A calculus for proving properties of while-programs

Ingrid Glasner

Jacques Loeckx

Fachbereich 10 der Universität
des Saarlandes
D-6600 Saarbrücken
W-Germany

June 1978

A 78/09

1. Introduction

Most commonly used methods for proving program properties - such as the inductive assertion method or the well-founded sets method - are only partially formalized. On the other hand, methods allowing completely formalized proofs - such as those proposed by Hoare [4], Manna and Pnueli [7] or Milner [8] - generally lead to lengthy calculations and are wearisome when performed by hand. The goal of the present paper is to propose a calculus which allows formal proofs of properties of while-programs according to the inductive assertion method, the subgoal induction method and the well-founded sets method; while being completely formal the proofs remain understandable and may easily be performed by hand.

The method to be described bears strong similarities with LUCID[1]. As a main difference the authors of LUCID propose a new programming language while the present paper refers to while-programs.

2. While-programs

2.1 Definitions

Informally, a *while-program* (see e.g. [6], p. 2o3) consists of a sequence of statements, each statement being either an assignment or a while-statement.

A while-program is called *elementary* when all while-statements are nested. Syntactically such a while-program is defined by the non-terminal symbol E together with the context-free productions

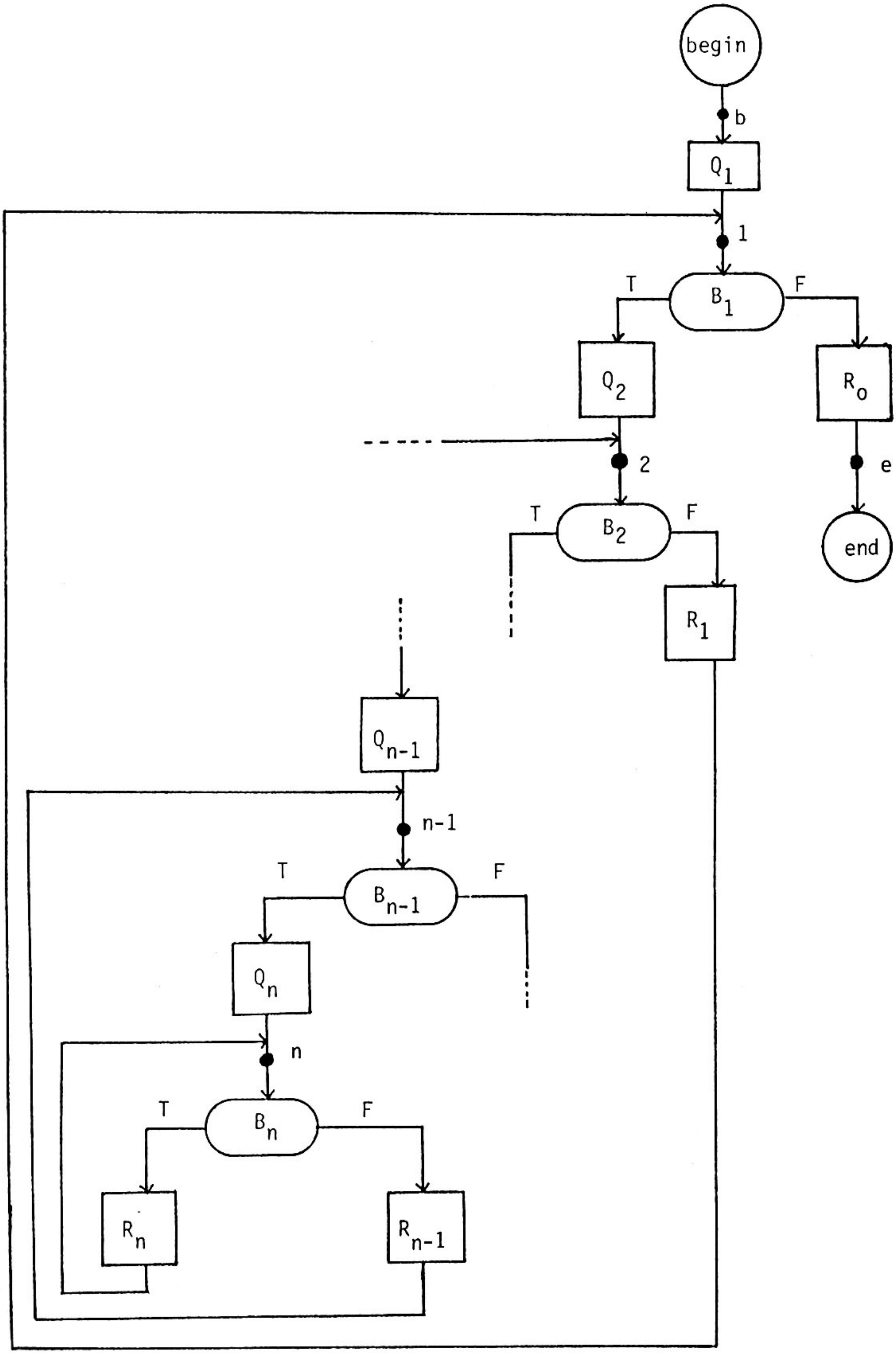


Figure 1: The flowchart of an elementary while-program with nesting depth n, $n \geq 0$.

