Solutions to Programming Theory exam 2008-12-16

Jonathan Cederberg
jonathan.cederberg@it.uu.se

April 8, 2009

Abstract

These solutions are supplied for your convenience, and may contain a lot of errors. Such errors are entirely the fault of the author, and indications as to where they are would be greatly appreciated.

Problem 1 (25 p)

Prove the following:

$$\left(\begin{array}{c} w \leq y \\ \wedge \\ w \leq z \\ \wedge \\ (x \leq y) \vee (x \leq z) \end{array}\right) \Rightarrow (x + w \ \leq \ y + z)$$

How to approach this problem

As we can see this problem is about proving an implication. If we look through the three main proof layouts discussed in the notes from the proof theory tutorial, the first and third can be directly applied. The second is a method for proving an equivalence, and can also be used if we can prove the stronger claim where the implication is substituted for an equivalence.

To see whether this can be done, one has to decide if the left hand side of the implication is stronger than the right hand side. If so, the second proof method cannot be employed. Here, we can see that this is the case, since we for example cannot decide that $w \leq z$ is true just because $(x + w \ \leq \ y + z)$ is true.

So, we are left with method one or three. Which one to choose is a matter of taste really, but one thing to note here is that we have relatively little to work with on the right hand side of the implication. In such cases, it might be useful to start with the whole expression, and try to reduce it to T, i.e. employ the first proof method.

Also, before we start, a very good thing would be to convince ourself of why this actually true. Not only to make sure that the exam is not inconsistent, but because we can use the intuition of why the statement is true to guide us when writing the proof.

Start by looking at the premises of the statement. We can think of it as either we have the first two conjuncts and the *first* part of the third

conjunct (i.e. $w \leq y \wedge w \leq z \wedge x \leq y$) in which case we can see that using the first and third, we can easily prove the consequence. Or, we have the first two conjuncts and the *second* part of the third conjunct (i.e. $w \leq y \wedge w \leq z \wedge x \leq z$) in which case we can see that using the second and third, we can easily prove the consequence.

Now, let's use this insight to create a proof.

Proof

$$
\begin{array}{ll}
 & \left(\begin{array}{c} w \leq y \\ \wedge \\ w \leq z \\ \wedge \\ (x \leq y) \vee (x \leq z) \end{array}\right) \Rightarrow (x + w \ \leq \ y + z) \\
= & \{ \text{ Distributivity } \} \\
 & \left(\begin{array}{c} w \leq y \wedge w \leq z \wedge x \leq y \\ \vee \\ w \leq y \wedge w \leq z \wedge x \leq z \end{array}\right) \Rightarrow (x + w \ \leq \ y + z) \\
= & \{ \text{ and-simplification } \} \\
 & \left(\begin{array}{c} w \leq y \wedge w \leq z \wedge x \leq y \wedge T \\ \vee \\ w \leq y \wedge w \leq z \wedge x \leq z \wedge T \end{array}\right) \Rightarrow (x + w \ \leq \ y + z) \\
= & \{ \text{ Arithmetic: } a \leq b \wedge c \leq d \ \Rightarrow \ a + c \leq b + d \ \} \\
 & \left(\begin{array}{c} w \leq y \wedge w \leq z \wedge x \leq y \wedge (w \leq z \wedge x \leq y \ \Rightarrow \ x + w \leq y + z) \\ \vee \\ w \leq y \wedge w \leq z \wedge x \leq z \wedge (w \leq y \wedge x \leq z \ \Rightarrow \ x + w \leq y + z) \end{array}\right) \\
 & \qquad \Rightarrow \\
 & \qquad (x + w \ \leq \ y + z) \\
= & \{ \text{ Elimination of } \Rightarrow \ \} \\
 & \left(\begin{array}{c} w \leq y \wedge w \leq z \wedge x \leq y \wedge x + w \leq y + z \\ \vee \\ w \leq y \wedge w \leq z \wedge x \leq z \wedge x + w \leq y + z \end{array}\right) \Rightarrow (x + w \ \leq \ y + z) \\
= & \{ \text{ Distributivity } \} \\
 & \left(\left(\begin{array}{c} w \leq y \wedge w \leq z \wedge x \leq y \\ \vee \\ w \leq y \wedge w \leq z \wedge x \leq z \end{array}\right) \wedge x + w \leq y + z\right) \Rightarrow (x + w \ \leq \ y + z) \\
= & \{ \text{ Weakening } \} \\
 & T \quad \Box
\end{array}
$$

Important points

Call the statement to prove "$p \ \Rightarrow \ q$". The main form of this proof is that we take the left hand side of the statement, p, which we have already established as being a stronger statement than q, and transform it such that it becomes obvious that it is stronger than q. This means transforming it to the form $p' \wedge q \ \Rightarrow \ q$.

