Formal Verification of Software – Examples

(Formal Methods in Computer Science)

Gernot Salzer

May 6, 2011

Example 1 (Semantic equivalence). Prove that the two programs

$$\begin{array}{ll} & \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od} \\ \text{and} & \mathsf{if}\ e\ \mathsf{then}\ p; \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi} \end{array}$$

are semantically equivalent, for all expressions $e \in \mathcal{E}$ and all programs $p \in \mathcal{P}$.

Solution. Let p_1 be the first program and p_2 be the second one. We have to prove $[p_1] = [p_2]$, i.e., for every state σ either $[p_1]\,\sigma$ and $[p_2]\,\sigma$ both are undefined, or $[p_1]\,\sigma = [p_2]\,\sigma$.

Proof via structural operational semantics. Let σ be an arbitrary state. We have to show that either (p_1, σ) and (p_2, σ) both loop/abort, or $(p_1, \sigma) \overset{*}{\Rightarrow} \tau$ and $(p_2, \sigma) \overset{*}{\Rightarrow} \tau$ for some state τ. Note that to ensure that both programs behave identical it is sufficient to find a configuration (q, τ) such that we have $(p_1, \sigma) \overset{*}{\Rightarrow} (q, \tau)$ and $(p_2, \sigma) \overset{*}{\Rightarrow} (q, \tau)$.

We distinguish two cases depending on the value of e.

$[e]\,\sigma = 0$:

$$\begin{aligned} (p_1, \sigma) &= (\mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}, \sigma) \\ &\Rightarrow \sigma \\ (p_2, \sigma) &= (\mathsf{if}\ e\ \mathsf{then}\ p; \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}, \sigma) \\ &\Rightarrow (\mathsf{skip}, \sigma) \\ &\Rightarrow \sigma \end{aligned}$$

$[e]\,\sigma \neq 0$:

$$\begin{aligned} (p_1, \sigma) &= (\mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}, \sigma) \\ &\Rightarrow (p; \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}, \sigma) \\ (p_2, \sigma) &= (\mathsf{if}\ e\ \mathsf{then}\ p; \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}, \sigma) \\ &\Rightarrow (p; \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}, \sigma) \end{aligned}$$

In both cases the evaluation of the programs leads to the same configuration (σ and $(p; \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}, \sigma)$, respectively). Therefore the programs p_1 and p_2 are equivalent.

Proof via natural semantics. Let σ be an arbitrary state. We use the theorem on slide 32 and distinguish again two cases.

$$\begin{aligned}
[p_2]\,\sigma &= [\textsf{if } e \textsf{ then } p; \textsf{while } e \textsf{ do } p \textsf{ od else skip fi}]\,\sigma \\
&= \begin{cases} [p; \textsf{while } e \textsf{ do } p \textsf{ od}] & \text{if } [e]\,\sigma \neq 0 \\ [\textsf{skip}]\,\sigma & \text{if } [e]\,\sigma = 0 \end{cases} \\
&= \begin{cases} [\textsf{while } e \textsf{ do } p \textsf{ od}]\,[p]\,\sigma & \text{if } [e]\,\sigma \neq 0 \\ \sigma & \text{if } [e]\,\sigma = 0 \end{cases} \\
&= [\textsf{while } e \textsf{ do } p \textsf{ od}]\,\sigma \\
&= [p_1]\,\sigma
\end{aligned}$$

Therefore we have $[p_1] = [p_2]$, i.e., the programs p_1 and p_2 are equivalent.

Example 2 (Comparison of verification methods). Show that the program

$$p \quad = \quad x := x - y;\; y := x + y;\; x := y - x$$

swaps the values of the variables x and y, using

(a) the Hoare calculus,

(b) the annotation calculus,

(c) weakest preconditions, and

(d) strongest postconditions.

This means: Find a suitable precondition F and postcondition G and prove the correctness of $\{\,F\,\}\,p\,\{\,G\,\}$ with each of the four methods.

Solution. We choose $F = (x = x_0 \wedge y = y_0)$ and $G = (x = y_0 \wedge y = x_0)$.

(a) Hoare calculus:

$$\dfrac{\dfrac{F \Rightarrow C \quad \overset{\text{(as)}}{\{\,C\,\}\,x := x - y\,\{\,B\,\}}}{\{\,F\,\}\,x := x - y\,\{\,B\,\}}\,\text{(lc)} \qquad \dfrac{\overset{\text{(as)}}{\{\,B\,\}\,y := x + y\,\{\,A\,\}} \quad \overset{\text{(as)}}{\{\,A\,\}\,x := y - x\,\{\,G\,\}}}{\{\,B\,\}\,y := x + y;\; x := y - x\,\{\,G\,\}}\,\text{(sc)}}{\{\,F\,\}\,p\,\{\,G\,\}}\,\text{(sc)}$$

where $A = G[x/y - x]$, $B = A[y/x + y]$, and $C = B[x/x - y]$. The three correctness assertions occurring as premises are axioms. It remains to prove the implication $F \Rightarrow C$.

$$\begin{aligned}
&F \Rightarrow C \\
&F \Rightarrow B[x/x - y] \\
&F \Rightarrow A[y/x + y][x/x - y]
\end{aligned}$$

$$\begin{aligned}
&F \Rightarrow G[x/y-x][y/x+y][x/x-y]\\
&F \Rightarrow (x = y_0 \wedge y = x_0)[x/y-x][y/x+y][x/x-y]\\
&F \Rightarrow (y - x = y_0 \wedge y = x_0)[y/x+y][x/x-y]\\
&F \Rightarrow ((x+y) - x = y_0 \wedge x + y = x_0)[x/x-y]\\
&F \Rightarrow (((x-y)+y) - (x-y) = y_0 \wedge (x-y) + y = x_0)
\end{aligned}$$

By elementary arithmetic and the commutativity of conjunction we see that the right-hand side is equivalent to F.

