PROGRAM VERIFICATION III: LOOPS

CS 340, FALL 2004

Purpose

This document describes how to prove that a loop satisfies some specification. The process is described for while-do loops, and then the same process is applied to other kinds of loops.

Contents

1. While-Do Loops

Structured programs can be created from sequence, if-then-else, and while-do. An unstructured program can be converted to a structured program using the constructive proof of the Structure Theorem. Trace tables can be used to verify sequences, and a procedure exists for comparing arbitrary conditional rules. It remains only to verify the correctness of loops. Each different kind of loop has its own correctness conditions, but they all follow the same pattern. This section illustrates the pattern in detail, using the while-do loop.

Consider the while loop of figure 1.1. This loop can be "unrolled" once to obtain the structure of figure 1.2. It should be obvious that these two proper programs are execution equivalent, and hence function equivalent. Assume the loop always terminates, and let its program function be f. Using referential transparency and the Axiom of Replacement we obtain the function equivalent proper program of figure 1.3. This program consists solely of an if-then-else (with the else part the identity) and a sequence, both of which we already know how to verify.

Based on this, we can conclude that the verification question for loops (where we know the loop always terminates) can be reduced to $f \subseteq [\text{if } (p)\ \{g; f;\}] = (\neg p \implies I \,|\, p \implies f \circ g)$, where I is the identity function.

Given the above, we can prove that an arbitrary while-do loop has program function f by induction. In order to do this, a special predicate **term** is introduced, defined as follows:

$$\mathbf{term}(p,g,x,n) = \left[\bigwedge_{0 \le i < n} p(g^i(x)) \right] \wedge \neg p(g^n(x)).$$

Date: 12th October 2004.

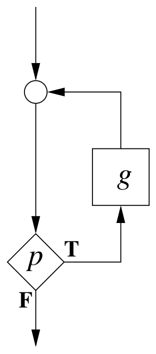

FIGURE 1.1. A while-do Loop

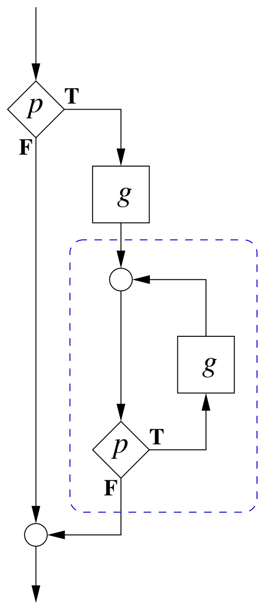

FIGURE 1.2. The while-do Loop, Unrolled Once

This predicate states that the predicate p is true for x, $g(x)$, $g(g(x))$, etc., up to $g^{n-1}(x)$, and then it fails for $g^n(x)$. In short, this captures the conditions under which the loop terminates after exactly n iterations. In particular, note that:

$$\mathrm{Dom}[P] = \{x | \exists n \geq 0, \mathbf{term}(p,g,x,n)\}.$$

(1) **Termination:** To satisfy the assumption made when applying the Axiom of Replacement, we must prove that the loop terminates for all elements of the domain of f; in other words, that its domain contains the domain of f. If the loop terminates for initial states not in the domain of f, then the loop will be, at best, only

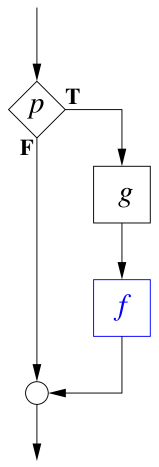

FIGURE 1.3. Equivalent Flowchart

sufficiently correct. This can be expressed as $\forall x \in \text{Dom} f, \exists n \geq 0, \textbf{term}(p,g,x,n)$, or $f \subseteq \text{Dom}[P]$.[1]

(2) **Basis Case:** Show that if p is false, then f is the identity function. This arises since the "else" case of the unrolled while-do is the identity. This can be expressed as $\forall x \in \text{Dom} f, \neg p(x) \implies f(x) = x$.

(3) **Inductive Hypothesis:** Assume that if the body of the loop is executed n times for some $n \geq 0$ and the loop then terminates, the result is correct. In other words $\forall x \in \text{Dom} f, \textbf{term}(p,g,x,n) \implies f(x) = g^n(x)$. Proving the basis case establishes that at least one such n does exist: $n = 0$.

