Assignment 0 – solution

Theodore S. Norvell

6892 Due 2014 Sept 18

Q0 [25] Division

(a) [10] List all the boolean expressions that need to be shown to be universally true in order to check that this proof outline is correct. All variables hold natural numbers. The notation $d|x$ means d divides x, i.e., there exists a natural number q such that $d \times q = x$.

```
{$d > 0$}
$r := a$
{$d|(a - r)$}
while $r \geq d$ do
    {$d|(a - r) \wedge r \geq d$}
    $e := d$
    {$d|(a - r) \wedge d|e$}
    while $r \geq e$ do
        {$d|(a - r) \wedge d|e \wedge r \geq e$}
        $r := r - e$
        {$d|(a - r) \wedge d|\,(2 \times e)$}
        $e := 2 \times e$
    end while
end while
{$d|(a - r) \wedge r < d$}
```

Solution:

From $\{d > 0\}$ $r := a$ $\{d|(a - r)\}$ we get

$$d > 0 \Rightarrow d|(a - a) \tag{0}$$

From the outer while loop we get

$$d|(a - r) \wedge r \geq d \quad \Rightarrow \quad d|(a - r) \wedge r \geq d \tag{1}$$

$$d|(a - r) \wedge \neg(r \geq d) \quad \Rightarrow \quad d|(a - r) \wedge r < d \tag{2}$$

From $\{d|(a - r) \wedge r \geq d\}$ $e := d$ $\{d|(a - r) \wedge d|e\}$ we get

$$d|(a - r) \wedge r \geq d \Rightarrow d|(a - r) \wedge d|d \tag{3}$$

From the inner while loop we get

$$d|(a-r) \wedge d|e \wedge r \geq e \quad \Rightarrow \quad d|(a-r) \wedge d|e \wedge r \geq e \tag{4}$$
$$d|(a-r) \wedge d|e \wedge \neg(r \geq e) \quad \Rightarrow \quad d|(a-r) \tag{5}$$

(N.B. The postcondition of the outer loop's body is the invariant of the outer loop; so $d|(a-r)$ is the postcondition of the inner loop.)

From $\{d|(a-r) \wedge d|e \wedge r \geq e\}$ $r := r - e$ $\{d|(a-r) \wedge d|\,(2 \times e)\}$ we get

$$d|(a-r) \wedge d|e \wedge r \geq e \Rightarrow d|(a-(r-e)) \wedge d|\,(2 \times e) \tag{6}$$

Furthermore, since r holds only natural numbers, we should also check that

$$d|(a-r) \wedge d|e \wedge r \geq e \Rightarrow (r-e) \in \mathbb{N} \tag{7}$$

(N.B. The range checks for the other three assignments are trivial, which is why I only mention the range checking condition here. Furthermore the definedness of all expressions is trivial, so I haven't included definedness checks.)

From $\{d|(a-r) \wedge d|\,(2 \times e)\}$ $e := 2 \times e$ $\{d|(a-r) \wedge d|e\}$ we get

$$d|(a-r) \wedge d|\,(2 \times e) \Rightarrow d|(a-r) \wedge d|\,(2 \times e) \tag{8}$$

(N.B. The postcondition of the inner loop's body is the invariant of the inner loop. That's why $\{d|(a-r) \wedge d|e\}$ is the postcondition here.)

(b) [10] For each of the expressions from part (a), explain why it is universally true, or why it is not. Some useful facts about the divides relation include

$$\begin{aligned} & d|d \\ & d|0 \\ d|x \wedge d|y \quad \Rightarrow \quad & d|(x-y) \\ d|x \quad \Rightarrow \quad & d|(2 \times x) \end{aligned}$$

Solution: In several places, below I use the idea of tautology. A tautology is a propositional formula that is universally true. If a formula can be derived from a tautology by substituting a boolean expression for each propositional variable, then it too will be universally true. For example: $p \wedge q \Rightarrow p$ is a tautology. (You can check this by considering all 4 cases if you like.), so $d|x \wedge d|y \Rightarrow d|x$ is universally true.