Alternative derivation: If we assume the second semicolon to be the outermost sequential composition, we obtain a different derivation, but the same axioms and implication:

$$\dfrac{\dfrac{\dfrac{F \Rightarrow C \quad \overset{\text{(as)}}{\{\,C\,\}\, x := x-y\, \{\,B\,\}}}{\{\,F\,\}\, x := x-y\, \{\,B\,\}}\,\text{(lc)} \quad \overset{\text{(as)}}{\{\,B\,\}\, y := x+y\, \{\,A\,\}}}{\{\,F\,\}\, x := x-y;\ y := x+y\, \{\,A\,\}}\,\text{(sc)} \quad \overset{\text{(as)}}{\{\,A\,\}\, x := y-x\, \{\,G\,\}}}{\{\,F\,\}\, p\, \{\,G\,\}}\,\text{(sc)}$$

(b) Annotation calculus:

$$\begin{array}{l}
\{\,F\colon x = x_0 \wedge y = y_0\,\}\\
\{\,G[x/y-x][y/x+y][x/x-y]\,\} \quad \text{as}\uparrow\\
x := x - y;\\
\{\,G[x/y-x][y/x+y]\,\} \quad \text{as}\uparrow\\
y := x + y;\\
\{\,G[x/y-x]\,\} \quad \text{as}\uparrow\\
x := y - x\\
\{\,G\colon x = y_0 \wedge y = x_0\,\}
\end{array}$$

For the proof of the implication $F \Rightarrow G[x/y{-}x][y/x{+}y][x/x{-}y]$ (logical consequence rule) see above.

(c) Weakest precondition: To prove $\{\,F\,\}\,p\,\{\,G\,\}$ via weakest preconditions, we have to prove $F \Rightarrow \mathrm{wp}(p, G)$.

$$\begin{aligned}
\mathrm{wp}(p, G) &= \mathrm{wp}(x := x-y;\ y := x+y;\ x := y-x,\ G)\\
&= \mathrm{wp}(x := x-y,\ \mathrm{wp}(y := x+y;\ x := y-x,\ G))\\
&= \mathrm{wp}(x := x-y,\ \mathrm{wp}(y := x+y,\ \mathrm{wp}(x := y-x,\ G)))\\
&= \mathrm{wp}(x := x-y,\ \mathrm{wp}(y := x+y,\ G[x/y-x]))\\
&= \mathrm{wp}(x := x-y,\ G[x/y-x][y/x+y])\\
&= G[x/y-x][y/x+y][x/x-y]
\end{aligned}$$

For the proof of the implication $F \Rightarrow \mathrm{wp}(p, G)$ see above.

Alternative calculation: If we assume the second semicolon to be the outermost sequential composition, we obtain a different calculation, but the same weakest precondition:

$$\begin{aligned}
\mathrm{wp}(p, G) &= \mathrm{wp}(x := x - y;\ y := x + y;\ x := y - x,\ G) \\
&= \mathrm{wp}(x := x - y;\ y := x + y,\ \mathrm{wp}(x := y - x,\ G)) \\
&= \mathrm{wp}(x := x - y,\ \mathrm{wp}(y := x + y,\ \mathrm{wp}(x := y - x,\ G))) \\
&= \cdots \text{ see above } \cdots
\end{aligned}$$

(d) Strongest postcondition: To prove $\{\, F \,\}\, p \,\{\, G \,\}$ via strongest postconditions, we have to prove $\mathrm{sp}(F, p) \Rightarrow G$.

$$\begin{aligned}
\mathrm{sp}(F, p) &= \mathrm{sp}(F,\ x := x - y;\ y := x + y;\ x := y - x) \\
&= \mathrm{sp}(\mathrm{sp}(F,\ x := x - y),\ y := x + y;\ x := y - x) \\
&= \mathrm{sp}(\mathrm{sp}(\mathrm{sp}(F,\ x := x - y),\ y := x + y),\ x := y - x) \\
&= \mathrm{sp}(\mathrm{sp}(\exists x'((x = x_0 \wedge y = y_0)[x/x'] \wedge x = x' - y),\ y := x + y),\ x := y - x) \\
&= \mathrm{sp}(\mathrm{sp}(\exists x'(x' = x_0 \wedge y = y_0 \wedge x = x' - y),\ y := x + y),\ x := y - x) \\
&= \mathrm{sp}(\mathrm{sp}(\exists x'(x' = x_0) \wedge y = y_0 \wedge x = x_0 - y),\ y := x + y),\ x := y - x) \\
&= \mathrm{sp}(\mathrm{sp}(\mathrm{true} \wedge y = y_0 \wedge x = x_0 - y),\ y := x + y),\ x := y - x) \\
&= \mathrm{sp}(\mathrm{sp}(y = y_0 \wedge x = x_0 - y),\ y := x + y),\ x := y - x) \\
&= \mathrm{sp}(\exists y'((y = y_0 \wedge x = x_0 - y)[y/y'] \wedge y = x + y'),\ x := y - x) \\
&= \mathrm{sp}(\exists y'(y' = y_0 \wedge x = x_0 - y' \wedge y = x + y'),\ x := y - x) \\
&= \mathrm{sp}(\exists y'(y' = y_0 \wedge x = x_0 - y_0 \wedge y = x + y_0),\ x := y - x) \\
&= \mathrm{sp}(\exists y'(y' = y_0) \wedge x = x_0 - y_0 \wedge y = x + y_0,\ x := y - x) \\
&= \mathrm{sp}(\mathrm{true} \wedge x = x_0 - y_0 \wedge y = x + y_0,\ x := y - x) \\
&= \mathrm{sp}(x = x_0 - y_0 \wedge y = x + y_0,\ x := y - x) \\
&= \exists x'((x = x_0 - y_0 \wedge y = x + y_0)[x/x'] \wedge x = y - x') \\
&= \exists x'(x' = x_0 - y_0 \wedge y = x' + y_0 \wedge x = y - x') \\
&= \exists x'(x' = x_0 - y_0 \wedge y = (x_0 - y_0) + y_0 \wedge x = y - (x_0 - y_0)) \\
&= \exists x'(x' = x_0 - y_0) \wedge y = x_0 \wedge x = y - (x_0 - y_0) \\
&= \mathrm{true} \wedge y = x_0 \wedge x = x_0 - (x_0 - y_0) \\
&= (y = x_0 \wedge x = y_0)
\end{aligned}$$

The implication $\mathrm{sp}(F, p) \Rightarrow G$ is obviously valid.

