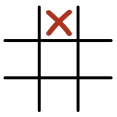
Programming 2 (SS 2023)
Saarland University
Faculty MI
Compiler Design Lab

Sample Solution 13
Verification

Prof. Dr. Sebastian Hack
Pascal Lauer, M. Sc.
Marcel Ullrich, B. Sc.

The calendar indicates which script chapters you should study in conjuction with each lecture. The exercises are designed to enhance your understanding of the lecture material and prepare you for the mini-tests and the final exam. Additional exercises can be found at the end of each chapter in the script.
The difficulty of an exercise on the sheet is determined by the number of annotated 'X' and 'O' marks in the tic-tac-toe field, with four levels (1-4) increasing by one mark per level.

Exercise 13.1:



Find a valid precondition P (as weak as possible) for the following Hoare triples.

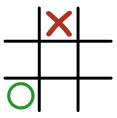
1. $\vdash \{P\}$ `x = x - 1;` $\{x == 5\}$
2. $\vdash \{P\}$ `x = 7;` $\{x == 5\}$
3. $\vdash \{P\}$ `x = 7;` $\{x == 7\}$
4. $\vdash \{P\}$ `a = a - b;` $\{a > b\}$

Solution

Using the rule [Assign], we can derive the following preconditions:

1. $P = (x == 5)[x - 1/x] = (x - 1 == 5) \Leftrightarrow (x == 6)$
2. $P = (x == 5)[7/x] = (7 == 5) \Leftrightarrow false$
3. $P = (x == 7)[7/x] = (7 == 7) \Leftrightarrow true$
4. $P = (a > b)[a - b/a] = (a - b > b) \Leftrightarrow (a > 2b)$

Exercise 13.2:

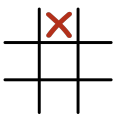


Use the rules for Hoare triples to derive a valid precondition P.

1. $\vdash \{P\}$ `a = a + 2;` $\{a == 42\}$
2. $\vdash \{P\}$ `{ a = b * c; b = c + a; c = c - 17; }` $\{$`a == b` $\wedge$ `b == c + 59`$\}$
3. $\vdash \{P\}$ `if (a == 0) abort(); else x = x + 1;` $\{x > 0\}$
4. $\vdash \{P\}$ `if (x < 0) y = -x; else y = x;` $\{$`y > 0` $\wedge$ `X * X == y * y`$\}$
5. $\vdash \{P\}$ `x = x / z;` $\{$`x != 0` $\wedge$ `z == 0`$\}$

Solution

1. Using [Assign], we can derive: $P = (a == 42)[a + 2/a] = (a + 2 == 42) \Leftrightarrow (a == 40)$
2. $P = (b * c == c + b * c \wedge c + b * c == c - 17 + 59) \Leftrightarrow (c == 0 \wedge b * c == 42) \Leftrightarrow false$
3. $P =$ (`a == 0` $\wedge\, false$) $\vee$ (`a != 0` $\wedge$ `x + 1 > 0`) $\Leftrightarrow$ (`a != 0` $\wedge$ `x > -1`)
4. $P = ((x < 0 \wedge (-x) > 0 \wedge X * X == (-x) * (-x)) \vee (\neg(x < 0) \wedge$ `x > 0` $\wedge$ `X * X == x * x`$)) \Leftrightarrow$ (`x != 0` $\wedge$ `X * X == x * x`)
5. $P =$ (`z != 0` $\wedge$ (`x / z != 0` $\wedge$ `z == 0`$)) \Leftrightarrow false$



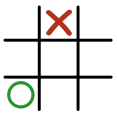
Exercise 13.3:

Which sets of states are described by the assertions *true* and $false$?

Solution

The assertion *true* describes the set of all states, i.e. Σ. The assertion $false$ describes the empty set $\emptyset$.

Exercise 13.4:



a) Give programs that make the following Hoare triples valid for any P and Q.

1) $\{P\}\ ?\ \{false\}$
2) $\{P\}\ ?\ \{true\}$
3) $\{false\}\ ?\ \{Q\}$
4) $\{true\}\ ?\ \{Q\}$

b) Prove that your programs actually satisfy the pre- and post-conditions.

Solution

1.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \{P\}\ \varepsilon\ \{P\}}\text{[Term]}}{\vdash \{P\}\ \{\}\ \{P\}}\text{[Block]} \quad P \wedge true \Rightarrow P}{\vdash \{P \wedge true\}\ \{\}\ \{P\}}\text{[Consequence]}}{\vdash \{P\}\ \texttt{while}\ (true)\ \{\}\ \{P \wedge \neg true\}}\text{[While]} \quad P \wedge \neg true \Rightarrow false}{\vdash \{P\}\ \texttt{while}\ (true)\ \{\}\ \{false\}}\text{[Consequence]} \quad \dfrac{}{\vdash \{false\}\ \varepsilon\ \{false\}}\text{[Term]}}{\vdash \{P\}\ \texttt{while}\ (true)\ \{\}\ \{false\}}\text{[Seq]}$$

2.

