Conditional Branches

An Example Program with Conditional Branches

```
1   biggest(x: Int, y: Int, z: Int): Int
2   // PRE: true                                        (P)
3   // POST: r = max{x, y, z}                           (Q)
4   {
5       var res: Int
6       if (x >= y){
7           res = x
8       } else {
9           res = y
10      }
11      // MID:  res = max{x, y}                        (M1)
12      if (z >= res){
13          res = z
14      }
15      // MID:  res = max{z, max{x, y}}                (M2)
16      return res
17  }
```

When choosing the mid-condition on line 15, you need to be careful with what you write.

One common mistake is to write:
$res = max\{res, z\}$

The mid-condition should be a logical assertion and not code. It does not describe an assignment to the variable res, but rather makes a claim about its value being equal to that of the larger of res or z. This can only be true if $res \geq z$ (i.e. we guarantee that max{res, z} returns res) which is not general enough for our proof.

Another common mistake is to write:

$res = max\{res_{old}, z\}$

This describes the current value of res in relation to some program variable res_{old}, but there is no such variable in our code. Remember that we only use the $_{old}$ annotations in our **proofs** to distinguish between the values stored in a variable before/after some code has run. Such annotations have no place in our assertions.

Proof Obligations

The choice of mid-conditions leads to the following proof obligations:

$$P \;\wedge\; \texttt{var res:Int; if(x >= y)\{res = x\}else\{res = y\}} \;\longrightarrow\; M_1$$

$$\mathit{true} \wedge \texttt{res} = \mathit{max}\{\texttt{x},\texttt{y}\} \;\longrightarrow\; \texttt{res} = \mathit{max}\{\texttt{x},\texttt{y}\}$$

$$M_1[\texttt{res} \mapsto \texttt{res}_{old}] \;\wedge\; \texttt{if(z >= res)\{res = z\}} \;\longrightarrow\; M_2$$

$$\texttt{res}_{old} = \mathit{max}\{\texttt{x},\texttt{y}\} \wedge \texttt{res} = \mathit{max}\{\texttt{z},\texttt{res}_{old}\} \;\longrightarrow\; \texttt{res} = \mathit{max}\{\texttt{z},\mathit{max}\{\texttt{x},\texttt{y}\}\}$$

$$M_2 \;\wedge\; \texttt{return res} \;\longrightarrow\; Q$$

$$\texttt{res} = \mathit{max}\{\texttt{z},\mathit{max}\{\texttt{x},\texttt{y}\}\} \;\wedge\; \texttt{r} = \texttt{res} \;\longrightarrow\; \texttt{r} = \mathit{max}\{\texttt{x},\texttt{y},\texttt{z}\}$$

Reasoning about Conditional Branches

How do we prove that branching code satisfies its specification?

```
// PRE: P
if (cond){
    code1
} else {
    code2
}
// POST: Q
```

Using P and the case of cond as assumptions we have to show that Q holds after executing each branch code1 and code2.

$$\frac{\{\, P \wedge \texttt{cond} \,\}\;\; \texttt{code1} \;\;\{\, Q \,\} \qquad \{\, P \wedge \neg\texttt{cond} \,\}\;\; \texttt{code2} \;\;\{\, Q \,\}}{\{\, P \,\}\;\; \texttt{if(cond)\{ code1 \} else \{ code2 \}} \;\;\{\, Q \,\}}$$

Notice above that there is a slightly unfortunate clash between the syntax of our pro gramming language and that of our proof system, both making use of curly-brackets { }. In the code, these delimit the scope of our conditional and looping statements, whereas in the proof system there separate the assertions from the code. We have to take a little care not to get confused by this symbolic overloading.

Reasoning about Conditional Branches

We can introduce appropriate mid-conditions to guide our proof:

```
// PRE: P
if (cond){
    // MID: P ∧ cond
    code1
    // MID: R₁
} else {
    // MID: P ∧ ¬cond
    code2
    // MID: R₂
}
// POST: Q
```

- code1 and code2 may "assume" P and their respective case for free.
- It is the "responsibility" of both code1 and code2 to establish Q (i.e. $R_1 \longrightarrow Q$ and $R_2 \longrightarrow Q$).

When reasoning about a conditional branch we get to assume that the pre-condition holds at the start of each branch. We also get to assume that the condition cond holds in the then branch and does not hold in the else branch. We then have to show that the post-condition holds after running either branch of the code. We can either do this directly at the end of each branch (i.e. establish Q as in the rule) or we can show that the mid-condition at the end of each branch implies the post-condition Q (as we have structured things in the slide above). This latter approach can be helpful to break down the complexity of our proofs. It is possible that $P \wedge cond \implies false$ or $P \wedge \neg cond \implies false$. Whilst this might seem like a problem, it actually just means that one branch or the other is unreachable. We can carry false through our mid-conditions (the triple {false} code {false} is always true) and then the proof obligation at the end of the branch becomes $false \implies Q$ which holds trivially.