As the arithmetic rule we use is an implication, it cannot be used to immediately substitute a statement, unless we use conditional substitution. This would require that we had the premises "$a \leq b$" and "$c \leq d$"

as assumptions, which we clearly do not. The solution to still be able to use the rule, is called "Elimination of $\Rightarrow$", as can be seen in the fourth step.

Problem 2 (20 p)

Consider the program S:

$$\textbf{do} \quad 0 < x \quad \rightarrow \quad x := x - 2 \quad \textbf{od}$$

Give $wp\,(\text{“}S\text{”},\ R)$ for the following values of R:

- $R = odd(x)$
- $R = (x = -1)$
- $R = (x = -3)$
- $R = (x = 0)$
- $R = (x \geq 2)$

Do not explain and do not prove anything!

How to approach this problem

Let's take a step back to and contemplate the actual question here. We have to provide the weakest precondition of the iterative command. For this, we have no computation rule as for the other constructs in our programming language. What we have is a list of points (the iterative command theorem) that proves that a certain predicate *implies* the weakest precondition if the iterative command. The reason for not having a general rule is both simple and profound: to be able to give a general way fof computing the weakest precondition of an iterative command, one has to, among other things, decide termination. This was famously proven to be impossible in 1936 by Alan Turing, and is a very important thing to know.

So, since we have no computation rule for computing the answer, we have to rely on the definition og the weakest precondition. From the course book (which you of course do not have at your disposal at the exam):

> $[\ldots]$ $wp(S, R)$ $[\ldots]$ represents the set of *all* states such that execution of S begun in any one of them is guaranteed to terminate in a finite amount of time in a state satisfying R.

So, next, we have to establish what the program actually does. It is trivial to see that as long as x is positive, 2 is subtracted from x. This means that for this problem, we do not really have to worry about termination, as no matter what number we start from, we will always eventually get to a negative number if we subtract 2 long enough.

Now let's take the first problem as an illustrating example. To end up in a state where x is odd, one only has to start from a state where x is od,, as subtracting two from an odd number keeps the number odd and subtracting 2 from an even nmber keeps the number even. Thus $wp\,(S,\ odd(x)) = odd(x)$.

The answers

- $wp\,(S,\ odd(x)) = odd(x)$
- $wp\,(S,\ x = -1) = odd(x) \wedge x \geq -1$
- $wp\,(S,\ x = -3) = (x = -3)$
- $wp\,(S,\ x = 0) = even(x) \wedge x \geq 0$
- $wp\,(S,\ x \geq 2) = F$

More points

The last one has proven to be tricky to some people. It is not a trick question. F is the predicate that no state can satisfy, and thus the set of states satisfying F is the empty set.

Problem 3 (25 p)

Using the alternative command theorem, show that the following program is correct.

$$
\begin{array}{lllll}
\{Q:\ (x = \max(x,y,z)) \vee (y = \max(x,y,z))\} & & & & \\
\textbf{if} & (x \leq z) \wedge (y \leq z) & \rightarrow & w := z & \\
\square & T & \rightarrow & w := x + y & \\
\textbf{fi} & & & & \\
\{R:\ (w \geq \max(x,y,z))\} & & & &
\end{array}
$$

where x, y, and z are natural numbers.