E ::= begin P end

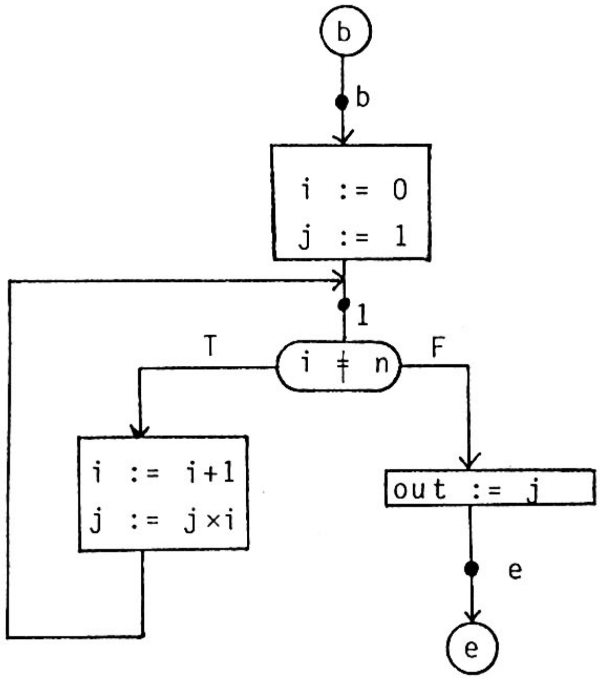
P ::= Q; while B do P od; Q | Q

Q ::= Q; A | ε

where A stands for an assignment, B a boolean expression and ε the empty string. An elementary while-program with nesting depth n is represented by the flowchart of Figure 1; in this flowchart $Q_1, Q_2, \ldots, Q_n$, $R_0, R_1, \ldots, R_n$ are elements of the syntactical class Q and $B_1, B_2, \ldots, B_n$ elements of the syntactical class B.

A while-program is called *normalized* when the following three conditions are satisfied. First, each variable occurs at most twice in the lefthand side of an assignment; next, in the case of two such occurrences one must be in a block Q_i and the other in the block R_i $(1 \leq i \leq n)$; finally, in the case of one such occurrence this occurrence must be in a block R_i $(0 \leq i \leq n)$. Examples of normalized while-programs are in Figure 2 and in the Appendix.

In the sequel only elementary normalized while-programs will be considered. This restriction is not essential as results from the following two arguments. First, each elementary while-program is easily transformed into a normalized one at the cost of a few supplementary variables; an algorithm performing this transformation is described in [5]. Second, the results of the present paper may easily be generalized for (non-elementary) while-programs.



$\vdash$ b1 : $i = o$ (A1)

$\vdash$ b1 : $j = 1$ (A2)

$\vdash$ 11 : $i' \neq n$ (A3)

$\vdash$ 1e : $i = n$ (A4)

$\vdash$ 11 : $i = i' + 1$ (A5)

$\vdash$ 11 : $j = j' \times i$ (A6)

$\vdash$ 1e : $out = j$ (A7)

Figure 2: A while-program and its semantics

The *rank* of a variable is defined as the index i of the block Q_i and/or R_i in which it occurs as the lefthand side of an assignment ($0 \leq i \leq n$). In Figure 2, for instance, the rank of j is 1 and that of out is 0.

2.2 An operational semantics

Consider the flowchart of a while-program with nesting depth n and introduce the n+2 *cutpoints* b,e,1,2,...,n as indicated by Figure 1.

It is then easy to define an operational semantics of this while-program. To this end one may introduce configurations of the form

$$(i,\bar{z})$$

where i is a cutpoint and $\bar{z}$ the vector constituted by (the values of) the different program variables.

A *computation* is defined as a sequence of configurations

$$(i_1,\bar{z}_1) \Rightarrow (i_2,\bar{z}_2) \Rightarrow \ldots \Rightarrow (i_m,\bar{z}_m) \qquad (m \geq 2)$$

with

$$(i_j,\bar{z}_j) \Rightarrow (i_{j+1},\bar{z}_{j+1}) \qquad (1 \leq j \leq m-1)$$

meaning that the flow of control passed from cutpoint i_j to cutpoint i_{j+1} (without passing a cutpoint in between).

Of course one is interested essentially in the computations with $i_1 = b$ and $i_m = e$.

For a more detailed description of the operational semantics the reader is referred to [3].

3. The calculus

3.1 Definition

The calculus is defined as an extension of the first-order predicate calculus.

Let n be an integer, $n \geq 0$.

