MATHEMATICAL THEORY OF COMPUTATION

COMPUTER
SCIENCE
SERIES

CHAPTER 3

Verification of Programs

Introduction

The purpose of this chapter is to describe methods for verifying computer programs. Suppose we are given a computer program with a description of its behavior, that is, a characteristic predicate (later called an *output predicate*), which describes the relationships among the program variables that must be satisfied at the completion of the program execution. Sometimes we are also given an *input predicate*, which defines the input restrictions that must be satisfied to make execution of the program meaningful. Our task is to prove that the program is correct with respect to such input and output predicates; that is, for all program executions with inputs satisfying the input predicate, we must guarantee that the program is terminated and that at the completion of execution the output predicate is satisfied. In this chapter we shall describe the construction of such proofs of correctness.

In order to discuss programs and their correctness we must specify a programming language. We shall consider flowchart programs without arrays (Sec. 3-1) and with arrays (Sec. 3-2), as well as simple Algol-like programs (Sec. 3-3). Methods for proving correctness of recursive programs are discussed in detail in Chap. 5.

3-1 FLOWCHART PROGRAMS

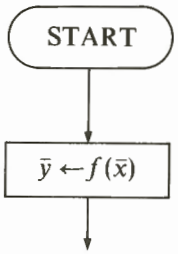
First let us consider a very simple class of flowchart programs. We distinguish among three types of variables (grouped as three vectors): (1) an *input vector* $\bar{x} = (x_1, x_2, \ldots, x_n)$, which consists of the given input values and therefore never changes during computation; (2) a *program vector* $\bar{y} =$

$(y_1, y_2, \ldots, y_b)$, which is used as temporary storage during computation; and (3) an *output vector* $\bar{z} = (z_1, z_2, \ldots, z_c)$, which yields the output values when computation terminates. Correspondingly, we also distinguish among three types of (nonempty) domains: (1) an *input domain* $D_{\bar{x}}$, (2) a *program domain* $D_{\bar{y}}$, and (3) an *output domain* $D_{\bar{z}}$. Each domain is actually a cartesian product of subdomains:

$$D_{\bar{x}} = D_{x_1} \times D_{x_2} \times \cdots \times D_{x_a}$$
$$D_{\bar{y}} = D_{y_1} \times D_{y_2} \times \cdots \times D_{y_b}$$
$$D_{\bar{z}} = D_{z_1} \times D_{z_2} \times \cdots \times D_{z_c}$$

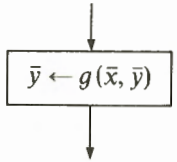
We also distinguish among four types of statements:†

1. START statement



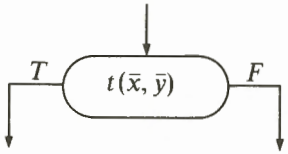
where $f(\bar{x})$ is a total function mapping $D_{\bar{x}}$ into $D_{\bar{y}}$.

2. ASSIGNMENT statement



where $g(\bar{x}, \bar{y})$ is a total function mapping $D_{\bar{x}} \times D_{\bar{y}}$ into $D_{\bar{y}}$

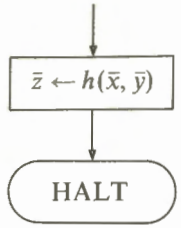
3. TEST statement



where $t(\bar{x}, \bar{y})$ is a total predicate over $D_{\bar{x}} \times D_{\bar{y}}$

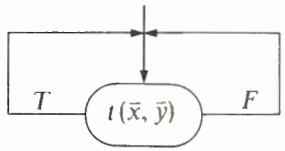
† In addition, we clearly allow the JOINT statement for combining two or more arcs (arrows) into a single one.

4. HALT statement



where $h(\bar{x}, \bar{y})$ is a total function mapping $D_{\bar{x}} \times D_{\bar{y}}$ into $D_{\bar{z}}$

A *flowchart program* is simply any flow-diagram constructed from these statements (with exactly one START statement) such that every ASSIGNMENT or TEST statement is on a path from the START statement to some HALT statement. In other words, flowchart programs are not allowed to include "dead-end" TEST statements such as



Given such a flowchart program P and an input value $\bar{\xi} \in D_{\bar{x}}$ for the input vector $\bar{x}$, the program can be executed. Execution always begins at the START statement by initializing the value of $\bar{y}$ to $f(\bar{\xi})$ and then proceeds in the normal way, following the arcs from statement to statement. Whenever an ASSIGNMENT statement is reached, the value of $\bar{y}$ is replaced by the value of $g(\bar{x}, \bar{y})$ for the current values $\bar{x}$ and $\bar{y}$. Whenever a TEST statement is reached, execution follows the T or F branch, depending on whether the current value of $t(\bar{x}, \bar{y})$ is *true* or *false*; the value of $\bar{y}$ is unchanged by a TEST statement. If the execution terminates (i.e., reaches a HALT statement), $\bar{z}$ is assigned the current value, say, $\bar{\zeta}$, of $h(\bar{x}, \bar{y})$ and we say that $P(\bar{\xi})$ *is defined and* $P(\bar{\xi}) = \bar{\zeta}$; otherwise, i.e., if the execution never terminates, we say that $P(\bar{\xi})$ *is undefined.* In other words, *the program* P *should be considered as representing a partial function* $\bar{z} = P(\bar{x})$ *mapping* $D_{\bar{x}}$ *into* $D_{\bar{z}}$.

For example, the flowchart program in Fig. 3-1 performs the integer division of x_1 by x_2, where $x_1 \geqq 0$ and $x_2 > 0$, yielding a quotient z_1 and a remainder z_2; that is, $z_1 = div(x_1, x_2)$, and $z_2 = rem(x_1, x_2)$. Here $\bar{x} = (x_1, x_2)$, $\bar{y} = (y_1, y_2)$, $\bar{z} = (z_1, z_2)$, and $D_{\bar{x}} = D_{\bar{y}} = D_{\bar{z}} = \{\text{all pairs of integers}\}$.

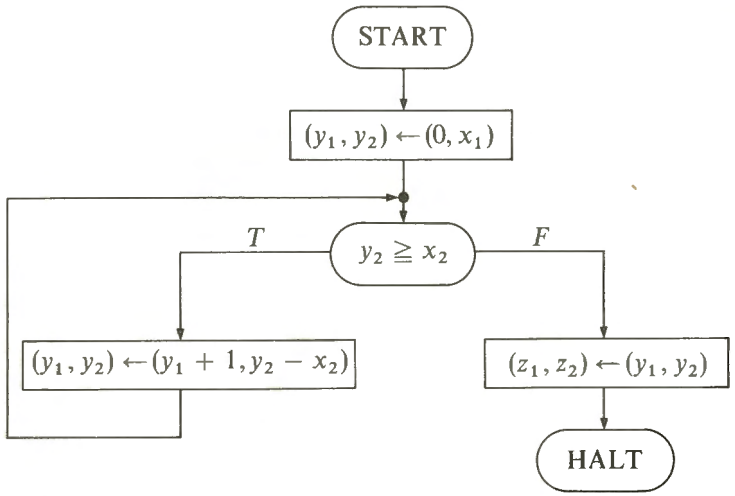


Figure 3-1 Flowchart program for performing integer division.

Note that $(y_1, y_2) \leftarrow (0, x_1)$ means that y_1 is replaced by 0 and y_2 is replaced by x_1; similarly $(y_1, y_2) \leftarrow (y_1 + 1, y_2 - x_2)$ means that y_1 is replaced by $y_1 + 1$ and y_2 is replaced by $y_2 - x_2$. In general, we shall use the notation $(y_1, y_2, \ldots, y_n) \leftarrow (g_1(\bar{x}, \bar{y}), g_2(\bar{x}, \bar{y}), \ldots, g_n(\bar{x}, \bar{y}))$ to indicate that the variables y_i, $1 \leqq i \leqq n$, are replaced by $g_i(x, y)$ *simultaneously*; that is, *all* the g_i's are evaluated before any y_i is changed. For example, if $y_1 = 1$ and $y_2 = 2$, the assignment $(y_1, y_2) \leftarrow (y_1 + 1, y_1 + y_2)$ yields $y_2 = 3$, not $y_2 = 4$. The assignment $(y_1, y_2) \leftarrow (y_2, y_1)$ has the effect of exchanging the contents of y_1 and y_2.

The *verification* of a flowchart program depends on two given predicates:

1. A total predicate $\varphi(\bar{x})$ over $D_{\bar{x}}$, called an *input predicate*, which describes those elements of $D_{\bar{x}}$ that may be used as inputs. In other words, we are interested in the program's performance only for those elements of $D_{\bar{x}}$ satisfying the predicate $\varphi(\bar{x})$. In the special case where we are interested in the program's performance for all elements of $D_{\bar{x}}$, we shall let $\varphi(\bar{x})$ be T; that is, $\varphi(\bar{x})$ is *true* for all elements of $D_{\bar{x}}$.

2. A total predicate $\psi(\bar{x}, \bar{z})$ over $D_{\bar{x}} \times D_{\bar{z}}$, called an *output predicate*, which describes the relationships that must be satisfied between the input variables and the output variables at the completion of program execution.

We may then define the following:

1. *P terminates over* φ if for every input $\bar{\xi}$, such that $\varphi(\bar{\xi})$ is *true*, the computation of the program terminates.

2. *P is partially correct with respect to* (*wrt*) φ *and* ψ if for every $\bar{\xi}$ such that $\varphi(\bar{\xi})$ is *true* and the computation of the program terminates, $\psi(\bar{\xi}, P(\bar{\xi}))$ is *true*.

3. *P is totally correct with respect to* (*wrt*) φ *and* ψ if for every $\bar{\xi}$ such that $\varphi(\bar{\xi})$ is *true*, the computation of the program terminates and $\psi(\bar{\xi}, P(\bar{\xi}))$ is *true*.

