Compilerconstructie

najaar 2017

`http://www.liacs.leidenuniv.nl/~vlietrvan1/coco/`

Rudy van Vliet
kamer 140 Snellius, tel. 071-527 2876
rvvliet(at)liacs(dot)nl

college 7, vrijdag 3 november 2017

\+ werkcollege

Intermediate Code Generation 2

FME Hackathon

http://www.fme.nl/hackathon

Today

- Translation of control flow
  - Top-down passing of labels (inherited attributes)
  - Backpatching (synthesized attributes)
- Translation of switch-statements

6.6 Control Flow

- Boolean expressions used to

  1. Alter flow of control: **if** (E) S

  2. Compute logical values, cf. arithmetic expressions

- Generated by

$$
\begin{array}{lcl}
P & \rightarrow & S \\
S & \rightarrow & \textbf{id} = \textbf{num}; \mid SS \\
 & \mid & \textbf{if}\ (B)\ S \mid \textbf{if}\ (B)\ S\ \textbf{else}\ S \mid \textbf{while}\ (B)\ S \\
B & \rightarrow & B||B \mid B\&\&B \mid !B \mid (B) \mid E\ \textbf{rel}\ E \mid \textbf{true} \mid \textbf{false}
\end{array}
$$

- In $B_1||B_2$, if B_1 is true, then expression is true
  In $B_1\&\&B_2$, if . . .

6.6.2 Short-Circuit Code

or jumping code

Boolean operators ||, && and ! translate into jumps

Example

```
if ( x < 100 || x > 200 && x!=y ) x = 0;
```

Precedence: || < && < !

```
      if x < 100 goto L2
      ifFalse x > 200 goto L1
      ifFalse x != y goto L1
L2:   x = 0
L1:
```

6.6.3 Flow-of-Control Statements

Translation using

- synthesized attributes $B.code$ and $S.code$
- inherited attributes (labels) $B.true$, $B.false$ and $S.next$

Syntax-Directed Definition

Production	Semantic Rules
$P \rightarrow S$	

Syntax-Directed Definition

Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$
	$P.code = S.code \ \| \ label(S.next)$
$S \rightarrow$ **id = num;**	

Syntax-Directed Definition

Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$
	$P.code = S.code \mid\mid label(S.next)$
$S \rightarrow \mathbf{id} = \mathbf{num};$	$S.code = gen(\mathbf{id}.addr\ ' =' \mathbf{num}.val);$
$S \rightarrow S_1 S_2$	

Syntax-Directed Definition

Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$
	$P.code = S.code \ \| \ label(S.next)$
$S \rightarrow$ **id = num;**	$S.code = gen(\textbf{id}.addr \ ' =' \ \textbf{num}.val);$
$S \rightarrow S_1 S_2$	$S_1.next = newlabel()$
	$S_2.next = S.next$
	$S.code = S_1.code \ \| \ label(S_1.next) \ \| \ S_2.code$
$S \rightarrow$ **if** (B) S_1	

Syntax-Directed Definition

Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$
	$P.code = S.code \ \| \ label(S.next)$
$S \rightarrow$ **id = num;**	$S.code = gen(\textbf{id}.addr \ ' =' \ \textbf{num}.val);$
$S \rightarrow S_1 S_2$	$S_1.next = newlabel()$
	$S_2.next = S.next$
	$S.code = S_1.code \ \| \ label(S_1.next) \ \| \ S_2.code$
$S \rightarrow$ **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	