In addition to the vocabularies required in the predicate calculus a vocabulary of *program variables* is introduced. With each program variable x is associated an integer called *rank* and noted rank(x), with $0 \leq \text{rank}(x) \leq n$.

If x is a program variable then x, x' and x" are called *instances* of this program variable; the *rank* of an instance is that of its program variable.

A *sentence* is either a sentence of the first-order predicate calculus or it has one of the following four forms:

$(i-1)i : q$	with $1 \leq i \leq n$	(1)
$ii : q$	with $0 \leq i \leq n$	(2)
$i(i-1) : q$	with $1 \leq i \leq n$	(3)
$i : q$	with $1 \leq i \leq n$	(4)

where q is the expression obtained from a sentence of the first-order predicate calculus by replacing a certain number - possibly zero - of free variables by instances of program variables with a rank not superior to i; in other words, q is a sentence of the predicate calculus being understood that the instances of program variables x with $\text{rank}(x) \leq i$ may be used in the place of free variables. As a notational convention intended to facilitate the description of the interpretation of the calculus we write

b1 : q	instead of	01 : q
be : q	instead of	00 : q
1e : q	instead of	10 : q

Examples of sentences are for instance in Figure 2.

3.2 The intended interpretation

The interpretation of a sentence of the predicate calculus is the classical one.

The interpretation of another sentence is a property of a while-program with nesting depth n. Roughly speaking, a sentence such as

$$ij : q$$

expresses a property of computations starting in cutpoint i and ending in cutpoint j; the instances x, resp. x', of a program variable are interpreted as the value of this program variable in the last, resp. the first, configuration of this computation; the instance x" is interpreted as the value of the program variable at a moment which is not further specified (*). The interpretation of these sentences will now be considered more carefully.

A sentence

$$(i-1)i : q$$

expresses that q holds for all computations

$$(i-1, \bar{z}_1) \Rightarrow (i, \bar{z}_2)$$

For instance

$$b1 : j = 1$$

(*) such instances stand for dummies and will be of use in the subgoal induction method only.

(see Figure 2) expresses that the value of the program variable j (contained in the vector $\bar{z}_2$) is 1 whenever reaching cutpoint 1 from cutpoint b.

A sentence

$$ii : q$$

expresses that q holds for all computations

$$(i, \bar{z}_1) \Rightarrow \ldots \Rightarrow (i, \bar{z}_m) \qquad (m \geq 2)$$

where ... stands for configurations with cutpoints > i.

For instance

$$11 : i = i' + 1$$

expresses that in a loop leading from cutpoint 1 to cutpoint 1 (possibly through some inner cutpoints) the value of i is increased by 1. By the way, a sentence such as

$$ii : x \neq x'$$

implies rank (x) = i because the while-programs considered are normalized.

A sentence

$$i(i-1) : q$$

expresses that q holds for all computations

$$(i, \bar{z}_1) \Rightarrow \ldots \Rightarrow (i-1, \bar{z}_m) \qquad (m \geq 2)$$

where ... stands for configurations with cutpoints $\geq$ i.

A sentence

$$i : q$$

expresses that q holds for all computations

$$(i - 1, \bar{z}_1) \Rightarrow \ldots \Rightarrow (i, \bar{z}_m) \qquad (m \geq 2)$$

where ... stands for configurations with cutpoints $\geq$ i. For more precision and details the reader is referred to [3].

3.3 Applying the calculus for proving program properties

In section 4 it will be shown how the semantics of a while-program may be expressed as a set of axioms of the form

$$\vdash (i-1)i : q$$

$$\vdash ii : q$$

or

$$\vdash i(i-1) : q$$

In sections 5 to 7 it will be shown how some methods for proving program properties may be implemented by a few rules of inference.

As the calculus is an extension of the first-order predicate calculus all of its axioms, inference rules and theorems hold. Moreover, if

$$\frac{\vdash A_1 \quad \vdash A_2 \quad \ldots \quad \vdash A_m}{\vdash B} \qquad (m \geq 0)$$

is an inference rule of the first-order predicate calculus then

$$\frac{\vdash \alpha : A_1^* \quad \vdash \alpha : A_2^* \quad \dots \quad \vdash \alpha : A_m^*}{\vdash \alpha : B^*}$$

is also an inference rule; in this rule $A_1^*,\dots,B^*$ are obtained from $A_1,\dots,B$ by consistently replacing a certain number (possibly zero) of free variables by instances of program variables such that $\vdash \alpha : A_1^*, \dots, \vdash \alpha : B$ are sentences of the form (1) to (4) of section 3.1 .

The consistency of these inference rules with the intended interpretation is intuitively clear; see [3] for a proof.

The following notation will be used in the sequel. If w is a substring of a sentence containing no primed instances of program variables of rank i, then

$$w'_{(i)}$$

is the string obtained from w by replacing each instance of rank i, say x, by x'. The notation

$$w''(i)$$

is defined similarly.