Thus, in partial correctness we "don't care" about termination, but in total correctness termination is essential. Verifying a program for a given input predicate $\varphi(\bar{x})$ and an output predicate $\psi(\bar{x}, \bar{z})$ means showing that it is totally correct wrt φ and ψ. However, usually it is most convenient to prove the program in two separate steps: first prove partial correctness wrt φ and ψ, and then prove termination over φ.

We shall now introduce methods for proving both partial correctness and termination of flowchart programs. Let us demonstrate the correctness of the division program introduced previously. First we show that the program is partially correct wrt the input predicate

$$\varphi(x_1, x_2): \quad x_1 \geqq 0 \wedge x_2 \geqq 0$$

(which asserts that we are interested in the program's performance when both x_1 and x_2 are nonnegative) and the output predicate

$$\psi(x_1, x_2, z_1, z_2): \quad x_1 = z_1 x_2 + z_2 \wedge 0 \leqq z_2 < x_2$$

(which is essentially a definition of integer division). Then we show that the program terminates over

$$\varphi'(x_1, x_2): \quad x_1 \geqq 0 \wedge x_2 > 0$$

Thus, since the program is partially correct wrt φ and ψ and terminates over φ', it follows that the program is totally correct wrt φ' and ψ. Note the difference between φ and φ'; it is essential to exclude the case $x_2 = 0$ when termination is discussed because the program does not terminate for $x_2 = 0$.

Partial correctness To prove the partial correctness of the division program wrt φ and ψ (see Fig. 3-2), we attach the input predicate φ to point A and the output predicate ψ to point C. The main problem in verifying programs is how to handle loops. The loop of this program becomes manageable by "cutting" the program at point B, which decomposes the program flow into three paths: the first path is from A to B (arcs 1 and 2); the second is from B around the loop and back to B (arcs 3, 4, and 5); and the third is from B to C (arcs 3, 6, and 7). We identify these three paths as

α (START), β (LOOP), and γ (HALT), respectively. All terminating executions of the program must first follow path α, then pass some number of times (possibly zero) around the loop β, and finally finish with path γ; thus all executions are "covered" by these three paths.

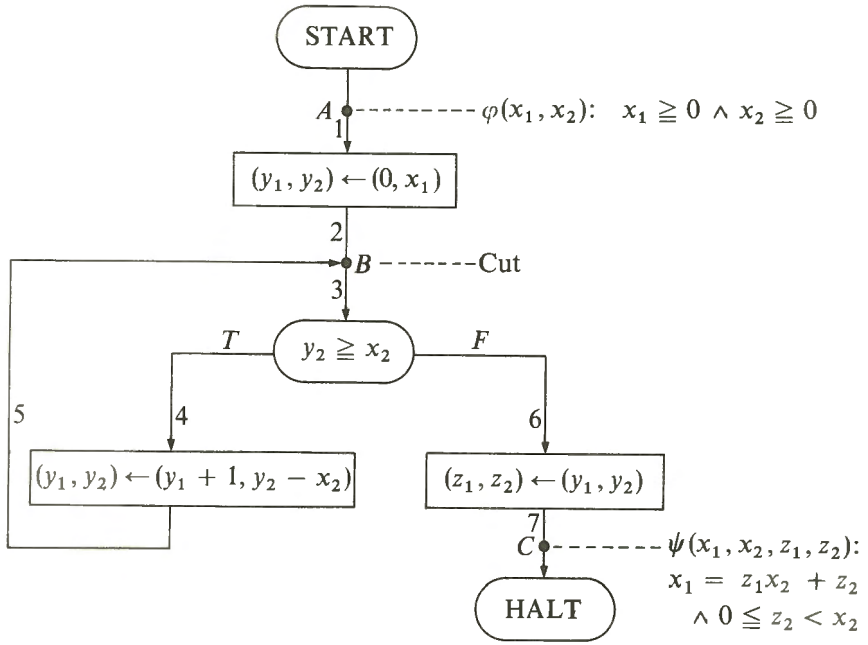


Figure 3-2 Flowchart program for performing integer division (with the input and output predicates).

In order to prove the partial correctness of the program, first we must find a predicate $p(x_1, x_2, y_1, y_2)$ describing the relationships among the program variables at cutpoint B. An appropriate predicate for this purpose is obtained by taking $p(x_1, x_2, y_1, y_2)$ to be

$$x_1 = y_1 x_2 + y_2 \ \wedge y_2 \geqq 0$$

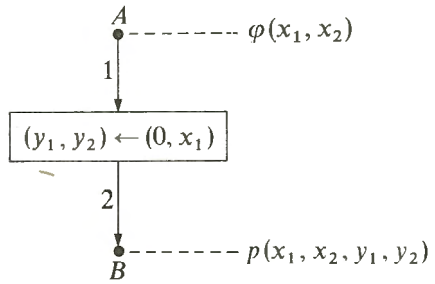
Having obtained this predicate, we have covered the program by three paths, each of which begins and ends with a predicate. The partial correctness of the program is proved by verifying each one of the three paths α, β, and γ by showing that if its initial predicate is *true* for some values of $\bar{x}$ and $\bar{y}$, and the path is executed then its final predicate will be *true* for the new values of $\bar{x}$ and $\bar{y}$.

In the integer division program, verification of path α establishes that $p(x_1, x_2, y_1, y_2)$ is *true* on the first entrance to the loop of the program (assuming the input predicate of the program is satisfied). Verification of path β shows that if $p(x_1, x_2, y_1, y_2)$ is *true* at the entrance of the loop the first time, it will be *true* the second time; if it is *true* the second time, it will be *true* the third time; etc. Verification of path γ shows that when the loop is left with predicate $p(x_1, x_2, y_1, y_2)$ *true*, then the output predicate of the program is *true*. In other words, verification of paths α and β guarantees that $p(x_1, x_2, y_1, y_2)$ has the property that whenever the computation of the program reaches the cutpoint B, $p(x_1, x_2, y_1, y_2)$ is *true* for the current values of x_1, x_2, y_1, and y_2. Therefore, verification of path γ implies that *whenever* the computation of the program reaches point C, the output predicate of the program is *true*.

In order to complete the proof of the partial correctness of the division program, we must verify the paths α, β, and γ. The verification of a path is performed in two steps: First we construct a verification condition for each path in terms of the given predicates, and then we prove it.

Constructing a verification condition of a path is usually done by moving *backward* through the path, considering each statement in turn.

Path α: Let's consider first path α of Fig. 3-2.



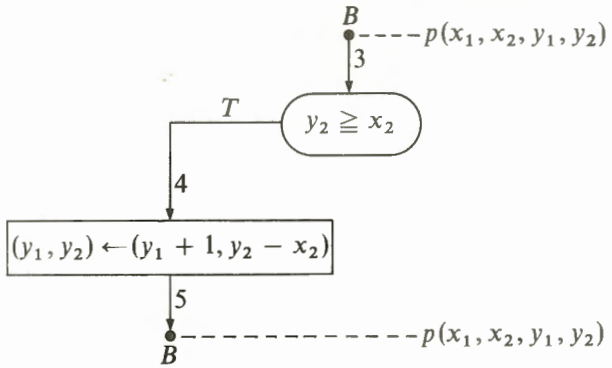
What must be *true* at point A, that is, before the execution of the statement $(y_1, y_2) \leftarrow (0, x_1)$, to ensure that $p(x_1, x_2, y_1, y_2)$ is *true* after its execution? The answer is $p(x_1, x_2, 0, x_1)$, which is formed by substituting 0 for all occurrences of y_1 in the predicate and x_1 for all occurrences of y_2. Thus, the verification condition of this path is

$$\varphi(x_1, x_2) \supset p(x_1, x_2, 0, x_1)$$

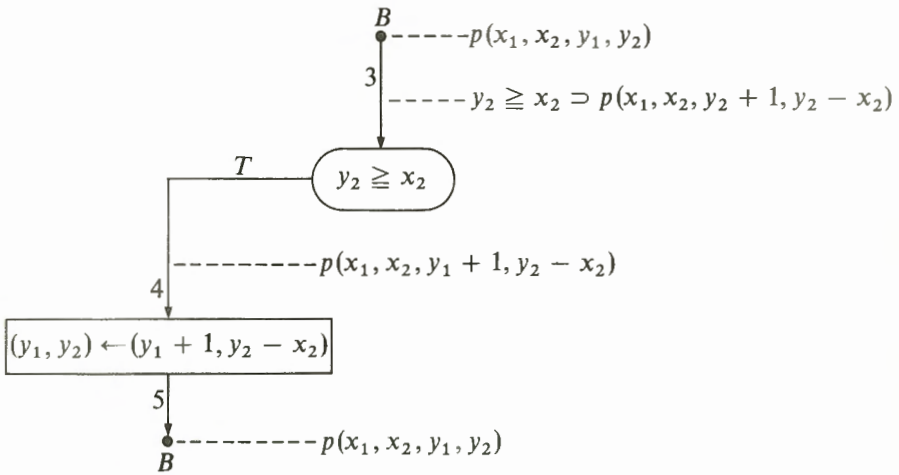
that is,

$$[x_1 \geqq 0 \ \wedge x_2 \geqq 0] \supset [x_1 = 0 \cdot x_2 + x_1 \ \wedge x_1 \geqq 0]$$

Path β: To construct the verification condition of path β, the TEST statement must be handled. As we follow path β, the TEST statement finds $y_2 \geqq x_2$, which can be shown as an annotated piece of flowchart:



Deriving the predicate $p(x_1, x_2, y_1, y_2)$ backward past the ASSIGNMENT statement yields $p(x_1, x_2, y_1 + 1, y_2 - x_2)$ at arc 4. Although the TEST statement does not change the value of any program variable, it does supply additional useful information because, after the test, it is clear that $y_2 \geqq x_2$. To handle this case, the same question used on ASSIGNMENT statements is asked: What must be *true* at arc 3 so that when control takes the T branch of the TEST statement, that is, when $y_2 \geqq x_2$ is *true*, the predicate $p(x_1, x_2, y_1 + 1, y_2 - x_2)$ will be *true* at arc 4? In this case, the answer is simply $y_2 \geqq x_2 \supset p(x_1, x_2, y_1 + 1, y_2 - x_2)$. Thus, the complete analysis of path β is



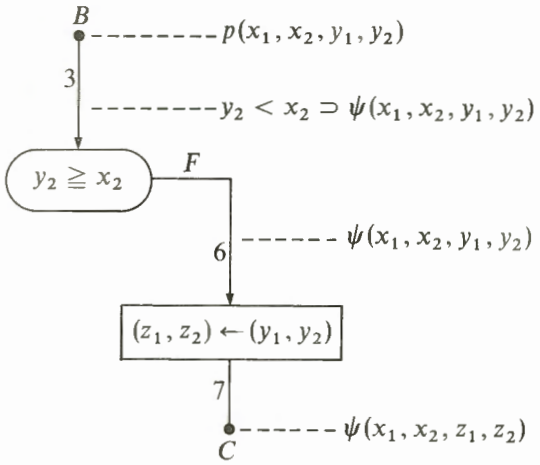
In this case the verification condition to be proved is

$$p(x_1, x_2, y_1, y_2) \supset [y_2 \geqq x_2 \supset p(x_1, x_2, y_1 + 1, y_2 - x_2)]$$

or, equivalently,

$$p(x_1, x_2, y_1, y_2) \wedge y_2 \geqq x_2 \supset p(x_1, x_2, y_1 + 1, y_2 - x_2)\dagger$$

Path γ: The analysis of path γ results in



The verification condition to be proved is

$$p(x_1, x_2, y_1, y_2) \supset [y_2 < x_2 \supset \psi(x_1, x_2, y_1, y_2)]$$

or equivalently,

$$p(x_1, x_2, y_1, y_2) \wedge y_2 < x_2 \supset \psi(x_1, x_2, y_1, y_2)$$

Thus, we have formed the three verification conditions:

$$\varphi(x_1, x_2) \supset p(x_1, x_2, 0, x_1) \qquad (\alpha)$$
$$p(x_1, x_2, y_1, y_2) \wedge y_2 \geqq x_2 \supset p(x_1, x_2, y_1 + 1, y_2 - x_2) \qquad (\beta)$$
$$p(x_1, x_2, y_1, y_2) \wedge y_2 < x_2 \supset \psi(x_1, x_2, y_1, y_2) \qquad (\gamma)$$

where

$\varphi(x_1, x_2)$ is $x_1 \geqq 0 \wedge x_2 \geqq 0$
$p(x_1, x_2, y_1, y_2)$ is $x_1 = y_1 x_2 + y_2 \wedge y_2 \geqq 0$
$\psi(x_1, x_2, z_1, z_2)$ is $x_1 = z_1 x_2 + z_2 \wedge 0 \leqq z_2 < x_2$

The reader can check for himself that for all integers x_1, x_2, y_1, and y_2, the three verification conditions are *true*; thus, the program is partially correct wrt φ and ψ.

†Throughout this chapter it should be understood that an expression of the form $A_1 \wedge A_2 \supset B$ stands for $(A_1 \wedge A_2) \supset B$ and, in general, $A_1 \wedge A_2 \wedge \ldots \wedge A_n \supset B$ stands for $(A_1 \wedge A_2 \wedge \ldots \wedge A_n) \supset B$.

Termination Actually, so far we have only partially verified our program: We have proved that whenever a computation of the program terminates (that is, reaches a HALT statement), the output predicate is *true*; however, we have not proved at all that the computation of the program indeed terminates. Thus, in order to complete the verification of the division program, we must prove termination as well.

We shall now prove that the program terminates for every input x_1 and x_2 where

$$\varphi'(x_1, x_2): \quad x_1 \geqq 0 \wedge x_2 > 0$$

(Note, again, that the program does not terminate for $x_2 = 0$.) First we show that the predicate

$$q(x_1, x_2, y_1, y_2): \quad y_2 \geqq 0 \wedge x_2 > 0$$

has the property that whenever we reach point B during the computation, $q(x_1, x_2, y_1, y_2)$ is *true* for the current values of the variables. For this purpose, we must prove the following two verification conditions:

$$\varphi'(x_1, x_2) \supset q(x_1, x_2, 0, x_1) \qquad (\alpha)$$

$$q(x_1, x_2, y_1, y_2) \wedge y_2 \geqq x_2 \supset q(x_1, x_2, y_1 + 1, y_2 - x_2) \qquad (\beta)$$

that is,

$$(x_1 \geqq 0 \wedge x_2 > 0) \supset (x_1 \geqq 0 \wedge x_2 > 0)$$

$$(y_2 \geqq 0 \wedge x_2 > 0 \wedge y_2 \geqq x_2) \supset (y_2 - x_2 \geqq 0 \wedge x_2 > 0)$$

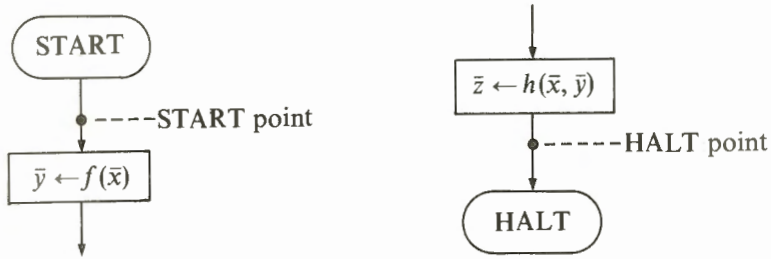
which are clearly *true*.

We observe now that since we always have $x_2 > 0$ at point B, whenever we go around the loop (from B back to B), *the value of* y_2 *decreases*. Also, we know that $y_2 \geqq 0$ whenever we reach point B. Thus, since there is no infinite decreasing sequence of natural numbers, we cannot go infinitely many times around the loop; in other words, the computation must terminate. Using this approach, we present a general technique for proving termination of programs in Sec. 3-1.2.

3-1.1 Partial Correctness

Let us generalize the technique demonstrated in verifying the division program. Suppose we are given a flowchart program P, an input predicate φ, and an output predicate ψ; to prove that P is partially correct wrt φ and ψ we proceed as follows.

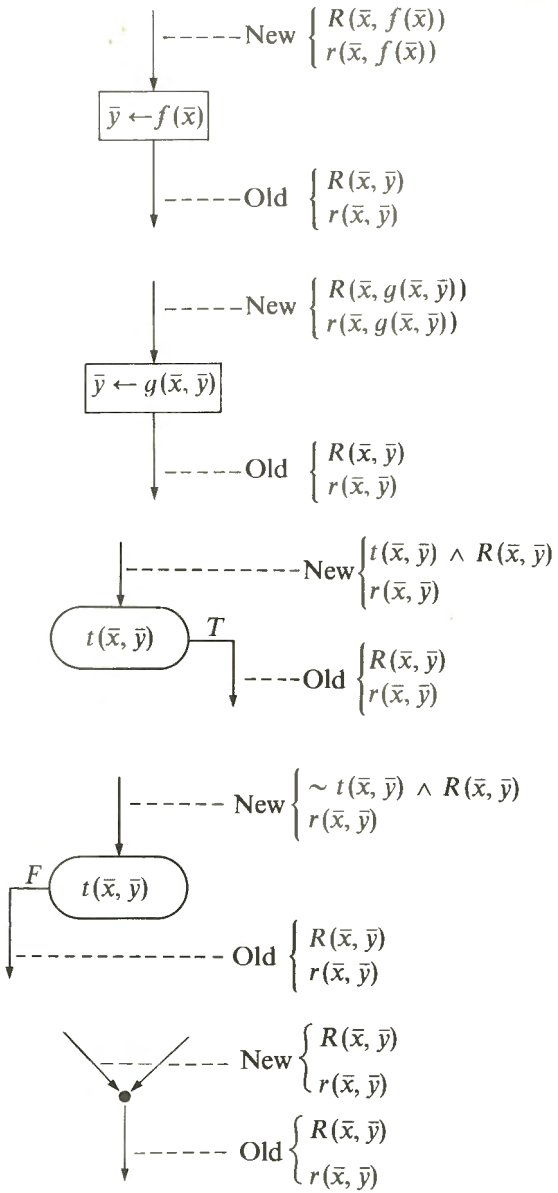
Step 1. *Cutpoints.* The first step is to cut the loops of the program by choosing on the arcs of the flowchart a finite set of points, called *cutpoints*, in such a way that every loop includes at least one such cutpoint. To this set of cutpoints we add a point on the arc leading from the START box, called the *START point*, and, for each HALT statement, a point on the arc leading to the HALT box, called a *HALT* point.



We consider only paths which start and end at cutpoints and which have no intermediate cutpoints. Each such path is finite because of the restriction that every loop includes a cutpoint, and there can be only a finite number of paths. Let α be a path leading from cutpoint i to j.† In the following discussion, we shall make use of the predicate $R_\alpha(\bar{x}, \bar{y})$, which indicates the condition for this path to be traversed, and $r_\alpha(\bar{x}, \bar{y})$, which describes the transformation of the values of $\bar{y}$ effected while the path is traversed. Thus, $R_\alpha(\bar{x}, \bar{y})$ is a predicate over $D_{\bar{x}} \times D_{\bar{y}}$, and $r_\alpha(\bar{x}, \bar{y})$ is a function mapping $D_{\bar{x}} \times D_{\bar{y}}$ into $D_{\bar{y}}$. Both R_α and r_α are expressed in terms of the functions and tests used in α; a simple method for deriving them is to use the *backward-substitution technique.*

Initially, $R(\bar{x}, \bar{y})$ is set to T and $r(\bar{x}, \bar{y})$ is set to $\bar{y}$, and both are attached to cutpoint j; then, in each step, the old R and r are used to construct the new R and r, moving *backward* toward cutpoint i. The final R and r obtained at cutpoint i are the desired R_α and r_α. The rules for constructing the new R and r in each step are:

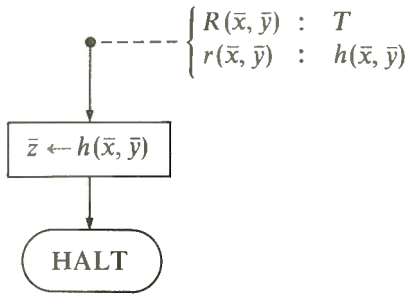
†Note that $i = j$ is possible, as in the division example.