Alternative calculation: If we assume the second semicolon to be the outermost sequential composition, we obtain a different calculation, but the same strongest postcondition:

$$\mathrm{sp}(F, p) = \mathrm{sp}(F,\ x := x - y;\ y := x + y;\ x := y - x)$$

$$
\begin{aligned}
&= \mathrm{sp}(\mathrm{sp}(F,\, x := x - y;\, y := x + y),\, x := y - x)\\
&= \mathrm{sp}(\mathrm{sp}(\mathrm{sp}(F,\, x := x - y),\, y := x + y),\, x := y - x)\\
&= \cdots \text{ see above } \cdots
\end{aligned}
$$

Example 3 (Integer division). Show that the following assertion is totally correct.

```
{ $F\colon m > 0 \land n \geq 0$ }
$a := 0$;
$b := n + 1$;
while $a + 1 \neq b$ do
   $d := (a + b)/2$;
   if $d * m \leq n$ then
      $a := d$;
   else
      $b := d$
   fi
od
{ $G\colon a * m \leq n < (a + 1) * m$ }
```

Describe the function computed by the program.

Solution. We prove partial correctness and termination separately. We start by adding intermediate conditions following the rules of the annotation calculus for partial correctness; see figure 1.

By the logical consequence rule, we have to prove the implications $F \Rightarrow 6$, $8 \Rightarrow 12$, $9 \Rightarrow 13$, and $4 \Rightarrow G$. To this end we have first to determine a suitable invariant. G itself is too rigid: the value of a is tightly coupled to the value of n, since a occurs in the lower as well as in the upper bound for n. Guided by the implication $4 \Rightarrow G$, we rewrite G as $am \leq n < bm \land a + 1 = b$ and try the first part as invariant: $\mathit{Inv} \equiv am \leq n < bm$.

Now we are able to prove the four implications.

$$
\begin{aligned}
F &\Rightarrow 6\\
F &\Rightarrow \mathit{Inv}[b/n+1][a/0]\\
F &\Rightarrow am \leq n < bm\,[b/n+1][a/0]\\
F &\Rightarrow am \leq n < (n+1)m\,[a/0]\\
m > 0 \land n \geq 0 &\Rightarrow 0m \leq n < (n+1)m
\end{aligned}
$$

We have to show that each formula in the conclusion is true, i.e., we have to prove $0m \leq n$ and $n < (n + 1)m$. The first is true because of the premise $n \geq 0$, whereas the second is a consequence of the premise $m > 0$. (Note that the condition $m > 0$ is essential.)

$$
\begin{gathered}
8 \Rightarrow 12\\
\mathit{Inv} \land a + 1 \neq b \land d = \tfrac{a+b}{2} \land dm \leq n \Rightarrow \mathit{Inv}[a/d]\\
am \leq n < bm \land a + 1 \neq b \land d = \tfrac{a+b}{2} \land dm \leq n \Rightarrow dm \leq n < bm
\end{gathered}
$$

```
{ $F\colon m > 0 \wedge n \geq 0$ }
{ $6\colon \mathit{Inv}[b/n+1][a/0]$ }    as ↑
$a := 0;$
{ $5\colon \mathit{Inv}[b/n+1]$ }    as ↑
$b := n+1;$
{ $1\colon \mathit{Inv}$ }
while $a + 1 \neq b$ do
  { $2\colon \mathit{Inv} \wedge a+1 \neq b$ }    wh
  $d := (a+b)/2;$
  { $7\colon \mathit{Inv} \wedge a+1 \neq b \wedge d = \frac{a+b}{2}$ }    as′ ↓
  if $d * m \leq n$ then
    { $8\colon \mathit{Inv} \wedge a+1 \neq b \wedge d = \frac{a+b}{2} \wedge dm \leq n$ }    if ↓
    { $12\colon \mathit{Inv}[a/d]$ }    as ↑
    $a := d;$
    { $10\colon \mathit{Inv}$ }    fi ↑
  else
    { $9\colon \mathit{Inv} \wedge a+1 \neq b \wedge d = \frac{a+b}{2} \wedge dm > n$ }    if ↓
    { $13\colon \mathit{Inv}[b/d]$ }    as ↑
    $b := d$
    { $11\colon \mathit{Inv}$ }    fi ↑
  fi
  { $3\colon \mathit{Inv}$ }    wh
od
{ $4\colon \mathit{Inv} \wedge a+1 = b$ }    wh
{ $G\colon am \leq n < (a+1)m$ }
```

Figure 1: Partial correctness of a program for integer division

Both conclusions, $dm \le n$ and $n < bm$, occur among the premises on the left-hand side, therefore the implication is valid.

$$\begin{aligned}
9 &\Rightarrow 13 \\
Inv \wedge a+1 \neq b \wedge d = \tfrac{a+b)}{2} \wedge dm > n &\Rightarrow Inv[b/d] \\
am \le n < bm \wedge a+1 \neq b \wedge d = \tfrac{a+b)}{2} \wedge dm > n &\Rightarrow am \le n < dm
\end{aligned}$$

Both conclusions, $am \le n$ and $n < dm$, occur among the premises on the left-hand side, therefore the implication is valid.

$$\begin{aligned}
4 &\Rightarrow G \\
Inv \wedge a+1 = b &\Rightarrow G \\
am \le n < bm \wedge a+1 = b &\Rightarrow G \\
am \le n < (a+1)m \wedge a+1 = b &\Rightarrow G \\
G \wedge a+1 = b &\Rightarrow G
\end{aligned}$$

After replacing b by $a+1$ (because of $b = a+1$) the conclusion is part of the premise, therefore the formula is valid.

Termination. The bound function (also called variant) is usually related to the loop condition, which controls termination of the loop. The condition $a+1 \neq b$ can be rewritten as $a+1 > b \vee a+1 < b$. Looking at the program, in particular at the initialisation of the variables, we see that a approximates the result from below and b from above, hence $a+1 > b$ will probably never occur. This leaves us with $a+1 < b$, which is the same as $b-a > 1$. Therefore we choose $b-a$ as bound function t, since this way the loop condition ensures the property $t \ge 0$ required for bound functions.