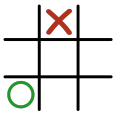
$$\dfrac{\dfrac{P \Rightarrow true \quad \dfrac{}{\vdash \{true\}\ \varepsilon\ \{true\}}\text{[Term]}}{\vdash \{P\}\ \varepsilon\ \{true\}}\text{[Consequence]} \quad \dfrac{}{\vdash \{true\}\ \varepsilon\ \{true\}}\text{[Term]}}{\vdash \{P\}\ \varepsilon\ \{true\}}\text{[Seq]}$$

3.

$$\dfrac{\dfrac{}{\vdash \{false\}\ \texttt{abort();}\ \{Q\}}\text{[Abort]} \quad \dfrac{}{\vdash \{Q\}\ \varepsilon\ \{Q\}}\text{[Term]}}{\vdash \{false\}\ \texttt{abort();}\ \{Q\}}\text{[Seq]}$$

4.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \{true\}\ \varepsilon\ \{true\}}\text{[Term]}}{\vdash \{true\}\ \{\}\ \{true\}}\text{[Block]} \quad true \wedge true \Rightarrow true}{\vdash \{true \wedge true\}\ \{\}\ \{true\}}\text{[Consequence]}}{\vdash \{true\}\ \texttt{while}\ (true)\ \{\}\ \{true \wedge \neg true\}}\text{[While]} \quad (true \wedge \neg true) \Rightarrow Q}{\vdash \{true\}\ \texttt{while}\ (true)\ \{\}\ \{Q\}}\text{[Consequence]} \quad \dfrac{}{\vdash \{Q\}\ \varepsilon\ \{Q\}}\text{[Term]}}{\vdash \{true\}\ \texttt{while}\ (true)\ \{\}\ \{Q\}}\text{[Seq]}$$



Exercise 13.5:

Verify the following program

```
1 d = x - y;
2 if (d < 0)
3     r = 0;
4 else
5     r = d;
```

against the specification (P, Q)

$$P = true \qquad Q = \underbrace{(\texttt{x - y < 0} \wedge \texttt{r = 0})}_{Q_1} \vee \underbrace{(\texttt{x - y >= 0} \wedge \texttt{r = x - y})}_{Q_2}.$$

Solution

Let $R := (\texttt{d = x - y})$.

$$\text{[SeqS]}\ \frac{\text{[Consequence]}\ \dfrac{true \implies R[x-y/d] \qquad \text{[Assign]}\ \dfrac{}{\vdash \{R[x-y/d]\}\ \texttt{d = x - y;}\ \{R\}}}{\vdash \{true\}\ \texttt{d = x - y;}\ \{R\}} \qquad \text{if-tree}}{\vdash \{true\}\ \texttt{d = x - y; if (d < 0) r = 0; else r = d;}\ \{Q\}}$$

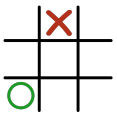
if-tree:

$$\text{[If]}\ \frac{\text{[Consequence]}\ \dfrac{d < 0 \wedge R \implies Q_1[0/r] \qquad \text{[Assign]}\ \dfrac{}{\vdash \{Q_1[0/r]\}\ \texttt{r = 0;}\ \{Q_1\}} \qquad Q_1 \implies Q}{\vdash \{\texttt{d < 0} \wedge R\}\ \texttt{r = 0;}\ \{Q\}} \qquad \text{else-tree}}{\vdash \{R\}\ \texttt{if (d < 0) r = 0; else r = d;}\ \{Q\}}$$

else-tree:

$$\text{[Consequence]}\ \frac{d \geq 0 \wedge R \implies Q_2[d/r] \qquad \text{[Assign]}\ \dfrac{}{\vdash \{Q_2[d/r]\}\ \texttt{r = d;}\ \{Q_2\}} \qquad Q_2 \implies Q}{\vdash \{\texttt{d >= 0} \wedge R\}\ \texttt{r = d;}\ \{Q\}}$$

Exercise 13.6:



Take a look at the following statement:

```
1     while (b != 0) {
2         if (a > b)
3             a = a - b;
4         else
5             b = b - a;
6     }
```

Let A and B be ghost variables which correspond to the initial values of a and b, respectively. Argue whether or not the following statements are invariants for the loop.

1. $I_1 = a > 0 \wedge b \geq 0$
2. $I_2 = a > 0 \wedge b > 0$
3. $I_3 = A \mod a \neq 0 \wedge B \mod a \neq 0$

Solution

1. Yes, I_1 is a valid loop invariant. The proof derivation of the statement given the invariance I_1 is:

$$\dfrac{\dfrac{\dfrac{\dfrac{I_1 \wedge b \neq 0 \wedge a > b \Rightarrow a - b > 0 \wedge b \geq 0 \quad \dfrac{}{\{a - b > 0 \wedge b \geq 0\}\ \texttt{a = a - b;}\ \{I_1\}}\,[\text{Assign}]}{\{I_1 \wedge b \neq 0 \wedge a > b\}\ \texttt{a = a - b;}\ \{I_1\}}\,[\text{Consequence}] \quad \dfrac{I_1 \wedge b \neq 0 \wedge a \leq b \Rightarrow a > 0 \wedge b - a \geq 0 \quad \dfrac{}{\{a > 0 \wedge b - a \geq 0\}\ \texttt{b = b - a;}\ \{I_1\}}\,[\text{Assign}]}{\{I_1 \wedge b \neq 0 \wedge a \leq b\}\ \texttt{b = b - a;}\ \{I_1\}}\,[\text{Consequence}]}{\{I_1 \wedge b \neq 0\}\ \texttt{if (a > b) a = a - b; else b = b - a;}\ \{I_1\}}\,[\text{If}]}{\{I_1 \wedge b \neq 0\}\ \texttt{\{if (a > b) a = a - b; else b = b - a;\}}\ \{I_1\}}\,[\text{Block}]}{\{I_1\}\ \texttt{while (b != 0) \{if (a > b) a = a - b; else b = b - a;\}}\ \{I_1 \wedge b = 0\}}\,[\text{While}]$$