4. The semantics of a while-program

Consider the while-program of Figure 1. Its semantics is expressed by the following (2n+s) axioms, s being the number of assignments.

To the predicate p of a block B_i correspond two axioms:

$$\vdash i(i+1) : p \qquad \text{if } i < n$$

or $\vdash nn : p'_{(n)}$ if $i = n$

and $\vdash i(i-1) : \neg p'_{(i-1)}$

Intuitively these axioms express that p holds when leaving B_i through the T-exit and does not hold when leaving B_i through the F-exit; the introduction of the primes in the case i = n is necessary because the program variables of rank n are updated (in block R_n) on the path leading from cutpoint n to cutpoint n.

To an assignment of block Q_i such as

$$x := f(u,v)$$

with rank (u) < i and rank (v) = i corresponds the axiom.

$$\vdash (i-1)i : x = f(u,v) \ .$$

To an assignment of block R_i, $i < n$, such as

$$x := f(u,v,x,y,z)$$

with

$$\text{rank }(u) < i$$

rank (v) = i and the assignment to v in the block R_i precedes

$$\text{rank }(x) = i$$

rank (y) = i and the assignment to y in the block R_i follows

$$\text{rank }(z) = i+1$$

corresponds the axiom

$$\vdash (i+1)i : x = f(u,v,x',y',z)$$

An assignment of block R_n leads to a similar axiom but with (i+1)i replaced by nn.

An example is in Figure 2; more elaborate examples are in the Appendix.

The consistency of these axioms with the model of Section 3.2 is proved in [3]; note that this proof heavily draws upon the fact that the while-program is normalized.

5. The inductive assertions method

The inductive assertion method is implemented by two inference rules :

$$\frac{\vdash (i-1)i : q \qquad \vdash ii : q'_{(i)} \supset q}{\vdash i : q} \qquad \text{(I1)}$$

($1 \leq i \leq n$, q contains no primed instances of rank i)

$$\frac{\vdash i : r \quad \vdash i(i-1) : r'_{(i-1)} \supset q}{\vdash (i-1)(i-1) : q} \qquad (I2)$$

($1 \leq i \leq n$, r contains no primed instances of rank i-1 or rank i)

Intuitively the rule (I1) inductively proves that $\vdash i : q$, i.e. that q is an invariant of cutpoint i; the rule (I2) deduces from the invariant of cutpoint i and from the properties of path i(i-1) a property of the loop (i-1)(i-1).

The consistency of the inference rules with the model of Section 3.2 is proved in [3]. This proof is based on the fact that the while-program is normalized and that according to the definition of a sentence e.g. q of rule (I1) may only contain instances of rank $\leq$ i.

A simple example is the proof of the partial correctness of the program of Figure 2, i.e. the proof of

$$\vdash be : out = n! \qquad (a)$$

We first prove j = i! to be an invariant, i.e.

$$\vdash 1 : j = i! \qquad (b)$$

According to rule (I1) it is sufficient to prove

$$\vdash b1 : j = i! \qquad (b1)$$

and $\vdash 11 : j' = i'! \supset j = i!$ (b2) (b1) directly follows from the axioms (A1) and (A2) of Figure 2; (b2) follows from the axioms (A5) and (A6) because

$$\vdash 11 : j' = i'! \supset j' \times (i' + 1) = (i' + 1)!$$

We now prove

$$\vdash 1e : j = i! \supset out = n! \qquad (c)$$

This directly follows from (A4) and (A7).

(a) directly follows from (b) and (c) by the inference rule (I2) with j = i! for r.

A less trivial example is in Appendix I.

6. The subgoal induction method

The subgoal induction method [9] is also implemented by two inference rules

$$\frac{\vdash i(i-1) : q \qquad \vdash ii : q''_{(i-1)} \supset (q'_{(i)})''_{(i-1)}}{\vdash i(i-1) : q'_{(i)}} \qquad (S1)$$

($1 \leq i \leq n$, q contains no primed instances of rank i or i-1)

$$\frac{\vdash i(i-1) : (r'_{(i)})'_{(i-1)} \supset q \quad \vdash (i-1)i : r}{\vdash (i-1)(i-1) : q} \qquad (S2)$$

($1 \le i \le n$, r contains no primed instances of rank i or i-1)

Intuitively (S1) inductively proves (by "backward" induction) that the loop ii defines a function with property $q'_{(i)}$; (S2) deduces a property of the loop (i-1)(i-1).

The consistency of these rules is proved in [3].

A simple example is the proof of the partial correctness of the program of Figure 2. Again

$$\vdash be : out = n! \qquad (a)$$

is to be proved.

First we prove the subgoal

$$\vdash le : out = j' \times \frac{n!}{i'!} \qquad (b)$$

According to (S1) it is sufficient to prove

$$\vdash le : out = j \times \frac{n!}{i!} \qquad (b1)$$

and

$$\vdash \text{ll: } out'' = j \times \frac{n!}{i!} \supset out'' = j' \times \frac{n!}{i'!} \qquad \text{(b2)}$$

(b1) directly results from (A4) and (A7) of Figure 2.