By rule (wht) we have to show $\{\, Inv \wedge e \wedge t = t_0 \,\}\, p \,\{\, t < t_0 \,\}$ and $Inv \Rightarrow t \ge 0$, where e is the loop condition and p is the loop body. Starting with the implication, we see that Inv is too weak: If m is negative, the invariant implies $a > b$ and therefore $t < 0$. According to the precondition we may assume $m > 0$, hence we construct the stronger invariant $Inv' \equiv Inv \wedge m > 0$. Now we succeed:

$$Inv' \equiv (am \le n < bm \wedge m > 0) \Rightarrow (am < bm \wedge m > 0) \Rightarrow a < b \equiv b - a > 0 \equiv t \ge 0 \ .$$

Note that we have to redo our proof above for partial correctness using the new invariant Inv' instead of Inv. There is not much to do, however: If an assertion $\{\, F \,\}\, p \,\{\, G \,\}$ is correct and A is a formula whose variables are not modified by program p, then the assertion $\{\, F \wedge A \,\}\, p \,\{\, G \wedge A \,\}$ is also correct. Therefore the correctness of $\{\, Inv \wedge e \,\}\, p \,\{\, Inv \,\}$ implies the correctness of $\{\, Inv' \wedge e \,\}\, p \,\{\, Inv' \,\}$, as $m > 0$ is not modified by the loop body p. Moreover, if $Inv \wedge \neg e$ implies the postcondition, then the stronger condition $Inv' \wedge \neg e$ also does. Finally, the implication $F \Rightarrow 6$ becomes $F \Rightarrow (6 \wedge m > 0)$, since m is not modified by the initialisation statements; obviously this implication is also valid.

It remains to show that t decreases with each iteration:

```
{ 14: $Inv' \wedge a+1 \neq b \wedge t = t_0$ }    wht
$d := (a+b)/2$;
{ 16: $Inv' \wedge a+1 \neq b \wedge b-a = t_0 \wedge d = \frac{a+b}{2}$ }    as' ↓
if $d * m \leq n$ then
  { 17: $Inv' \wedge a+1 \neq b \wedge b-a = t_0 \wedge d = \frac{a+b}{2} \wedge dm \leq n$ }    if ↓
  { 21: $b - d < t_0$ }    as ↑
  $a := d$;
  { 19: $b - a < t_0$ }    fi ↑
else
  { 18: $Inv' \wedge a+1 \neq b \wedge b-a = t_0 \wedge d = \frac{a+b}{2} \wedge dm > n$ }    if ↓
  { 22: $d - a < t_0$ }    as ↑
  $b := d$
  { 20: $b - a < t_0$ }    fi ↑
fi
{ 15: $t < t_0$ }    wht
```

We have to prove two implications.

$$\begin{array}{r}
17 \Rightarrow 21 \\
Inv' \wedge a+1 \neq b \wedge b-a = t_0 \wedge d = \frac{a+b}{2} \wedge dm \leq n \Rightarrow b-d < t_0 \\
a < b \wedge a+1 \neq b \Rightarrow b - \frac{a+b}{2} < b-a \\
a < b \wedge a+1 \neq b \Rightarrow a < \frac{a+b}{2}
\end{array}$$

From the second to the third line we use the fact $Inv' \Rightarrow a < b$ (see above), replace t_0 and d by the equivalent expressions and omit all irrelevant premises. The two premises $a < b$ and $a+1 \neq b$ together imply $a+2 \leq b$, hence we obtain $a < a+1 = \frac{a+(a+2)}{2} \leq \frac{a+b}{2}$.

$$\begin{array}{r}
18 \Rightarrow 22 \\
Inv' \wedge a+1 \neq b \wedge b-a = t_0 \wedge d = \frac{a+b}{2} \wedge dm > n \Rightarrow d-a < t_0 \\
a < b \Rightarrow \frac{a+b}{2} - a < b-a \\
a < b \Rightarrow \frac{a+b}{2} < b
\end{array}$$

This implication holds, since $\frac{a+b}{2} < \frac{b+b}{2} = b$

Note that this careful analysis is necessary in the context of integer division. E.g., $a < b$ does not imply $a < \frac{a+b}{2}$, the additional premise $a+1 \neq b$ is indeed needed: We have $1 < 2$ but $1 \not< \frac{1+2}{2}$.

Function computed by the program. It is sufficient to analyse the postcondition, which expresses exactly those properties of the program we are interested in. Dividing by m, we obtain $a \leq n/m < a+1$, which is the same as $a = \lfloor n/m \rfloor$. Hence the program computes integer division.

Remark. The two annotated programs above (one for partial and one for total correctness) can be combined into a single one by applying annotation rules derived from the Hoare rules wht$''$ or wht$'''$.

Example 4 (Strongest postcondition of if). Specify $\mathrm{sp}(F, \textsf{if } e \textsf{ then } p \textsf{ else } q \textsf{ fi})$, the strongest postcondition of an if-statement with respect to the precondition F. (Assume that e is a total function.)

Solution. Using the definition of sp:

$$\mathrm{sp}(\{\,F\,\}, p) = \{\, [p]\,\sigma \mid \sigma \in \{\,F\,\} \text{ and } [p]\,\sigma \text{ defined} \,\} = [p]\,(\{\,F\,\})$$

and the (natural) semantics of the if-statement:

$$[\textsf{if } e \textsf{ then } p \textsf{ else } q \textsf{ fi}]\,\sigma = \begin{cases} [p]\,\sigma & \text{if } [e]\,\sigma \neq 0 \\ [q]\,\sigma & \text{if } [e]\,\sigma = 0 \end{cases}$$

we obtain the strongest postcondition for the if-statement as follows:

$$\begin{array}{ll}
\mathrm{sp}(\{\,F\,\}, \textsf{if } e \textsf{ then } p \textsf{ else } q \textsf{ fi}) & \\
\quad = [\textsf{if } e \textsf{ then } p \textsf{ else } q \textsf{ fi}]\,(\{\,F\,\}) & \text{definition of sp} \\
\quad = [\textsf{if } e \textsf{ then } p \textsf{ else } q \textsf{ fi}]\,(\{\,F \wedge e\,\} \cup \{\,F \wedge \neg e\,\}) & \text{propositional logic and sets} \\
\quad = [\textsf{if } e \textsf{ then } p \textsf{ else } q \textsf{ fi}]\,(\{\,F \wedge e\,\}) & \text{property of functions and sets:} \\
\qquad \cup\, [\textsf{if } e \textsf{ then } p \textsf{ else } q \textsf{ fi}]\,(\{\,F \wedge \neg e\,\}) & \quad f(A \cup B) = f(A) \cup f(B) \\
\quad = [p]\,(\{\,F \wedge e\,\}) \cup [q]\,(\{\,F \wedge \neg e\,\}) & \text{semantics of the } \textsf{if}\text{-statement} \\
\quad = \mathrm{sp}(\{\,F \wedge e\,\}, p) \cup \mathrm{sp}(\{\,F \wedge \neg e\,\}, q) & \text{definition of sp}
\end{array}$$

