# Program analysis: from proving correctness to proving incorrectness

**Roberto Bruni, Roberta Gori**
**(University of Pisa)**
**Lecture #02**

**BISS 2024**
**March 11-15, 2024**

# Before we start…

# Please answer questions

There are the 3 possible answers to the verification problem
“does my program $c$ satisfy the specification $S$ ?”

- ○ yes
- ○ no
- ○ don’t know

please pick one option whenever we ask questions in these classes

# This is a course on (in)correctness

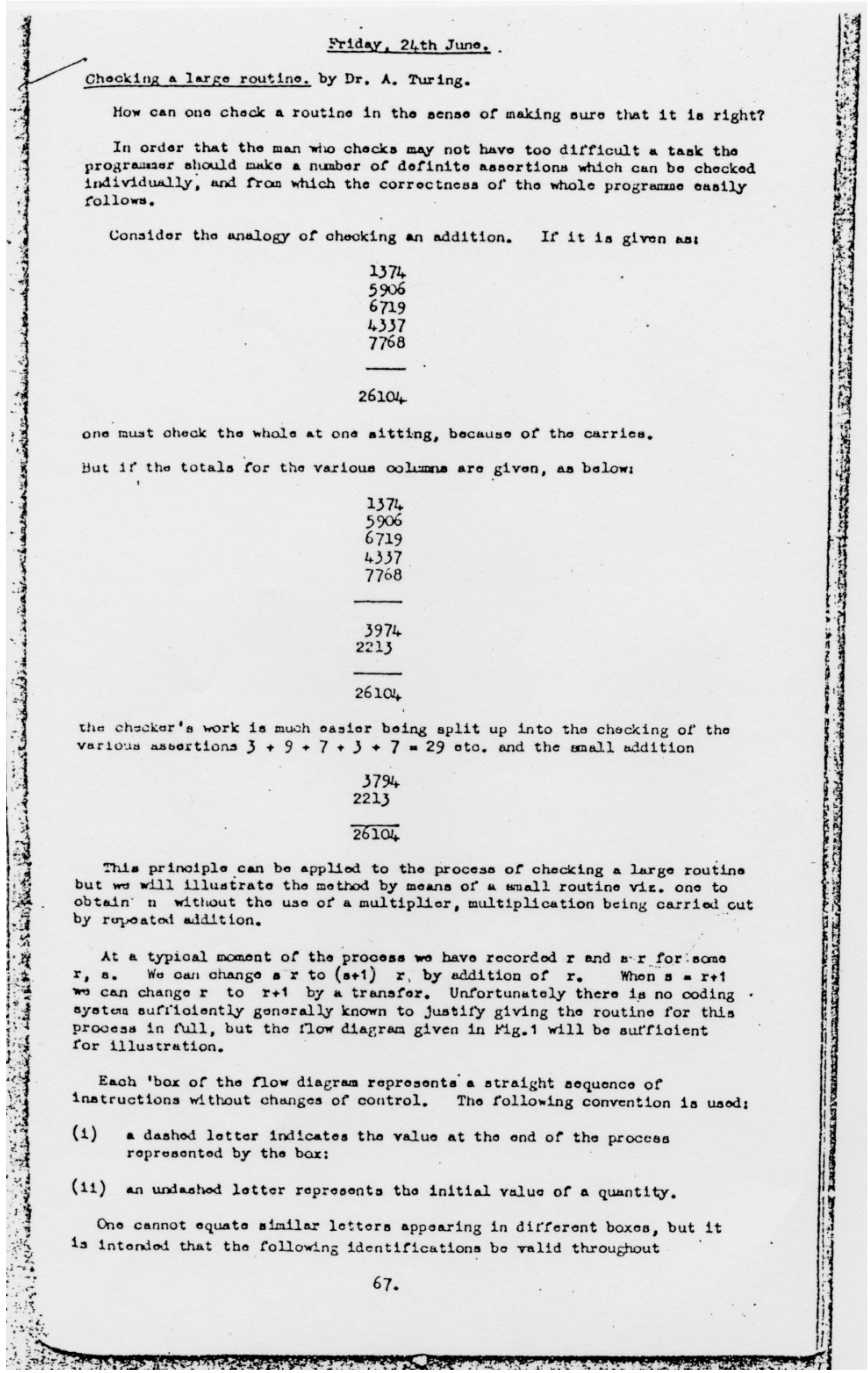
Inevitably, there will be errors in the slides, help us to correct them