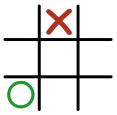
   Note that the implication $I_1 \wedge b \neq 0 \wedge a > b \Rightarrow a - b > 0 \wedge b \geq 0$ can be derived as following:

$$\begin{aligned} I_1 \wedge b \neq 0 \wedge a > b &\Leftrightarrow a > 0 \wedge b \geq 0 \wedge b \neq 0 \wedge a > b \\ &\Rightarrow b \geq 0 \wedge a > b \\ &\Leftrightarrow b \geq 0 \wedge a - b > 0 \\ &\Leftrightarrow a - b > 0 \wedge b \geq 0 \end{aligned}$$

2. No, I_2 is not a valid loop invariant, since for all $a, b > 0$ with $a = b$, the assertion $b - a > 0$ doesn't hold.

3. No, I_3 is not a valid loop invariant. For all $A, B > 0$ where the greatest common divisor of them is 1 (as a more concrete counter-example, let us say $A = 7$ and $B = 4$), the loop may take some iterations until we reach the state where $a = 1$ or $b = 1$. Then $A \mod a = A \mod 1 = 0$.

Exercise 13.7:



Take a look at the following loop, find an invariant and prove that your invariant is valid using Hoare logic derivation rules.

```
1 sum = 0;
2 i = 1;
3 while (i <= n) {
4     sum = sum + i;
5     i = i + 1;
6 }
```

Solution

The loop computes the sum of all natural numbers leading up to n. Therefore an invariant we can try is the Gaussian summation formula up to $i - 1$:

$$\texttt{I := sum = (i-1) * i / 2.}$$

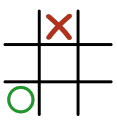
To prove the invariant valid, we need to show that it holds together with the loop condition before and after the execution of the loop body. The Hoare triple will thus consist of precondition $P = I \wedge (i<=n)$, the loop body and postcondition $Q = I$.

$$\dfrac{\dfrac{I \wedge i \leq n \Rightarrow R_2 \quad \dfrac{\dfrac{}{\{R_2\}\ \texttt{sum = sum + i;}\ \{R_1\}}\,[\text{Assign}] \quad \dfrac{}{\{R_1\}\ \texttt{i = i + 1;}\ \{I\}}\,[\text{Assign}]}{\vdash \{R_2\}\ \texttt{sum = sum + i; i = i + 1;}\ \{I\}}\,[\text{SeqS}]}{\vdash \{I \wedge (i <= n)\}\ \texttt{sum = sum + i; i = i + 1;}\ \{I\}}\,[\text{Consequence}]}{\vdash \{I \wedge (i <= n)\}\ \texttt{\{sum = sum + i; i = i + 1;\}}\ \{I\}}\,[\text{Block}]$$

R_1 and R_2 can be derived from the postcondition Q:

$$\begin{aligned} R_1 &= Q[i+1/i] = \texttt{sum = (i+1-1) * (i+1) / 2} \equiv \texttt{sum = (i+1) * i / 2} \\ R_2 &= R_1[sum + i/sum] = \texttt{sum + i = (i+1-1) * (i+1) / 2} \equiv \texttt{sum = (i-1) * i / 2.} \end{aligned}$$

Lastly, $I \wedge (i <= n)$ is stronger than R_2, since we just showed that $R_2 \equiv I$.



Exercise 13.8:

```
1 {
2     q = 0;
3     while (x >= y) {
4         q = q + 1;
5         x = x - y;
6     }
7     r = x;
8 }
```

C0 program for division. r contains the remainder of the division and q the quotient.

```
1 let div x y =
2 if x < y then
3 (1,x)
4 else
5 let (q,r) = div (x-y) y in
6 (q+1,r)
```

OCaml program for division.

$$div(x,y) := \begin{cases} (0,x) & (x < y) \\ (q+1,r) & (x \geq y), (q,r) = div(x-y,y) \end{cases}$$

Recursive function for division.
Prove the specification $0 \leq x \wedge y > 0 \rightarrow q \cdot y + r = x \wedge 0 \leq r < y$ for the OCaml program / the mathematical definition of div.
Compare your proof to the correctness proof of the C0 program done using hoare logic. What are the similarities and differences?

Solution

```
1 {
2     // {x = X & 0 <= x [2]}
3     q = 0;
4     // {q*y + x = X [3] & 0 <= x}
5     while (x >= y) {
6         // {q*y + x = X & 0 <= x & x >= y}
7         // {q*y + x + y - y = X [b] & 0 <= x-y [a]}
8         q = q + 1;
9         // {q*y + x - y = X [b] & 0 <= x-y}
10        x = x - y;
11    // {q*y + x = X & 0 <= x}
12    }
13    // {q*y + x = X & 0 <= [2] x < y [1,c]}
14    r = x;
15    // {q*y + r = X & 0 <= r < y}
16 }
```

The C0 program annotated with its correctness proof as performed using hoare logic. The numbers and letters in brackets show the correspondence to the induction proof.
In hoare logic, the case for $y < 0$ is not considered as the program diverges. Nevertheless, the post-condition is able to prove $0 < y$.
Proof by complete induction on x.

$$\text{IH: } \forall x'. x' < x \rightarrow x' > 0 \rightarrow \forall m'\, r'. (m', r') = div\ x'\ y' \rightarrow m'y + r' = x' \wedge 0 \leq r' < y$$

Case distinction on $x \geq y$.