If $\mathrm{sp}(\{\,F \wedge e\,\}, p)$ and $\mathrm{sp}(\{\,F \wedge \neg e\,\}, q)$ are given as formulas, we have

$$\mathrm{sp}(F, \textsf{if } e \textsf{ then } p \textsf{ else } q \textsf{ fi}) = \mathrm{sp}(F \wedge e, p) \vee \mathrm{sp}(F \wedge \neg e, q) \enspace .$$

Annotation calculus. Another way of deriving the strongest postcondition is to use the annotation rules if$\downarrow$ and fi$\downarrow$:

```
{ 1: $F$ }
if $e$ then
  { 2: $F \wedge e$ }   if$\downarrow$
  $p$
  { 4: $\mathrm{sp}(F \wedge e, p)$ }
else
  { 3: $F \wedge \neg e$ }   if$\downarrow$
  $q$
  { 5: $\mathrm{sp}(F \wedge \neg e, q)$ }
fi
{ 6: $\mathrm{sp}(F \wedge e, p) \vee \mathrm{sp}(F \wedge \neg e, q)$ }   fi$\downarrow$
```

This approach can be used to compute a candidate for the strongest postcondition, but it is no complete proof, since it relies on the assumption that the annotation rules yield strongest conditions. This is the case but has been neither stated nor proved in the lecture. Proving it essentially amounts to the calculation above.

Example 5 (Partial vs. total correctness). Let p be the program

```
while $x \neq 0$ do
  if $x > 0$ then
    $x := x - 2$
  else
    $x := x + 2$
  fi
od
```

For each of the four correctness assertions

$$\{ F \} \, p \, \{ \text{true} \}, \ \{ F \} \, p \, \{ \text{false} \}, \ \{ \text{true} \} \, p \, \{ F \}, \text{ and } \{ \text{false} \} \, p \, \{ F \}$$

find formulas F neither equivalent to true nor to false such that the assertion is

(a) partially but not totally correct,

(b) totally correct.

In total, these may be up to eight formulas. The formulas need not be different from each other. Some of the required formulas may not exist; in this case argue why this is so.

Solution. The program terminates if the inital value of x is an even number, and loops otherwise. If it terminates, the final state satisfies the condition $x = 0$.

	partially but not totally correct	totally correct
$\{ F \} \, p \, \{ \text{true} \}$	$x \neq 0$, $x = 1$, … (any formula satisfied by at least one odd number, except formulas equivalent to true)	$x = 0$, $x = 2$, "x is even" (or $x/2 * 2 = x$ or $\exists y(x = 2y)$ or $x \bmod 2 = 0$), … (any formula implying "x is even", except formulas equivalent to false)
$\{ F \} \, p \, \{ \text{false} \}$	$x = 1$, "x is odd", … (any formula only satisfied by odd numbers, except formulas equivalent to false)	F does not exist: For even numbers, p terminates, but no final state can satisfy false. For odd numbers, p does not terminate. The only remaining choice is $F \equiv$ false, which is excluded by the specification of the example.
$\{ \text{true} \} \, p \, \{ F \}$	$x = 0$, $x \geq 0$, … (any formula implied by $x = 0$, except formulas equivalent to true)	F does not exist, since the program does not terminate for at least one input (e.g. if x has value 1).
$\{ \text{false} \} \, p \, \{ F \}$	F does not exist, since the assertion is totally correct for arbitrary F	any formula F except those equivalent to true or false

Example 6 (Correctness of if-rule). Prove that the rule

$$\frac{\{\,F \wedge e\,\}\, p\, \{\,G\,\} \quad \{\,F \wedge \neg e\,\}\, q\, \{\,G\,\}}{\{\,F\,\}\, \mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q\ \mathsf{fi}\, \{\,G\,\}}\ \text{(if)}$$

of Hoare calculus is correct, i.e., show that the conclusion of the rule is totally correct, if both premises are totally correct. The proof should only refer to the operational semantics of TPL and to the semantics of correctness assertions, but should not use the notion of weakest pre- or strongest postcondition, and it should not assume the correctness of the alternative if-rule.

Solution. *Short argument.* Let τ be a state satisfying the precondition F. We distinguish two cases depending on the truth value of e.

Suppose that τ satisfies e. Then τ satisfies $F \wedge e$, and by the total correctness of the premise $\{\,F \wedge e\,\}\, p\, \{\,G\,\}$ we have that

- $\tau' = [p]\,\tau$ is defined, i.e., $(p,\tau) \overset{*}{\Rightarrow} \tau'$ for some state τ'.
- τ' satisfies the postcondition G.

By the semantics of TPL we obtain

$$(\mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q\ \mathsf{fi}, \tau) \Rightarrow (p,\tau) \overset{*}{\Rightarrow} \tau' \ \ .$$

Hence $[\mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q\ \mathsf{fi}]\,\tau = \tau'$ is defined and satisfies the postcondition.

For the dual case of τ not satisfying e we use the second premise and conclude in an analogous manner that the result of the if-statement is defined and satisfies G, too.

Therefore $\{\,F\,\}\, \mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q\ \mathsf{fi}\, \{\,G\,\}$ is totally correct, provided the two premises are totally correct.

Detailed proof. Let r be an abbreviation for the program "if e then p else q fi". By the definition of total correctness, we have to prove

> For all states $\sigma \in \mathcal{S}$,
> if $[F]\,\sigma$ is true, then $[r]\,\sigma$ is defined and $[G]\,[r]\,\sigma$ is true.

This universally quantified statement holds if we can prove for a fixed state τ that

> if $[F]\,\tau$ is true, then $[r]\,\tau$ is defined and $[G]\,[r]\,\tau$ is true.

