CSI2101 Discrete Structures Winter 2009: Program Correctness and Verification

Lucia Moura

Winter 2009

Proving the correctness of recursive programs

Mathematical induction (and strong induction) can be used to prove that a recursive algorithm is correct:

to prove that the algorithm produces the desired output for all possible input values.

We will see some examples next.

Recursive algorithm for computing a^n

procedure power(a: nonzero real number, n: nonnegative integer)
    if $(n = 0)$ **then** return 1
        else return $a \times$power$(a, n - 1)$

We will prove by mathematical induction on n that the algorithm above is correct.
We will show $P(n)$ is true for all $n \geq 0$, for
$P(n)$: For all nonzero real numbers a, power(a, n) correctly computes a^n.

Proving $power(a, n)$ is correct

Basis: If $n = 0$, the first step of the algorithm tells us that power(a,0)=1. This is correct because $a^0 = 1$ for every nonzero real number a, so $P(0)$ is true.

Inductive step:
Let $k \geq 0$.
Inductive hypothesis: power$(a, k) = a^k$, for all $a \neq 0$.
We must show next that power$(a, k + 1) = a^{k+1}$.
Since $k + 1 > 0$ the algorithm sets power$(a, k + 1) = a \times$power(a, k).
By inductive hypotheses $power(a, k) = a^k$, so
power$(a, k + 1) = a \times$power$(a, k) = a \times a^k = a^{k+1}$.

Recursive algorithm for computing $b^n \mod m$

procedure mpower(b, n, m: integers with $m \geq 2, n \geq 0$)
if $n = 0$ then return 1;
else if n is even then return mpower$(b, n/2, m)^2 \mod m$
else return ((mpower$(b, \lfloor n/2 \rfloor, m)^2 \mod m) * (b \mod m)) \mod m$

Examples:

$$
\begin{aligned}
& power(2,5,6) = \\
= \; & ((power(2,2,6)^2 \mod 6) * (2 \mod 6)) \mod 6 \\
= \; & (((power(2,1,6)^2 \mod 6)^2 \mod 6) * (2)) \mod 6 \\
= \; & ((((power(2,0,6)^2 \mod 6) * (2 \mod 6)) \mod 6)^2 \mod 6)^2 \mod 6 \\
& * 2) \mod 6 \\
= \; & ((((1^2 \mod 6) * 2) \mod 6)^2 \mod 6)^2 \mod 6) * 2) \mod 6 \\
= \; & 2.
\end{aligned}
$$

Proving $mpower(a, n, m)$ is correct, using induction on n

Basis: Let b and m be integers with $m \geq 2$, and $n = 0$. In this case, the algorithm returns 1. This is correct because $b^0 \mod m = 1$.

Inductive step:

Induction hypothesis: assume power$(b, j, m) = b^j \mod m$ for all integers j with $0 \leq j \leq k-1$, whenever b is a positive integer and m is an integer with $m \geq 2$.

We must show next that power$(b, k, m) = b^k \mod m$. There are two cases to consider.

- Case 1: k is even. In this case, the algorithm returns mpower$(b, k/2, m)^2 \mod m =$ $(i.h.)$ $(b^{k/2} \mod m)^2 \mod m = b^k \mod m$.
- Case 2: k is odd. In this case, the algorithm returns
  $((\text{mpower}(b, \lfloor k/2 \rfloor, m)^2 \mod m) * (b \mod m)) \mod m$
  $=$ $(i.h.)$ $(b^{\lfloor k/2 \rfloor} \mod m)^2 \mod m) * (b \mod m)) \mod m$
  $= (b^{2\lfloor k/2 \rfloor + 1} \mod m) = b^k \mod m$.

Program verification

We want to be able to prove that a given program meets the intended specifications.
This can often be done manually, or even by automated program verification tools. One example is PVS (People's Verification System).

A program is **correct** if it produces the correct output for every possible input.
A program has **partial correctness** if it produces the correct output for every input for which the program eventually halts.
Therefore, a program is correct if and only if it has partial correctnes and terminates.

Hoare's triple notation

- A program's I/O specification can be given using initial and final assertions.
  - The **initial assertion** p is the condition that the program's input (its initial state) is guaranteed (by its user) to satisfy.
  - The **final assertion** q is the condition that the output produced by the program (its final state) is required to satisfy.
- Hoare triple notation:
  - The notation $p\{S\}q$ means that, for all inputs I such that $p(I)$ is true, if program S (given input I) halts and produces output $O = S(I)$, then $q(O)$ is true.
  - That is, S is partially correct with respect to specification p, q.

A simple example

- Let S be the program fragment
  `y:= 2; z:= x+y`
- Let p be the initial assertion `x=1`.
  The variable x will hold 1 in all initial states.
- Let q be the final assertion `z= 3`.
  The variable z must hold 3 in all final states.
- Prove $p\{S\}q$.
  Proof: If x=1 in the program's input state, then after running y:=2 and z:=x+y, then z will be $1 + 2 = 3$.

Rules of inference for Hoare triples

- The composition rule:

$$\frac{p\{S_1\}q \qquad q\{S_2\}r}{\therefore p\{S_1; S_2\}r}$$

- It says: If program S_1 given condition p produces condition q, and S_2 given q produces r,
  then the program "S_1 followed by S_2", if given p, yields r.