**Can you find the the mistake?**

1 2 3 4 5 6 7 8 9

# Program correctness:
# a long standing problem

# Origins? Turing's assertions



Friday, 24th June.

Checking a large routine. by Dr. A. Turing.

How can one check a routine in the sense of making sure that it is right?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

Consider the analogy of checking an addition. If it is given as:

1374
5906
6719
4337
7768

26104

one must check the whole at one sitting, because of the carries.

But if the totals for the various columns are given, as below:

1374
5906
6719
4337
7768

3974
2213

26104

the checker's work is much easier being split up into the checking of the various assertions 3 + 9 + 7 + 3 + 7 = 29 etc. and the small addition

3794
2213

26104

This principle can be applied to the process of checking a large routine but we will illustrate the method by means of a small routine viz. one to obtain n without the use of a multiplier, multiplication being carried out by repeated addition.

At a typical moment of the process we have recorded r and s r for some r, s. We can change s r to (s+1) r, by addition of r. When s = r+1 we can change r to r+1 by a transfer. Unfortunately there is no coding system sufficiently generally known to justify giving the routine for this process in full, but the flow diagram given in Fig.1 will be sufficient for illustration.

Each 'box of the flow diagram represents a straight sequence of instructions without changes of control. The following convention is used:

(i) a dashed letter indicates the value at the end of the process represented by the box:

(ii) an undashed letter represents the initial value of a quantity.

One cannot equate similar letters appearing in different boxes, but it is intended that the following identifications be valid throughout

67.

"how can one check a routine in the sense of making sure that it is right?"

Alan Turing (1949)