This implication is valid if we can prove

$$[r]\,\tau \text{ is defined} \tag{1}$$

and

$$[G]\,[r]\,\tau \text{ is true} \tag{2}$$

assuming that

$$[F]\,\tau \text{ is true.} \tag{3}$$

We distinguish two cases.

$[e]\,\tau$ is true: By assumption, the assertion $\{\,F \wedge e\,\}\,p\,\{\,G\,\}$ (first premise of the rule) is totally correct, i.e.:

> For all states $\sigma \in \mathcal{S}$,
> if $[F \wedge e]\,\sigma$ is true, then $[p]\,\sigma$ is defined and $[G]\,[p]\,\sigma$ is true.

This universally quantified statement holds in particular for the fixed state τ from above, i.e.,

> if $[F \wedge e]\,\tau$ is true, then $[p]\,\tau$ is defined and $[G]\,[p]\,\tau$ is true.

By assumption 3 and the assumption that $[e]\,\tau$ is true we obtain that $[F \wedge e]\,\tau$ is true (semantics of propositional logic). Hence we conclude that

$$[p]\,\tau \text{ is defined} \tag{4}$$

and

$$[G]\,[p]\,\tau \text{ is true.} \tag{5}$$

$[p]\,\tau$ being defined means $(p,\tau) \overset{*}{\Rightarrow} \tau'$ and $[p]\,\tau = \tau'$ for some state τ'. Since $[e]\,\tau$ is true, we have

$$(r,\tau) \Rightarrow (p,\tau) \overset{*}{\Rightarrow} \tau' \quad \text{and} \quad [r]\,\tau = \tau'$$

(structural operational semantics of Tpl), i.e., the result of the if-statement is defined. Moreover, from statement 5 we may conclude that the postcondition of r is satisfied:

> $[G]\,[r]\,\tau = [G]\,\tau' = [G]\,[p]\,\tau$ is true.

Thus we have proved the statements 1 and 2 for the case that e evaluates to true.

$[e]\,\tau$ is false: By a similar line of reasoning we conclude from the total correctness of the second premise $\{\,F \wedge \neg e\,\}\,q\,\{\,G\,\}$ that the statements 1 and 2 also hold in the case where $[e]\,\tau$ evaluates to false.

Therefore $[r]\,\tau$ is defined and $[G]\,[r]\,\tau$ is true regardless of the value of e, i.e., $\{\,F\,\}\,r\,\{\,G\,\}$ is totally correct.

To make the structure of the proof clearer we rewrite it in the style of natural deduction. Expressions like $[F \wedge e]\,\tau$ and $[G]\,[p]\,\sigma$ are to be interpreted as *"τ is a defined state satisfying $F \wedge e$"* and *"$[p]\,\sigma$ is a defined state satisfying $[G]$"*, respectively.

$$
\begin{array}{c}
\dfrac{
\dfrac{\dfrac{
\dfrac{
\dfrac{
{}^*[F]\,\tau \qquad
\dfrac{
\dfrac{
\begin{array}{c}\{\,F \wedge e\,\}\,p\,\{\,G\,\}\\ \forall\sigma\colon [F \wedge e]\,\sigma \Rightarrow [G]\,[p]\,\sigma\end{array}
}{[F \wedge e]\,\tau \Rightarrow [G]\,[p]\,\tau}\,\forall e
}{[F]\,\tau \Rightarrow [e]\,\tau \Rightarrow [G]\,[p]\,\tau}\,\mathrm{PL}_0
}{[e]\,\tau \Rightarrow [G]\,[p]\,\tau}\,{\Rightarrow}e
\qquad
\dfrac{
{}^*[F]\,\tau \qquad
\dfrac{
\dfrac{
\begin{array}{c}\{\,F \wedge \neg e\,\}\,q\,\{\,G\,\}\\ \forall\sigma\colon [F \wedge \neg e]\,\sigma \Rightarrow [G]\,[q]\,\sigma\end{array}
}{[F \wedge \neg e]\,\tau \Rightarrow [G]\,[q]\,\tau}\,\forall e
}{[F]\,\tau \Rightarrow [\neg e]\,\tau \Rightarrow [G]\,[q]\,\tau}\,\mathrm{PL}_0
}{[\neg e]\,\tau \Rightarrow [G]\,[q]\,\tau}\,{\Rightarrow}e
}{[G]\,[\mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q\ \mathsf{fi}]\,\tau}\,\mathrm{TPL}
}{{}^*[F]\,\tau \Rightarrow [G]\,[\mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q\ \mathsf{fi}]\,\tau}\,{\Rightarrow}i
}{\forall\sigma\colon [F]\,\sigma \Rightarrow [G]\,[\mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q\ \mathsf{fi}]\,\sigma}\,\forall i
\\
\{\,F\,\}\ \mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q\ \mathsf{fi}\ \{\,G\,\}
\end{array}
$$

A short proof using two properties. In the proof below we use the following properties of correctness assertions:

- Let p and p' be programs such that $[p]\,\sigma = [p']\,\sigma$ for all states σ satisfying a formula F. (This means that p and p' are semantically equivalent for the states in $\{\,F\,\}$.) Then $\{\,F\,\}\,p\,\{\,G\,\}$ is partially/totally correct if and only if $\{\,F\,\}\,p'\,\{\,G\,\}$ is.

- Let F, F_1, and F_2 be formulas such that $\{\,F\,\} = \{\,F_1\,\} \cup \{\,F_2\,\}$, i.e., F is logically equivalent to $F_1 \vee F_2$. Then the assertion $\{\,F\,\}\,p\,\{\,G\,\}$ is partially/totally correct if and only if both assertions $\{\,F_1\,\}\,p\,\{\,G\,\}$ and $\{\,F_2\,\}\,p\,\{\,G\,\}$ are.

Now observe that by the Tpl-semantics of the if-statement,

- $[\mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q\ \mathsf{fi}]\,\sigma = [p]\,\sigma$ for all states σ satisfying $F \wedge e$ and
- $[\mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q\ \mathsf{fi}]\,\sigma = [q]\,\sigma$ for all states σ satisfying $F \wedge \neg e$.