Consider, for example, the backward-substitution technique for path α described in Fig. 3-3. Starting with T for R and $\bar{y}$ for r at cutpoint j, we proceed backward and at cutpoint i finally obtain

$$R_\alpha(\bar{x}, \bar{y}): \; t_1(\bar{x}, g_1(\bar{x}, \bar{y})) \wedge \sim t_2(\bar{x}, g_2(\bar{x}, g_1(\bar{x}, \bar{y})))$$
$$r_\alpha(\bar{x}, \bar{y}): \; g_3(\bar{x}, g_2(\bar{x}, g_1(\bar{x}, \bar{y})))$$

In the special case that j is a HALT point, r is initialized to $\bar{z}$ and R to T; then in the first step of moving backward we obtain



The process then continues by moving backward, as described previously. In the special case that i is the START point, both $R_\alpha(\bar{x}, \bar{y})$ and $r_\alpha(\bar{x}, \bar{y})$ contain no $\bar{y}$'s; R_α is a predicate over $D_{\bar{x}}$, and r_α is a function mapping $D_{\bar{x}}$ into $D_{\bar{y}}$.

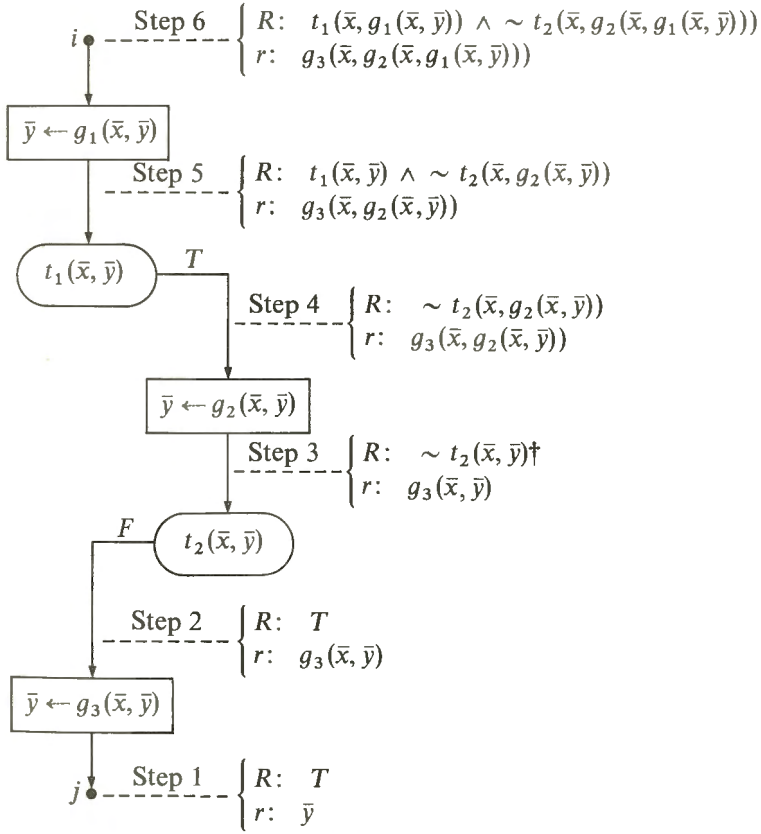


Figure 3-3 Backward-substitution technique.

†Note that we have used the fact that $\sim t_2(\bar{x}, \bar{y}) \wedge T$ is logically equivalent to $\sim t_2(\bar{x}, \bar{y})$.

Step 2. *Inductive assertions.* The second step is to associate with each cutpoint i of the program a predicate $p_i(\bar{x}, \bar{y})$ (often called *inductive assertion*), which purports to characterize the relation among the variables at this point; that is, $p_i(\bar{x}, \bar{y})$ will have the property that whenever control reaches point i, then $p_i(\bar{x}, \bar{y})$ must be *true* for the current values of $\bar{x}$ and $\bar{y}$ at this point. The input predicate $\varphi(\bar{x})$ is attached to the START point, and the output predicate $\psi(\bar{x}, \bar{z})$ is attached to the HALT points.

Step 3. *Verification conditions.* The third step is to construct for every path α leading from cutpoint i to j the verification condition:

$$\forall \bar{x} \forall \bar{y} [p_i(\bar{x}, \bar{y}) \wedge R_\alpha(\bar{x}, \bar{y}) \supset p_j(\bar{x}, r_\alpha(\bar{x}, \bar{y}))]$$

This condition states simply that if p_i is *true* for some values of $\bar{x}$ and $\bar{y}$, and $\bar{x}$ and $\bar{y}$ are such that starting with them at point i the path α will indeed be selected, then p_j is *true* for the new values of $\bar{x}$ and $\bar{y}$ after the path α is traversed.

In the special case that j is a HALT point, the verification condition is

$$\forall \bar{x} \forall \bar{y} [p_i(\bar{x}, \bar{y}) \wedge R_\alpha(\bar{x}, \bar{y}) \supset \psi(\bar{x}, r_\alpha(\bar{x}, \bar{y}))]$$

and in the case that i is the START point, the verification condition is

$$\forall \bar{x} [\varphi(\bar{x}) \wedge R_\alpha(\bar{x}) \supset p_j(\bar{x}, r_\alpha(\bar{x}))]$$

The final step is to prove that all these verification conditions for our choice of inductive assertions are *true.* Proving the verification conditions implies that each predicate attached to a cutpoint has the property that whenever control reaches the point, the predicate is *true* for the current values of the variables; in particular, whenever control reaches a HALT point, $\psi(\bar{x}, \bar{z})$ is *true* for the current values of $\bar{x}$ and $\bar{z}$. In other words, proving the verification conditions implies that the given program P is partially correct wrt φ and ψ.

To summarize, we have the following theorem.

THEOREM 3-1 [INDUCTIVE-ASSERTATIONS METHOD (Floyd)]

For a given flowchart program P, an input predicate $\varphi(\bar{x})$, and an output predicate $\psi(\bar{x}, \bar{z})$, apply the following steps: (1) Cut the loops; (2) find an appropriate set of inductive assertions; and (3) construct the verification conditions. If all the verification conditions are true, then P is partially correct wrt φ and ψ.

In general, all the steps are quite mechanical, except for step 2; discovering the proper inductive assertion requires a deep understanding of the program's performance. Let us illustrate the inductive-assertions method with a few examples.

EXAMPLE 3-1

The program P_1 over the integers (Fig. 3-4) computes $z = \lfloor \sqrt{x} \rfloor$ for every natural number x; that is, the final value of z is the largest integer k such that $k \leqq \sqrt{x}$. The computation method is based on the fact that $1 + 3 + 5 + \cdots + (2n + 1) = (n + 1)^2$ for every $n \geqq 0$; n is computed in y_1, the odd number $2n + 1$ in y_3, and the sum $1 + 3 + 5 + \cdots + (2n + 1)$ in y_2.

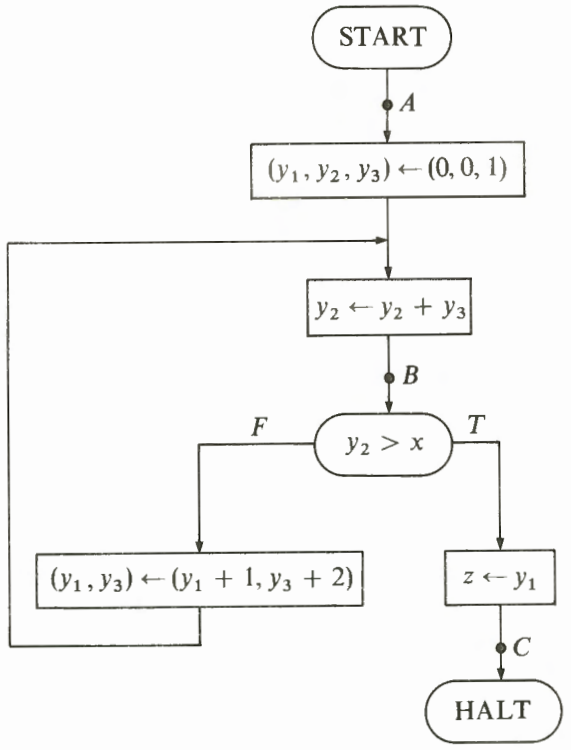
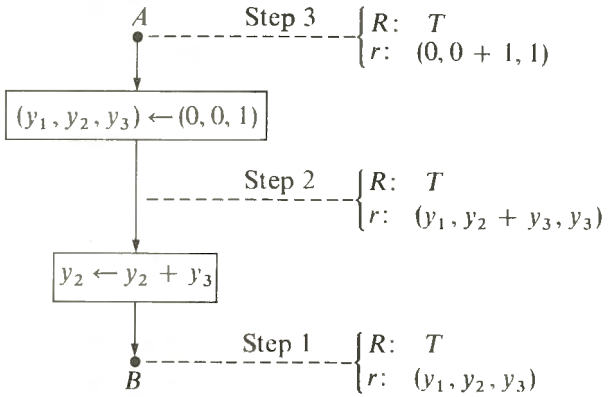