- $$d > 0 \Rightarrow d|(a-a) \tag{0}$$

  Since $a - a = 0$, and since $d|0$ is true, this simplifies to $d > 0 \Rightarrow \mathit{true}$, which is true, as $p \Rightarrow \mathit{true}$ is a tautology.[0]

[0] It's a curious thing that we never actually use the fact that $d > 0$. The reason is that we haven't worried about termination. If we were to prove that the loops terminate, we would need to use $d > 0$. Note that $0|x$ is perfectly well defined: $0|0$ is true and $0|x$ is false if $x > 0$. So we don't need $d > 0$ to know that our annotations are sensible.

- $$d|(a-r) \wedge r \geq d \Rightarrow d|(a-r) \wedge r \geq d \tag{1}$$

  This is true based on the tautology $p \Rightarrow p$.

- $$d|(a-r) \wedge \neg(r \geq d) \Rightarrow d|(a-r) \wedge r < d \tag{2}$$

  Since $\neg(r \geq d)$ is equivalent to $r < d$, we can rewrite the left-hand side to be the same as the right-hand side. So again it is true based on $p \Rightarrow p$.

- $$d|(a-r) \wedge r \geq d \Rightarrow d|(a-r) \wedge d|d \tag{3}$$

  Since $d|d$ is true, the left-hand side simplifies to $d|(a-r)$, which is obviously implied by the right-hand side, based on the tautology that $p \wedge q \Rightarrow p$.

- $$d|(a-r) \wedge d|e \wedge r \geq e \Rightarrow d|(a-r) \wedge d|e \wedge r \geq e \tag{4}$$

  The left and right-hand sides are the same; so this is true, based on the tautology $p \Rightarrow p$.

- $$d|(a-r) \wedge d|e \wedge \neg\,(r \geq e) \Rightarrow d|(a-r) \tag{5}$$

  This is true, based on the tautology $p \wedge q \Rightarrow p$

- $$d|(a-r) \wedge d|e \wedge r \geq e \Rightarrow d|(a-(r-e)) \wedge d|\,(2 \times e) \tag{6}$$

  I'll start with the consequent and work toward the antecedent

  $$\begin{array}{ll}
  & d|(a-(r-e)) \wedge d|\,(2 \times e) \\
  = & \quad \text{plain old algebra} \\
  & d|(a-r+e) \wedge d|\,(e+e) \\
  \Leftarrow & \quad \text{applying the law } d|\,(x) \wedge d|\,(y) \Rightarrow d|\,(x+y) \text{ to } d|(a-r+e) \\
  & d|(a-r) \wedge d|e \wedge d|\,(e+e) \\
  \Leftarrow & \quad \text{applying the law } d|\,(x) \wedge d|\,(y) \Rightarrow d|\,(x+y) \text{ to } d|(e+e) \\
  & d|(a-r) \wedge d|e \\
  \Leftarrow & \quad \text{tautology} \\
  & d|(a-r) \wedge d|e \wedge r \geq e
  \end{array}$$

- $$d|(a-r) \wedge d|e \wedge r \geq e \Rightarrow (r-e) \in \mathbb{N} \tag{7}$$

  $(r-e) \in \mathbb{N}$ is equivalent to $r \geq e$, which is assured by the antecedent.

- $$d|(a-r) \wedge d|\,(2 \times e) \Rightarrow d|(a-r) \wedge d|\,(2 \times e) \tag{8}$$

  This is true based on the tautology $p \Rightarrow p$.

(c) [5] As you can see, the algorithm above calculates the remainder of a divided by d, using only adders, subtractors, and comparators. Modify the algorithm to also compute the quotient; use only adders, subtractors, and comparators. Be sure to modify the assertions, as well as the executable code, so that the modified proof outline is correct. Present the modified proof outline. (You don't need to hand in proof that the modified outline is correct, but you should satisfy yourself that it is.) The modified postcondition should be

$$d|(a-r) \wedge r < d \wedge d \times q + r = a$$

Solution:

The key to my solution is to add the conjunct $d \times q + r = a$ to the invariant of both loops. This ensures it is true at the end, as required. To maintain the truth of $d \times q + r = a$, we need to increment q by e/d whenever r is decremented by e. (Since $d|e$, e/d will be an integer.) In order to do that, we can use a variable m such that $m \times d = e$. If we have such a variable, we can increment q by m whenever r is decremented by e. To maintain $m \times d = e$: whenever e is doubled, m must also be doubled.