Syntax-Directed Definition

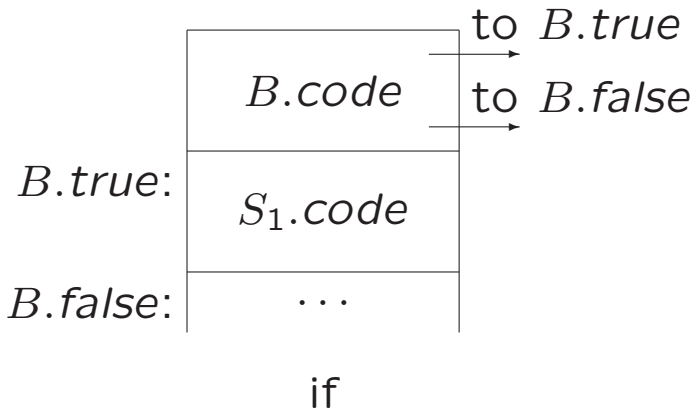
Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$
$S \rightarrow$ **id = num;**	$S.code = gen(\textbf{id}.addr\ '='\ \textbf{num}.val);$
$S \rightarrow S_1 S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$
$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ S.next) \mid\mid label(B.false) \mid\mid S_2.code$
$S \rightarrow$ **while** (B) S_1	

Syntax-Directed Definition

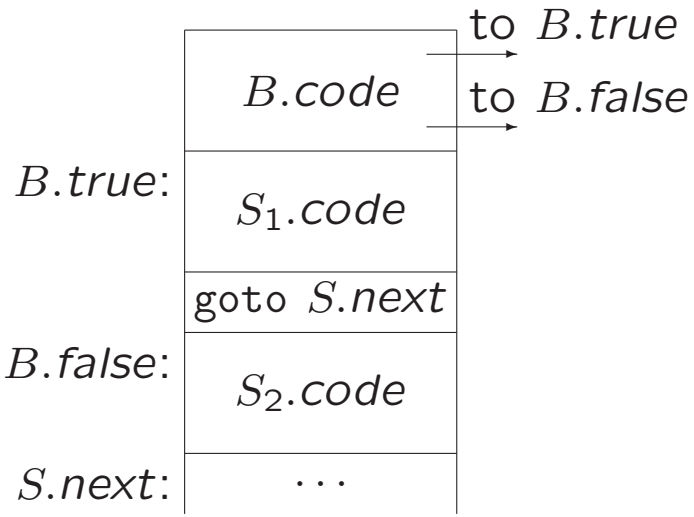
Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \;\|\|\; label(S.next)$
$S \rightarrow$ **id = num;**	$S.code = gen(\textbf{id}.addr\ '='\ \textbf{num}.val);$
$S \rightarrow S_1 S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \;\|\|\; label(S_1.next) \;\|\|\; S_2.code$
$S \rightarrow$ **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	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code \;\|\|\; label(B.true) \;\|\|\; S_1.code$ $\;\;\;\;\|\|\; gen('\texttt{goto}'\ S.next) \;\|\|\; label(B.false) \;\|\|\; S_2.code$
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \;\|\|\; B.code \;\|\|\; label(B.true)$ $\;\;\;\;\|\|\; S_1.code \;\|\|\; gen('\texttt{goto}'\ begin)$

6.6.3 Flow-of-Control Statements

$$S \rightarrow \textbf{if } (B)\ S_1$$
$$S \rightarrow \textbf{if } (B)\ S_1 \textbf{ else } S_2$$
$$S \rightarrow \textbf{while } (B)\ S_1$$



if



if-else

Syntax-Directed Definition

Production	Semantic Rules
$B \rightarrow E_1$ **rel** E_2	

Syntax-Directed Definition

Production	Semantic Rules
$B \rightarrow E_1$ **rel** E_2	$B.code = E_1.code \mid\mid E_2.code$ $\mid\mid$ *gen*('`if`' $E_1.addr$ **rel**.*op* $E_2.addr$ '`goto`' $B.true$) $\mid\mid$ *gen*('`goto`' $B.false$)
$B \rightarrow B_1 \mid\mid B_2$	

Syntax-Directed Definition

Production	Semantic Rules
$B \rightarrow E_1$ **rel** E_2	$B.code = E_1.code \mid\mid E_2.code$
	$\quad \mid\mid gen('\texttt{if}'\ E_1.addr\ \textbf{rel}.op\ E_2.addr\ '\texttt{goto}'\ B.true)$
	$\quad \mid\mid gen('\texttt{goto}'\ B.false)$
$B \rightarrow B_1 \mid\mid B_2$	$B_1.true = B.true$
	$B_1.false = newlabel()$
	$B_2.true = B.true$
	$B_2.false = B.false$
	$B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$
$B \rightarrow B_1 \&\& B_2$	$B_1.true = newlabel()$
	$B_1.false = B.false$
	$B_2.true = B.true$
	$B_2.false = B.false$
	$B.code = B_1.code \mid\mid label(B_1.true) \mid\mid B_2.code$