(4) **Inductive Step:** Now show that the inductive hypothesis implies that the loop is correct when executed exactly $n+1$ times. That is, $\forall x \in \text{Dom} f, \textbf{term}(p,g,x,n+1) \implies f(x) = g^{n+1}(x)$. Considering this further we obtain the following:

$$\begin{aligned}
\forall x \in \text{Dom} f, \textbf{term}(p,g,x,n+1) &\implies f(x) = g^{n+1}(x)\\
\forall x \in \text{Dom} f, p(x) \wedge \textbf{term}(p,g,g(x),n) &\implies f(x) = g^{n}(g(x))\\
\forall x \in \text{Dom} f, p(x) \wedge \textbf{term}(p,g,g(x),n) &\implies f(x) = f(g(x))\\
\forall x \in \text{Dom} f, p(x) \wedge g(x) \in \text{Dom}[P] &\implies f(x) = f(g(x))
\end{aligned}$$

Note that if the termination condition is proven above, we can conclude that $x \in \text{Dom} f$ implies that P terminates for x. Since $p(x)$ must be true, it follows that $g(x) \in \text{Dom}[P]$, since P must terminate. We can thus drop the requirement that $g(x) \in \text{Dom}[P]$, and obtain the simpler rule $\forall x \in \text{Dom} f, p(x) \implies f(x) = f(g(x))$.

Verification that a loop `while(p){g();}` correctly implements specification f can be boiled down to three steps. Let d denote the characteristic predicate for the domain of f. That is, $(d \implies f) = f$. The following should be evident from figure 1.3:

(1) Prove the loop terminates for everything in the domain of f. This is the point at which you should consider sufficient versus complete correctness. Prove that d

[1]For those who care, this step establishes that every chain of function applications terminates; that it is *Noetherian*.

implies termination. This can often be done with a simple informal argument, but sometimes requires an inductive proof.

(2) Prove that $\forall x \in \operatorname{Dom} f, \neg p(x) \implies f(x) = x$. Alternately, prove that $(d \wedge \neg p \implies f) = (d \wedge \neg p \implies I)$.

(3) Prove that $\forall x \in \operatorname{Dom} f, p(x) \implies f(x) = f(g(x))$. Alternately, prove that $(d \wedge p \implies f) = (d \wedge p \implies f \circ g)$.

Note that when $x \in \operatorname{Dom} f \wedge p(x) \implies f(x) = f(g(x))$, this translates to a domain restriction on *both* functions; the functions $f(x)$ and $f(g(x))$ only have to agree on $\{x | x \in \operatorname{Dom} f \wedge p(x)\}$. Thus $(d \wedge \neg p \implies f) = (d \wedge \neg p \implies I)$ and $(d \wedge p \implies f) = (d \wedge p \implies f \circ g)$.

Suppose f is itself written as a conditional rule of the form $f = (q \implies r)$. Then note that $(p \implies f) \iff (p \implies (q \implies r)) \iff (p \wedge q \implies r)$.[2]

WHILE-DO CORRECTNESS QUESTION: Let $T = \{x | \exists n \geq 0, \mathbf{term}(p, g, x, n)\}$. For complete correctness $\operatorname{Dom} f = T$, $(d \wedge \neg p \implies f) = (d \wedge \neg p \implies I)$, and $(d \wedge p \implies f) = (d \wedge p \implies f \circ g)$. For sufficient correctness $\operatorname{Dom} f \subseteq T$ is required, instead.

2. EXAMPLES

Example 2.1. Consider the following C code, where `x`, `y`, and `a` are of type `unsigned int`:

```
while (x > 0) {
    x--;
    y += a;
}
```

Let f be the conditional rule $f = (x \geq 0 \implies x, y, a := 0, ax + y, a)$. Ignoring issues related to the computer representation of integers, is the above program correct with respect to this specification?

First, it must be shown that the loop terminates for every element in the domain of f. The domain of the conditional rule is $x \geq 0$. When x is zero, the loop does not execute.