Figure 3-4 Program P_1 for computing $z = \lfloor \sqrt{x} \rfloor$

We shall prove that the program is partially correct wrt the input predicate $\varphi(x)$: $x \geqq 0$ and the output predicate $\psi(x, z)$: $z^2 \leqq x < (z + 1)^2$. For this purpose first we cut the only loop of the program at

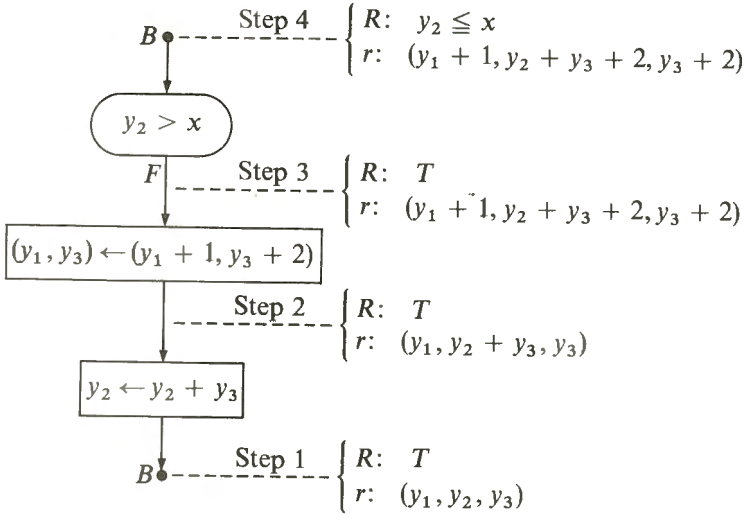
point B and therefore have three control paths to consider. Using the backward-substitution technique, we obtain the predicate R and term r for each path as follows.

Path α (from A to B):



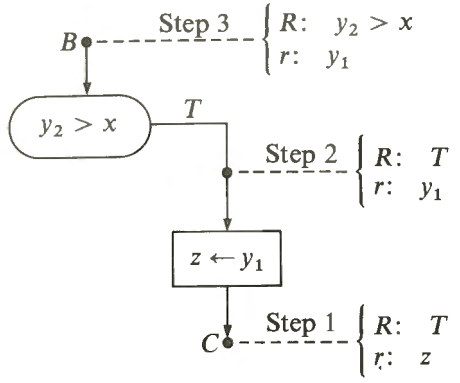
Thus, R_α is T, and r_α is $(0, 1, 1)$.

Path β (from B to B):



Thus, R_β is $y_2 \leqq x$, and r_β is $(y_1 + 1, y_2 + y_3 + 2, y_3 + 2)$.

Path γ (from B to C):



Thus, R_γ is $y_2 > x$, and r_γ is y_1.

Now we are ready to verify the given program. For this purpose we attach the inductive assertion

$$p(x, y_1, y_2, y_3): \quad y_1^2 \leqq x \wedge y_2 = (y_1 + 1)^2 \wedge y_3 = 2y_1 + 1$$

to cutpoint B, in addition to attaching the input predicate $\varphi(x)$: $x \geqq 0$ to cutpoint A and the output predicate $\psi(x, z)$: $z^2 \leqq x < (z + 1)^2$ to cutpoint C (see Fig. 3-5).

The three verification conditions to be proved (for all integers x, y_1, y_2, and y_3) are:

1. For path α

   $$[\varphi(x) \wedge T] \supset p(x, 0, 1, 1)$$

   that is,

   $$x \geqq 0 \supset [0^2 \leqq x \wedge 1 = (0 + 1)^2 \wedge 1 = 2 \cdot 0 + 1]$$

2. For path β

   $$[p(x, y_1, y_2, y_3) \wedge y_2 \leqq x] \supset p(x, y_1 + 1, y_2 + y_3 + 2, y_3 + 2)$$

   that is,

   $$[y_1^2 \leqq x \wedge y_2 = (y_1 + 1)^2 \wedge y_3 = 2y_1 + 1 \wedge y_2 \leqq x]$$

   $$\supset [(y_1 + 1)^2 \leqq x \wedge y_2 + y_3 + 2 = (y_1 + 2)^2$$

   $$\wedge\ y_3 + 2 = 2(y_1 + 1) + 1]$$

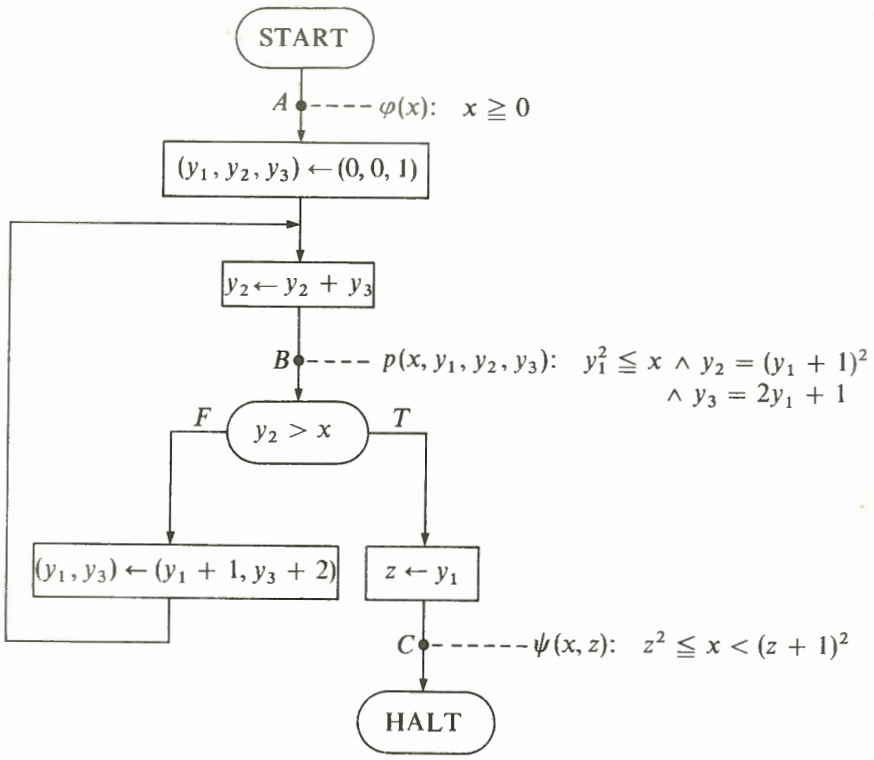


Figure 3-5 Program P_1 for computing $z = \lfloor \sqrt{x} \rfloor$ (partial correctness).

3. For path γ

$$[p(x, y_1, y_2, y_3) \wedge y_2 > x] \supset \psi(x, y_1)$$

that is,

$$[y_1^2 \leqq x \wedge y_2 = (y_1 + 1)^2 \wedge y_3 = 2y_1 + 1 \wedge y_2 > x] \supset y_1^2 \leqq x < (y_1 + 1)^2$$

Since the three verification conditions are *true*, it follows that the given program P_1 is partially correct wrt the input predicate $x \geqq 0$ and the output predicate $z^2 \leqq x < (z + 1)^2$.

□

EXAMPLE 3-2

The program P_2 over the integers (Fig. 3-6) computes $z = x_1^{x_2}$ for any integer x_1 and any natural number x_2 (we define 0^0 as equal to 1). The

computation is based on the binary expansion of x_2, i.e., that for every $y_2 \geqq 0$

$$y_1^{y_2} = y_1 \cdot y_1^{y_2 - 1} \quad \text{if } y_2 \text{ is odd}$$
$$y_1^{y_2} = (y_1 \cdot y_1)^{y_2/2} \quad \text{if } y_2 \text{ is even}$$

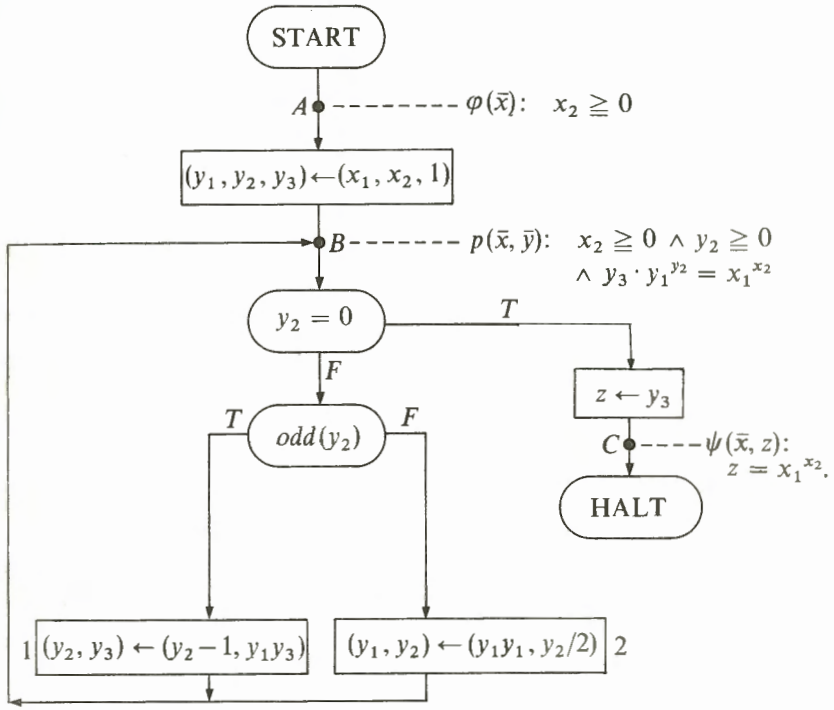


Figure 3-6 Program P_2 for computing $z = x_1^{x_2}$ (partial correctness).

