Phases of a compiler

Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text
backpatching if
### 6.7.3 Backpatching Flow-of-Control statements

| S \(\rightarrow\) if (B) S1 else S2 | \(B\).true = newlabel()  
\(B\).false = newlabel()  
S1.next = S2.next = S.next  
S.code = B.code || label(B.true) || S1.code || gen('goto', S.next) || label(B.false) || S2.code |
|-----------------------|--------------------------------------------------|
| S \(\rightarrow\) if (B) M1 S1 N else M2 S2 | backpatch(B.truelist, M1.instr)  
backpatch(B.falselist, M2.instr)  
temp = merge(S1.nextlist, N.nextlist)  
S.nextlist = merge(temp, S2.nextlist) |
| M \(\rightarrow\) ε | M.instr = nextinstr |
| N \(\rightarrow\) ε | N.nextlist = makelist(nextinstr)  
gen('goto _') |
6.7.3 Backpatching Flow-of-Control statements

S -> if (B) S1 else S2

B.true = newlabel()
B.false = newlabel()
S1.next = S2.next = S.next
S.code = B.code || label(B.true) || S1.code ||
gen('goto', S.next) || label(B.false) || S2.code
6.7.3 Backpatching Flow-of-Control statements

S → if (B) M1 S1
    N else M2 S2

M1.instr
temp = merge(S1.nextlist, N.nextlist)
S.nextlist = merge(temp, S2.nextlist)

M2.instr
N.nextlist = makelist(nextinstr)
gen('goto _')

M → ε
N → ε

B
true
false
S
M1
M2
S1
S2
N
S.next
Example 6.24 - extended

if (x < 100 || x > 200 && x != y) S1 else S2

Let's extend the Boolean expression example from part 1 by embedding that expression into an if-then-else statement (using the textbook syntax, not alpha syntax).
Example 6.24 - extended

if \((x < 100 \| x > 200 \&\& x \neq y)\) S1 else S2

100: if \(x < 100\) goto ___
101: goto 102
102: if \(x > 200\) goto 104
103: goto ___
104: if \(x \neq y\) goto ___
105: goto ___

truelist = \{100,104\}
falselist = \{103,105\}

Let's remember where we left off...
Example 6.24 - extended
if (x < 100 || x > 200 && x != y) S1 else S2

100: if x < 100 goto ___
101: goto 102
102: if x > 200 goto 104
103: goto ___
104: if x != y goto ___
105: goto ___
106: instruction for S1
107: instruction for S1
108: instruction for S1
109: instruction for S1
110: instruction for S1
111: goto ___
112: instruction for S2
113: instruction for S2
114: instruction for S2

truelist = \{100,104\}
falselist = \{103,105\}

In the example above we have not spelled out what S1 and S2 are.

Let's assume S1 requires 5 instructions and S2 requires 3 instructions.
Example 6.24 - extended
if \((x < 100 \lor x > 200 \land x \neq y)\) S1 else S2

100: if \(x < 100\) goto 106
101: goto 102
102: if \(x > 200\) goto 104
103: goto 112
104: if \(x \neq y\) goto 106
105: goto 112
106: instruction for S1
107: instruction for S1
108: instruction for S1
109: instruction for S1
110: instruction for S1
111: goto _____
112: instruction for S2
113: instruction for S2
114: instruction for S2
115:

truelist = \{100,104\} 
falselist = \{103,105\} 
nextlist = \{111\}

Embedded in the context of this if-then-else statement we can backpatch truelist and falselist from the Boolean expression, and we introduce nextlist.
backpatching while
6.7.3 Backpatching Flow-of-Control statements

The end-of-rule actions for a while statement are shown on the next slide.

Exercise:
Extend example 6.24 as a while statement where the body of the while requires 5 instructions.

```
while (x < 100 || x > 200 && x != y) S1
```

Show how the backpatching in the instruction array works. We'll review what you came up with in the Q&A session.
### 6.7.3 Backpatching Flow-of-Control statements

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow \text{while } (B) \ S_1$</td>
<td>\begin{align*} \text{begin} &amp;= \text{newlabel()} \ \text{B.true} &amp;= \text{newlabel()} \ \text{B.false} &amp;= \text{S.next()} \ \text{S1.next} &amp;= \text{begin} \ \text{S.code} &amp;= \text{label(begin)}</td>
</tr>
<tr>
<td>$S \rightarrow \text{while } M_1 \ (B) \ M_2 \ S_1$</td>
<td>\begin{align*} \text{backpatch}(S1.nextlist, \ M1.instr) \ \text{backpatch}(B.truelist, \ M2.instr) \ \text{S.nextlist} &amp;= \text{B.falselist} \ \text{gen('goto' M1.instr)} \end{align*}</td>
</tr>
<tr>
<td>$M \rightarrow \epsilon$</td>
<td>\begin{align*} \text{M.instr} &amp;= \text{nextinstr} \end{align*}</td>
</tr>
</tbody>
</table>
6.7.3 Backpatching Flow-of-Control statements

Here’s what I came up with...