[2]This is true because $p \implies (q \implies r)$ is equivalent to $p \wedge q \implies r$. You can prove this with a truth table, or by the following derivation:

$$\begin{aligned}
& p \implies (q \implies r) \\
\iff \quad & p \implies (\neg q \vee r) \\
\iff \quad & \neg p \vee (\neg q \vee r) \\
\iff \quad & (\neg p \vee \neg q) \vee r \\
\iff \quad & \neg (p \wedge q) \vee r \\
\iff \quad & p \wedge q \implies r.
\end{aligned}$$

If you don't like this, think of domain restrictions on a function $f : A \to B$:

$$\begin{aligned}
& (p \implies (q \implies f)) \\
= \quad & (p \implies f \cap \{x | q(x)\} \times B) \\
= \quad & (f \cap \{x | q(x)\} \times B) \cap (\{x | p(x)\} \times B) \\
= \quad & f \cap (\{x | q(x)\} \times B \cap \{x | p(x)\} \times B) \\
= \quad & f \cap ((\{x | q(x)\} \cap \{x | p(x)\}) \times B) \\
= \quad & f \cap (\{x | q(x) \wedge p(x)\} \times B) \\
= \quad & (p \wedge q \implies f).
\end{aligned}$$

Assume as the inductive hypothesis that the loop terminates for all $x \le n$ for some $n \ge 0$,[3] and consider $x = n+1$. Then certainly $x > 0$ and the loop body executes. In the loop, x is reduced by one, and we have $x - 1 = n + 1 - 1 = n$. By the inductive hypothesis, the loop terminates. Thus we can conclude that the loop terminates for all $x \ge 0$, as required.

An equally good argument is the following. Assume $x = 0$. Then the loop terminates immediately. Assume $x > 0$. Then each time through the loop x is decremented by one, and thus eventually $x = 0$ and the loop will terminate. Thus the loop terminates for all values of $x \ge 0$.

Second, it must be shown that $(d \wedge \neg p \implies f) = (d \wedge \neg p \implies I)$:

$$\begin{array}{rcl}
(d \wedge \neg p & \implies & f) \\
(x \ge 0 \wedge x \le 0 & \implies & (x \ge 0 \implies x,y,a := 0, ax+y, a)) \\
(x \ge 0 \wedge x \le 0 \wedge x \ge 0 & \implies & x,y,a := 0, ax+y, a) \\
(x = 0 & \implies & x,y,a := x, a(0)+y, a) \\
(x = 0 & \implies & x,y,a := x,y,a)
\end{array}$$

$$\begin{array}{rcl}
(d \wedge \neg p & \implies & I) \\
(x \ge 0 \wedge x \le 0 & \implies & x,y,a := x,y,a) \\
(x = 0 & \implies & x,y,a := x,y,a).
\end{array}$$

These two functions are identical, as required.

Finally, it must be shown that $(d \wedge p \implies f) = (d \wedge p \implies f \circ g)$. Here a trace table will be used to obtain $[g; f] = f \circ g$.

Part	Cond	x	y	a
`x--`		$x_1 = x_0 - 1$	$y_1 = y_0$	$a_1 = a_0$
`y += a`		$x_2 = x_1$	$y_2 = y_1 + a_1$	$a_2 = a_1$
$f: (x \ge 0 \implies x,y,a := 0, ax+y, a)$	$x_2 \ge 0$	$x_3 = 0$	$y_3 = a_2 x_2 + y_2$	$a_3 = a_2$

Using back-substitution the following condition is obtained:

$$\begin{array}{rl}
x_2 \ge 0 & \\
\iff & x_1 \ge 0 \\
\iff & x_0 - 1 \ge 0 \\
\iff & x_0 \ge 1.
\end{array}$$

The following assignments are obtained:

$$x_3 = 0 \qquad \begin{array}{rcl}
y_3 & = & a_2 x_2 + y_2 \\
& = & a_1 x_1 + y_1 + a_1 \\
& = & a_0(x_0 - 1) + y_0 + a_0 \\
& = & a_0 x_0 - a_0 + y_0 + a_0 \\
& = & a_0 x_0 + y_0
\end{array} \qquad \begin{array}{rcl}
a_3 & = & a_2 \\
& = & a_1 \\
& = & a_0
\end{array}$$

[3]There is at least one such n: $n = 0$. This was established by the basis case.