(b2) directly results from (A5) and (A6) because

$$\vdash \text{ll: } j \times \frac{n!}{i!} = j' \times (i'+1) \times \frac{n!}{(i'+1)!} = j' \times \frac{n!}{i'!}$$

Because of (A1) and (A2)

$$\vdash \text{bl: } i = 0 \wedge j = 1$$

Consider rule (S2) with $i = 0 \wedge j = 1$ for r; for proving (a) it suffices to prove

$$\vdash \text{le: } (i' = 0 \wedge j' = 1) \supset out = n!$$

This is trivially true because of (b).

7. The well-founded sets method

Expressing termination requires the introduction of a supplementary symbol T. The set of sentences is augmented as follows: if

$$i : q$$

with $1 \leq i \leq n$ is a sentence containing <u>no instances of rank i</u> then

$$i : q_T$$

with q_T being obtained from q by the replacement of some free variables by T is also a sentence.

The interpretation of the sentence

$$i : q_T$$

is as usual but with the following supplementary rule: in a computation

$$(i-1,\bar{z}_1) \Rightarrow \ldots \Rightarrow (i,\bar{z}_m) \qquad (*)$$

where ... stands for configurations with cutpoints $\geq$ i, the value of T is <u>true</u> if and only if the computation - when pursued - eventually leads back to cutpoint i-1. Less formally, in $i:q_T$ T expresses that the $i^{\underline{th}}$ loop terminates. For more precision the reader is referred to [3].

Proving that a program terminates for input variables (**) satisfying the property q consists in proving

$$\vdash 1 : q \supset T$$

The well-founded sets method is then implemented by a single rule of inference

(*) cf the interpretation of i:q in Section 3.2

(**) an input variable is a variable not occurring in the lefthand side of an assignment; it behaves as a program variable of rank 0.

$$\frac{\vdash i{:}q\supset t>0 \quad \vdash ii{:}q'_{(i)}\supset t'_{(i)}>t \quad \vdash i{+}1{:}s\supset T \quad \vdash i{:}r\supset q\wedge s}{i : r \supset T}$$

($1 \leq i \leq n-1$, q,t and s contain no primed instances of rank i, t has an integer value)

For i = n the inference rule is the same except that the third premise is lacking and that s is taken to be <u>true</u>.

A trivial example is the proof of termination of the program of Figure 2 under the assumption $n \geq 0$. We have to prove

$$\vdash 1 : n \geq 0 \supset T \qquad (a)$$

Applying the inference rule with n-i+1 for t and $i \leq n$ for q we have to prove

$$\vdash 1 : i \leq n \supset n-i+1 > 0 \qquad (a1)$$

$$\vdash 11 : i' \leq n \supset n-i'+1 > n-i+1 \qquad (a2)$$

and $$\vdash 1 : n \geq 0 \supset i \leq n \qquad (a3)$$

(a1) trivially holds; (a2) holds by (A5) of Figure 2; for proving (a3) we apply the inference rule (I1) and prove

$$\vdash b1 : n \geq 0 \supset i \geq n \qquad (a3\text{-}1)$$

$$\vdash 11 : (n \geq 0 \supset i' \leq n) \supset (n \geq 0 \supset i \leq n) \qquad (a3\text{-}2)$$

(a3-1) holds by (A1); (a3-2) holds by (A3) and (A5).

8. Concluding remark

The calculus has been applied to three proof methods: the inductive assertion method, the subgoal induction method and the well-founded sets method. The calculus may in principle also be applied to other methods or used for proving other properties. Non-termination, for instance, is expressed by

$$\vdash \text{le} : \underline{\text{false}}$$

Note also that different proof methods may be combined. In Appendix I, for instance, the lemma

$$\vdash \text{ll} : \text{r } \underline{\text{mod}} \text{ d} = \text{r' } \underline{\text{mod}} \text{ d}$$