# Checking factorial

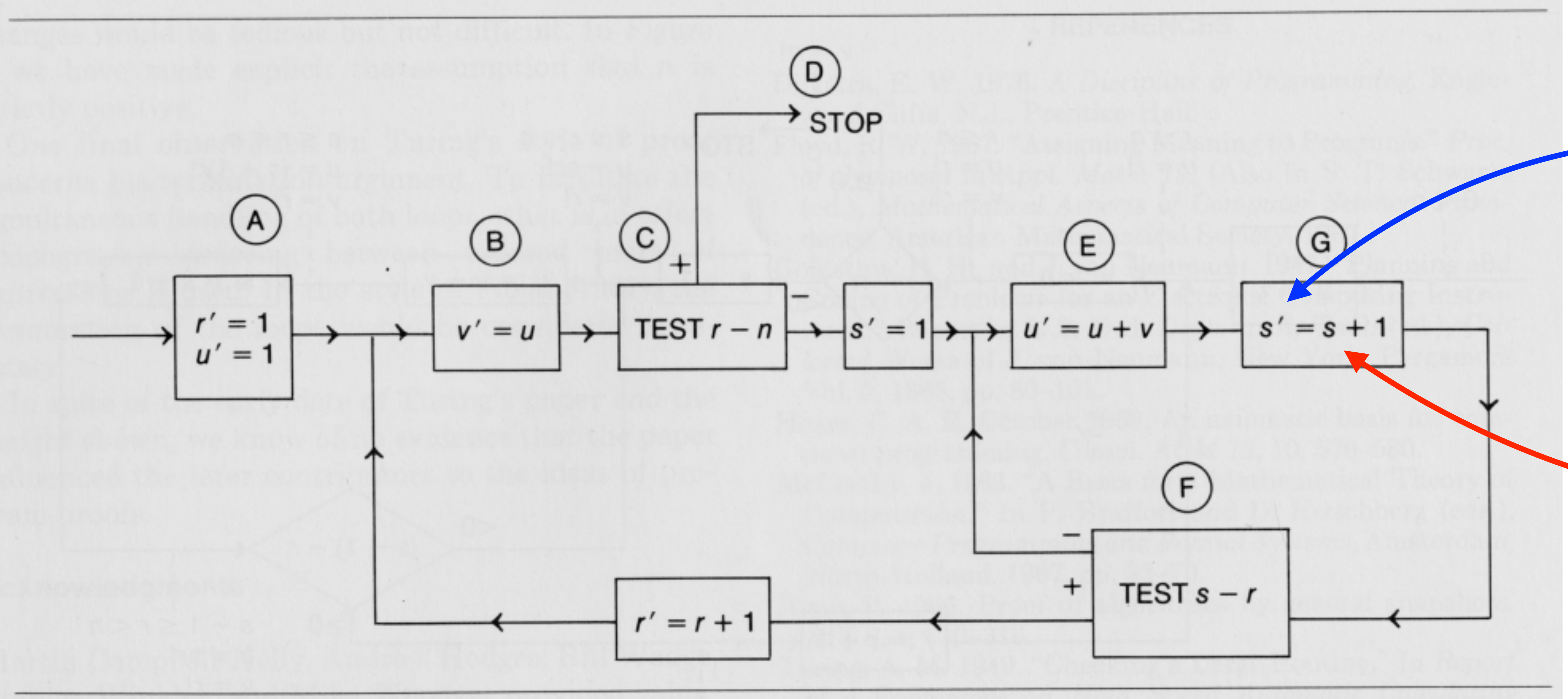



**Figure 1** (Redrawn from Turing's original)

| STORAGE LOCATION | (INITIAL) A $k = 6$ | B $k = 5$ | C $k = 4$ | (STOP) D $k = 0$ | E $k = 3$ | F $k = 1$ | G $k = 2$ |
|---|---|---|---|---|---|---|---|
| 27 s | | | | | $s$ | $s + 1$ | $s$ |
| 28 r | | $r$ | $r$ | | $r$ | $r$ | $r$ |
| 29 n | $n$ | $n$ | $n$ | $n$ | $n$ | $n$ | $n$ |
| 30 u | | $\lfloor r$ | $\lfloor r$ | | $s \lfloor r$ | $(s + 1) \lfloor r$ | $(s + 1) \lfloor r$ |
| 31 v | | | $\lfloor r$ | $\lfloor n$ | $\lfloor r$ | $\lfloor r$ | $\lfloor r$ |
| | TO B WITH $r' = 1$ $u' = 1$ | TO C | TO D IF $r = n$ TO E IF $r < n$ | | TO G | TO B WITH $r' = r + 1$ IF $s \geq r$ TO E WITH $s' = s + 1$ IF $s < r$ | TO F |

**Figure 2** (Redrawn from Turing's original)

- a dashed letter indicates the value at the end of the process represented by the box
- an undashed letter represents the initial value of a quantity
- TEST is test for zero
- $\lfloor$ denotes factorial
- at the end (D) $v = n!$

# General snapshots (P. Naur, 1966)

BIT 6 (1966), 310–316

PROOF OF ALGORITHMS BY GENERAL SNAPSHOTS

PETER NAUR

**Abstract.**

A constructive approach to the question of proofs of algorithms is to consider proofs that an object resulting from the execution of an algorithm possesses certain static characteristics. It is shown by an elementary example how this possibility may be used to prove the correctness of an algorithm written in ALGOL 60. The stepping stone of the approach is what is called General Snapshots, i.e. expressions of static conditions existing whenever the execution of the algorithm reaches particular points. General Snapshots are further shown to be useful for constructing algorithms.

Key words: Algorithm, proof, computer, programming.