This gives the concurrent assignment $(x \geq 1 \implies x,y,a := 0, ax+y, a)$. Thus we have the following two conditional rules:

$$\begin{aligned}
(d \wedge p &\implies f) \\
(x \geq 0 \wedge x > 0 &\implies (x \geq 0 \implies x,y,a := 0, ax+y, a)) \\
(x \geq 0 \wedge x > 0 \wedge x \geq 0 &\implies x,y,a := 0, ax+y, a) \\
(x > 0 &\implies x,y,a := 0, ax+y, a)
\end{aligned}$$

$$\begin{aligned}
(d \wedge p &\implies f \circ g) \\
(x \geq 0 \wedge x > 0 &\implies (x \geq 1 \implies x,y,a := 0, ax+y, a)) \\
(x \geq 0 \wedge x > 0 \wedge x \geq 1 &\implies x,y,a := 0, ax+y, a) \\
(x > 0 &\implies x,y,a := 0, ax+y, a).
\end{aligned}$$

These are obviously the same function, as required.

The program is *completely correct* with respect to the specification.[4]

Example 2.2. Consider the following C source code, where all variables are of type `unsigned int`:

```
while (y != 0) {
    z += x;
    y--;
}
```

This program is claimed to be correct with respect to specification $f = (y \geq 0 \implies x,y,z := x, 0, xy + z)$. Ignoring computer number representation issues, is this program completely correct, sufficiently correct, or not correct?

The domain of the specification is $y \geq 0$. If $y = 0$, then the loop test fails, and the program terminates. Assume the program terminates for all $y \leq n$, for some $n \geq 0$, and consider $y = n+1$. Then clearly $y > 0$, and the loop executes. Within the loop, the value of y is reduced by one, to $y = n$, and by the inductive hypothesis, the loop terminates for this case. Thus the loop terminates for all $y \geq 0$, as required. Since the value of y is restricted to unsigned integers, it terminates for *only* these values, and thus the loop may be completely correct.

An equally good argument is the following. Assume $y = 0$. Then the loop terminates immediately. Assume $y > 0$. Then each time through the loop y is decremented by one, and thus eventually $y = 0$ and the loop will terminate. Thus the loop terminates for all values of $y \geq 0$.

[4]Note that when you consider the limit of the integer type (probably $2^{32} - 1$), the program is only *partially correct* on some restricted domain, since $x \geq 0$ is an unbounded domain. There are also limits on a and y, and it is possible for $ax+y$ to overflow. Thus there are elements of the domain of f for which the program gives the *wrong* answer. If this program were to become part of a numerical library, all these limits should be expressed in the documentation of the code.

Next it must be shown that $(d \wedge \neg p \implies f) = (d \wedge \neg p \implies I)$:

$$\begin{aligned}
(d \wedge \neg p &\implies f) \\
(y \geq 0 \wedge y = 0 &\implies (y \geq 0 \implies x,y,z := x,0,xy+z)) \\
(y \geq 0 \wedge y = 0 \wedge y \geq 0 &\implies x,y,z := x,0,xy+z) \\
(y = 0 &\implies x,y,z := x,y,x(0)+z) \\
(y = 0 &\implies x,y,z := x,y,z)
\end{aligned}$$

$$\begin{aligned}
(d \wedge \neg p &\implies I) \\
(y \geq 0 \wedge y = 0 &\implies x,y,z := x,y,z) \\
(y = 0 &\implies x,y,z := x,y,z).
\end{aligned}$$

These are obviously the same function.

Finally, it is necessary to show that $(d \wedge p \implies f) = (d \wedge p \implies f \circ g)$. Again, $f \circ g = [g; f]$ will be found with a trace table.