may be proved by the inductive assertions method and the theorem

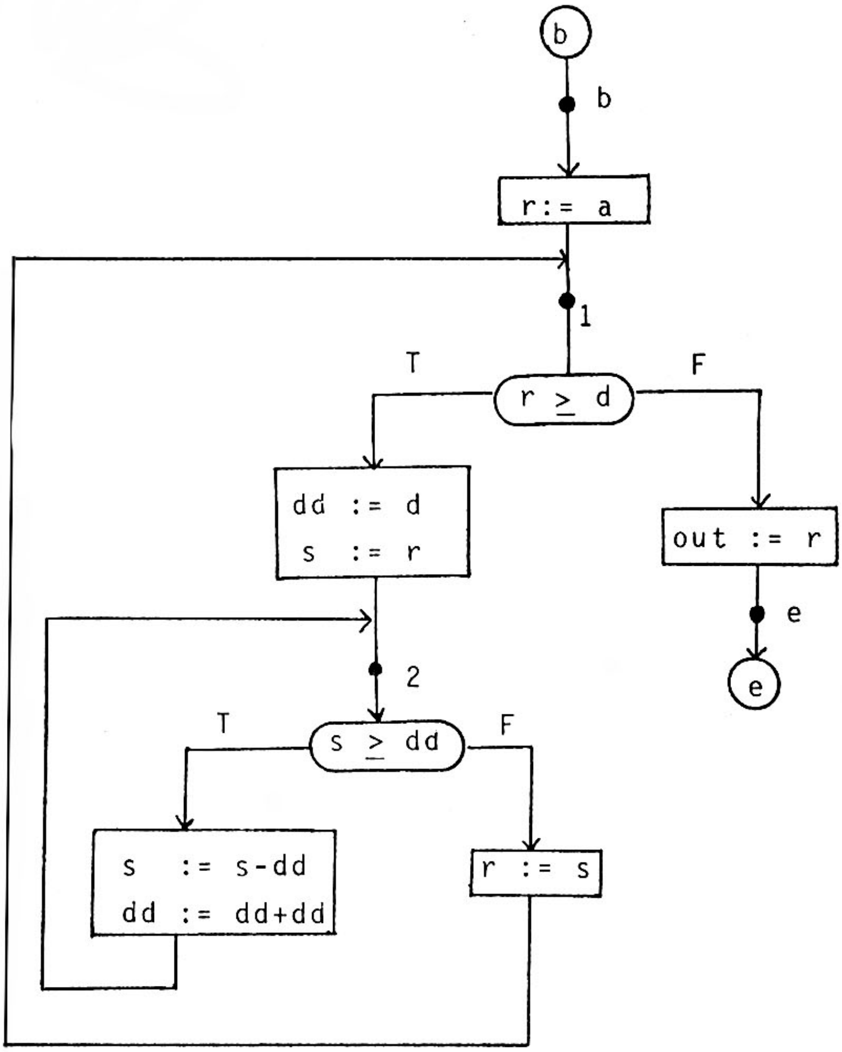
$$\vdash \text{be} : \text{out} = \text{a } \underline{\text{mod}} \text{ d}$$

by subgoal induction.

Appendix I: Illustration of the inductive assertion method and the subgoal induction method.

A.1. The program and its semantics

The program computes a mod d (see [2], p. 59)



⊢ b1:r=a	(A1)
⊢ le:r<d	(A2)
⊢ le:out=r	(A3)
⊢ 12:r≥d	(A4)
⊢ 12:dd=d	(A5)
⊢ 12:s=r	(A6)
⊢ 21:s<dd	(A7)
⊢ 21:r=s	(A8)
⊢ 22:s'≥dd'	(A9)
⊢ 22:s=s'-dd'	(A10)
⊢ 22:dd=dd'+dd'	(A11)

The partial correctness of this program will now be proved successively by the inductive assertion and the subgoal induction method.

A.2. The inductive assertion method

A.2.1 Lemma (invariant in cutpoint 2):

$\vdash$ 2: (s mod d = r mod d) $\wedge$ (dd mod d = 0)

Proof

According to rule (I1) it suffices to prove:

$\vdash$ 12: (s mod d = r mod d) $\wedge$ (dd mod d = 0) (a)

and

$\vdash$ 22: (s'mod d = r mod d) $\wedge$ (dd'mod d = 0)

$\supset$ (s mod d = r mod d) $\wedge$ (dd mod d = 0) (b)

(a) holds by (A6) and (A5).

(b) holds by (A10) and (A11) and by the properties of mod.

A.2.2 Lemma:

$\vdash$ 11: r'mod d = r mod d

Proof

According to rule (I2) it is sufficient to prove

$\vdash$ 2: s mod d = r mod d

and

$\vdash$ 21: s mod d = r'mod d $\supset$ r'mod d = r mod d (a)

(a) holds by the previous lemma.

(b) holds by (A8).

A.2.3 Lemma (invariant in cutpoint 1):

$\vdash$ 1 : r mod d = a mod d

Proof

Applying (I1):

⊢ b1: r <u>mod</u> d = a <u>mod</u> d (a)

⊢ 11: r' <u>mod</u> d = a <u>mod</u> d

⊃ r <u>mod</u> d = a <u>mod</u> d (b)

(a) holds by (A1).

(b) holds by the previous lemma.

A.2.4 Theorem (partial correctness):

⊢ be: out = a <u>mod</u> d

Proof

Applying (I2):

⊢ 1: r <u>mod</u> d = a <u>mod</u> d (a)

⊢ 1e: r <u>mod</u> d = a <u>mod</u> d ⊃ out = a <u>mod</u> d (b)

(a) holds by the previous lemma.