Syntax-Directed Definition

Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$
	$P.code = S.code \ \| \ label(S.next)$
$S \rightarrow$ **if** $(B)\ S_1$	$B.true = newlabel()$
	$B.false = S_1.next = S.next$
	$S.code = B.code \ \| \ label(B.true) \ \| \ S_1.code$
$B \rightarrow B_1 \| \| B_2$	$B_1.true = B.true$
	$B_1.false = newlabel()$
	$B_2.true = B.true$
	$B_2.false = B.false$
	$B.code = B_1.code \ \| \ label(B_1.false) \ \| \ B_2.code$
$B_1 \rightarrow E_1$ **rel** E_2	$B_1.code = E_1.code \ \| \ E_2.code$
	$\| \ gen('\texttt{if}'\ E_1.addr$ **rel**$.op\ E_2.addr\ '\texttt{goto}'\ B_1.true)$
	$\| \ gen('\texttt{goto}'\ B_1.false)$
$B_2 \rightarrow B_3 \&\& B_4$	$B_3.true = newlabel()$
	$B_3.false = B_2.false$
	$B_4.true = B_2.true$
	$B_4.false = B_2.false$
	$B_2.code = B_3.code \ \| \ label(B_3.true) \ \| \ B_4.code$
$S_1 \rightarrow$ **id = num;**	$S_1.code = gen($**id**$.addr\ '='$ **num**$.val);$

Example: `if ( x < 100 || x > 200 && x != y ) x = 0;`

6.6.5 Avoiding Redundant Gotos

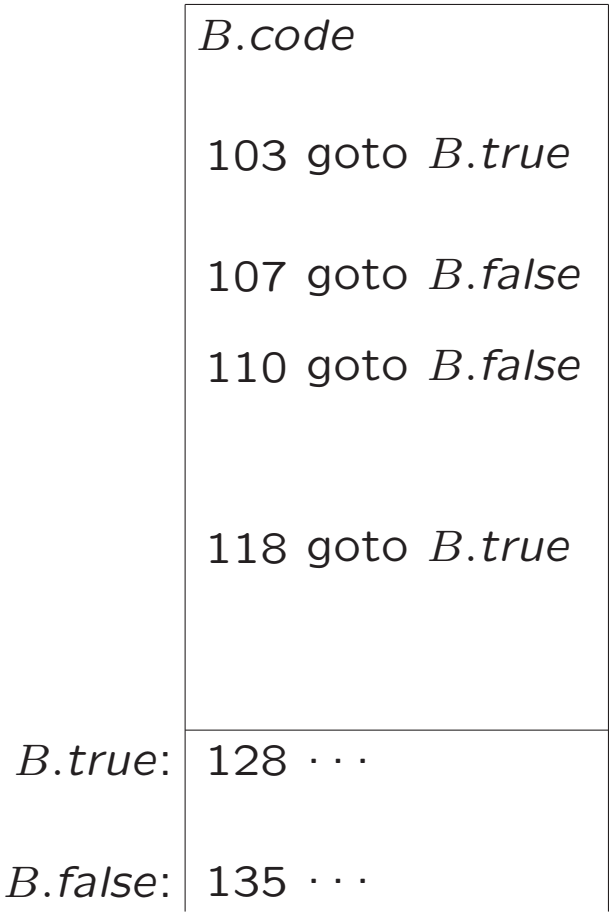
```
      if x < 100 goto L2
      goto L3
L3:   if x > 200 goto L4
      goto L1
L4:   if x != y goto L2
      goto L1
L2:   x = 0
L1:
```