**Introduction.**

It is a deplorable consequence of the lack of influence of mathematical thinking on the way in which computer programming is currently being pursued, that the regular use of systematic proof procedures, or even the realization that such proof procedures exist, is unknown to the large majority of programmers. Undoubtedly, this fact accounts for at least a large share of the unreliability and the attendant lack of over-all effectiveness of programs as they are used to-day.

Historically this state of affairs is easily explained. Large scale computer programming started so recently that all of its practitioners are, in fact, amateurs. At the same time the modern computers are so effective that they offer advantages in use even when their powers are largely wasted. The stress has been on always larger, and, allegedly, more powerful systems, in spite of the fact that the available programmer competence often is unable to cope with their complexities.

However, a reaction is bound to come. We cannot indefinitely continue to build on sand. When this is realized there will be an increased interest in the less glamorous, but more solid, basic principles. This will go in parallel with the introduction of these principles in the elementary school curricula. One subject which will then come up for attention is that of proving the correctness of algorithms. The purpose of the present article is to show in an elementary way that this subject not only exists, but is ripe to be used in practise. The illustrations are phrased in ALGOL 60, but the technique may be used with any programming language.

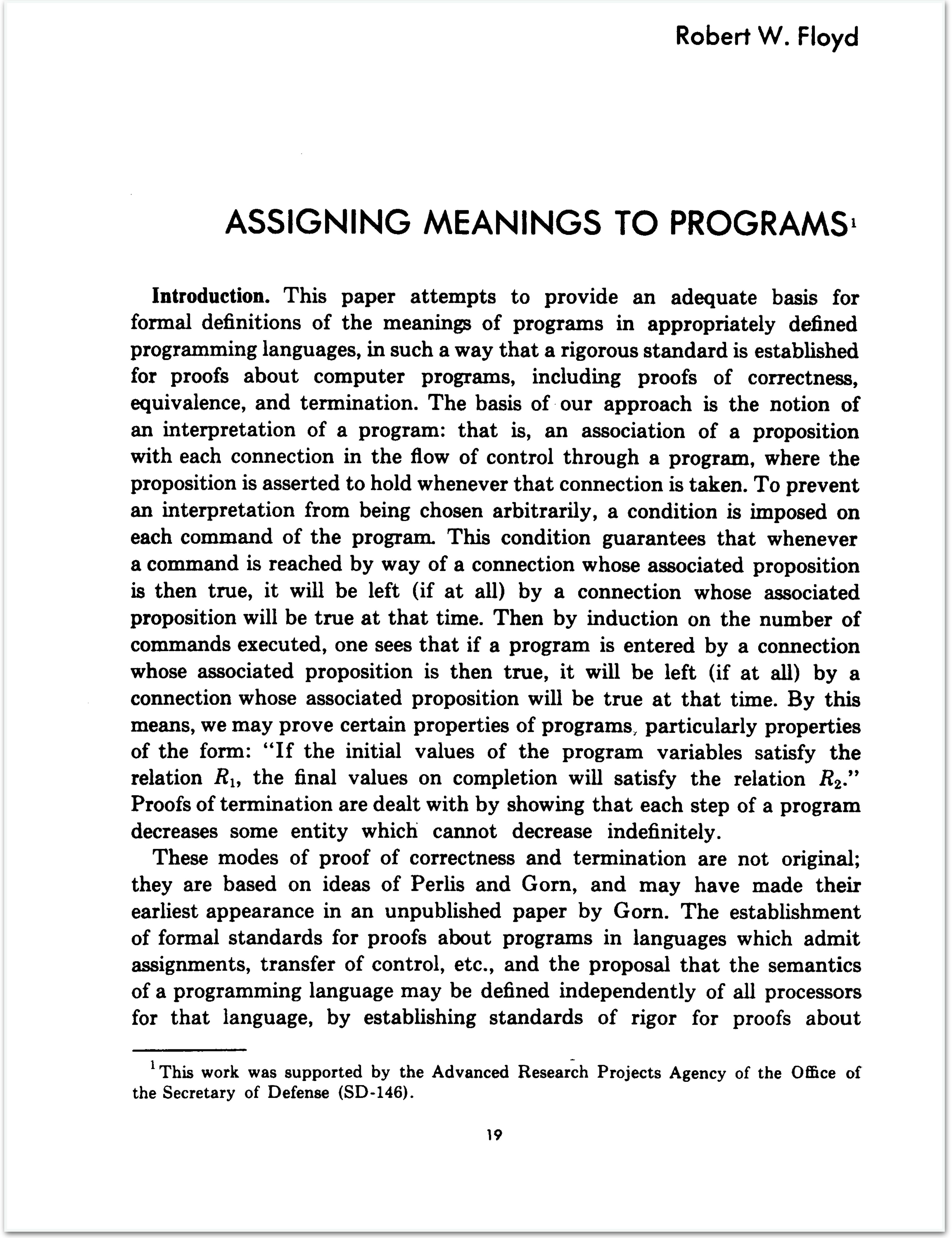


"expression of static conditions existing whenever the execution of the algorithm reaches particular points"

Greatest number, with snapshots

**comment** *General Snapshot* 1: $1 \leqq N$;
$r := 1$;
**comment** *General Snapshot* 2: $1 \leqq N$, $r = 1$;
**for** $i := 2$ **step** 1 **until** $N$ **do**
  **begin** **comment** *General Snapshot* 3: $2 \leqq i \leqq N$, $1 \leqq r \leqq i-1$, $A[r]$ *is the greatest among the elements* $A[1], A[2], \ldots, A[i-1]$;
  **if** $A[i] > A[r]$ **then** $r := i$;
  **comment** *General Snapshot* 4: $2 \leqq i \leqq N$, $1 \leqq r \leqq i$, $A[r]$ *is the greatest among the elements* $A[1], A[2], \ldots, A[i]$;
  **end**;
**comment** *General Snapshot* 5: $1 \leqq r \leqq N$, $A[r]$ *is the greatest among the elements* $A[1], A[2], \ldots, A[N]$;
$R := A[r]$;