Since F is equivalent to $(F\wedge e)\vee(F\wedge\neg e)$, we conclude by the properties and observations above that

$\{\,F\,\}\ \mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q\ \mathsf{fi}\ \{\,G\,\}$ is totally correct

if and only if

$\{\,F \wedge e\,\}\ \mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q\ \mathsf{fi}\ \{\,G\,\}$ and
$\{\,F \wedge \neg e\,\}\ \mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q\ \mathsf{fi}\ \{\,G\,\}$ are totally correct,

if and only if

$\{\,F \wedge e\,\}\,p\,\{\,G\,\}$ and
$\{\,F \wedge \neg e\,\}\,q\,\{\,G\,\}$ are totally correct.

Example 7 (Square root). Suppose we want to write a program for computing $y = \lfloor\sqrt{x}\rfloor$, the integer square root of x. This can be specified by the postcondition

$$\{\,0 \le y^2 \le x < (y+1)^2\,\}\ .$$

Derive an invariant by deleting some conjunct, and construct a correct program in a systematic way.

Solution. Given the postcondition G (=specification of the result), a program computing the result can sometimes be constructed systematically by the following steps:

1. *Choose an invariant by weakening the postcondition.* After the loop, the invariant and the negated loop condition should imply the postcondition ($\mathit{Inv}\wedge\neg e \Rightarrow G$). To obtain Inv from G, we 'split' G into Inv and $\neg e$, which amounts to weakening G. This can be done for instance by deleting some conjunct from G, maybe after rewriting the postcondition (e.g. by replacing constants by variables).

2. *Make sure the invariant holds before the loop.* Find some state that satisfies the invariant. Put statements in front of the loop that establish this state before entering the loop. If it is difficult to find such a state or if establishing the state requires complex computations, then look for a simpler state or choose a weaker invariant.

3. *Choose the loop condition.* Since *Inv* is weaker than G, the negated loop condition has to provide the missing information to conclude G from *Inv*. If *Inv* was obtained from G by deleting some conjunct, then the negation of this conjunct may serve as loop condition.

4. *Make sure the loop body advances the loop towards termination.* Determine the variables in the loop condition that may be modified within the loop. Put statements into the loop that modify one or more of the variables such that it becomes more likely that the loop condition fails.

5. *Make sure the invariant stays invariant.* After modifying some variables to make the loop eventually terminate (see previous step), the invariant will usually no longer hold. Put some more statements into the loop balancing the effect of the statements from the last step such that the invariant holds again at the end of an iteration.

Regarding this example, we observe that the postcondition can be simplified, since $y^2 \geq 0$ holds for all values of y. Hence the postcondition consists of two conjuncts, $y^2 \leq x$ and $x < (y+1)^2$, giving rise to two different programs.

Solution 1.

1. We delete the conjunct $y^2 \leq x$ and obtain the invariant $\mathit{Inv} = x < (y+1)^2$.

2. Since x is the input, we have to find some initial value for y such that *Inv* is satisfied. One possibility is to set y to the value of x, by executing the statement $y := x$.

3. As loop condition we choose the negation of the deleted conjunct, which results in the program $y := x;\ \mathsf{while}\ y^2 > x\ \mathsf{do}\ \cdots\ \mathsf{od}$.

4. Since x is the input, the loop body has to modify y to influence the loop condition. Obviously the value of y has to be decreased within the loop to let y^2 eventually drop below the value of x. The easiest way to do this is to decrement y by one, resulting in the program $y := x;\ \mathsf{while}\ y^2 > x\ \mathsf{do}\ y := y - 1;\ \cdots\ \mathsf{od}$.

5. For the invariant $x < (y+1)^2$ to hold at the end of an iteration (after decrementing y) we have to make sure that $x < y^2$ holds at the start of the iteration. Incidentally, this coincides with the loop condition, so we don't have to add further statements to the loop. Hence the final program is

$$y := x;\ \mathsf{while}\ y^2 > x\ \mathsf{do}\ y := y - 1\ \mathsf{od}\ .$$

Solution 2.

1. We delete the conjunct $x < (y+1)^2$ and obtain the invariant $\mathit{Inv} = y^2 \leq x$.
2. Since x is the input, we have to find some initial value for y such that Inv is satisfied. One possibility is to set y to zero, by executing the statement $y := 0$.
3. As loop condition we choose the negation of the deleted conjunct, which results in the program $y := 0;\ \mathsf{while}\ x \geq (y+1)^2\ \mathsf{do}\ \cdots\ \mathsf{od}$.
4. Since x is the input, the loop body has to modify y to influence the loop condition. Obviously the value of y has to be increased within the loop to let $(y+1)^2$ eventually exceed the value of x. The easiest way to do this is to increment y by one, resulting in the program $y := 0;\ \mathsf{while}\ x \geq (y+1)^2\ \mathsf{do}\ y := y+1;\ \cdots\ \mathsf{od}$.
5. For the invariant $y^2 \leq x$ to hold at the end of an iteration (after incrementing y) we have to make sure that $(y+1)^2 \leq x$ holds at the start of the iteration. Incidentally, this coincides with the loop condition, so we don't have to add further statements to the loop. Hence the final program is

$$y := 0;\ \mathsf{while}\ x \geq (y+1)^2\ \mathsf{do}\ y := y+1\ \mathsf{od}\ .$$

Correctness proofs

The programs constructed above are correct (as we will prove below) provided that we add $x \geq 0$ as precondition. Moreover, to show termination we have to strengthen the invariants by the conjuncts $y \geq 0$ and $x \geq 0$. This is no restriction, since the square root is only defined for nonnegative numbers and is a nonnegative number itself. To make the proofs less boring we use the rule wht$'''$ for the while-loop.

Solution 1.

```
{ 1: $x \geq 0$ }
{ 7: $Inv[y/x]$ }    as ↑
$y := x$;
{ $Inv$: $x < (y+1)^2 \wedge y \geq 0 \wedge x \geq 0$ }    wht'''
while $y^2 > x$ do
   { 3: $Inv \wedge y^2 > x \wedge t = t_0$ }    wht'''
   { 6: $(Inv \wedge (y^2 > x \Rightarrow 0 \leq t < t_0))[y/y-1]$ }    as ↑
   $y := y - 1$
   { 4: $Inv \wedge (y^2 > x \Rightarrow 0 \leq t < t_0)$ }    wht'''
od
{ 5: $Inv \wedge y^2 \leq x$ }    wht'''
{ 2: $y^2 \leq x < (y+1)^2$ }
```

Implication 1 ⇒ 7:

$$x \geq 0 \Rightarrow Inv[y/x]$$
$$x \geq 0 \Rightarrow x < (x+1)^2 \wedge x \geq 0 \wedge x \geq 0$$

The conjunct $x < (x+1)^2$ is valid over the integers, and $x \geq 0$ occurs as premise.

