desvios
- A idéia é permitir ações semânticas junto com a análise sintática
- As construções permitem gerar código de três endereços

Geração de código intermediário

- Exemplo:
  - if (x< 100 || x>200 && x!=y) x=0;
  - É gerado como:

    If x<100 goto L2

    goto L3

    L3: if x>200 goto L4

    goto L1

    If x!=y goto L2

    goto L1

    x=0;

Geração de código intermediário

- Nas regras a seguir, considere:
  - newlabel(): retorna um novo label
  - next, true e false: são atributos herdados dos não-terminais
  - code: é um atributo sintetizado
  - label(L): retorna o nome do label armazenado por L
  - gen('string'): retorna o valor string
  - Operador de concateção ||: “une” expressões

Geração de código intermediário

Produção	Regra Semântica
$P \rightarrow S$	S.next=newlabel() P.code=S.code || label(S.next)
$S \rightarrow$ assign	S.code=assign.code
$S \rightarrow$ if (B) S_1	B.true=newlabel() B.false=S_1.next=S.next S.code=B.code || label(B.true) || S_1.code
$S \rightarrow$ if (B) S_1 else S_2	B.true=newlabel() B.false=newlabel() S_1.next=S_2.next=S.next S.code=B.code || label(B.true) || S_1.code || gen('goto' S.next) || label(B.false) || S2.code

Geração de código intermediário

Produção	Regra Semântica
$S \rightarrow$ while (B) S_1	begin=newlabel() B.true=newlabel() B.false=next S_1.next=begin S.code=label(begin) || B.code || label(B.true) || S_1.code || gen('goto' begin)
$S \rightarrow$ do S_1 while (B) $S \rightarrow$ if (B) S_1	begin=newlabel() B.true=begin B.false=S.next S_1.next=newlabel() S.code=label(begin) || S_1.code || label(S_1.next) || B.code

Geração de código intermediário

Produção	Regra Semântica
$B \rightarrow B_1 \,\|\|\, B_2$	B_1.true=B.true B_1.false=newlabel() B_2.true=B.true B_2.false=B.false B.false=next B.code=B_1.code || label(B_1.false) || B_2.code
$B \rightarrow B_1 \,\&\&\, B_2$	B_1.true=newlabel() B_1.false=B.false B_2.true=B.true B_2.false=B.false B.code=B_1.code || label(B_1.true) || B_2.true

Geração de código intermediário

Produção	Regra Semântica
$B \rightarrow !B_1$	B_1.true=B.false B_1.false=B.true B.code=B_1.code
$B \rightarrow E_1$ rel E_2	B.code=E_1.code || E_2.code || gen('if' E_1.addr rel.op E_2.addr 'goto' B.true) || gen('goto' B.false)

Backpatching

- Na tradução de comandos de fluxo de controle, a abordagem que usa atributos herdados faz com que o processo de tradução seja, necessariamente, em duas etapas
- Em expressões do tipo if (B) S, as regras anteriores geram o código de B antes de gerar S, qual será, então o alvo do goto para o caso de B ser falso?
  - A solução anterior usa atributos herdados que capturam labels onde os gotos devem apontar

Backpatching

- A motivação para usar backpatching é possibilitar que a geração de código para comandos de fluxo de controle seja feita em um único passo
- Para tanto, faz-se uso das definições a seguir. Considere a expressão **if (B) S:**
  - Listas de gotos são passadas como atributos sintetizados
  - Quando um goto é gerado, o alvo do goto não é especificado imediatamente
  - Cada goto é colocado em uma lista de gotos cujos labels devem ser iguais, no momento em que for definido

Backpatching

- Existe agora dois atributos sintetizados: truelist e falselist de um símbolo não-terminal B, que são usados para manter os labels dos gotos
- Existe um atributo sintetizado nextlist denotando uma lista de gotos para a instrução após o código de S
- Marcadores M e N são usados para “guardar” uma posição dentro do código que está sendo gerado
- A variável nextinstr é um ponteiro para o endereço da instrução atual

Backpatching

- A função makelist(i) cria uma nova lista de gotos contendo apenas a instrução na linha i
- merge(p1, p2) concatena as listas apontadas por p1 e p2
- backpatch(p,i) insere i como o label alvo das instruções contidas na lista p

Backpatching

Produção	Regra Semântica
$B \rightarrow B_1 \,\|\|\, M\ B_2$	backpatch(B_1.falselist, M.inst) B.truelist=merge(B_1.truelist, B_2.truelist) B.falselist=B_2.falselist
$B \rightarrow B_1 \,\&\&\, M\ B_2$	backpatch(B_1.truelist, M.inst) B.truelist=B_2.truelist B.falselist=merge(B_1.falselist, B_2.falselist)
$B \rightarrow E_1$ rel E_2	B.truelist=makelist(nextinstr) B.falselist=makelist(++nextinstr) B.code= gen('if' E_1.addr rel.op E_2.addr 'goto _') || gen('goto _')
$M \rightarrow \varepsilon$	M.inst=++nextinstr

Backpatching

Produção	Regra Semântica
$S \rightarrow$ if (B) M S_1	backpatch(B.truelist, M.inst) S.nextlist=merge(B.falselist, S_1.nextlist)
$S \rightarrow$ if (B) M_1 S_1 N_1 else M_2 S_2	backpatch(B.truelist, M_1.inst) backpatch(B.falselist, M_2.inst) temp=merge(S_1.nextlist, N.nextlist) S.nextlist=merge(temp, S_2.nextlist)
$M \rightarrow \varepsilon$	M.inst=++nextinstr
$N \rightarrow \varepsilon$	N.nextlist=makelist(++nextinstr) write('goto _')
$S \rightarrow$ assign	S.nextlist=NULL S.code=assign.code gen(S.code)

Backpatching

- Como seria a geração de código, baseado em backpatching, para a expressão a seguir?
- If (x<100 || x> 200 && x!=y) x=0;