Putting these ideas together we get (underlined parts are new) .

```
$\{d > 0\}$
$r := a \quad \underline{q := 0}$
$\{d|(a-r) \underline{\wedge d \times q + r = a}\}$
while $r \geq d$ do
    $\{d|(a-r) \wedge r \geq d \underline{\wedge d \times q + r = a}\}$
    $e := d \quad \underline{m := 1}$
    $\{d|(a-r) \wedge d|e \underline{\wedge d \times q + r = a \wedge m \times d = e}\}$
    while $r \geq e$ do
        $\{d|(a-r) \wedge d|e \wedge r \geq e \underline{\wedge d \times q + r = a \wedge m \times d = e}\}$
        $r := r - e \quad \underline{q := q + m}$
        $\{d|(a-r) \wedge d|\,(2 \times e) \underline{\wedge d \times q + r = a \wedge m \times d = e}\}$
        $e := 2 \times e \quad \underline{m := 2 \times m}$
    end while
end while
$\{d|(a-r) \wedge r < d \underline{\wedge d \times q + r = a}\}$
```

Q1 [20]
The Fibonacci function is such that

$$\begin{aligned} \text{fib}(0) &= 0 \\ \text{fib}(1) &= 1 \\ \text{fib}(i+2) &= \text{fib}(i) + \text{fib}(i+1), \text{ for all } i \in \mathbb{N} \end{aligned}$$

Develop a program to efficiently[1] compute the value of $\text{fib}(k)$, for any $k > 0$. Present a correct proof outline and show that it is correct.

Solution:
I'll compute the result into b, so the specification is

$\{k > 0\}$
?
$\{b = \text{fib}(k)\}$

Define I to be $0 < j \le k \wedge a = \text{fib}(j-1) \wedge b = \text{fib}(j)$. Now split the problem into two

$\{k > 0\}$
?
$\{I\}$
?
$\{b = \text{fib}(k)\}$

The first problem can be solved by 3 assignments

$\{k > 0\}$
$a := 0$
$b := 1$
$j := 1$
$\{I : 0 < j \le k \wedge a = \text{fib}(j-1) \wedge b = \text{fib}(j)\}$

To check this

$$\begin{aligned} & I[j:1][b:1][a:0] \\ = \quad & \quad \text{substitute} \\ & 0 < 1 \le k \wedge 0 = \text{fib}(1-1) \wedge 1 = \text{fib}(1) \\ = \quad & \quad \text{simplify using } 0 = \text{fib}(0) \text{ and } 1 = \text{fib}(1) \\ & 1 \le k \\ = \quad & \quad \text{rewrite} \\ & k > 0 \end{aligned}$$

The second problem can be solved with a loop

[1] If your program is such that the number of addition operations that it does is roughly proportional to k, it is efficient enough.

```
$\{I : 0 < j \leq k \wedge a = \text{fib}(j-1) \wedge b = \text{fib}(j)\}$
while $j \neq k$ do
    $\{ I \wedge j \neq k \}$
    ?
end while
$\{b = \text{fib}(k)\}$
```

To check this we check

$$\begin{array}{ll}
 & \neg(j \neq k) \wedge 0 < j \leq k \wedge a = \text{fib}(j-1) \wedge b = \text{fib}(j) \\
= & \quad \text{rewrite} \\
 & j = k \wedge 0 < j \leq k \wedge a = \text{fib}(j-1) \wedge b = \text{fib}(j) \\
= & \quad \text{one point} \\
 & j = k \wedge 0 < k \leq k \wedge a = \text{fib}(k-1) \wedge b = \text{fib}(k) \\
\Rightarrow & \quad \text{tautology} \\
 & b = \text{fib}(k)
\end{array}$$

What remains is to find a loop body

```
$\{ I \wedge j \neq k \}$
?
$\{ I : 0 < j \leq k \wedge a = \text{fib}(j-1) \wedge b = \text{fib}(j) \}$
```

To get closer to termination, we should increment j; this means we need adjust a and b to keep the invariant true.

```
$\{ I \wedge j \neq k \}$
    $t := a$
    $a := b$
    $b := b + t$
    $j := j + 1$
$\{ I : 0 < j \leq k \wedge a = \text{fib}(j-1) \wedge b = \text{fib}(j)\}$
```