Hint: formulate the predicates $b = \max(a_1, a_2, a_3)$ and $b \geq \max(a_1, a_2, a_3)$

How to approach this problem

One general comment that can be made here, which is valid for any exam in any course, is that it there is a hint written on the exam, it is very good policy to utilize it when solving the problem. So, let's start by doing as the hint says:

- $(b = \max(a_1, a_2, a_3)) \stackrel{\text{def}}{=}$
  $(b \geq a_1 \wedge b \geq a_2 \wedge b \geq a_3 \wedge (b = a_1 \vee b = a_2 \vee b = a_3))$
- $(b \geq \max(a_1, a_2, a_3)) \stackrel{\text{def}}{=}$
  $(b \geq a_1 \wedge b \geq a_2 \wedge b \geq a_3)$

After doing as the hint says, we should, without thinking, write down the following:

> According to the alternative command theorem, we need to prove the following:
>
> 1. $Q \ \Rightarrow\ ((x \leq z) \wedge (y \leq z)) \vee T$
> 2. $Q \wedge (x \leq z) \wedge (y \leq z) \ \Rightarrow\ wp\,(\text{“}w := z\text{”},\ R)$
> 3. $Q \wedge T \ \Rightarrow\ wp\,(\text{“}w := x + y\text{”},\ R)$

Proofs

The proofs are rather straight forward, and we should keep in mind that often there is a few steps in the start of each proof where we just apply the definitions and simplify. This first start usually reduces the thing to prove quite a bit.

Proof of 1

$$
\begin{array}{ll}
Q \;\Rightarrow\; ((x \le z) \wedge (y \le z)) \vee T \\
\quad = & \{ \text{ or-simplification } \} \\
& Q \;\Rightarrow\; T \\
\quad = & \{ \text{ Right zero of } \Rightarrow \; \} \\
& T \quad \Box
\end{array}
$$

Proof of 2

Assume Q
Assume $x \le z$
Assume $y \le z$

$$
\begin{array}{ll}
wp\,(\text{“}w :=\; z\text{”},\; R) \\
\quad = & \{ \text{ Definition of } R \; \} \\
& wp\,(\text{“}w :=\; z\text{”},\; (w \ge \max(x,y,z))) \\
\quad = & \{ \text{ Definition of max } \} \\
& wp\,(\text{“}w :=\; z\text{”},\; (w \ge x \wedge w \ge y \wedge w \ge z)) \\
\quad = & \{ \text{ Definition of assignment } \} \\
& z \ge x \wedge z \ge y \wedge z \ge z \\
\quad = & \{ \text{ Assumption: “}x \le z\text{” } \} \\
& T \wedge z \ge y \wedge z \ge z \\
\quad = & \{ \text{ Assumption: “}y \le z\text{” } \} \\
& T \wedge T \wedge z \ge z \\
\quad = & \{ \text{ Arithmetic: “}a \ge a\text{” } \} \\
& T \wedge T \wedge T \\
\quad = & \{ \text{ and-simplification } \} \\
& T \quad \Box
\end{array}
$$

Proof of 3

$$
\begin{array}{ll}
Q \wedge T \;\Rightarrow\; wp\,(\text{“}w :=\; x + y\text{”},\; R) \\
\quad = & \{ \text{ Definition of } R \text{ and and-simplification } \} \\
& Q \;\Rightarrow\; wp\,(\text{“}w :=\; x + y\text{”},\; (w \ge \max(x,y,z))) \\
\quad = & \{ \text{ Definition of assignment } \} \\
& Q \;\Rightarrow\; (x + y \ge \max(x,y,z)) \\
\quad = & \{ \text{ Definition of max } \} \\
& Q \;\Rightarrow\; (x + y \ge x \wedge x + y \ge y \wedge x + y \ge z) \\
\quad = & \{ \text{ Arithmetic: “}a + b \ge a\text{” } \} \\
& Q \;\Rightarrow\; (T \wedge T \wedge x + y \ge z) \\
\quad = & \{ \text{ Definition of } Q \text{ and and-simplification } \} \\
& \left(\begin{array}{c} x = \max(x,y,z) \vee \\ y = \max(x,y,z) \end{array} \right) \;\Rightarrow\; (x + y \ge z)
\end{array}
$$