Case $x < y$. To show:

$$0 \cdot y + x = x \quad (I)$$
$$0 \leq x < y \quad (II)$$

(I) follows by reflexivity of equality and corresponds to [3]. The first inequality of (II) follows by assumption and corresponds to [2]. The second part is a consequence of the current case distinction and corresponds to [1].
Case $x \geq y$. To show:

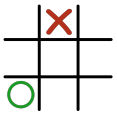
$$m' \cdot y + r' + y = x \quad (I)$$
$$0 \leq r' \geq y \quad (II)$$

for $(m', r') = div\ (x - y)\ y$. We have $x' = x - y > 0$ from $x \geq y$ (current case). This step corresponds to [a]. To apply the induction hypothesis, $x' = x - y < x$ is needed which follows from the assumption $y > 0$. We get:

$$m' \cdot y + r' = x - y \quad (IH_I)$$
$$0 \leq r' \geq y \quad (IH_{II})$$

(II) is a direct consequence of (IH_{II}) and corresponds to [c]. (I) follows from (IH_I) using arithmetic and corresponds to [b].

Exercise 13.9:



Consider the following C0 program that computes the greatest common divisor:

```
1 while (a != b)
2     if (a > b)      // \
3         a = a - b; //  \
4     else            //   } =: s
5         b = b - a; //  /
```

1. Compute the static semantics of this program with $\Gamma = \{a \mapsto int, b \mapsto int\}$.
2. Generate MIPS machine code for it according to the rules of the lecture with $offs = \{a \mapsto 0, b \mapsto 4\}$
3. Verify that this program is correct with respect to the specification (P, Q)

$$P = \texttt{a} > 0 \land \texttt{b} > 0 \land A = a \land B = b \qquad Q = \texttt{a} = gcd(A, B)$$

Hint: Pick an appropriate loop invariant.

Solution

1. Static semantics:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \texttt{a} = \texttt{int}}{\Gamma \vdash \texttt{a}}\,[\text{TVar}]}{\Gamma \vdash \texttt{a} : \texttt{int}}\,[\text{TComp}] \quad \dfrac{\dfrac{\Gamma \texttt{b} = \texttt{int}}{\Gamma \vdash \texttt{b}}\,[\text{TVar}]}{\Gamma \vdash \texttt{b} : \texttt{int}}\,[\text{TComp}] \quad \overline{int \leftrightarrow int}}{\Gamma \vdash \texttt{a != b} : \texttt{int}}\,[\text{TWhile}] \quad \dfrac{\text{Subtree 1} \quad \text{Subtree 2} \quad \text{Subtree 3}}{\Gamma \vdash \texttt{if (a > b) a = a - b; else b = b - a;}}\,[\text{TIf}]}{\Gamma \vdash \texttt{while (a != b) if (a > b) a = a - b; else b = b - a;}}\,[\text{TSeq}] \quad \dfrac{}{\Gamma \vdash \epsilon}\,[\text{TTerm}]}{\Gamma \vdash \texttt{while (a != b) if (a > b) a = a - b; else b = b - a;}}$$

Subtree 1

$$\dfrac{\dfrac{\Gamma \texttt{a} = int}{\texttt{a}}\,[\text{TVar}] \quad \dfrac{\Gamma \texttt{b} = int}{\texttt{b}}\,[\text{TVar}] \quad \overline{int \leftrightarrow int}}{\Gamma \vdash \texttt{a > b} : int}\,[\text{TComp}]$$

Subtree 2

$$\dfrac{\dfrac{\dfrac{\Gamma \texttt{a} = int}{\texttt{a} : int}\,[\text{TVar}] \quad \dfrac{\Gamma \texttt{b} = int}{\texttt{b} : int}\,[\text{TVar}]}{\texttt{a - b} : int}\,[\text{TArith}] \quad \dfrac{\Gamma \texttt{a} = int}{\texttt{a} : int}\,[\text{TVar}] \quad \overline{int \leftrightarrow int}}{\texttt{a = a - b;}}\,[\text{TAssign}]$$

Subtree 3

$$\dfrac{\dfrac{\dfrac{\Gamma \texttt{b} = \mathit{int}}{\texttt{b} : \mathit{int}}\,[\text{TVar}] \quad \dfrac{\Gamma \texttt{a} = \mathit{int}}{\texttt{a} : \mathit{int}}\,[\text{TVar}]}{\texttt{b - a} : \mathit{int}}\,[\text{TArith}] \quad \dfrac{\Gamma \texttt{b} = \mathit{int}}{\texttt{b} : \mathit{int}}\,[\text{TVar}] \quad \dfrac{}{\mathit{int} \leftrightarrow \mathit{int}}}{\texttt{b = b - a;}}\,[\text{TAssign}]$$