(b) holds by (A3),(A2) and a property of <u>mod</u>.

A.3 The subgoal induction method

A.3.1 Lemma (subgoal of loop 2):

⊢ 21: r <u>mod</u> dd' = s'<u>mod</u> dd'

Proof

Applying rule (S1):

⊢ 21: r <u>mod</u> dd = s <u>mod</u> dd (a)

⊢ 22: r"<u>mod</u> dd = s <u>mod</u> dd

⊃ r" <u>mod</u> dd' = s' <u>mod</u> dd' (b)

(a) holds by (A8).

For proving (b) it is sufficient to prove (because of (A10) and (A11)) that:

$$\vdash 22: r''\underline{mod}\ (dd'+dd') = (s'-dd')\ \underline{mod}\ (dd'+dd') \supset r''\underline{mod}\ dd' = s'\underline{mod}\ dd' \qquad (b_1)$$

(b_1) holds by (A9) and by a property of $\underline{mod}$ (consider successively the cases

$$0 \leq r''\underline{mod}\ (dd'+dd') < dd'$$

and $$dd' \leq r''\underline{mod}\ (dd'+dd') < dd'+dd')$$

A.3.2 Lemma:

$$\vdash 11: r'\underline{mod}\ d = r\ \underline{mod}\ d$$

Proof

Applying rule (S2):

$$\vdash 12: dd = d \wedge s = r \qquad (a)$$

$$\vdash 21: dd' = d \wedge s' = r' \supset r'\underline{mod}\ d = r\ \underline{mod}\ d \qquad (b)$$

(a) holds by (A5) and (A6)

(b) holds by the previous lemma.

A.3.3 Lemma (subgoal of loop 1):

$$\vdash 1e: out = r'\underline{mod}\ d$$

Proof

Applying rule (S1):

$$\vdash 1e: out = r\ \underline{mod}\ d \qquad (a)$$

$$\vdash 11: out'' = r\ \underline{mod}\ d \supset out'' = r'\ \underline{mod}\ d \qquad (b)$$

(a) holds by (A3) and (A2)

(b) holds by the previous lemma

A.3.4 Theorem (partial correctness):

$\vdash$ be: out = a <u>mod</u> d

Proof

Applying rule (S2):

$\vdash$ b1: r = a (a)

$\vdash$ le: r'= a $\supset$ out = a <u>mod</u> d (b)

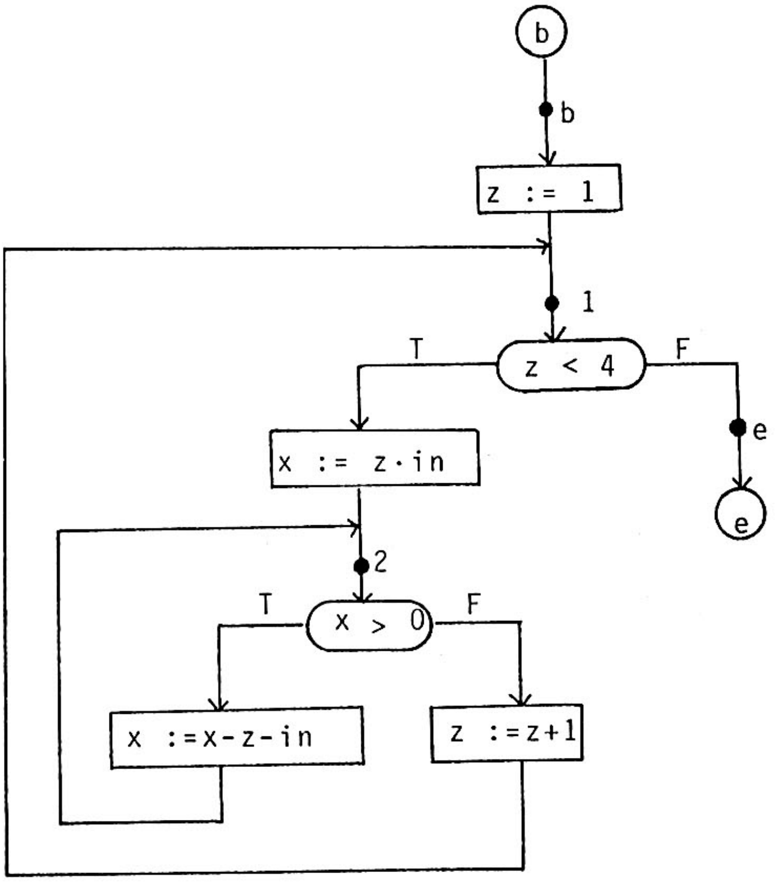
(a) holds by (A1)

(b) holds by the previous lemma

Appendix II: Illustration of the well-founded sets method.