Versus

```
      if x < 100 goto L2
      ifFalse x > 200 goto L1
      ifFalse x != y goto L1
L2:   x = 0
L1:
```

6.7 Backpatching

- Code generation problem:
  - Labels (addresses) that control must go to may not be known at the time that jump statements are generated
- One solution:
  - Separate pass to bind labels to addresses



6.7 Backpatching

- Code generation problem:
  - Labels (addresses) that control must go to may not be known at the time that jump statements are generated

- Other solution: backpatching
  - Generate jump statements with empty target
  - Add such statements to a list
  - Fill in targets when proper address is determined

6.7.1 One-Pass Code Generation Using Backpatching

- Synthesized attributes $B.truelist$, $B.falselist$, $S.nextlist$ containing lists of jumps

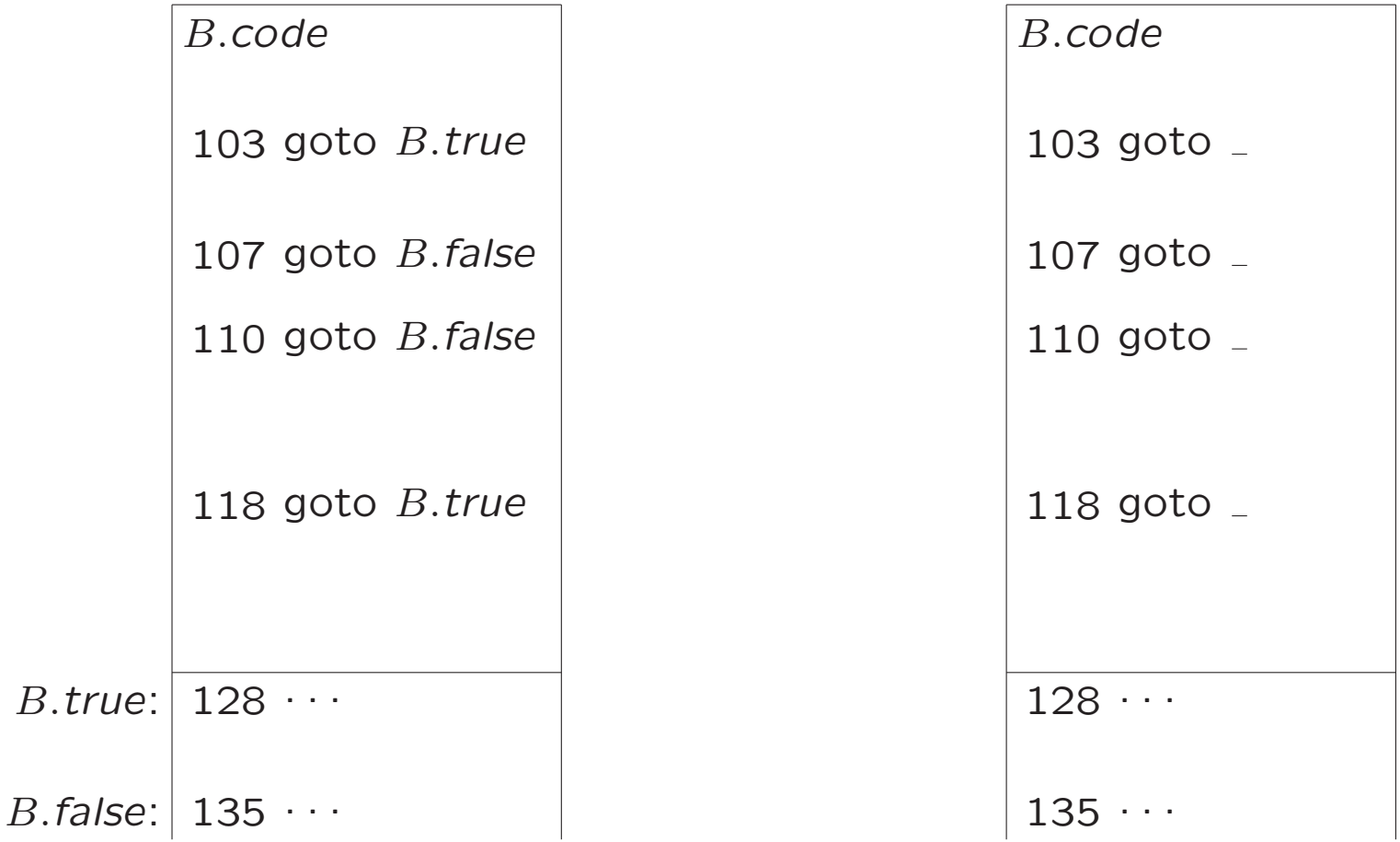
- Three functions

  1. *makelist*(i) creates new list containing index i

  2. *merge*(p_1, p_2) concatenates lists pointed to by p_1 and p_2

  3. *backpatch*(p, i) inserts i as target label for each instruction on list pointed to by p

$B.true$/$B.false$ **vs** $B.truelist$/$B.falselist$



$B.truelist = \{103, 118\}$
$B.falselist = \{107, 110\}$

Analogous: $S.next$ vs $S.nextlist$

Grammars for Backpatching

- Grammar for boolean expressions:

$$
\begin{aligned}
B &\rightarrow B_1 || M B_2 \mid B_1 \&\& M B_2 \mid !B_1 \mid (B_1) \\
&\quad \mid E_1 \textbf{ rel } E_2 \mid \textbf{true} \mid \textbf{false} \\
M &\rightarrow \epsilon
\end{aligned}
$$

  M is marker nonterminal

- Grammar for flow-of-control statements