$$
\begin{array}{ll}
= & \{ \text{ Definition of max } \} \\
& \left(\begin{array}{c} (x \geq x \wedge x \geq y \wedge x \geq z \wedge (x = x \vee x = y \vee x = z)) \vee \\ (y \geq x \wedge y \geq y \wedge y \geq z \wedge (y = x \vee y = y \vee y = z)) \end{array} \right) \\
& \qquad \Rightarrow \ (x + y \geq z) \\
= & \{ \text{ Arithmetic: "} a \geq a \text{" and "} a = a \text{" } \} \\
& \left(\begin{array}{c} (T \wedge x \geq y \wedge x \geq z \wedge (T \vee x = y \vee x = z)) \vee \\ (y \geq x \wedge T \wedge y \geq z \wedge (y = x \vee T \vee y = z)) \end{array} \right) \ \Rightarrow \ (x + y \geq z) \\
= & \{ \text{ or-simplification } \} \\
& \left(\begin{array}{c} (T \wedge x \geq y \wedge x \geq z \wedge T) \vee \\ (y \geq x \wedge T \wedge y \geq z \wedge T) \end{array} \right) \ \Rightarrow \ (x + y \geq z) \\
= & \{ \text{ and-simplification } \} \\
& \left(\begin{array}{c} (x \geq y \wedge x \geq z) \vee \\ (y \geq x \wedge y \geq z) \end{array} \right) \ \Rightarrow \ (x + y \geq z) \\
= & \{ \text{ and-simplification } \} \\
& \left(\begin{array}{c} (x \geq y \wedge x \geq z \wedge T) \vee \\ (y \geq x \wedge y \geq z \wedge T) \end{array} \right) \ \Rightarrow \ (x + y \geq z) \\
= & \{ \text{ Arithmetic: "} (a \geq b \ \Rightarrow \ a + c \geq b) \text{" } \} \\
& \left(\begin{array}{c} (x \geq y \wedge x \geq z \wedge (x \geq z \ \Rightarrow \ x + y \geq z)) \vee \\ (y \geq x \wedge y \geq z \wedge (y \geq z \ \Rightarrow \ x + y \geq z)) \end{array} \right) \ \Rightarrow \ (x + y \geq z) \\
= & \{ \text{ Elimination of } \Rightarrow \ \} \\
& \left(\begin{array}{c} (x \geq y \wedge x \geq z \wedge x + y \geq z) \vee \\ (y \geq x \wedge y \geq z \wedge x + y \geq z) \end{array} \right) \ \Rightarrow \ (x + y \geq z) \\
= & \{ \text{ Distributivity } \} \\
& \left(\begin{array}{c} (x \geq y \wedge x \geq z) \vee \\ (y \geq x \wedge y \geq z) \end{array} \right) \wedge (x + y \geq z) \ \Rightarrow \ (x + y \geq z) \\
= & \{ \text{ Weakening } \} \\
& T \quad \square
\end{array}
$$

Important points

Statement 1 and 2 are rather straight forward to prove. The third is somewhat trickier (but still easy). Since we are attentive, we can see that the proof approach in the third proof is different from the standard approach of proving an implication using assumptions, as we do in 2. We can also recognize the structure of the last part of the proof from the proof in the first problem.

The reason for approaching this proof differently, is that the premise Q is a big disjunction. A disjunction is problematic to use as an assumption, because it is not so strong a statement. Therefore, the first proof method is good also in this case.

Also, we should not be too alarmed by the length of the proof. If we look closely, we can see that the first 9 steps of the proof are just application of definitions, simplifications and very simple arithmetic rules.

As a last note, the additional information of the numbers being natural numbers is only used implicitly, in the choice of the arithmetic rules to use. In particular, the rule "$(a \geq b \ \Rightarrow \ a + c \geq b)$" is not generally true, but only if a and b are natural.

Problem 4 (30 p)

Consider the following program:

$$
\begin{array}{l}
\{Q:\ (s=1)\wedge(i=0)\wedge(n>0)\} \\
\{\text{inv } P: ??\} \\
\{\text{bound } t: ??\} \\
\quad \textbf{do} \quad i<n \quad \rightarrow \quad s,i:=\ s+2i+3, i+1 \quad \textbf{od} \\
\{R:\ s=(n+1)^2\}
\end{array}
$$

Define an invariant P and a bound function t. Show the correctness of the program through the iterative command theorem, using your definitions of P and t.