To verify this, we find the substituted postcondition and show that the precondition implies it. It is important to do the substitutions in the right order here; so I'll show them one at a time.

$$
\begin{array}{cl}
& I[j : j + 1][b : b + t][a : b][t : a] \\
= & \quad \text{substitute for } j \\
& (0 < j + 1 \leq k \wedge a = \text{fib}(j + 1 - 1) \wedge b = \text{fib}(j + 1))\,[b : b + t][a : b][t : a] \\
= & \quad \text{substitute for } b \\
& (0 < j + 1 \leq k \wedge a = \text{fib}(j) \wedge b + t = \text{fib}(j + 1))\,[a : b][t : a] \\
= & \quad \text{substitute for } a \\
& (0 < j + 1 \leq k \wedge b = \text{fib}(j) \wedge b + t = \text{fib}(j + 1))\,[t : a] \\
= & \quad \text{substitute for } t \\
& 0 < j + 1 \leq k \wedge b = \text{fib}(j) \wedge b + a = \text{fib}(j + 1) \\
= & \quad \text{rewrite the inequalities} \\
& 0 < j + 1 \wedge j < k \wedge b = \text{fib}(j) \wedge b + a = \text{fib}(j + 1) \\
\Leftarrow & \quad \text{since } 0 < j \text{ implies } 0 < j + 1\text{, and since } j \leq k \wedge j \neq k \text{ implies } j < k \\
& j \neq k \wedge 0 < j \leq k \wedge b = \text{fib}(j) \wedge b + a = \text{fib}(j + 1) \\
\Leftarrow & \quad \text{since } b = \text{fib}(j) \text{ and } a = \text{fib}(j - 1) \text{ and } j > 0 \text{ imply } b + a = \text{fib}(j + 1) \\
& j \neq k \wedge 0 < j \leq k \wedge b = \text{fib}(j) \wedge a = \text{fib}(j - 1) \\
= & \quad \text{definition of } I \\
& j \neq k \wedge I
\end{array}
$$

That completes the development. In summary the proof outline is

```
{$k > 0$}
$a := 0$
$b := 1$
$j := 1$
{$I : 0 < j \leq k \wedge a = \text{fib}(j - 1) \wedge b = \text{fib}(j)$}
while $j \neq k$ do
    { $I \wedge j \neq k$ }
    $t := a$
    $a := b$
    $b := b + t$
    $j := j + 1$
end while
{$b = \text{fib}(k)$}
```

An alternative loop body is

```
{ $I \wedge j \neq k$ }
    $b := b + a$
    $a := b - a$
    $j := j + 1$
{ $I : 0 < j \leq k \wedge a = \text{fib}(j - 1) \wedge b = \text{fib}(j)$}
```

The number of additions is $2(k - 1)$, so this is sufficiently efficient.

Bonus [6].

(a)[3] Consider a new kind of command: the nondeterministic assignment statement. A non-deterministic assignment statement looks like this $x :\in S$ —pronounced "x becomes an element of S"— where x is a variable and S is a set. The meaning is that an arbitrary member of S is assigned to x. Give a rule that reduces the programming question

$$\text{Is } \{P\}\ x :\in S\ \{Q\} \text{ correct?}$$

to a mathematical question about universal truth.

Solution: There are a few reasonable possibilities here. The simplest rule is

If $P \Rightarrow (\forall x \in S \cdot Q)$ is universally true
then $\{P\}\ x :\in S\ \{Q\}$ is correct.

However, it is a bit troubling that, when S is empty, we have a command that can seemingly work miracles. For example we can show

$$\{\text{true}\}\ x :\in \emptyset\ \{1 = 2\} \quad ,$$

as

$$\begin{array}{l} \quad \text{true} \Rightarrow (\forall x \in \emptyset \cdot 1 = 2) \\ = \\ \quad \text{true} \Rightarrow \text{true} \\ = \\ \quad \text{true.} \end{array}$$

[2]

Another approach is to consider that, if the set is empty, it is an error

If $P \Rightarrow S \neq \emptyset$ is universally true
and $P \Rightarrow (\forall x \in S \cdot Q)$ is universally true
then $\{P\}\ x :\in S\ \{Q\}$ is correct.

