CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall
Phases of a compiler

Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text
CSE443 Compilers

Lecture 26

Part 1 - backpatching while
Part 2 - backpatching for
CSE443 Compilers

Lecture 26

Part 1 - backpatching while
Exercise from last time
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>$S \rightarrow \text{while (B) S1}$</th>
</tr>
</thead>
</table>
| $S \rightarrow \text{while M1 (B) M2 S1}$ | $\begin{align*}
\text{begin} &= \text{newlabel}() \\
\text{B.true} &= \text{newlabel}() \\
\text{B.false} &= \text{S.next()} \\
\text{S1.next} &= \text{begin} \\
\text{S.code} &= \text{label}('begin') || \text{B.code} || \text{label}('B.true') \\
&\quad || \text{S1.code} || \text{gen('goto' 'begin')}
\end{align*}$

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

© 2020 Carl Alphonce - Reproduction of this material is prohibited without the author's consent
Here's what I came up with...

...first the diagram,

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

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

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

\[ M \rightarrow \epsilon \]
\[ \text{M.instr} = \text{nextinstr} \]
Example 6.24 - extended

while \((x < 100 \lor x > 200 \land x \neq y)\) S1

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 S1
107: instruction for S1
108: instruction for S1
109: instruction for S1
110: instruction for S1
111: goto 100

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

Notice that we backpatch only those instructions whose targets are within the (while) instruction's code block.
As requested by students in the last Q&A session, 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

\[ \text{for ( } S1 ; B ; S2 \text{ ) } S3 \]

and give the translation of this one in particular:

\[ \text{for ( } S1 ; x < 100 \text{ || } x > 200 \text{ && } x \neq y \text{ ) } S3 \]
End of part 1
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?

Instruction array

Note order of S3 and S2!
Where are jump targets?

Instruction array
Jumps

Instruction array

M1 → S1 → nextList → M1
M2 → B → trueList → M3
M3 → goto M1 → nextList → M1
M4 → goto M2 → nextList → M2
Jumps

for (s1; M, B; M2, S2) M3 S3

\{ M1. instr = nextInstr; \}
\{ M2. instr = nextInstr; \}
\{ gen('Goto', M1.instr); \}
\{ M3. instr = nextInstr; \}

{\text{mergeList}(s1.nextList, s2.nextList)}
\text{backpatch}(\ell, M1.instr)
\text{backpatch}(B.trueList, M3.instr)
\text{backpatch}(M2.nextList, M2.instr)
\text{S.nextList = B.falseList}
\text{go('Goto', M1.instr)}
\text{nextList} \rightarrow M1

trueList \rightarrow M3
\text{S.nextList = B.falseList}
\text{nextList} \rightarrow M1

Goto M1
S2
M2
B
S1
M1
Goto M1
M3
Goto M2
S3
## Jumps

<table>
<thead>
<tr>
<th>Transition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \rightarrow )</td>
<td>for ( ( S1 ); ( M1 ) ( B ); ( M2 ) ( S2 ) ) ( M3 ) ( S3 )</td>
</tr>
<tr>
<td>( M1 \rightarrow \epsilon )</td>
<td>( M1.\text{instr} = \text{nextInstr} )</td>
</tr>
<tr>
<td>( M2 \rightarrow \epsilon )</td>
<td>( M2.\text{instr} = \text{nextInstr} )</td>
</tr>
</tbody>
</table>
| \( M3 \rightarrow \epsilon \) | \( \text{gen('goto', M1.instr)} \)
\( M3.\text{instr} = \text{nextInstr} \) |
| \( t = \text{mergeList}(S1.nextList, S2.nextList) \) | backpatch(\( t \), \( M1.\text{instr} \)) |
| \( \) | backpatch(\( B.\text{truelist} \), \( M3.\text{instr} \)) |
| \( \) | backpatch(\( S3.nextList \), \( M2.\text{instr} \)) |
| \( S.\text{nextList} = B.\text{falseList} \) | \( \) |
| \( \) | \( \text{gen('goto', M2.instr)} \) |
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.
Switch [p. 419]

Textbook

switch (E) {
    case C₁ : S₁
    case C₂ : S₂
    ...
    case Cₙ₋₁ : Sₙ₋₁
    default : Sₙ
}

Our Language

switch (E) {
    case C₁ : sblock₁
    case C₂ : sblock₂
    ...
    case Cₙ₋₁ : sblockₙ₋₁
    otherwise : sblockₙ
}
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
}
```
End of part 2