  (marker nonterminals omitted for readability)

$$
\begin{aligned}
S &\rightarrow \textbf{if } (B) \ S_1 \mid \textbf{if } (B) \ S_1 \textbf{ else } S_2 \\
&\quad \mid \textbf{while } (B) \ S_1 \mid \{L\} \mid \textbf{id} = \textbf{num}; \\
L &\rightarrow L_1 S \mid S
\end{aligned}
$$

Translation Scheme for Backpatching

$B \rightarrow E_1$ **rel** E_2

Translation Scheme for Backpatching

$B \rightarrow E_1$ **rel** E_2 { $B.truelist = makelist(nextinstr)$;
$B.falselist = makelist(nextinstr + 1)$;
$gen('\texttt{if}'\ E_1.addr$ **rel**$.op\ E_2.addr\ '\texttt{goto}\ \_')$;
$gen('\texttt{goto}\ \_')$; }

$M \rightarrow \epsilon$ { $M.instr = nextinstr$; }

$B \rightarrow B_1 || M B_2$

Translation Scheme for Backpatching

$B \rightarrow E_1$ **rel** E_2 { $B.truelist = makelist(nextinstr)$;
$B.falselist = makelist(nextinstr + 1)$;
$gen('\texttt{if}'\ E_1.addr$ **rel**$.op\ E_2.addr\ '\texttt{goto}\ \_')$;
$gen('\texttt{goto}\ \_')$; }

$M \rightarrow \epsilon$ { $M.instr = nextinstr$; }

$B \rightarrow B_1 || M B_2$ { $backpatch(B_1.falselist, M.instr)$;
$B.truelist = merge(B_1.truelist, B_2.truelist)$;
$B.falselist = B_2.falselist$; }

$B \rightarrow B_1 \&\& M B_2$ { $backpatch(B_1.truelist, M.instr)$;
$B.truelist = B_2.truelist$;
$B.falselist = merge(B_1.falselist, B_2.falselist)$; }

$S \rightarrow$ **id = num**;

Translation Scheme for Backpatching

$B \to E_1$ **rel** E_2 { $B.truelist = makelist(nextinstr)$;
$B.falselist = makelist(nextinstr + 1)$;
$gen('\texttt{if}'\ E_1.addr$ **rel**$.op\ E_2.addr\ '\texttt{goto}\ \_')$;
$gen('\texttt{goto}\ \_')$; }

