
Department of Computer Science, Australian National University
COMP2600 — Formal Methods for Software Engineering
Semester 2, 2011

— Assignment 4 —
Program Verification

Due: 11am Monday 24th October 2011

The submission of your assignment must be done via the assignment boxes in the student foyer. Failure to submit by the due date will result in late penalties of ten percent per weekday. Other arrangements that are required because of truly exceptional circumstances need to be negotiated before your deadline.

Your submission must be well-presented on clean A4 paper with a fully completed standard cover page, including your *tutor's name* and your *tutorial group*. Failure to follow this instruction will result in a penalty. The COMP2600 Assignments page has a link to an appropriate cover page.

Introduction

Two integers x, y are *coprime* if their greatest common divisor is 1. The program below computes what is called *Euler's Totient function*. The totient $\varphi(n)$ of a positive, non-zero integer n is the number of integers m in the range $1 \leq m \leq n$ such that m, n are coprime. The table below gives some examples:

n	coprimes in range	$\varphi(n)$
1	{1}	1
2	{1}	1
3	{1, 2}	2
4	{1, 3}	2
5	{1, 2, 3, 4}	4
18	{1, 5, 7, 11, 13, 17}	6

The program *Totient*

```
t:=1;
x:=1;
while (n ≠ 1) do
  x:=x+1;
  if (x | n) then
    t:=t*(x-1);
    n:=n/x;
    while (x | n) do
      t:=t*x;
      n:=n/x
```

A few remarks on this code:

- We may assume that all variables are typed as positive non-zero integers.
- Note that we are using tabbing to indicate nesting: the second `while` loop is inside the body of the `if` statement, which is in turn inside the first `while` loop.
- We will use the abbreviations *Loop1* for the main loop, *Body1* for the body of the main loop, and *Loop2* for the smaller loop nested inside. If you wish to use any other abbreviation in your answers please make that clear when you introduce them.
- $x \mid y$ (read as *x divides y*) is a boolean test that succeeds when $\frac{y}{x}$ is an integer. We will write its failure as $x \nmid y$.
- The `if . . then . .` statement is in a mildly different format from that given in lectures, as there is no `else` case. Hoare Logic and WP calculus rules for this construct are given in the appendix, and are easily derivable from the the rules you have seen for `if . . then . . else . .`

1 Hoare Logic

We wish to show that the program computes the totient of the original value of n . That is:

$$\{n = N\} \textit{Totient} \{t = \varphi(N)\}$$

Here are some facts about number theory that may help you with this proof. Make sure you cite the relevant fact by its number when using it for precondition strengthening, postcondition weakening, or simplification:

Fact 1. If $x \nmid y$, then $x \nmid \frac{y}{z}$;

Fact 2. If x is the smallest positive divisor of y not equal to 1, then x is prime;

Fact 3. If x is prime and $x \nmid y$, then x, y are coprime;

Fact 4. $\varphi(1) = 1$;

Fact 5. If x is prime, then $\varphi(x) = x - 1$;

Fact 6. If x is prime and $x \mid y$, then $\varphi(yx) = \varphi(y)x$;

Fact 7. If x, y are coprime, then $\varphi(xy) = \varphi(x)\varphi(y)$.

We have two loops, *Loop1* and *Loop2*, and therefore will need two invariants. Let

$$\textit{Inv1} \equiv (\varphi(N) = \varphi(n)t \wedge \forall y. (1 < y \leq x \rightarrow y \nmid n))$$

$$\textit{Inv2} \equiv (\varphi(N) = \frac{\varphi(nx)t}{x-1} \wedge \forall y. (1 < y < x \rightarrow y \nmid n)) \wedge x \text{ is prime})$$

Now use Hoare Logic to prove the following. **Make sure that every step of your proof is justified by citing the rule that you are using.**

- Prove $\{\textit{Inv2}\} \textit{Loop2} \{\textit{Inv1}\}$ using the invariant *Inv2*.
- Hence prove $\{\textit{Inv1}\} \textit{Body1} \{\textit{Inv1}\}$.
- Hence prove $\{n = N\} \textit{Totient} \{t = \varphi(N)\}$, using the invariant *Inv1* for *Loop1*.

2 Weakest Precondition Calculus

In this section we wish to use the rules of the Weakest Precondition Calculus to find the weakest precondition which ensures that $\varphi(n) = 1$. That is, we want to find

$$wp(Totient, t = 1)$$

Answer the following questions. Make sure you simplify your answers as far as possible, and justify any simplifications that are not immediately obvious.

(i) We will need to find $wp(Loop1, t = 1)$. First, state P_0 (the predicate expressing success after zero loop iterations) for this weakest precondition.

(ii) Find $wp(Body1, P_0)$ and hence state P_1 for $wp(Loop1, t = 1)$.

This will involve calculating $wp(Loop2, P_0)$. Call the sequence of predicates for this weakest precondition $Q_0, Q_1, Q_2 \dots$ (so that you don't get confused with P_0, P_1, P_2, \dots). Make sure you justify clearly each step of your argument when finding the weakest precondition for $Loop2$.

(iii) Find $wp(Body1, P_1)$, and hence state P_2 for $wp(Loop1, t = 1)$.

This will involve calculating $wp(Loop2, P_1)$. Call the sequence of predicates for this weakest precondition $R_0, R_1, R_2 \dots$

(iv) Use these results to state P_k for $wp(Loop1, t = 1)$ for all $k > 2$. Remember to justify this step.

(v) Find $wp(Loop1, t = 1)$ and, using this result, find $wp(Totient, t = 1)$.

Appendix — Hoare Logic Rules

- Precondition Strengthening:

$$\frac{\{P_w\} S \{Q\} \quad P_s \rightarrow P_w}{\{P_s\} S \{Q\}}$$

- Postcondition Weakening:

$$\frac{\{P\} S \{Q_s\} \quad Q_s \rightarrow Q_w}{\{P\} S \{Q_w\}}$$

- Assignment:

$$\{Q(e)\} x := e \{Q(x)\}$$

- Sequence:

$$\frac{\{P\} S_1 \{Q\} \quad \{Q\} S_2 \{R\}}{\{P\} S_1; S_2 \{R\}}$$

- Conditional:

$$\frac{\{P \wedge b\} S \{Q\} \quad P \wedge \neg b \rightarrow Q}{\{P\} \text{ if } b \text{ then } S \{Q\}}$$

- While Loop:

$$\frac{\{P \wedge b\} S \{P\}}{\{P\} \text{ while } b \text{ do } S \{P \wedge \neg b\}}$$

Appendix — Weakest Precondition Rules

$$wp(x := e, Q(x)) \equiv Q(e)$$

$$wp(S_1; S_2, Q) \equiv wp(S_1, wp(S_2, Q))$$

$$wp(\text{if } b \text{ then } S, Q) \equiv (b \rightarrow wp(S, Q)) \wedge (\neg b \rightarrow Q)$$

$$\equiv (b \wedge wp(S, Q)) \vee (\neg b \wedge Q)$$

P_k is the weakest predicate that must be true before `while b do S` executes, in order for the loop to terminate after exactly k iterations in a state that satisfies Q .

$$P_0 \equiv \neg b \wedge Q$$

$$P_{k+1} \equiv b \wedge wp(S, P_k)$$

$$wp(\text{while } b \text{ do } S, Q) \equiv \exists k. (k \geq 0 \wedge P_k)$$