Finally, we should be concerned about whether the set expression is well defined and about

[2] There is a way to defend this simple law, while avoiding miracles. Suppose you command a genie to assign to x any member of an empty set. The only way the genie can avoid an unacceptable final state is to take infinitely long to complete this task. I.e., to not terminate. If we consider that $x :\in S$ does not terminate when S is empty, then in fact $\{\text{true}\}\ x :\in \emptyset\ \{1 = 2\}$ is correct.

This argument may seem like grasping at straws, however, this is the approach taken in some well-respected theories of programming. In concurrent programming, it actually makes a lot of sense: We can regard $x :\in y$ to mean that (in the a case y is empty) that the thread should wait until another thread puts something in set y.

whether all its member are in range. This suggests the following rule

If $P \Rightarrow \mathrm{df}(S)$ is universally true,
$P \Rightarrow S \subseteq \mathrm{rng}(x)$ is universally true,
$P \Rightarrow S \neq \emptyset$ is universally true,
and $P \Rightarrow (\forall x \in S \cdot Q)$ is universally true
then $\{P\}\ x :\in S\ \{Q\}$ is correct.

By the way, this sort of nondeterminism, where an arbitrary value is picked and thus we must write our program to be prepared for every eventuality is traditionally called *demonic nondeterminism* — appropriate for our story.

(b)[3] The Devil gives Faust a chance to win his soul back. The Devil produces a pile of 100 pebbles. The players take turns removing anywhere from 1 to 9 (inclusive) pebbles. The game ends when the pile is empty. The last player to move wins Faust's soul. Being a gentleman, the Devil allows Faust to move first.

We can model the game as a program:

```
$f :=$ true // Faust to move
$p := 100$
while $p \neq 0$ do
    if $f$ then
        $r :\in \{1, .., \max(9, p)\}$ // Faust's move
        $p := p - r$
    else
        $r :\in \{1, .., \max(9, p)\}$ // The Devil's move
        $p := p - r$
    end if
    $f := \neg f$
end while
```

Propose a condition Q such that Q will be true after $f :=$ true $p := 100$ is executed and such that $p = 0 \wedge Q \Rightarrow f$ is universally true. A consequence is that, if Q were a loop invariant, then f would be a postcondition, meaning that the Devil wins. Now modify the Devil's move —staying within the rules— so that Q is preserved by the loop's body.

Solution: The devil's strategy is to ensure that $Q : f = 10|p$ is an invariant of the loop. The way to do this is to pick a number r from $\{1, .., \max(9, p)\}$ such that $10|(p - r)$. In fact the value $r = p - 10\lfloor p/10 \rfloor$ will do. We can see that this won't be 0 since, when it is the Devil's turn, we have $\neg f$ and so by the invariant $\neg 10|p$. Furthermore this number is not bigger either 9 or p, so it is within the set $\{1, .., \max(9, p)\}$.

Note also that Faust's move, however he picks, preserves the invariant, since

$$\{p \neq 0 \wedge f \wedge 10|p\}\ r :\in \{1, .., \max(9, p)\}\ p := p - r\ \{f \wedge \neg 10|p\}$$

is correct, as

$$p \neq 0 \wedge f \wedge 10|p \Rightarrow (\forall r :\in \{1, .., \max(9, p)\} \cdot f \wedge \neg 10|(p - r))$$

is universally true.

After refining the Devil's strategy we have

```
$f :=$ true // Faust to move
$p := 100$
$\{f = 10|p\}$
while $p \neq 0$ do
    if $f$ then
        $\{p \neq 0 \wedge f \wedge 10|p\}$
        $r :\in \{1, .., \max(9, p)\}$ // Faust's move
        $p := p - r$
        $\{f \wedge \neg 10|p\}$
    else
        $\{p \neq 0 \wedge \neg f \wedge \neg 10|p\}$
        $r := p - 10\lfloor p/10 \rfloor$ // The Devil's move
        $p := p - r$
        $\{\neg f \wedge 10|p\}$
    end if
    $\{f \neq 10|p\}$
    $f := \neg f$
end while
$\{f\}$ // Faust has lost
```