2. Generated code:

The derivation tree for the expression `x + 1`

$$\dfrac{\dfrac{\text{Subtree 1} \qquad \text{Subtree 2}}{\mathit{codeS}\ \mathit{offs}\ [\![\texttt{while (a != b) s}]\!] = c_1}\,[\text{CWhile}] \qquad \dfrac{}{\mathit{codeP}\ \mathit{offs}\ [\![\epsilon]\!] = \texttt{\# c\_term}}\,[\text{CTerm}]}{\mathit{codeP}\ \mathit{offs}\ [\![\texttt{while (a != b) s}]\!] = c_2}\,[\text{CSeq}]$$

Subtree 1

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathit{offs}\ a = 0}\,[\text{CVar}]}{\mathit{codeL}\ \mathit{offs}\ (\$t0 :: \$t1 :: rs)\ [\![\texttt{a}]\!]\ \mathit{int} = \texttt{\# c\_var}\ \texttt{addiu \$t0 \$sp 0}}\,[\text{CLtoR}] \quad \dfrac{\dfrac{}{\mathit{offs}\ b = 4}\,[\text{CVar}]}{\mathit{codeL}\ \mathit{offs}\ (\$t1 :: rs)\ [\![\texttt{b}]\!]\ \mathit{int} = \texttt{\# c\_var}\ \texttt{addiu \$t1 \$sp 4}}\,[\text{CLtoR}]}{\mathit{codeR}\ \mathit{offs}\ (\$t0 :: \$t1 :: rs)\ [\![\texttt{a}]\!]\ \mathit{int} = \texttt{\# c\_LtoR}\ \texttt{\# c\_var}\ \texttt{addiu \$t0 \$sp 0}\ \texttt{lw \$t0 (\$t0)}} \quad \dfrac{}{\mathit{codeR}\ \mathit{offs}\ (\$t1 :: rs)\ [\![\texttt{b}]\!]\ \mathit{int} = \texttt{\# c\_LtoR}\ \texttt{\# c\_var}\ \texttt{addiu \$t1 \$sp 4}\ \texttt{lw \$t1 (\$t1)}}\,[\text{CLtoR}]}{\mathit{codeR}\ \mathit{offs}\ (\$t0 :: \$t1 :: rs)\ [\![\texttt{a != b}]\!]\ \mathit{int} = \texttt{\# c\_Binary}\ \texttt{\# c\_LtoR}\ \texttt{\# c\_var}\ \texttt{addiu \$t0 \$sp 0}\ \texttt{lw \$t0 (\$t0)}\ \texttt{\# c\_LtoR}\ \texttt{\# c\_var}\ \texttt{addiu \$t1 \$sp 4}\ \texttt{lw \$t1 (\$t1)}\ \texttt{xor \$t0 \$t0 \$t1}}\,[\text{CBinary}]$$

Subtree 2

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathit{offs}\ a = 0}\,[\text{CVar}]}{\mathit{codeL}\ \mathit{offs}\ (\$t0 :: \$t1 :: rs)\ [\![\texttt{a}]\!]\ \mathit{int} = \texttt{\# c\_var}\ \texttt{addiu \$t0 \$sp 0}}\,[\text{CLtoR}] \quad \dfrac{\dfrac{}{\mathit{offs}\ b = 4}\,[\text{CVar}]}{\mathit{codeL}\ \mathit{offs}\ (\$t1 :: rs)\ [\![\texttt{b}]\!]\ \mathit{int} = \texttt{\# c\_var}\ \texttt{addiu \$t1 \$sp 4}}\,[\text{CLtoR}]}{\mathit{codeR}\ \mathit{offs}\ (\$t0 :: \$t1 :: rs)\ [\![\texttt{a}]\!]\ \mathit{int} = \texttt{\# c\_LtoR}\ \texttt{\# c\_var}\ \texttt{addiu \$t0 \$sp 0}\ \texttt{lw \$t0 (\$t0)}} \quad \dfrac{}{\mathit{codeR}\ \mathit{offs}\ (\$t1 :: rs)\ [\![\texttt{b}]\!]\ \mathit{int} = \texttt{\# c\_LtoR}\ \texttt{\# c\_var}\ \texttt{addiu \$t1 \$sp 4}\ \texttt{lw \$t1 (\$t1)}}\,[\text{CLtoR}]}{\dfrac{\mathit{codeR}\ \mathit{offs}\ (\$t0 :: \$t1 :: rs)\ [\![\texttt{a > b}]\!]\ \mathit{int} = c_3 \qquad \text{Subtree 3} \qquad \text{Subtree 5}}{\mathit{codeS}\ \mathit{offs}\ [\![\texttt{if (a > b) a = a - b; else b = b - a;}]\!] = c_4}\,[\text{CIf}]}\,[\text{CBinary}]$$

Subtree 3

$$\dfrac{\dfrac{\dfrac{}{\mathit{offs}\ a = 0}\,[\text{CVar}]}{\mathit{codeL}\ \mathit{offs}\ (\$t0 :: \$t1 :: rs)\ [\![\texttt{a}]\!]\ \mathit{int} = \texttt{\# c\_var}\ \texttt{addiu \$t0 \$sp 0} \qquad \text{Subtree 4}}\,[\text{CAssign}]}{\begin{array}{l}\mathit{codeS}\ \mathit{offs}\ [\![\texttt{a = a - b;}]\!] = \\ \quad \texttt{\# c\_assign}\ \texttt{\# c\_var}\ \texttt{addiu \$t0 \$sp 0}\ \texttt{\# c\_Binary}\ \texttt{\# c\_LtoR}\ \texttt{\# c\_var}\ \texttt{addiu \$t1 \$sp 0}\ \texttt{lw \$t1 (\$t1)}\ \texttt{\# c\_LtoR}\ \texttt{\# c\_var}\ \texttt{addiu \$t2 \$sp 4}\ \texttt{lw \$t2 (\$t2)}\ \texttt{sub \$t1 \$t1 \$t2}\ \texttt{sw \$t1 (\$0)}\end{array}}$$

