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
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>Rule</th>
<th>Code</th>
</tr>
</thead>
</table>
| $S \rightarrow \text{while (B) } S_1$ | \[
\begin{align*}
\text{begin} &= \text{newlabel()} \\
\text{B.true} &= \text{newlabel()} \\
\text{B.false} &= S.\text{next()} \\
S_1.\text{next} &= \text{begin} \\
S.\text{code} &= \text{label}(\text{begin}) \ || \ B.\text{code} \ || \ \text{label}(\text{B.true}) \\
& \ || \ S_1.\text{code} \ || \ \text{gen('goto' begin)}
\end{align*}
\] |
| $S \rightarrow \text{while } M_1 \ (B) \ M_2 \ S_1$ | \[
\begin{align*}
\text{backpatch}(S_1.\text{nextlist}, \ M_1.\text{instr}) \\
\text{backpatch}(\text{B.true}list, \ M_2.\text{instr}) \\
S.\text{nextlist} &= \text{B.false}list \\
\text{gen('goto' } M_1.\text{instr})
\end{align*}
\] |
| $M \rightarrow \epsilon$ | \[
\begin{align*}
M.\text{instr} &= \text{nextinstr}
\end{align*}
\] |
6.7.3 Backpatching Flow-of-Control statements

Here's what I came up with...

...first the diagram,
...then the intermediate code.
### Backpatching

**Flow-of-Control statements**

```
S -> while M1
  (B) M2 S1
```

```
M -> ε
```

Backpatching:

- `backpatch(S1.nextlist, M1.instr)`
- `backpatch(B.truelist, M2.instr)`
- `S.nextlist = B.falselist`
- `gen('goto' M1.instr)`
- `M.instr = nextinstr`

**Diagram**

- `M1.instr` → `B`
- `B`:
  - `true` → `M2.instr`
  - `false` → `S1`
- `S1`
- `S.next`
Example 6.24 - extended
while (x < 100 || x > 200 && x != y) S1

100: if x < 100 goto 106
101: goto 102
102: if x > 200 goto 104
103: goto ___
104: if x != 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.
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

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

and give the translation of this one in particular:

\[
\text{for } ( \text{S1}; x < 100 \; || \; x > 200 \; && \; x \neq y; \text{S2} ) \text{ S3}
\]
Part 2

backpatching for
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 $S_2$!

Instruction array
Where are jump targets?

Instruction array
Jumps

Instruction array

Diagram:

- S1
  - M1
  - B
  - M2
  - S2
    - M3
    - GOTO M1
    - S3
      - M2
      - GOTO M2
  - nextList -> M1
  - trueList -> M3
  - falseList
  - nextList -> M1
  - nextList -> M2
for ( S1 ; M1, B ; M2, S2 ) M3, S3

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

\{ \text{mergeList( S1.nextList, S2.nextList) } \}
\text{backpatch}( B, M1.instr )
\text{backpatch}( M2.nextList, M2.instr )
S.nextList = B.falseList
\text{gen( GOTO, M1.instr )}
nextList \rightarrow M1
trueList \rightarrow M3
S.nextList = B.falseList
nextList \rightarrow M1
nextList \rightarrow M2
GOTO M1
GOTO M2
### Jumps

<table>
<thead>
<tr>
<th>Expression</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \rightarrow )</td>
<td>for ( S1 ; M1 B ; M2 S2 ) M3 S3</td>
</tr>
<tr>
<td>( t = \text{mergeList}(S1.nextList, S2.nextList) )</td>
<td>backpatch(t, M1.instr)</td>
</tr>
<tr>
<td> </td>
<td>backpatch(B.truelist, M3.instr)</td>
</tr>
<tr>
<td> </td>
<td>backpatch(S3.nextList, M2.instr)</td>
</tr>
<tr>
<td> </td>
<td>S.nextList = B.falseList</td>
</tr>
<tr>
<td> </td>
<td>gen('goto', M2.instr)</td>
</tr>
<tr>
<td>( M1 \rightarrow \varepsilon )</td>
<td>M1.instr = nextInstr</td>
</tr>
<tr>
<td>( M2 \rightarrow \varepsilon )</td>
<td>M2.instr = nextInstr</td>
</tr>
<tr>
<td>( M3 \rightarrow \varepsilon )</td>
<td>gen('goto', M1.instr)</td>
</tr>
<tr>
<td> </td>
<td>M3.instr = nextInstr</td>
</tr>
</tbody>
</table>
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

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**

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

**OUR LANGUAGE**

```java
switch (E) {
    case C1 : sblock1
    case C2 : sblock2
    ...
    case Cn-1 : sblockn-1
    otherwise : sblockn
}
```
Exercise: show how to translate a generic switch statement:

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