...first the diagram,
...then the intermediate code.
6.7.3 Backpatching Flow-of-Control statements

\[
S \rightarrow \text{while } M1 \\
(B) M2 S1
\]

\[
\begin{align*}
\text{backpatch}(S1.\text{nextlist}, M1.\text{instr}) \\
\text{backpatch}(B.\text{truelist}, M2.\text{instr}) \\
S.\text{nextlist} = B.\text{falselist} \\
\text{gen('goto' } M1.\text{instr}) \\
M.\text{instr} = \text{nextinstr}
\end{align*}
\]

\[
M \rightarrow \epsilon
\]

© 2020 Carl Alphonce - Reproduction of this material is prohibited without the author's consent
Example 6.24 - extended

while \((x < 100 \, \| \, x > 200 \, \&\& \, x \neq y)\) \(S_1\)

100: if \(x < 100\) goto 106
101: goto 102
102: if \(x > 200\) goto 104
103: goto ___
104: if \(x \neq y\) goto 106
105: goto ___
106: instruction for \(S_1\)
107: instruction for \(S_1\)
108: instruction for \(S_1\)
109: instruction for \(S_1\)
110: instruction for \(S_1\)
111: goto 100

\(B.\text{truelist} = \{100, 104\}\)
\(S.\text{nextlist} = B.\text{falselist} = \{103, 105\}\)

Notice that we backpatch only those instructions whose targets are within the (while) instruction's code block.
backpatching for
6.7.3 Backpatching Flow-of-Control statements

Here's an exercise to work on before you watch part 2. There will be another exercise at the end of part 2 for next class.

Exercise: show how to translate a generic for statement

for ( S1 ; B ; S2 ) S3

and give the translation of this one in particular:

for ( S1 ; x < 100 || x > 200 && x != y ; S2 ) S3
Exercise: show how to translate
for ( S1 ; B ; S2 ) S3

First show how to translate a generic for statement
for ( S1 ; B ; S2 ) S3
Exercise: show how to translate
for ( S1 ; B ; S2 ) S3
Exercise: show how to translate
for (S1; B; S2) S3

Note order of S3 and S2!
for ( S1 ; M1 B ; M2 S2 ) M3 S3
How is code generated?

Note order of $S_3$ and $S_2$!
Where are jump targets?

- Instruction array
- M1 → S1
- M2 → B
- M3 → S2
- S3
Jumps

for ( S1 ; M1, B ; M2, S2 ) M3 S3

{ M1.instr = nextInstr; }
{ M2.instr = nextInstr; }

{ to mergeList ( S1.nextList, S2.nextList) }
backpatch ( t, M1.instr ) backpatch ( B.trueList, M3.instr )
backpatch ( M2.nextList, M2.instr ) S.nextList = B.falseList

gen ( 'Goto', M1.instr );
M3.instr = nextInstr; }

S1

M1 → B

M2 → S2

M3 → Goto M1

S3

Goto M2
## Jumps

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S</strong></td>
<td>$\rightarrow$</td>
</tr>
</tbody>
</table>
| for ( **S1** ; **M1** B ; **M2** S2 ) **M3** S3 | $t = \text{mergeList}(S1.\text{nextList}, S2.\text{nextList})$
|   | $\text{backpatch}(t, M1.\text{instr})$
|   | $\text{backpatch}(B.\text{truelist}, M3.\text{instr})$
|   | $\text{backpatch}(S3.\text{nextList}, M2.\text{instr})$
|   | $S.\text{nextList} = B.\text{falseList}$
|   | $\text{gen('goto', M2.\text{instr})}$
| **M1** | $\rightarrow \epsilon$
|   | $M1.\text{instr} = \text{nextinstr}$
| **M2** | $\rightarrow \epsilon$
|   | $M2.\text{instr} = \text{nextinstr}$
| **M3** | $\rightarrow \epsilon$
|   | $\text{gen('goto', M1.\text{instr})}$
|   | $M3.\text{instr} = \text{nextinstr}$
Exercise: show how to translate
for ( S1 ; B ; S2 ) S3

Second give the translation of this one in particular:
for ( S1 ; x < 100 || x > 200 && x != y ; S2 ) S3
Example 6.24 - extended

for ( S1; x < 100 || x > 200 && x != y; S2 ) S3

097: ...S1 instruction...
098: ...S1 instruction...
099: ...S1 instruction...
100: if x < 100 goto 109
101: goto 102
102: if x > 200 goto 104
103: goto ___
104: if x != y goto 109
105: goto ___
106: ...S2 instruction...
107: ...S2 instruction...
108: goto 100
109: ...S3 instruction...
110: ...S3 instruction...
111: ...S3 instruction...
112: goto 106

B.truelist = {100,104}
S.nextlist = B.falselist = {103,105}

Notice that we backpatch only those instructions whose targets are within the (for) instruction's code block.
backpatching
switch
Switch [p. 419]

**Textbook**

```
switch (E) {
    case C_1 : S_1
    case C_2 : S_2
    ...
    case C_{n-1} : S_{n-1}
    default : S_n
}
```

**Our Language**

```
switch (E) {
    case C_1 : sblock_1
    case C_2 : sblock_2
    ...
    case C_{n-1} : sblock_{n-1}
    otherwise : sblock_n
}
```
6.7.3 Backpatching Flow-of-Control statements

Exercise: show how to translate a generic switch statement:

```java
switch (E) {
    case C1 : sblock1
    case C2 : sblock2
    ...
    case Cn-1 : sblockn-1
    otherwise : sblockn
}
```