Subtree 4

[CVar] $\dfrac{\textit{offs } a = 0}{\textit{codeL offs } (\texttt{\$t1} :: \texttt{\$t2} :: rs)\ [\![\texttt{a}]\!]\ \textit{int} =}$
```
# c_var
addiu $t1 $sp 0
```

[CLtoR] $\dfrac{}{\textit{codeR offs } (\texttt{\$t1} :: \texttt{\$t2} :: rs)\ [\![\texttt{a}]\!]\ \textit{int} =}$
```
# c_LtoR
# c_var
addiu $t1 $sp 0
lw $t1 ($t1)
```

[CVar] $\dfrac{\textit{offs } b = 4}{\textit{codeL offs } (\texttt{\$t2} :: rs)\ [\![\texttt{b}]\!]\ \textit{int} =}$
```
# c_var
addiu $t2 $sp 4
```

[CLtoR] $\dfrac{}{\textit{codeR offs } (\texttt{\$t2} :: rs)\ [\![\texttt{b}]\!]\ \textit{int} =}$
```
# c_LtoR
# c_var
addiu $t2 $sp 4
lw $t2 ($t2)
```

[CBinary] $\dfrac{}{\textit{codeR offs } (\texttt{\$t1} :: \texttt{\$t2} :: rs)\ [\![\texttt{a - b}]\!]\ \textit{int} =}$
```
# c_Binary
# c_LtoR
# c_var
addiu $t1 $sp 0
lw $t1 ($t1)
# c_LtoR
# c_var
addiu $t2 $sp 4
lw $t2 ($t2)
sub $t1 $t1 $t2
```

Subtree 5

[CVar] $\dfrac{\textit{offs } b = 4}{\textit{codeL offs } (\texttt{\$t0} :: \texttt{\$t1} :: rs)\ [\![\texttt{b}]\!]\ \textit{int} =}$
```
# c_var
addiu $t0 $sp 4
```

[CAssign] $\dfrac{\text{Subtree 6}}{\textit{codeS offs } [\![\texttt{b = b - a;}]\!] =}$
```
# c_assign
# c_var
addiu $t0 $sp 4
# c_Binary
# c_LtoR
# c_var
addiu $t1 $sp 4
lw $t1 ($t1)
# c_LtoR
# c_var
addiu $t2 $sp 0
lw $t2 ($t2)
sub $t1 $t1 $t2
sw $t1 ($0)
```

Subtree 6

[CVar] $\dfrac{\textit{offs } b = 4}{\textit{codeL offs } (\texttt{\$t1} :: \texttt{\$t2} :: rs)\ [\![\texttt{b}]\!]\ \textit{int} =}$
```
# c_var
addiu $t1 $sp 4
```

[CLtoR] $\dfrac{}{\textit{codeR offs } (\texttt{\$t1} :: \texttt{\$t2} :: rs)\ [\![\texttt{b}]\!]\ \textit{int} =}$
```
# c_LtoR
# c_var
addiu $t1 $sp 4
lw $t1 ($t1)
```

[CVar] $\dfrac{\textit{offs } a = 0}{\textit{codeL offs } (\texttt{\$t2} :: rs)\ [\![\texttt{a}]\!]\ \textit{int} =}$
```
# c_var
addiu $t2 $sp 0
```

[CLtoR] $\dfrac{}{\textit{codeR offs } (\texttt{\$t2} :: rs)\ [\![\texttt{a}]\!]\ \textit{int} =}$
```
# c_LtoR
# c_var
addiu $t2 $sp 0
lw $t2 ($t2)
```

[CBinary] $\dfrac{}{\textit{codeR offs } (\texttt{\$t1} :: \texttt{\$t2} :: rs)\ [\![\texttt{b - a}]\!]\ \textit{int} =}$
```
# c_Binary
# c_LtoR
# c_var
addiu $t1 $sp 4
lw $t1 ($t1)
# c_LtoR
# c_var
addiu $t2 $sp 0
lw $t2 ($t2)
sub $t1 $t1 $t2
```

c_1:

```
# c_while
b T
L:
# c_if
# c_binary
# c_LtoR
# c_var
addiu $t0 $sp 0
lw    $t0 $t0
# c_LtoR
# c_var
addiu $t1 $sp 4
lw    $t1 $t1

sgt   $t0 $t0 $t1
beqz  $t0 F
# c_assign
# c_var
addiu $t0 $sp 0

# c_binary
# c_LtoR
# c_var
addiu $t2 $sp 0
lw    $t2 $t2
# c_LtoR
# c_var
addiu $t1 $sp 4
lw    $t1 $t1
subiu  $t2 $t2 $t1
sw    $t2 ($t0)
b N
F:
# c_assign
# c_var
 addiu $t0 $sp 4

# c_binary
# c_LtoR
# c_var
addiu $t2 $sp 4
lw    $t2 $t2
# c_LtoR
# c_var
addiu $t1 $sp 0
lw    $t1 $t1
subiu  $t2 $t2 $t1
sw    $t2 ($t0)
N:
T:
# c_binary
# c_LtoR
# c_var
addiu $t0 $sp 0
lw    $t0 $t0
# c_LtoR
# c_var
addiu $t1 $sp 4
lw    $t1 $t1

xor   $t0 $t0 $t1
bnez  $t0 L
```