We shall prove that the program is partially correct wrt the input predicate $\varphi(\bar{x})$: $x_2 \geqq 0$ and the output predicate $\psi(\bar{x}, z)$: $z = x_1^{x_2}$. For this purpose we cut both loops of the program at point B and attach to it the inductive assertion

$$p(\bar{x}, \bar{y}): \quad x_2 \geqq 0 \wedge y_2 \geqq 0 \wedge y_3 \cdot y_1^{y_2} = x_1^{x_2}$$

The verification conditions to be proved (for all integers x_1, x_2, y_1, y_2, and y_3) are:

1. For path α (from A to B)

$$\varphi(\bar{x}) \supset p(\bar{x}, x_1, x_2, 1)$$

that is,

$$x_2 \geq 0 \supset [x_2 \geq 0 \wedge x_2 \geq 0 \wedge 1 \cdot x_1^{x_2} = x_1^{x_2}]$$

2. For path β_1 (from B to B via statement 1)

$$[p(\bar{x}, \bar{y}) \wedge y_2 \neq 0 \wedge odd(y_2)] \supset p(\bar{x}, y_1, y_2 - 1, y_1 y_3)$$

that is,

$$[x_2 \geq 0 \wedge y_2 \geq 0 \wedge y_3 \cdot y_1^{y_2} = x_1^{x_2} \wedge y_2 \neq 0 \wedge odd(y_2)]$$
$$\supset [x_2 \geq 0 \wedge y_2 - 1 \geq 0 \wedge (y_1 y_3) \cdot y_1^{y_2 - 1} = x_1^{x_2}]$$

3. For path β_2 (from B to B via statement 2)

$$[p(\bar{x}, \bar{y}) \wedge y_2 \neq 0 \wedge even(y_2)] \supset p(\bar{x}, y_1 y_1, y_2/2, y_3)$$

that is,

$$[x_2 \geq 0 \wedge y_2 \geq 0 \wedge y_3 \cdot y_1^{y_2} = x_1^{x_2} \wedge y_2 \neq 0 \wedge even(y_2)]$$

$$\supset [x_2 \geq 0 \wedge y_2/2 \geq 0 \wedge y_3 \cdot (y_1 y_1)^{y_2/2} = x_1^{x_2}]$$

4. For path γ (from B to C)

$$[p(\bar{x}, \bar{y}) \wedge y_2 = 0] \supset \psi(\bar{x}, y_3)$$

that is,

$$[x_2 \geq 0 \wedge y_2 \geq 0 \wedge y_3 \cdot y_1^{y_2} = x_1^{x_2} \wedge y_2 = 0] \supset y_3 = x_1^{x_2}$$

Since the four verification conditions are *true*, it follows that the program is partially correct wrt the input predicate $x_2 \geq 0$ and the output predicate $z = x_1^{x_2}$.

□

EXAMPLE 3-3

The program P_3 over the integers (Fig. 3-7) computes $z = gcd(x_1, x_2)$ for every pair of positive integers x_1 and x_2; that is, z is the greatest common divisor of x_1 and x_2 [for example, $gcd(14, 21) = 7$, and $gcd(13, 21) = 1$]. The computation method is based on the fact that

If $y_1 > y_2$, then $gcd(y_1, y_2) = gcd(y_1 - y_2, y_2)$
If $y_1 < y_2$, then $gcd(y_1, y_2) = gcd(y_1, y_2 - y_1)$
If $y_1 = y_2$, then $gcd(y_1, y_2) = y_1 = y_2$

We shall prove that the program is partially correct wrt the input predicate $\varphi(\bar{x})$: $x_1 > 0 \wedge x_2 > 0$ and the output predicate $\psi(\bar{x}, z)$: $z = gcd(x_1, x_2)$. For this purpose we cut the two loops of the program at point B and attach to the cutpoint B the assertion

$p(\bar{x}, \bar{y})$: $x_1 > 0 \wedge x_2 > 0 \wedge y_1 > 0 \wedge y_2 > 0 \wedge gcd(y_1, y_2) = gcd(x_1, x_2)$

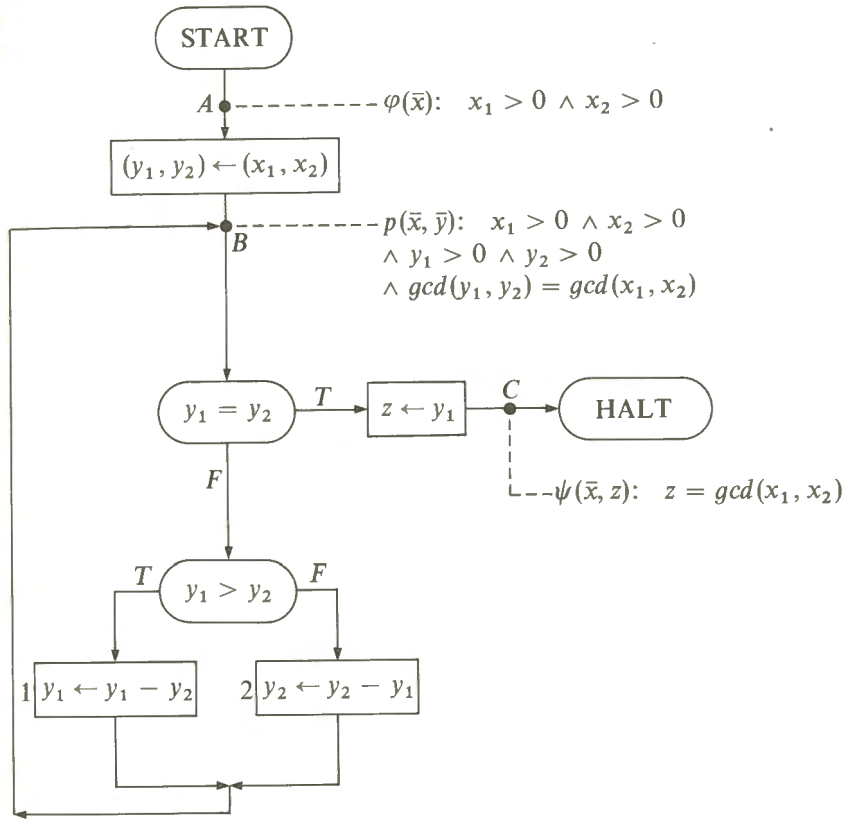


Figure 3-7 Program P_3 for computing $z = gcd(x_1, x_2)$ (partial correctness).

The verification conditions to be proved (for all integers x_1, x_2, y_1, and y_2) are:

1. For path α (from A to B)

$$\varphi(\bar{x}) \supset p(\bar{x}, x_1, x_2)$$

that is,

$$[x_1 > 0 \wedge x_2 > 0] \supset [x_1 > 0 \wedge x_2 > 0$$
$$\wedge x_1 > 0 \wedge x_2 > 0 \wedge gcd(x_1, x_2) = gcd(x_1, x_2)]$$

2. For path β_1 (from B to B via statement 1)

$$[p(\bar{x}, \bar{y}) \wedge y_1 \neq y_2 \wedge y_1 > y_2] \supset p(\bar{x}, y_1 - y_2, y_2)$$

that is,

$$\begin{aligned}&[x_1 > 0 \wedge x_2 > 0 \wedge y_1 > 0 \wedge y_2 > 0\\ &\wedge gcd(y_1, y_2) = gcd(x_1, x_2) \wedge y_1 \neq y_2 \wedge y_1 > y_2]\\ &\supset [x_1 > 0 \wedge x_2 > 0 \wedge y_1 - y_2 > 0 \wedge y_2 > 0\\ &\wedge gcd(y_1 - y_2, y_2) = gcd(x_1, x_2)]\end{aligned}$$

3. For path β_2 (from B to B via statement 2)

$$[p(\bar{x}, \bar{y}) \wedge y_1 \neq y_2 \wedge y_1 \leqq y_2] \supset p(\bar{x}, y_1, y_2 - y_1)$$

that is,

$$\begin{aligned}&[x_1 > 0 \wedge x_2 > 0 \wedge y_1 > 0 \wedge y_2 > 0\\ &\wedge gcd(y_1, y_2) = gcd(x_1, x_2) \wedge y_1 \neq y_2 \wedge y_1 \leqq y_2]\\ &\supset [x_1 > 0 \wedge x_2 > 0 \wedge y_1 > 0 \wedge y_2 - y_1 > 0\\ &\wedge gcd(y_1, y_2 - y_1) = gcd(x_1, x_2)]\end{aligned}$$

4. For path γ (from B to C)

$$[p(\bar{x}, \bar{y}) \wedge y_1 = y_2] \supset \psi(\bar{x}, y_1)$$

that is,

$$\begin{aligned}&[x_1 > 0 \wedge x_2 > 0 \wedge y_1 > 0 \wedge y_2 > 0\\ &\wedge gcd(y_1, y_2) = gcd(x_1, x_2) \wedge y_1 = y_2]\\ &\supset y_1 = gcd(x_1, x_2)\end{aligned}$$

Since the four verification conditions are *true*, it follows that the program is partially correct wrt the input predicate $x_1 > 0 \wedge x_2 > 0$ and the output predicate $z = gcd(x_1, x_2)$.

□

3-1.2 Termination

Next we shall describe a method for proving the termination of flowchart programs. We simply use an ordered set with no infinite decreasing sequences to establish that the program cannot go through a loop indefinitely. The most common ordered set used for this purpose is the set of natural numbers ordered with the usual $>$ (*greater than*) relation. Since, for any

n, we have $n > \cdots > 2 > 1 > 0$, there are no infinite decreasing sequences of natural numbers. Note that we may not use the set of all integers with the same ordering because it has infinite decreasing sequences such as $n > \cdots > 2 > 1 > 0 > -1 > -2 > \cdots$. In general, every well-founded set can be used for the purpose of proving the termination of flowchart programs as we shall demonstrate.