$M \to \epsilon$ { $M.instr = nextinstr$; }

$B \to B_1 || M B_2$ { $backpatch(B_1.falselist, M.instr)$;
$B.truelist = merge(B_1.truelist, B_2.truelist)$;
$B.falselist = B_2.falselist$; }

$B \to B_1 \&\& M B_2$ { $backpatch(B_1.truelist, M.instr)$;
$B.truelist = B_2.truelist$;
$B.falselist = merge(B_1.falselist, B_2.falselist)$; }

$S \to$ **id = num;** { $S.nextlist =$ **null**;
$gen($**id**$.addr\ '='$ **num**$.val)$; }

$S \to$ **if** $(B)\ MS_1$

Translation Scheme for Backpatching

$B \rightarrow E_1$ **rel** E_2	{	$B.truelist = makelist(nextinstr)$;
		$B.falselist = makelist(nextinstr + 1)$;
		$gen('\texttt{if}'\ E_1.addr$ **rel**$.op\ E_2.addr\ '\texttt{goto}\ \_')$;
		$gen('\texttt{goto}\ \_')$; }
$M \rightarrow \epsilon$	{	$M.instr = nextinstr$; }
$B \rightarrow B_1 \| \| M B_2$	{	$backpatch(B_1.falselist, M.instr)$;
		$B.truelist = merge(B_1.truelist, B_2.truelist)$;
		$B.falselist = B_2.falselist$; }
$B \rightarrow B_1 \&\& M B_2$	{	$backpatch(B_1.truelist, M.instr)$;
		$B.truelist = B_2.truelist$;
		$B.falselist = merge(B_1.falselist, B_2.falselist)$; }
$S \rightarrow$ **id** $=$ **num**;	{	$S.nextlist =$ **null**;
		$gen($**id**$.addr\ '='$ **num**$.val)$; }
$S \rightarrow$ **if** $(B)\ M S_1$	{	$backpatch(B.truelist, M.instr)$;
		$S.nextlist = merge(B.falselist, S_1.nextlist)$; }

Translation Scheme for Backpatching

$S \rightarrow$ **if** (B) MS_1 { *backpatch*(B.*truelist*, M.*instr*);
S.*nextlist* = *merge*(B.*falselist*, S_1.*nextlist*); }

$S_1 \rightarrow$ **id** = **num**; { S_1.*nextlist* = **null**;
gen(**id**.*addr* $'='$ **num**.*val*); }

$B \rightarrow B_1 || M_1 B_2$ { *backpatch*(B_1.*falselist*, M_1.*instr*);
B.*truelist* = *merge*(B_1.*truelist*, B_2.*truelist*);
B.*falselist* = B_2.*falselist*; }

$B_2 \rightarrow B_3 \&\& M_2 B_4$ { *backpatch*(B_3.*truelist*, M_2.*instr*);
B_2.*truelist* = B_4.*truelist*;
B_2.*falselist* = *merge*(B_3.*falselist*, B_4.*falselist*); }

$B \rightarrow E_1$ **rel** E_2 { B.*truelist* = *makelist*(*nextinstr*);
B.*falselist* = *makelist*(*nextinstr* + 1);
gen('`if`' E_1.*addr* **rel**.*op* E_2.*addr* '`goto _`');
gen('`goto _`'); }

$M \rightarrow \epsilon$ { M.*instr* = *nextinstr*; }

Example: `if (x < 100 || x > 200 && x != y) x = 0;`

Exercises 2 and 3

Translation Scheme for Backpatching

For Exercise 2

(Boolean Expressions)

$B \to B_1 \&\& M_1 B_2$ { *backpatch*(B_1.*truelist*, M_1.*instr*);
B.*truelist* = B_2.*truelist*;
B.*falselist* = *merge*(B_1.*falselist*, B_2.*falselist*); }

$B_2 \to (\ B_3\ )$ { B_2.*truelist* = B_3.*truelist*;
B_2.*falselist* = B_3.*falselist*; }