c_2:

```
# c_seq
# c_while
b T
L:
# c_if
# c_binary
# c_LtoR
# c_var
addiu $t0 $sp 0
lw    $t0 $t0
# c_LtoR
# c_var
addiu $t1 $sp 4
lw    $t1 $t1

sgt   $t0 $t0 $t1
beqz  $t0 F
# c_assign
# c_var
addiu $t0 $sp 0

# c_binary
# c_LtoR
# c_var
addiu $t2 $sp 0
lw    $t2 $t2
# c_LtoR
# c_var
addiu $t1 $sp 4
lw    $t1 $t1
subiu  $t2 $t2 $t1
sw    $t2 ($t0)
b N
F:
# c_assign
# c_var
addiu $t0 $sp 4

# c_binary
# c_LtoR
# c_var
addiu $t2 $sp 4
lw    $t2 $t2
# c_LtoR
# c_var
addiu $t1 $sp 0
lw    $t1 $t1
subiu  $t2 $t2 $t1
sw    $t2 ($t0)
N:
T:
# c_binary
# c_LtoR
# c_var
addiu $t0 $sp 0
lw    $t0 $t0
# c_LtoR
# c_var
addiu $t1 $sp 4
lw    $t1 $t1

xor   $t0 $t0 $t1
bnez  $t0 L

# c_term
```

c_3:

```
# c_binary
# c_LtoR
# c_var
  addiu $t0 $sp 0
  lw    $t0 $t0
# c_LtoR
# c_var
  addiu $t1 $sp 4
  lw    $t1 $t1
  sgt   $t0 $t0 $t1
```

c_4:

```
# c_if
# c_binary
# c_LtoR
# c_var
addiu $t0 $sp 0
lw    $t0 $t0
# c_LtoR
# c_var
addiu $t1 $sp 4
lw    $t1 $t1

sgt   $t0 $t0 $t1
beqz  $t0 F
# c_assign
# c_var
addiu $t0 $sp 0

# c_binary
# c_LtoR
# c_var
addiu $t2 $sp 0
lw    $t2 $t2
# c_LtoR
# c_var
addiu $t1 $sp 4
lw    $t1 $t1
subiu  $t2 $t2 $t1
sw    $t2 ($t0)
b N
F:
# c_assign
# c_var
addiu $t0 $sp 4

# c_binary
# c_LtoR
# c_var
addiu $t2 $sp 4
lw    $t2 $t2
# c_LtoR
# c_var
addiu $t1 $sp 0
lw    $t1 $t1
subiu  $t2 $t2 $t1
sw    $t2 ($t0)
N:
```

Complete Code:

```
1
2 # c_seq
3 # c_while
4     b T
5 L:
6 # c_if
7 # c_binary
8 # c_LtoR
9 # c_var
10     addiu $t0 $sp 0
```

```
11      lw      $t0 $t0
12  # c_LtoR
13  # c_var
14      addiu $t1 $sp 4
15      lw      $t1 $t1
16
17      sgt     $t0 $t0 $t1
18      beqz    $t0 F
19  # c_assign
20  # c_var
21    addiu $t0 $sp 0
22
23  # c_binary
24  # c_LtoR
25  # c_var
26      addiu $t2 $sp 0
27      lw      $t2 $t2
28  # c_LtoR
29  # c_var
30      addiu $t1 $sp 4
31      lw      $t1 $t1
32      subiu   $t2 $t2 $t1
33      sw      $t2 ($t0)
34      b N
35  F:
36  # c_assign
37  # c_var
38    addiu $t0 $sp 4
39
40  # c_binary
41  # c_LtoR
42  # c_var
43      addiu $t2 $sp 4
44      lw      $t2 $t2
45  # c_LtoR
46  # c_var
47      addiu $t1 $sp 0
48      lw      $t1 $t1
49      subiu   $t2 $t2 $t1
50      sw      $t2 ($t0)
51  N:
52  T:
53  # c_binary
54  # c_LtoR
55  # c_var
56      addiu $t0 $sp 0
57      lw      $t0 $t0
58  # c_LtoR
59  # c_var
60      addiu $t1 $sp 4
61      lw      $t1 $t1
62
63      xor     $t0 $t0 $t1
64      bnez    $t0 L
65
66  # c_term
```