**comment** *General Snapshot* 6: $R$ *is the greatest value of any element*, $A[1], A[2], \ldots, A[N]$;

# Floyd's interpretations (1967)



Robert W. Floyd

ASSIGNING MEANINGS TO PROGRAMS[1]

**Introduction.** This paper attempts to provide an adequate basis for formal definitions of the meanings of programs in appropriately defined programming languages, in such a way that a rigorous standard is established for proofs about computer programs, including proofs of correctness, equivalence, and termination. The basis of our approach is the notion of an interpretation of a program: that is, an association of a proposition with each connection in the flow of control through a program, where the proposition is asserted to hold whenever that connection is taken. To prevent an interpretation from being chosen arbitrarily, a condition is imposed on each command of the program. This condition guarantees that whenever a command is reached by way of a connection whose associated proposition is then true, it will be left (if at all) by a connection whose associated proposition will be true at that time. Then by induction on the number of commands executed, one sees that if a program is entered by a connection whose associated proposition is then true, it will be left (if at all) by a connection whose associated proposition will be true at that time. By this means, we may prove certain properties of programs, particularly properties of the form: "If the initial values of the program variables satisfy the relation $R_1$, the final values on completion will satisfy the relation $R_2$." Proofs of termination are dealt with by showing that each step of a program decreases some entity which cannot decrease indefinitely.

These modes of proof of correctness and termination are not original; they are based on ideas of Perlis and Gorn, and may have made their earliest appearance in an unpublished paper by Gorn. The establishment of formal standards for proofs about programs in languages which admit assignments, transfer of control, etc., and the proposal that the semantics of a programming language may be defined independently of all processors for that language, by establishing standards of rigor for proofs about

[1] This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-146).