Implication 5 ⇒ 2:

$$Inv \wedge y^2 \leq x \Rightarrow y^2 \leq x < (y+1)^2$$
$$x < (y+1)^2 \wedge y \geq 0 \wedge x \geq 0 \wedge y^2 \leq x \Rightarrow y^2 \leq x < (y+1)^2$$

The conjuncts of the conclusion are part of the premise.

Implication 3 ⇒ 6: The implication is of the form

$$Inv \wedge e \wedge t = t_0 \Rightarrow Inv' \wedge (e' \Rightarrow 0 \leq t' < t_0) \ .$$

where Inv', e', and t' denote the invariant, loop condition, and variant after applying the substitution $[y/y-1]$. The validity of the formula can be shown by proving the three implications

$$Inv \wedge e \Rightarrow Inv' \quad \text{(invariance of } Inv\text{)}$$
$$Inv \wedge e \wedge e' \Rightarrow t' \geq 0 \quad (t \text{ is bounded})$$
$$Inv \wedge e \wedge e' \Rightarrow t' < t \quad (t \text{ decreases})$$

This transformation is justified by the following basic equivalences:

$$A \Rightarrow (B \wedge C) \equiv (A \Rightarrow B) \wedge (A \Rightarrow C)$$
$$A \Rightarrow (B \Rightarrow C) \equiv (A \wedge B) \Rightarrow C$$
$$A \wedge x = t \Rightarrow B \equiv A[x/t] \Rightarrow B[x/t]$$

We use $t = y^2 - x$ as bound function, which corresponds to the loop condition. We could also use $t = y^2$ or $t = y$.

$$Inv \wedge e \Rightarrow Inv'$$

$$x < (y+1)^2 \wedge y \geq 0 \wedge x \geq 0 \wedge y^2 > x \Rightarrow x < y^2 \wedge y - 1 \geq 0 \wedge x \geq 0$$

$y - 1 \geq 0$ follows from $x \geq 0$, $y \geq 0$, and $y^2 > x$; the other conjuncts of the conclusion are part of the premise.

$$Inv \wedge e \wedge e' \Rightarrow t' \geq 0$$

$$Inv \wedge e \wedge (y-1)^2 > x \Rightarrow (y-1)^2 - x \geq 0$$

$(y-1)^2 - x \geq 0$ follows from $(y-1)^2 > x$.

$$Inv \wedge e \wedge e' \Rightarrow t' < t$$

$$Inv \wedge e \wedge e' \Rightarrow (y-1)^2 - x < y^2 - x$$

$$Inv \wedge e \wedge e' \Rightarrow (y-1)^2 < y^2$$

$$x < (y+1)^2 \wedge y \geq 0 \wedge x \geq 0 \wedge y^2 > x \wedge e' \Rightarrow y \geq 1$$

The conclusion can be simplified to $y \geq 1$, which follows from $x \geq 0$, $y \geq 0$, and $y^2 > x$.

Solution 2.

```
{ 1: $x \geq 0$ }
{ 7: $Inv[y/0]$ }    as ↑
$y := 0$;
{ $Inv$: $y^2 \leq x \wedge y \geq 0 \wedge x \geq 0$ }    wht'''
while $x \geq (y+1)^2$ do
   { 3: $Inv \wedge x \geq (y+1)^2 \wedge t = t_0$ }    wht'''
   { 6: $(Inv \wedge (x \geq (y+1)^2 \Rightarrow 0 \leq t < t_0))[y/y+1]$ }    as ↑
   $y := y + 1$
   { 4: $Inv \wedge (x \geq (y+1)^2 \Rightarrow 0 \leq t < t_0)$ }    wht'''
od
{ 5: $Inv \wedge x < (y+1)^2$ }    wht'''
{ 2: $y^2 \leq x < (y+1)^2$ }
```

Implication 1 ⇒ 7:

$$x \geq 0 \Rightarrow Inv[y/0]$$

$$x \geq 0 \Rightarrow 0^2 \leq x \wedge 0 \geq 0 \wedge x \geq 0$$

Obviously valid.

Implication 5 ⇒ 2:

$$Inv \wedge x < (y+1)^2 \Rightarrow y^2 \leq x < (y+1)^2$$

$$y^2 \leq x \wedge y \geq 0 \wedge x \geq 0 \wedge x < (y+1)^2 \Rightarrow y^2 \leq x < (y+1)^2$$

The conjuncts of the conclusion are part of the premise.

Implication 3 $\Rightarrow$ 6: We split the implication again into three parts (see above). As bound function we use $t = x - (y+1)^2$ obtained from the loop condition; another choice would be $t = x - y$.

$$\mathit{Inv} \wedge e \Rightarrow \mathit{Inv}'$$

$$y^2 \leq x \wedge y \geq 0 \wedge x \geq 0 \wedge x \geq (y+1)^2 \Rightarrow (y+1)^2 \leq x \wedge (y+1) \geq 0 \wedge x \geq 0$$

$y + 1 \geq 0$ follows from $y \geq 0$; the other conjuncts of the conclusion are part of the premise.

$$\mathit{Inv} \wedge e \wedge e' \Rightarrow t' \geq 0$$

$$\mathit{Inv} \wedge e \wedge x \geq (y+2)^2 \Rightarrow x - (y+2)^2 \geq 0$$

$x - (y+2)^2 \geq 0$ follows from $x \geq (y+2)^2$.

$$\mathit{Inv} \wedge e \wedge e' \Rightarrow t' < t$$

$$\mathit{Inv} \wedge e \wedge e' \Rightarrow x - (y+2)^2 < x - (y+1)^2$$

$$\mathit{Inv} \wedge e \wedge e' \Rightarrow (y+2)^2 > (y+1)^2$$

$$y^2 \leq x \wedge y \geq 0 \wedge x \geq 0 \wedge e \wedge e' \Rightarrow y \geq -1$$

The conclusion can be simplified to $y \geq -1$, which follows from $y \geq 0$.