B.1 The program and its semantics

The program is a "toy program"; we are only interested in proving its termination (for any integer value - positive, negative or zero - of the input variable *in*)



⊢ b1:	$z=1$	(A1)
⊢ 1e:	$z\geq 4$	(A2)
⊢ 12:	$z<4$	(A3)
⊢ 12:	$x=z\cdot in$	(A4)
⊢ 21:	$x\leq 0$	(A5)
⊢ 21:	$z=z'+1$	(A6)
⊢ 22:	$x'>0$	(A7)
⊢ 22:	$x=x'-z-in$	(A8)

B.2 The termination proof

B.2.1 Lemma (invariant in cutpoint 2):

$$\vdash 2 : z>0 \wedge x>0 \supset in>0$$

Proof

According to the inference rule (I1) it is sufficient to prove

$$\vdash 12 : z>0 \wedge x>0 \supset in>0 \qquad (a)$$

$$\vdash 22 : (z>0 \wedge x'>0 \supset in>0) \supset (z>0 \wedge x>0 \supset in>0) \qquad (b)$$

(a) holds by (A4)

(b) holds by (A7)

B.2.2 Lemma (conditional termination of loop 2):

$$\vdash 2 : z>0 \supset T$$

Proof

Applying the inference rule for termination with

$$\underline{\text{if}}\ x>0\ \underline{\text{then}}\ x+1\ \underline{\text{else}}\ 1$$

for t and

$$z>0 \wedge (x>0 \supset in>0)$$

for q we have to prove

$$\vdash 2: z>0 \wedge (x>0 \supset in>0) \supset (\underline{\text{if}}\ x>0\ \underline{\text{then}}\ x+1\ \underline{\text{else}}\ 1)>0 \qquad (a)$$

$\vdash$ 22: $z>0 \wedge (x'>0 \supset in>0)$

$\supset$ (if x'>0 then x'+1 else 1)>(if x>0 then x+1 else 1) (b)

and

$\vdash$ 2: $z>0 \supset (z>0 \wedge (x>0 \supset in>0))$ (c)

(a) holds by a property of if-then-else.

Because of (A7) and (A8) (b) is proved if we can prove

$\vdash$ 22: $z>0 \wedge in>0$

$\supset$ x'+1 > (if x'-z-in>0 then x'-z-in+1 else 1) (b')

(b') holds by a property of if-then-else (consider successively the cases $x'-z-in>0$ and $x'-z-in \leq 0$) and of (A7) .

(c) follows from the previous lemma.

B.2.3 Lemma (invariant in cutpoint 1):

$\vdash$ 1 : $0<z<5$

Proof

Applying (I1):

$\vdash$ b1: $0<z<5$ (a)

$\vdash$ 11: $0<z'<5 \supset 0<z<5$ (b)

(a) holds by (A1)

(b) holds if

$\vdash$ 11: $z=z'+1$ (b1)

and

$\vdash$ 11: $z'<4$ (b2)

(b1) results by (A6) from an application of the inference rule (I2) with <u>true</u> for r.

(b2) results by (A3) from an application of the inference rule (S2) with $z<4$ for r and $z'<4$ for q.

B.2.4 Theorem (termination):

$$\vdash 1 : T$$

Proof

Applying the inference rule for termination with <u>true</u> for r,

$$z>0$$

for q and s, and

$$5-z$$

for t we have to prove

$$\vdash 1: \; z>0 \supset 5-z>0 \qquad (a)$$

$$\vdash 11: \; z'>0 \supset 5-z' > 5-z \qquad (b)$$

$$\vdash 2: \; z>0 \supset T \qquad (c)$$

$$\vdash 1: \; \underline{true} \supset z>0 \qquad (d)$$

(a) and (d) follow from the previous lemma and (b) from (b1) in the proof of the previous lemma, (c) is proved in B.2.2 .

References

[1] E.A. Ashcroft, W.W. Wadge, "LUCID, a formal system for writing and proving programs", SIAM Journal Comp. 5, 3 (1976)

[2] E.W. Dijkstra, "A discipline of programming", Prentice Hall, 1976

[3] I. Glasner, "Formale Beweise über while-Programme: Ein Kalkül und sein Modell", Diplomarbeit, Universität des Saarlandes, Saarbrücken, 1978

[4] C.A.R. Hoare, "An axiomatic basis of computer programming", Comm. ACM 12, 10 (1969)

[5] S. Lehmann, J. Loeckx, "An algorithm normalizing elementary while-programs", Bericht A 76/14, Fachbereich 10, Universität des Saarlandes, Saarbrücken (1976).

[6] Z. Manna, "Mathematical theory of computation", McGraw-Hill, 1974

[7] Z. Manna, A. Pnueli, "Axiomatic approach to total correctness of programs", Acta Informatica 3, 3 (1974)

[8] R. Milner, "Implementation and application of Scott's logic for computable functions", SIGPLAN Notices 7,1 (1972)

[9] J.H. Morris, B. Wegbreit, "Subgoal induction", Comm. ACM 20, 4 (1977).