19

"an association of a proposition with each connection in the flow of control through a program, where the proposition is asserted to hold whenever that connection is taken"

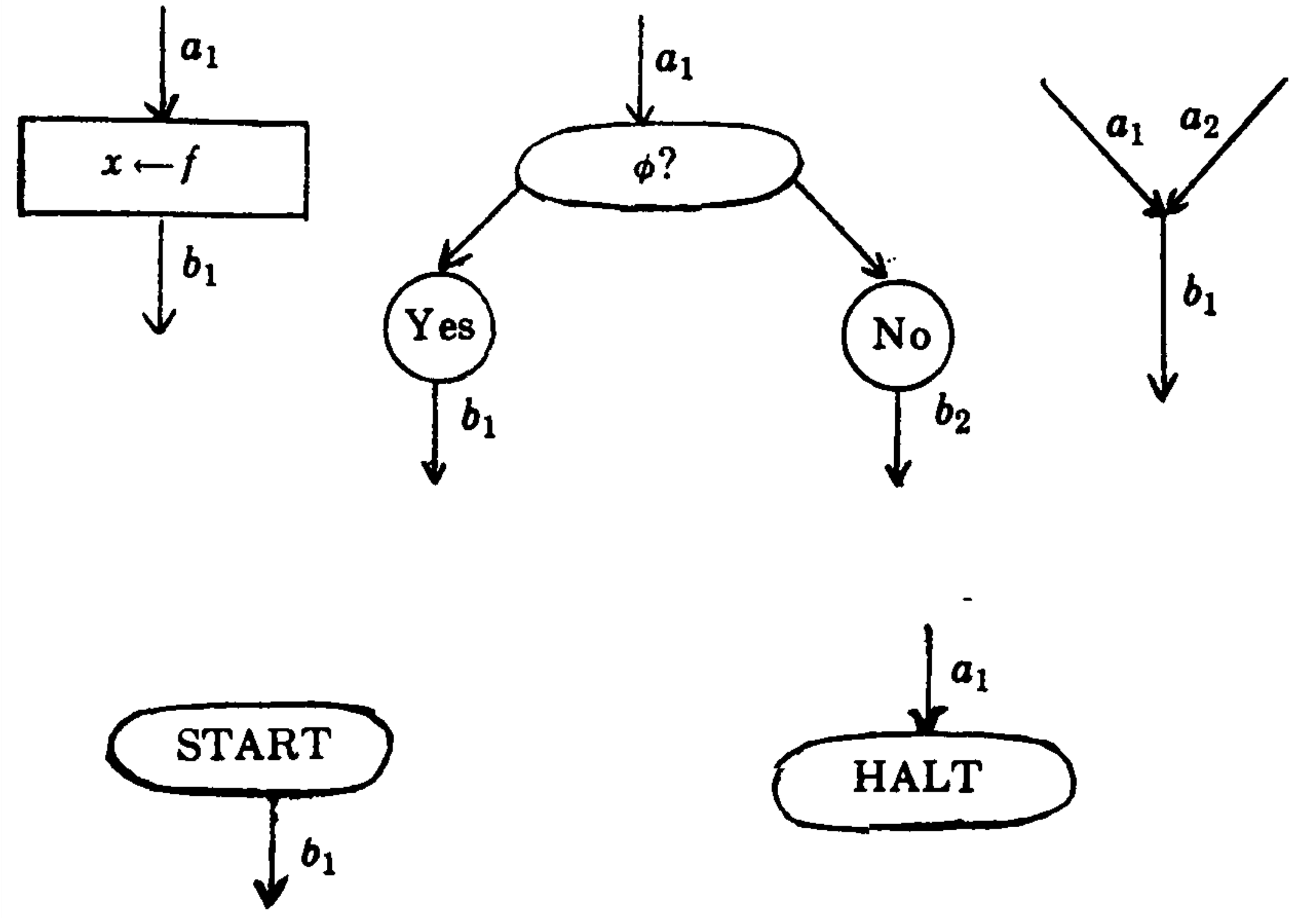



FIGURE 2

# Floyd’s examples

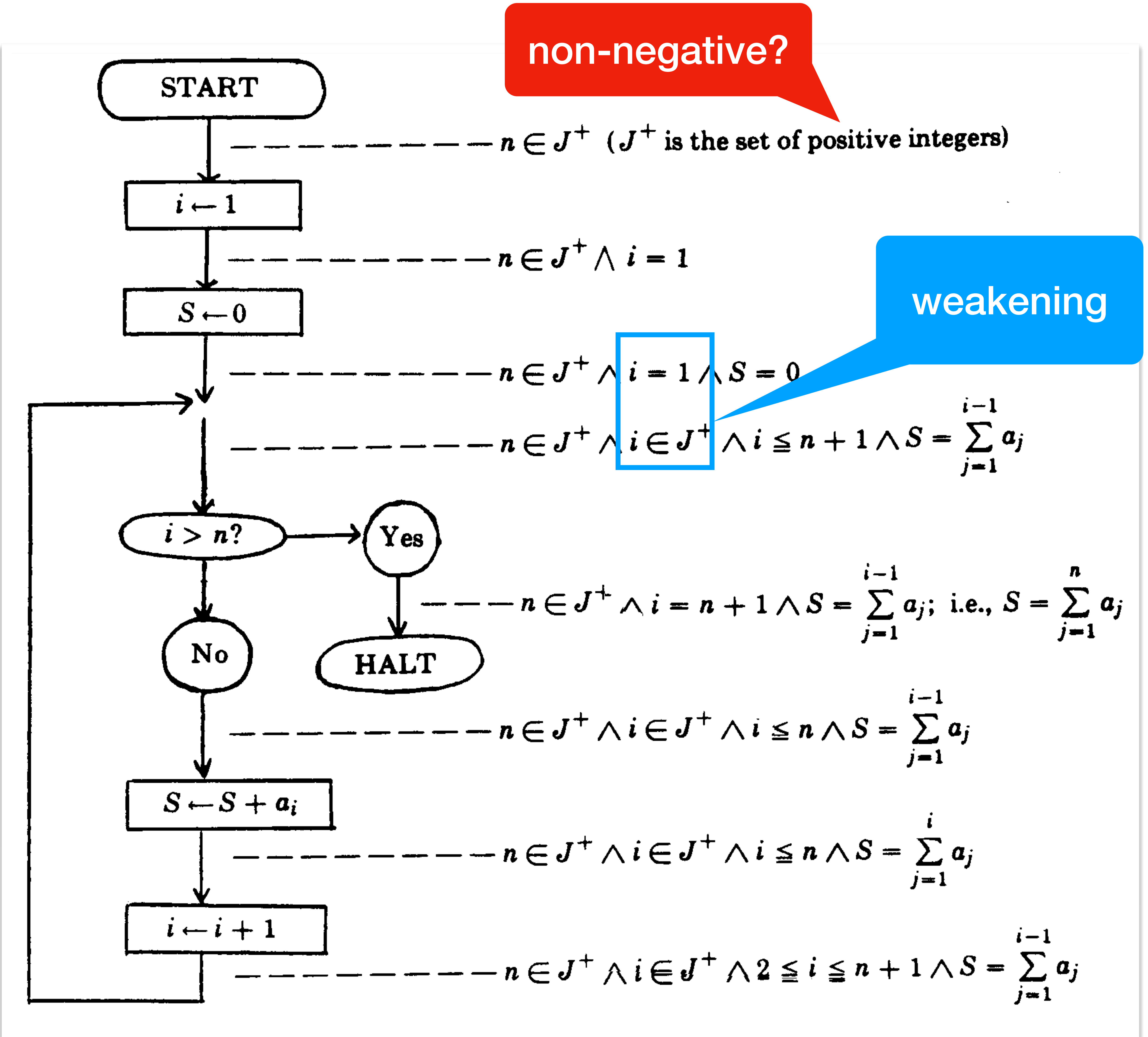



FIGURE 1. Flowchart of program to compute $S = \sum_{j=1}^{n} a_j$ $\boxed{(n \geqq 0)}$