How to approach this problem

Similarly to what was noted in the solution to the previous problem, we should, without thinking, write down the following:

> According to the iterative command theorem, we need to prove the following:
>
> 1. $Q \ \Rightarrow\ P$
> 2. $P\wedge(i<n)\ \Rightarrow\ wp\,(\text{“}s,i:=\ s+2i+3, i+1\text{”},\ P)$
> 3. $P\wedge\neg(i<n)\ \Rightarrow\ R$
> 4. $P\wedge(i<n)\ \Rightarrow\ t>0$
> 5. $P\wedge(i<n)\ \Rightarrow\ wp\,(\text{“}t_1:=\ t; s,i:=\ s+2i+3, i+1\text{”},\ t<t_1)$

What remains, and what also is the harder part of this problem, is to find P and t. Before this is possible, we need to understand the program, and what it actually does. If we look at the postcondition, we can see that it computes the square of the input n, through successive additions of some multiple of the iteration index i. To convince yourself that the program works, run it for a few iterations:

i	0	1	2	3
s	1	$1+2\cdot 0+3=4$	$4+2\cdot 1+3=9$	$9+2\cdot 2+3=16$

Let's start with finding a good invariant P. Looking at the table above, it seems clear that s is a series of squares. As discussed in the tutorial session, the way to develop a good invariant is to start with the postcondition and weaken it. Since R neither contains a conjunct to delete, or a variable whose range we could enlarge, we only have the "replace a constant with a variable"-approach to go with. As is fairly clear from the table, the n in R should be replaced by i to get the invariant $P=(s=(i=1)^2)$. The program structure also gives this away: the program iterates up until $i=n$, so at the end this replacement should give the postcondtition.

Now it is time to test whether we have a good invariant. Look at the list of things to prove. Clearly, point 1 holds. Point 2 is a bit harder to see directly, but judging from the table above, this is also true. Point 3, on the other hand is not clear. Let's examine it in more detail. We want to prove $(s=(i+1)^2)\wedge\neg(i<n)\ \Rightarrow\ (s=(n+1)^2)$ which is equivalent, using arithmetic relations, to $(s=(i+1)^2)\wedge(i\geq n)\ \Rightarrow\ (s=(n+1)^2)$. Clearly, a proof of this has to include some type of substitution between i and n. But we do not know that $i=n$ from the information we have. What to do about this?

The reason is that we, when we replace the constant with a variable, also should bound the variable using the constant. If we look at the program, the variable i is used in such a way that it is never dominated by n. This information is crucial for the proof of point 3. Therefore, a strong enough invariant P is $(s = (i+1)^2) \wedge (i \leq n)$.

A general comment about the iterative command theorem is that the five points can be divided into two categories. The first three concerns the correctness of the computation. That is, *if* we compute something, *then* we know that the result is correct. Points 4 and 5 ensures that we in each computation actually will compute something.

So, we need to devise a bound function to ensure termination. The way to do this is to look in the program for something that is uniformly changing, and use this as a starting point. Most programs have some variable which is a looping index or equivalent, and that is exactly what we want here. In this particular problem, this index is the variable i. To fulfill point 5, which states that the bound function should be decreasing, we only need to know in what direction the i changes through the iterations, and tweak the sign to make it decrease with every iteration. Here we can see that i increases with every iteration, and thus the bound function $-i$ is enough to satisfy point 5.

To satisfy also point 4, which states that the bound function should be positive as long as we keep looping, we need to shift the bound function in the positive direction, by adding a number larger than i will ever be as long as we loop. Inspecting the guard for looping, we can easily see that as long as we loop, n will be strictly larger than i, and thus the bound function $n - i$ fulfills our constraints.

Proofs

Simple. The hard part is coming up with the bound and invariant.

Additional points

Sometimes the bound function is not as easy to come up with as it is here. The loop might have more than one guard, and there might be two variables, x and y say, of which at most one is changing as the loop is executing. The solution then is to use the sum $x + y$ in the same way we used i above.

You might also be asked to provide a command S_{init}, that establishes P, rather than as being given a strong precondition as you are here. In such a case, a few testruns as above with different inputs are indispensible, or at least will save you a lot of time.