Part	Cond	x	y	z
`z += x`		$x_1 = x_0$	$y_1 = y_0$	$z_1 = z_0 + x_0$
`y--`		$x_2 = x_1$	$y_2 = y_1 - 1$	$z_2 = z_1$
$f:\ (y \geq 0 \implies x,y,z := x,0,xy+z)$	$y_2 \geq 0$	$x_3 = x_2$	$y_3 = 0$	$z_3 = x_2 y_2 + z_2$

Using back-substitution the following condition is obtained:

$$\begin{aligned}
&y_2 \geq 0 \\
\iff\ &y_1 - 1 \geq 0 \\
\iff\ &y_0 \geq 1.
\end{aligned}$$

The following assignments are obtained:

$$\begin{aligned}
x_3 &= x_2 \\
&= x_1 \\
&= x_0
\end{aligned}
\qquad
y_3 = 0
\qquad
\begin{aligned}
z_3 &= x_2 y_2 + z_2 \\
&= x_1(y_1 - 1) + z_1 \\
&= x_1 y_1 - x_1 + z_1 \\
&= x_0 y_0 - x_0 + z_0 + x_0 \\
&= x_0 y_0 + z_0
\end{aligned}$$

This gives the concurrent assignment $(y \geq 1 \implies x,y,z := x,0,xy+z)$. Thus the conditional rules to compare are the following.

$$\begin{aligned}
(d \wedge p &\implies f) \\
(y \geq 0 \wedge y \neq 0 &\implies (y \geq 0 \implies x,y,z := x,0,xy+z)) \\
(y \geq 0 \wedge y \neq 0 \wedge y \geq 0 &\implies x,y,z := x,0,xy+z) \\
(y > 0 &\implies x,y,z := x,0,xy+z)
\end{aligned}$$

$$\begin{aligned}(d \wedge p &\implies f \circ g)\\ (y \geq 0 \wedge y \neq 0 &\implies (y \geq 1 \implies x,y,z := x,0,xy+z))\\ (y \geq 0 \wedge y \neq 0 \wedge y \geq 1 &\implies x,y,z := x,0,xy+z)\\ (y \geq 1 &\implies x,y,z := x,0,xy+z).\end{aligned}$$

Since y is an integer, $y > 0 \iff y \geq 1$, and the two functions are identical.

The program is *completely correct* with respect to the specification.

Example 2.3. Consider the following C code where x and y are both of type int (thus signed).

```
while (y > 0) {
    x++;
    y--;
}
```

The following specification is given: $f = (x \geq 0 \wedge y \geq 0 \implies x,y := x+y,0)$. Is the program sufficiently, completely, or not correct with respect to this specification?

The domain of f is $x \geq 0 \wedge y \geq 0$. For the program if $y = 0$ then the loop does not execute. Assume that the loop terminates for all $0 \leq y \leq n$ for some $n \geq 0$, and consider $y = n+1$. Then since $y > 0$ the loop executes, and at the end $y = n$. By the inductive hypothesis, the loop terminates when $0 \leq y \leq n$, so we conclude the loop terminates for all $y \geq 0$. Note that $y \geq 0 \implies x \geq 0 \wedge y \geq 0$.

The domain of the program is the set of all values of x and y for which the program terminates. We already know the program terminates whenever $y \geq 0$. The loop does not execute when $y \leq 0$, so we can conclude that the domain of the program is $y \geq 0 \vee y \leq 0 \iff T$. This should not seem odd; remember that the loop above can be re-written as:

```
if (y > 0) {
    x++;
    y--;
    P;
} else {}
```

This has the domain $y > 0 \vee y \leq 0$, which is again just T. As with the if-then-else, you have to consider the case when the predicate is false.

Since the domain of the program is T, and the domain of the function f is $x \geq 0 \wedge y \geq 0$, we have the following:

- Sufficient correctness: $x \geq 0 \wedge y \geq 0 \implies T$. Since the conclusion is true, this implication is true, and the program is at least sufficiently correct.
- Complete correctness: $x \geq 0 \wedge y \geq 0 \iff T$. Since x and y are signed integers, this equality is contingent on the values of x and y, and is not necessarily true. Thus the program is not completely correct.[5]

An equally good argument about termination is the following. Assume $y = 0$. Then the loop terminates immediately. Assume $y > 0$. Then each time through the loop y is

[5]Remember that for sets $A = B$ is true iff $A \subseteq B$ and $B \subseteq A$. If we let a and b be the characteristic predicates of A and B, then we can translate this into the world of predicate logic, and get $a \iff b$ true iff $a \implies a$ and $b \implies a$. In this case, we'd have to show $x \geq 0 \wedge y \geq 0 \implies T$ (which we know is true), and $T \implies x \geq 0 \wedge y \geq 0$, which we known is not necessarily true. If x and y were unsigned, then we could replace $x \geq 0 \wedge y \geq 0$ with T, and the implication would be true.

decremented by one, and thus eventually $y = 0$ and the loop will terminate. Thus the loop terminates for all values of $y \geq 0$, irrespective of the value of x.