A *partially ordered set* $(W, \prec)$ consists of a nonempty set W and any binary relation $\prec$ on elements of W which satisfies the following properties:

1. For all $a, b, c \in W$, if $a \prec b$ and $b \prec c$, then $a \prec c$ (*transitivity*).
2. For all $a, b \in W$, if $a \prec b$, then $b \not\prec a$ (*asymmetry*).
3. For all $a \in W$, $a \not\prec a$ (*irreflexivity*).

As usual, we write $a \prec b$ or $b \succ a$; $a \not\prec b$ means "a does not precede b." Note that the ordering need not be total, i.e., it is possible that for some $a, b \in W$, neither $a \prec b$ nor $b \prec a$ holds.

A partially ordered set $(W, \prec)$ which contains no infinite decreasing sequences, $a_0 \succ a_1 \succ a_2 \succ \cdots$, of elements of W is called a *well-founded set*. For example:

(a) The set of all real numbers between 0 and 1, with the usual ordering $<$, is partially ordered but not well-founded (consider the infinite decreasing sequence $\frac{1}{2} > \frac{1}{3} > \frac{1}{4} > \cdots$).

(b) The set I of integers, with the usual ordering $<$, is partially ordered but not well-founded (consider the infinite decreasing sequence $0 > -1 > -2 > \cdots$).

(c) The set N of natural numbers, with the usual ordering $<$, is well-founded.

(d) If Σ is any alphabet, then the set Σ^* of all words over Σ with the substring relation $\prec$ (that is, $w_1 \prec w_2$ iff w_1 is a proper substring of w_2), is well-founded.

Now, suppose we are given a flowchart program P and an input predicate φ. To show that P terminates over φ, we propose the following procedure:

1. Choose a well-founded set $(W, \prec)$.
2. Select a set of cutpoints to cut the loops of the program.
3. With every cutpoint i, associate a function $u_i(\bar{x}, \bar{y})$ mapping $D_{\bar{x}} \times D_{\bar{y}}$ into W; that is,

(*) $\quad \forall \bar{x} \forall \bar{y} [u_i(\bar{x}, \bar{y}) \in W]$

If for every path α from cutpoint i to j with no intermediate cutpoints which is a part of some loop (i.e., there is also some path from j to i), we

have

(**) $\forall\bar{x}\forall\bar{y}\{\varphi(\bar{x}) \wedge R_\alpha(\bar{x}, \bar{y}) \supset [u_i(\bar{x}, y) \succ u_j(\bar{x}, r_\alpha(\bar{x}, \bar{y}))]\}$

then P terminates over φ.

In other words, for any computation, when we move from one cutpoint to another, there is associated a smaller and smaller element $u_i(\bar{x}, \bar{y})$ of W. Since W is a well-founded set, there is no infinite decreasing sequence of elements of W, which, in turn, implies that any computation of P must be finite.

The main drawback of the procedure just described is that conditions (*) and (**) are too restrictive, and it is possible only rarely to find an appropriate set of functions $u_i(\bar{x}, \bar{y})$ that will satisfy both conditions. The problem is that we require that the conditions will be *true* for all $\bar{x}$ and $\bar{y}$, while, in general, it suffices to show (*) and (**) only for those values of $\bar{x}$ and $\bar{y}$ which can indeed be reached at point i at some stage of the computation. This problem suggests attaching an inductive assertion $q_i(\bar{x}, \bar{y})$ to each cutpoint i in such a way that $q_i(\bar{x}, \bar{y})$ has the property that whenever the computation of the program reaches cutpoint i, $q_i(\bar{x}, \bar{y})$ is *true* for the current values of $\bar{x}$ and $\bar{y}$. Then we can state a more powerful method for proving termination, as follows.

Step 1. Select a set of cutpoints to cut the loops of the program and with every cutpoint i associate an assertion $q_i(\bar{x}, \bar{y})$ such that $q_i(\bar{x}, \bar{y})$ *are good assertions.* That is, for every path α from the START point to cutpoint j (with no intermediate cutpoints), we have

$$\forall\bar{x}[\varphi(\bar{x}) \wedge R_\alpha(\bar{x}) \supset q_j(\bar{x}, r_\alpha(\bar{x}))]$$

and for every path α from cutpoint i to j (with no intermediate cutpoints), we have

$$\forall\bar{x}\forall\bar{y}[q_i(\bar{x}, \bar{y}) \wedge R_\alpha(\bar{x}, \bar{y}) \supset q_j(\bar{x}, r_\alpha(\bar{x}, \bar{y}))]$$

Step 2. Choose a well-founded set $(W, \prec)$ and with every cutpoint i associate a partial function $u_i(\bar{x}, \bar{y})$ mapping $D_{\bar{x}} \times D_{\bar{y}}$ into W such that $u_i(\bar{x}, \bar{y})$ *are good functions.* That is, for every cutpoint i, we have

$$\forall\bar{x}\forall\bar{y}[q_i(\bar{x}, \bar{y}) \supset u_i(\bar{x}, \bar{y}) \in W]$$

Step 3. Show that *the termination conditions hold.* That is, for every path α from cutpoint i to j (with no intermediate cutpoints) which is a part of some loop, we have

$$\forall\bar{x}\forall\bar{y}\{q_i(\bar{x}, \bar{y}) \wedge R_\alpha(\bar{x}, \bar{y}) \supset [u_i(\bar{x}, \bar{y}) \succ u_j(\bar{x}, r_\alpha(\bar{x}, \bar{y}))]\}$$

The method just described is summarized in the following theorem.

THEOREM 3-2 [WELL-FOUNDED-SETS METHOD (Floyd)]. *For a given flowchart program P and an input predicate $\varphi(\bar{x})$, apply the following steps: (1) Cut the loops and find "good" inductive assertions; and (2) choose a well-founded set and find "good" partial functions. If all the termination conditions are true, then P terminates over φ.*

EXAMPLE 3-4

First we prove that the program P_1 of Example 3-1 terminates for every natural number x. To show that the program terminates over $\varphi(x)$: $x \geqq 0$, we use the well-founded set $(N, <)$, that is, the set of natural numbers with the usual ordering $<$. We cut the single loop at point B and attach to it the assertion $q(x, \bar{y})$: $y_2 \leqq x \wedge y_3 > 0$ and the function $u(x, \bar{y})$: $x - y_2$ (see Fig. 3-8).

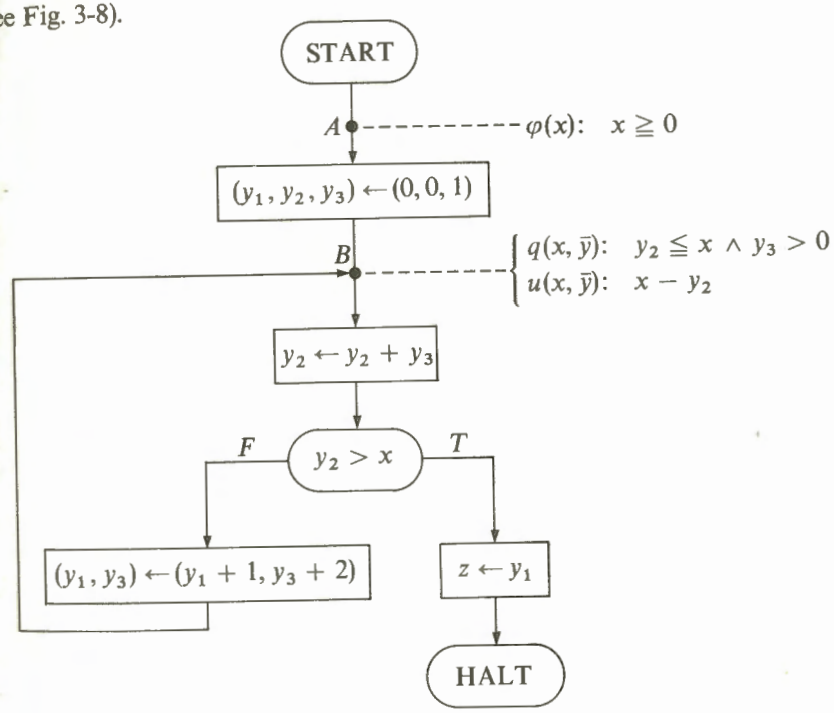


Figure 3-8 Program P_1 for computing $z = [\sqrt{x}]$ (termination).

Note that there are two paths of interest here:

Path α_1 (from A to B): R_{α_1} is T, and r_{α_1} is $(0, 0, 1)$.

Path α_2 (from B to B): R_{α_2} is $y_2 + y_3 \leqq x$, and r_{α_2} is $(y_1 + 1, y_2 + y_3, y_3 + 2)$.

Our proof consists of three steps. For all integers x, y_1, y_2, and y_3:

Step 1. *$q(x, \bar{y})$ is a good assertion.*
For path α_1

$$\varphi(x) \supset q(x, 0, 0, 1)$$

that is,

$$x \geqq 0 \supset [0 \leqq x \wedge 1 > 0]$$

For path α_2

$$[q(x, y_1, y_2, y_3) \wedge y_2 + y_3 \leqq x] \supset q(x, y_1 + 1, y_2 + y_3, y_3 + 2)$$

that is,

$$[y_2 \leqq x \wedge y_3 > 0 \wedge y_2 + y_3 \leqq x] \supset [y_2 + y_3 \leqq x \wedge y_3 + 2 > 0]$$

Step 2. *$u(x, \bar{y})$ is a good function.*

$$q(x, \bar{y}) \supset u(x, \bar{y}) \in N$$

that is,

$$[y_2 \leqq x \wedge y_3 > 0] \supset x - y_2 \geqq 0$$

Step 3. *The termination condition holds.*
For path α_2

$$[q(x, \bar{y}) \wedge y_2 + y_3 \leqq x] \supset [u(x, y_1, y_2, y_3) > u(x, y_1 + 1, y_2 + y_3, y_3 + 2)]$$

that is,

$$[y_2 \leqq x \wedge y_3 > 0 \wedge y_2 + y_3 \leqq x] \supset [x - y_2 > x - (y_2 + y_3)]$$

(Note that path α_1 is not considered because it is not part of any loop.)