Inference rule for **if-then** statements

$$\frac{(p \wedge cond)\{S\}q \quad (p \wedge \neg cond) \rightarrow q}{\therefore p\{\textbf{if } cond \textbf{ then } S\}q}$$

Example: Show that: $T\{\textbf{if } x > y \textbf{ then } y := x\}(y \geq x)$.

Proof:
When initially T is true, if $x > y$, then the **if**-body is executed, setting $y = x$, and so afterwards $y \geq x$ is true. Otherwise, $x \leq y$ and so $y \geq x$. In either case, the final assertion $y \geq x$ is true. So the rule applies, and so the fragment meets the specification.

Inference rule for **if-then-else** statements

$$\frac{(p \wedge cond)\{S_1\}q \quad (p \wedge \neg cond)\{S_2\}q}{\therefore p\{\textbf{if } cond \textbf{ then } S_1 \textbf{ else } S_2\}q}$$

Example: Prove that
$T\{\textbf{if } x < 0 \textbf{ then } abs := -x \textbf{ else } abs := x\}(abs = |x|)$

Proof:
If the initial assertion is true and $x < 0$ then after the **if**-body, abs will be $-x = |x|$.
If the initial assertion is true, but $\neg(x < 0)$ is true, i.e., $x \geq 0$, then after the **else**-body, $abs = x$, which is $|x|$.
So using the above rule, we get that this segment is true with respect to the final assertion.

Loop Invariants

For a **while**-loop "**while** cond S", we say that p is a **loop invariant** of this loop if $(p \wedge cond)\{S\}p$.

If p (and the continuation condition $cond$) is true before executing the body, then p remains true afterwards.

And so p stays true through all subsequent iterations.

This leads to the inference rule:

$$\frac{(p \wedge cond)\{S\}p}{\therefore p\{\textbf{while } cond\ S\}(\neg cond \wedge p)}$$

Example1: loop invariant

Prove that the following Hoare triple holds:
$T\{i := 1; fact := 1; \textbf{while } i < n\{i++; fact = fact * i\}\}(fact = n!)$

Proof:
Let p be the assertion "$fact = i! \wedge i \leq n$". We will show tht p is a loop invariant.
Assume that at the beginning of the **while**-loop p is true and the condition of the **while**-loop holds, in other words, assume that $fact = i!$ and $i < n$.
The new values i_{new} and $fact_{new}$ of i and $fact$ are
$i_{new} = i + 1$ and
$fact_{new} = fact \times (i + 1) = (i!) \times (i + 1) = (i + 1)! = i_{new}!$.
Since $i < n$, we also have $i_{new} = i + 1 \leq n$.
Thus p is true at the end of the execution of the loop. This shows p is a loop invariant.

Final example: combining all rules

procedure multiply$(m, n : integers)$

$p :=$ "$(m, n \in Z)$"

if $n < 0$ **then** $a := -n$ (segment S_1)

else $a := n$

$q :=$ "$(p \wedge (a = |n|))$"

$k := 0; x := 0$ (segment S_2)

$r :=$ "$q \wedge (k = 0) \wedge (x = 0)$"

$(x = mk \wedge k \leq a)$

while $k < a$ { (segment S_3)

$x = x + m; \quad k = k + 1;$

}Maintains loop invariant: $(x = mk \wedge k \leq a)$

$(x = mk \wedge k = a) \therefore s :=$ "$(x = ma) \wedge a = |n|)$"

$s \Rightarrow (n < 0 \wedge x = -mn) \vee (n \leq 0 \wedge x = mn)$

if $n < 0$ **then** $prod := -x$ (segment S_4)

else $prod := x$ $t =$ "$(prod = mn)$"

Correctness of multiply(m, n)

The proof is structured as follows, by using propositions p, q, r, s, t as defined in the previous page.

- Prove $p\{S_1\}q$ by using **if-then-else** inference rule.
- Prove $q\{S_2\}r$ by examining this trivial segment.
- Prove $r\{S_3\}s$ by using **while-loop** inference rule.
- Prove $s\{S_4\}t$ by using **if-then-else** inference rule.
- Use the rule of composition to show that $p\{S_1; S_2; S_3; S_4\}t$; recall that $p :=$ "$(m, n \in Z)$" and $t =$ "$(prod = mn)$", which is what we wanted to show for the partial correctness.

To complete the proof of correctness, given the partial correctness, we must verify that each segment terminates.
Termination is trivial for segments S_1, S_2 and S_4; for the **while-loop** (S_4) it is easy to see that it runs for a iterations.
(See general rule for proving termination of loops in the next page)
We leave the details of each step above as an exercise.

Proving termination of a loop

- Associate with each iteration i a natural number k_i, such that $< k_0, k_1, k_2, \ldots >$ is a decreasing sequence.

Proving termination of a loop

- Associate with each iteration i a natural number k_i, such that $< k_0, k_1, k_2, \ldots >$ is a decreasing sequence.
- Using the well-ordering principle, every decreasing sequence of natural numbers is finite.

Proving termination of a loop

- Associate with each iteration i a natural number k_i, such that $< k_0, k_1, k_2, \ldots >$ is a decreasing sequence.
- Using the well-ordering principle, every decreasing sequence of natural numbers is finite.
- Find a decreasing sequence of natural numbers for the while-loop in the previous example:

  Define $k_i = a - k$
  $< k_0, k_1, k_2, \ldots >$ is decresing as a is constant and k increases by 1 at each iteration.