$B_3 \to B_4 || M_2 B_5$ { *backpatch*(B_4.*falselist*, M_2.*instr*);
B_3.*truelist* = *merge*(B_4.*truelist*, B_5.*truelist*);
B_3.*falselist* = B_5.*falselist*; }

$B \to E_1$ **rel** E_2 { B.*truelist* = *makelist*(*nextinstr*);
B.*falselist* = *makelist*(*nextinstr* + 1);
gen('`if` E_1.*addr* **rel**.*op* E_2.*addr* '`goto _`');
gen('`goto _`'); }

$M \to \epsilon$ { M.*instr* = *nextinstr*; }

Translation Scheme for Backpatching

For Exercise 3

(Flow-of-Control Statements)

$S \to \{L\}$	{	$S.nextlist = L.nextlist;$ }
$L \to L_1 M_3 S_1$	{	$backpatch(L_1.nextlist, M_3.instr);$ $L.nextlist = S_1.nextlist;$ }
$L_1 \to S_2$	{	$L_1.nextlist = S_2.nextlist;$ }
$S_2 \to$ **if** (B) $M_4 S_3$	{	$backpatch(B.truelist, M_4.instr);$ $S_2.nextlist = merge(B.falselist, S_3.nextlist);$ }
$S_3 \to$ **id = num;**	{	$S_3.nextlist =$ **null**; $gen($**id**$.addr\ '='$ **num**$.val);$ }
$M \to \epsilon$	{	$M.instr = nextinstr;$ }

6.8 Switch-Statements

switch (E)
{
    case V_1: S_1
    case V_2: S_2
        ...
    case V_{n-1}: S_{n-1}
    default S_n
}

Translation:

1. Evaluate expression E
2. Find value V_j in list of cases that matches value of E
3. Execute statement S_j

Implementation Step 2 (Back End)

- Small number of cases: sequence of conditional jumps
- Larger number of cases: hash table
- Values in a small range [min–max]: array[min-max] of labels

Translation of Switch-Statement

Straightforward

```
      code to evaluate E into t
      if t != V1 goto L1
      code for S1
      goto next
L1:   if t != V2 goto L2
      code for S2
      goto next
L2:   ...
      ...
L_{n-2}:  if t != V_{n-1} goto L_{n-1}
          code for S_(n-1)
          goto next
L_{n-1}:  code for S_n
next:
```

Translation of Switch-Statement

Easier for back end to recognize multiway branch: combine tests

```
      code to evaluate E into t
      if t = V1 goto L1
      if t = V2 goto L2
      ...
      if t = V_{n-1} goto L_{n-1}
      goto L_{n}
L1:   code for S1
      goto next
L2:   code for S2
      goto next
      ...
L_{n-1}:  code for S_(n-1)
          goto next
L_{n}:    code for S_n
next:
```

Translation of Switch-Statement

Easier to generate: tests at the end

```
      code to evaluate E into t
      goto test
L1:   code for S1
      goto next
L2:   code for S2
      goto next
      ...
L_{n-1}:  code for S_(n-1)
          goto next
L_{n}:    code for S_n
          goto next
test: if t = V1 goto L1
      if t = V2 goto L2
      ...
      if t = V_{n-1} goto L_{n-1}
      goto L_{n}
next:
```

Translation of Switch-Statement

Even easier for back end to recognize: case statements

```
      code to evaluate E into t
      goto test
L1:   code for S1
      goto next
L2:   code for S2
      goto next
      ...
L_{n-1}:  code for S_(n-1)
          goto next
L_{n}:    code for S_n
          goto next
test: case t V1 L1
      case t V2 L2
      ...
      case t V_{n-1} L_{n-1}
      case t t L_{n}
next:
```

Vervolgens. . .

- Nu: introductie opdracht 3
- Vrijdag 10 november, 11.00-12.45: practicum
- Inleveren 23 november
- Vrijdag 17 november, hoorcollege + werkcollege, beide in 402

Compilerconstructie

college 7

Intermediate Code Generation 2

Chapters for reading:

6.6–top-of-page-406,

6.7–6.7.3, 6.8