Since all three conditions are *true*, it follows that the program terminates for every natural number x.

□

It is straightforward to prove that the program P_2 of Example 3-2 terminates over $\varphi(\bar{x})$: $x_2 \geqq 0$ [let $q_B(\bar{x}, \bar{y})$ be $y_2 \geqq 0$ and $u_B(\bar{x}, \bar{y})$ be y_2] and that the program P_3 of Example 3-3 terminates over $\varphi(\bar{x})$: $x_1 > 0 \wedge x_2 > 0$ [let $q_B(\bar{x}, \bar{y})$ be $y_1 > 0 \wedge y_2 > 0$ and $u_B(\bar{x}, \bar{y})$ be $max(y_1, y_2)$]. We proceed with a less trivial example.

EXAMPLE 3-5 (Knuth)
The program P_4 over the integers (Fig. 3-9) also computes $z = gcd(x_1, x_2)$ for every pair of positive integers x_1 and x_2; that is, z is the greatest common divisor of x_1 and x_2. We leave it as an exercise for the reader to prove that the program is partially correct wrt the input predicate $\varphi(\bar{x})$: $x_1 > 0 \wedge x_2 > 0$ and the output predicate $\psi(\bar{x}, z)$: $z = gcd(x_1, x_2)$.† We would like to prove that P_4 terminates over φ.

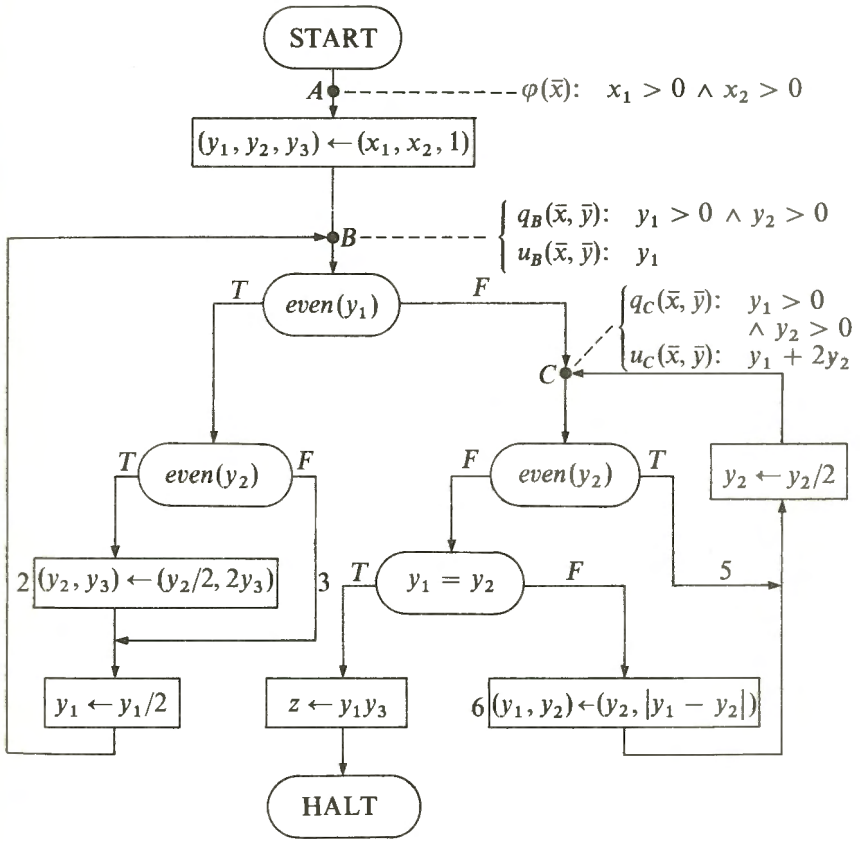


Figure 3-9 Program P_4 for computing $z = gcd(x_1, x_2)$ (termination).

Again we use the well-founded set $(N, <)$. We cut the loops of the program at points B and C and then attach to cutpoint B

$$q_B(\bar{x}, \bar{y})\text{: } y_1 > 0 \wedge y_2 > 0 \quad \text{and} \quad u_B(\bar{x}, \bar{y})\text{: } y_1$$

† Let $p_B(\bar{x}, \bar{y})$ be $x_1 > 0 \wedge x_2 > 0 \wedge y_1 > 0 \wedge y_2 > 0 \wedge y_3 \cdot gcd(y_1, y_2) = gcd(x_1, x_2)$ and let $p_C(\bar{x}, \bar{y})$ be $x_1 > 0 \wedge x_2 > 0 \wedge y_1 > 0 \wedge y_2 > 0 \wedge odd(y_1) \wedge y_3 \cdot gcd(y_1, y_2) = gcd(x_1, x_2)$.

and to cutpoint C

$$q_C(\bar{x}, \bar{y}): \quad y_1 > 0 \wedge y_2 > 0 \qquad \text{and} \qquad u_C(\bar{x}, \bar{y}): \quad y_1 + 2y_2$$

Note that there are six paths of interest in this program:

α_1 (from A to B): R_{α_1} is T, and r_{α_1} is $(x_1, x_2, 1)$.
α_2 (from B to B via statement 2): R_{α_2} is $even(y_1) \wedge even(y_2)$, and r_{α_2} is $(y_1/2, y_2/2, 2y_3)$.
α_3 (from B to B via arc 3): R_{α_3} is $even(y_1) \wedge odd(y_2)$, and r_{α_3} is $(y_1/2, y_2, y_3)$.
α_4 (from B to C): R_{α_4} is $odd(y_1)$, and r_{α_4} is (y_1, y_2, y_3).
α_5 (from C to C via arc 5): R_{α_5} is $even(y_2)$, and r_{α_5} is $(y_1, y_2/2, y_3)$.
α_6 (from C to C via statement 6): R_{α_6} is $odd(y_2) \wedge y_1 \neq y_2$, and r_{α_6} is $(y_2, |y_1 - y_2|/2, y_3)$.

Again our proof consists of three steps. For all integers x, y_1, y_2, and y_3:

Step 1. *q_B and q_C are good assertions.*
For path α_1

$$\varphi(\bar{x}) \supset q_B(\bar{x}, x_1, x_2, 1)$$

For path α_2

$$q_B(\bar{x}, \bar{y}) \wedge even(y_1) \wedge even(y_2) \supset q_B(\bar{x}, y_1/2, y_2/2, 2y_3)$$

For path α_3

$$q_B(\bar{x}, \bar{y}) \wedge even(y_1) \wedge odd(y_2) \supset q_B(\bar{x}, y_1/2, y_2, y_3)$$

For path α_4

$$q_B(\bar{x}, \bar{y}) \wedge odd(y_1) \supset q_C(\bar{x}, y_1, y_2, y_3)$$

For path α_5

$$q_C(\bar{x}, \bar{y}) \wedge even(y_2) \supset q_C(\bar{x}, y_1, y_2/2, y_3)$$

For path α_6

$$q_C(\bar{x}, \bar{y}) \wedge odd(y_2) \wedge y_1 \neq y_2 \supset q_C(\bar{x}, y_2, |y_1 - y_2|/2, y_3)$$

Step 2. *u_B and u_C are good functions.*

$$q_B(\bar{x}, \bar{y}) \supset u_B(\bar{x}, \bar{y}) \in N$$

$$q_C(\bar{x}, \bar{y}) \supset u_C(\bar{x}, \bar{y}) \in N$$

Step 3. *The termination conditions hold.*
For path α_2

$$[q_B(\bar{x}, \bar{y}) \wedge even(y_1) \wedge even(y_2)]$$
$$\supset [u_B(\bar{x}, y_1, y_2, y_3) > u_B(\bar{x}, y_1/2, y_2/2, 2y_3)]$$

For path α_3

$$[q_B(\bar{x}, \bar{y}) \wedge even(y_1) \wedge odd(y_2)]$$
$$\supset [u_B(\bar{x}, y_1, y_2, y_3) > u_B(\bar{x}, y_1/2, y_2, y_3)]$$

For path α_5

$$[q_C(\bar{x}, \bar{y}) \wedge even(y_2)] \supset [u_C(\bar{x}, y_1, y_2, y_3) > u_C(\bar{x}, y_1, y_2/2, y_3)]$$

For path α_6

$$[q_C(\bar{x}, \bar{y}) \wedge odd(y_2) \wedge y_1 \neq y_2]$$
$$\supset [u_C(\bar{x}, y_1, y_2, y_3) > u_C(\bar{x}, y_2, |y_1 - y_2|/2, y_3)]$$

Substituting the given assertions q_B and q_C and the given functions u_B and u_C, all the above statements can easily be verified; therefore it follows that the program terminates for every pair of positive integers x_1 and x_2.

□

3-2 FLOWCHART PROGRAMS WITH ARRAYS

An *array* is a programming feature used for describing a large family of related program variables. For example, to describe a group of 21 integer variables, instead of using letters $A, B, C, \ldots, T, U$, we prefer to identify the variables as $S[0], S[1], \ldots, S[20]$, where S is a 21-element array; this notation corresponds to the mathematical subscript notation S_0, $S_1, \ldots, S_{20}$. An expression like $S[i + j]$ indicates the $(i + j)$th element $(0 \leqq i + j \leqq 20)$ in this family, depending on the current value of $i + j$.

3-2.1 Partial Correctness

In this section we shall prove the partial correctness of several flowchart programs which use arrays and discuss some of the difficulties involved in treating arrays. However, in our examples we shall ignore one problem concerning the correctness of flowchart programs with arrays: We shall not verify that the array subscript actually lies within the boundaries of the array. For instance, suppose that a program uses an array S of 21