Next consider the while predicate to be false:

$$\begin{array}{rcl}
(d \wedge \neg p & \Longrightarrow & f) \\
(x \geq 0 \wedge y \geq 0 \wedge y \leq 0 & \Longrightarrow & (x \geq 0 \wedge y \geq 0 \implies x,y := x+y,0)) \\
(x \geq 0 \wedge y \geq 0 \wedge y \leq 0 \wedge x \geq 0 \wedge y \geq 0 & \Longrightarrow & x,y := x+y,0) \\
(y = 0 \wedge x \geq 0 & \Longrightarrow & x,y := x,y)
\end{array}$$

$$\begin{array}{rcl}
(d \wedge \neg p & \Longrightarrow & I) \\
(x \geq 0 \wedge y \geq 0 \wedge y \leq 0 & \Longrightarrow & x,y := x,y) \\
(y = 0 \wedge x \geq 0 & \Longrightarrow & x,y := x,y).
\end{array}$$

These are the same function, as required.

Finally, a trace table is used to compute $f \circ g = [g, f]$:

Part	Cond	x	y
`x++`		$x_1 = x_0 + 1$	$y_1 = y_0$
`y--`		$x_2 = x_1$	$y_2 = y_1 - 1$
$f: \ (x \geq 0 \wedge y \geq 0 \implies x,y := x+y,0)$	$x_2 \geq 0 \wedge y_2 \geq 0$	$x_3 = x_2 + y_2$	$y_3 = 0$

Using back-substitution the following condition is obtained:

$$\begin{array}{ll}
x_2 \geq 0 \wedge y_2 \geq 0 & \\
\Longleftrightarrow & x_1 \geq 0 \wedge y_1 - 1 \geq 0 \\
\Longleftrightarrow & x_0 + 1 \geq 0 \wedge y_0 - 1 \geq 0 \\
\Longleftrightarrow & x_0 \geq -1 \wedge y_0 \geq 1.
\end{array}$$

Because the variables are unsigned, we can reduce this to $x \geq 0 \wedge y \geq 1$. The following assignments are obtained:

$$\begin{array}{rcl}
x_3 & = & x_2 + y_2 \\
& = & x_1 + y_1 - 1 \\
& = & x_0 + 1 + y_0 - 1 \\
& = & x_0 + y_0
\end{array}
\qquad\qquad y_3 = 0$$

This gives the concurrent assignment $(x \geq 0 \wedge y \geq 1 \implies x,y := x+y,0)$. Thus the conditional rules to compare are the following.

$$\begin{array}{rcl}
(d \wedge p & \Longrightarrow & f) \\
(x \geq 0 \wedge y \geq 0 \wedge y > 0 & \Longrightarrow & (x \geq 0 \wedge y \geq 0 \implies x,y := x+y,0)) \\
(x \geq 0 \wedge y \geq 0 \wedge y > 0 \wedge x \geq 0 \wedge y \geq 0 & \Longrightarrow & x,y := x+y,0) \\
(y > 0 \wedge x \geq 0 & \Longrightarrow & x,y := x+y,0)
\end{array}$$

$$\begin{aligned}
(d \wedge p &\implies f \circ g) \\
(x \geq 0 \wedge y \geq 0 \wedge y > 0 &\implies (x \geq 0 \wedge y \geq 1 \implies x, y := x + y, 0)) \\
(x \geq 0 \wedge y \geq 0 \wedge y > 0 \wedge x \geq 0 \wedge y \geq 1 &\implies x, y := x + y, 0) \\
(y > 0 \wedge x \geq 0 &\implies x, y := x + y, 0).
\end{aligned}$$

These are the same function, as required.

The program is *sufficiently correct* with respect to the specification.