3. Verifying: Let $I :=$ `a > 0 ∧ b > 0` $\wedge gcd($`a,b`$) =$ `gcd(A, B)`.

$$\dfrac{\dfrac{P \Rightarrow I \qquad \dfrac{\dfrac{\text{Subtree 1} \qquad \text{Subtree 2}}{\vdash \{I \wedge \texttt{a != b}\}\ \texttt{if (a > b) a = a - b; else b = b - a;}\ \{I\}}\,[\text{If}]}{\vdash \{I\}\ \texttt{while (a != b) if (a > b) a = a - b; else b = b - a;}\ \{I \wedge \texttt{a = b}\}}\,[\text{While}] \qquad I \wedge \texttt{a = b} \Rightarrow Q}{\vdash \{P\}\ \texttt{while (a != b) if (a > b) a = a - b; else b = b - a;}\ \{Q\}}\,[\text{Csq}] \qquad \dfrac{}{\vdash \{Q\}\ \epsilon\ \{Q\}}\,[\text{Term}]}{\vdash \{P\}\ \texttt{while (a != b) if (a > b) a = a - b; else b = b - a;}\ \{Q\}}\,[\text{Seq}]$$

Subtree 1

$$\dfrac{\texttt{a > b} \wedge I \wedge \texttt{a != b} \Rightarrow I[a-b/a] \qquad \dfrac{}{\vdash \{I[a-b/a]\}\ \texttt{a = a-b}\ \{I\}}\,[\text{Assign}] \qquad I \Rightarrow I}{\vdash \{\texttt{a > b} \wedge I \wedge \texttt{a != b}\}\texttt{a = a - b;}\ \{I\}}\,[\text{Csq}]$$

Subtree 2

$$\dfrac{\texttt{a <= b} \wedge I \wedge \texttt{a != b} \Rightarrow I[b-a/b] \qquad \dfrac{}{\vdash \{I[b-a/b]\}\ \texttt{b = b-a}\ \{I\}}\,[\text{Assign}] \qquad I \Rightarrow I}{\vdash \{\texttt{a <= b} \wedge I \wedge \texttt{a != b}\}\texttt{b = b - a;}\ \{I\}}\,[\text{Csq}]$$

The remaining proof obligation is (and analogously with a and b swapped):

$$I \wedge a > b \Rightarrow I[a-b/b]$$

Let $a > b > 0$ and let $x := gcd(a, b)$. We have to show:

(a) $a - b > 0$

(b) $b > 0$

(c) $gcd(a - b, b) = x$ $(= gcd(a, b) = gcd(A, B))$

The goals (a) and (b) follow directly from the assumptions.

By the definition of gcd, there are $n, m > 0$ such that $a = nx$ and $b = mx$, and x is the greatest natural number with that property. Furthermore, we have $n > m$ and $n - m > 0$ and $a - b = nx - mx = (n - m)x$. Thus x is also a common divisor of $a - b$ and b.

Assume there is another common divisor $x' > x$. Then there are $n', m' > 0$ such that:

$$a - b = n'x' \wedge b = m'x'$$

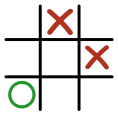
But then we would have

$$a = n'x' + m'x' = (n' + m')x'$$

and x' would also be a common divisor of a and b which is a contradiction to the assumption $x = gcd(a, b)$.

Consequently, $gcd(a - b, b) = x$.

Exercise 13.10:



We previously only talked a bit about the soundness of the rules we use. Now we want to go further and show for one rule that it is indeed correct to use it.

Sketch a soundness proof for the if-then-else rule. For this, utilize the premises of the rule, the definition of partial correctness and the semantics of C0.

Solution

The if-then-else rule is

$$\frac{\vdash \{e \wedge P\}\ \ s_1\ \ \{Q\} \qquad \vdash \{\neg e \wedge P\}\ \ s_2\ \ \{Q\}}{\vdash \{\texttt{def}\, e \wedge P\}\ \ \texttt{if}\,(e)\, s_1\, \texttt{else}\, s_2\ \ \{Q\}}\,.$$

The induction hypothesis tells us that the Hoare triples in the premises are sound, so we have $\vDash \{e \wedge P\}\ \ s_1\ \ \{Q\}$ and $\vDash \{\neg e \wedge P\}\ \ s_2\ \ \{Q\}$.

We need to show that:

$$\forall \sigma.\, (\sigma \vDash \texttt{def}\, e \wedge P) \Rightarrow \big(\forall c.\, \langle \texttt{if}\,(e)\, s_1\, \texttt{else}\, s_n | \sigma\rangle \downarrow c \Rightarrow \exists \sigma'.\, (c = \langle \varepsilon | \sigma'\rangle \wedge \sigma' \vDash Q)\big)$$

Let σ be a state such that $\sigma \vDash \texttt{def}\, e \wedge P$ and let c be a configuration such that

$$\langle \texttt{if}\,(e)\, s_1\, \texttt{else}\, s_n | \sigma\rangle \downarrow c$$

holds. Because of $\sigma \vDash \texttt{def}\, e \wedge P$ we either have $\sigma \vDash e \wedge P$ or $\sigma \vDash \neg e \wedge P$. Assume without loss of generality (the other case is analogous) that $\sigma \vDash e \wedge P$. By the semantics of C0 we know that $\langle s_1 | \sigma\rangle \downarrow c$. If $\langle s_1 | \sigma\rangle$ gets stuck or is aborted (i.e. $c = \langle p | \sigma''\rangle$ or $c = \langle \lightning | \sigma''\rangle$), then this would contradict the first induction hypothesis $\vDash \{e \wedge P\}\ \ s_1\ \ \{Q\}$. Thus, $\langle s_1 | \sigma\rangle$ must terminate properly in some state σ' with

$$c = \langle \varepsilon | \sigma'\rangle \wedge \sigma' \vDash Q$$

concluding the proof.