Conditional Branches - Example

Looking at the first part of our `biggest` function in more detail:

```
// PRE: true                                  (P)
var res: Int
if (x >= y){
    // MID:  x ≥ y                            (P ∧ cond)
    res = x
    // MID:  res = x  ∧  x ≥ y                (R1)
} else {
    // MID:  y > x                            (P ∧ ¬cond)
    res = y
    // MID:  res = y  ∧  y > x                (R2)
}
// MID:  res = max{x, y}                      (M1)
```

We are assuming that this code snippet exists in a program where the variables x and y have been declared, otherwise the program would not even compile.

Proof Obligations

The choice of mid-conditions leads to the following proof obligations:

$$P \land \texttt{cond} \land \texttt{res = x} \longrightarrow R_1$$
$$\texttt{x} \geq \texttt{y} \land \texttt{res} = \texttt{x} \longrightarrow \texttt{res} = \texttt{x} \land \texttt{x} \geq \texttt{y}$$

$$R_1 \longrightarrow M_1$$
$$\texttt{res} = \texttt{x} \land \texttt{x} \geq \texttt{y} \longrightarrow \texttt{res} = max\{\texttt{x}, \texttt{y}\}$$

$$P \land \neg\texttt{cond} \land \texttt{res = y} \longrightarrow R_2$$
$$\texttt{y} > \texttt{x} \land \texttt{res} = \texttt{y} \longrightarrow \texttt{res} = \texttt{y} \land \texttt{y} > \texttt{x}$$

$$R_2 \longrightarrow M_1$$
$$\texttt{res} = \texttt{y} \land \texttt{y} > \texttt{x} \longrightarrow \texttt{res} = max\{\texttt{x}, \texttt{y}\}$$

Note that if we had used M_1 for both R_1 and R_2, then we would only have two proof obligations:

$$\begin{array}{rcl} P \ \wedge \ \texttt{cond} \ \wedge \ \texttt{res = x} & \longrightarrow & M_1 \\ \texttt{x} \geq \texttt{y} \ \wedge \ \texttt{res} = \texttt{x} & \longrightarrow & \texttt{res} = max\{\texttt{x}, \texttt{y}\} \end{array}$$

and

$$\begin{array}{rcl} P \ \wedge \ \neg\texttt{cond} \ \wedge \ \texttt{res = y} & \longrightarrow & M_1 \\ \texttt{y} > \texttt{x} \ \wedge \ \texttt{res} = \texttt{y} & \longrightarrow & \texttt{res} = max\{\texttt{x}, \texttt{y}\} \end{array}$$

Whilst this would clearly be simpler in this case, in general our conditional branches might not be so similar. In fact, it is quite common for the mid-condition after the conditional branch to be a disjunction of the effects of each branch.

Conditional Branches - Example

Looking at the second part of our `biggest` method in more detail:

```
// MID:  res = max{x, y}                              (M1)
if (z >= res){
    // MID:  res = max{x, y}  ∧  z ≥ res              (M1 ∧ cond)
    res = z
    // MID:  res = z  ∧  z ≥ max{x, y}                (R3)
}
// MID:  res = max{z, max{x, y}}                      (M2)
```

We are assuming that this code snippet exists in a program where the variables x, y and z have been declared, otherwise the program would not even compile. Notice that we do not need a mid-condition for the (non-existent) else branch, although **we will still have a proof obligation that covers this case**.

Proof Obligations

The choice of mid-conditions leads to the following proof obligations:

$$\begin{array}{rcl} (M_1 \wedge \texttt{cond})[\texttt{res} \mapsto \texttt{res}_{old}] \ \wedge \ \texttt{res} = \texttt{z} & \longrightarrow & R_3 \\ \texttt{res}_{old} = max\{\texttt{x},\texttt{y}\} \wedge \texttt{z} \geq \texttt{res}_{old} \wedge \texttt{res} = \texttt{z} & \longrightarrow & \texttt{res} = \texttt{z} \wedge \texttt{z} \geq max\{\texttt{x},\texttt{y}\} \end{array}$$

$$\begin{array}{rcl} R_3 & \longrightarrow & M_2 \\ \texttt{res} = \texttt{z} \ \wedge \ \texttt{z} \geq max\{\texttt{x},\texttt{y}\} & \longrightarrow & \texttt{res} = max\{\texttt{z}, max\{\texttt{x},\texttt{y}\}\} \end{array}$$

$$\begin{array}{rcl} M_1 \ \wedge \ \neg\texttt{cond} & \longrightarrow & M_2 \\ \texttt{res} = max\{\texttt{x},\texttt{y}\} \ \wedge \ \texttt{z} < \texttt{res} & \longrightarrow & \texttt{res} = max\{\texttt{z}, max\{\texttt{x},\texttt{y}\}\} \end{array}$$

The last proof obligation above corresponds to the execution path that does not enter the conditional branch.