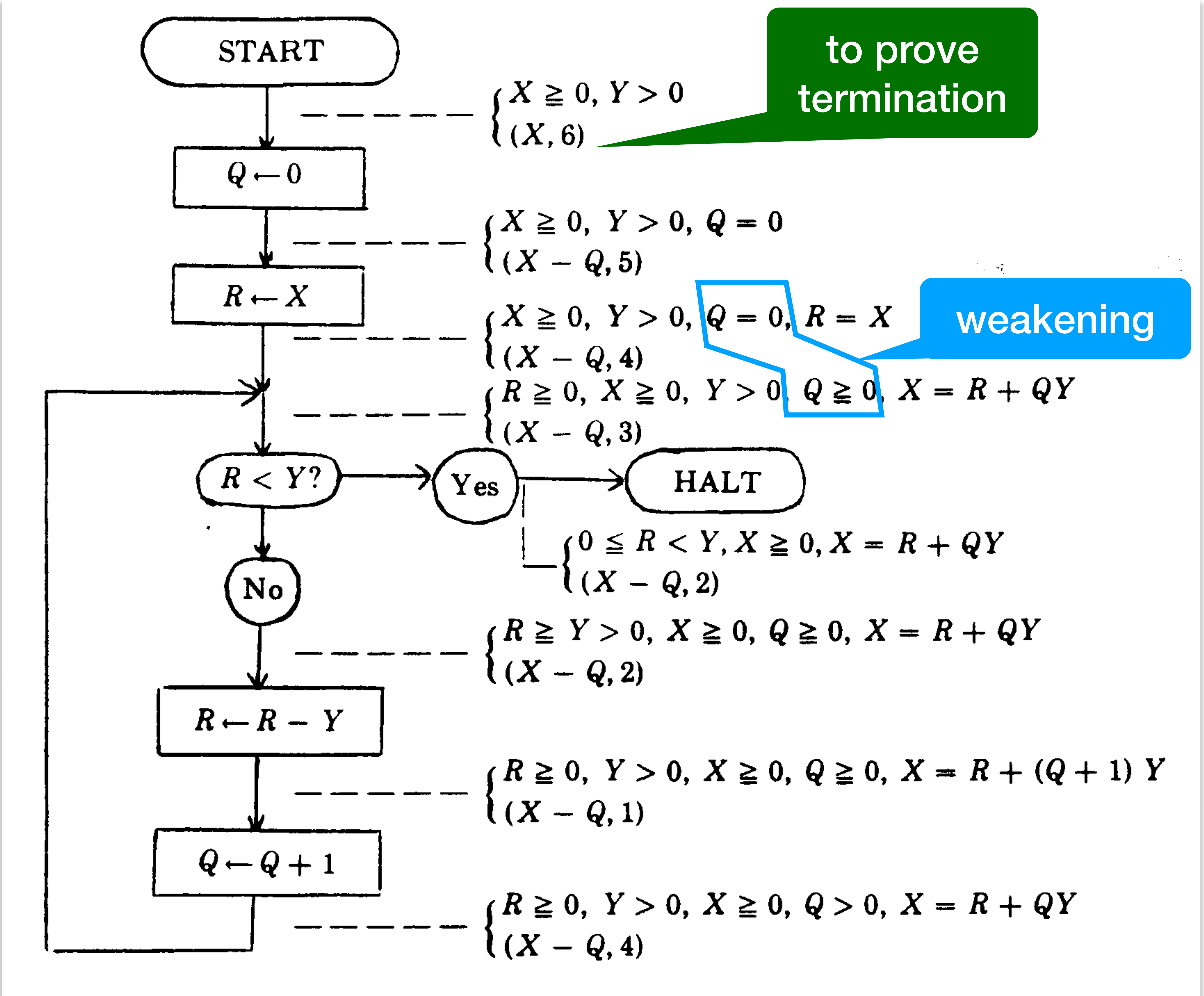
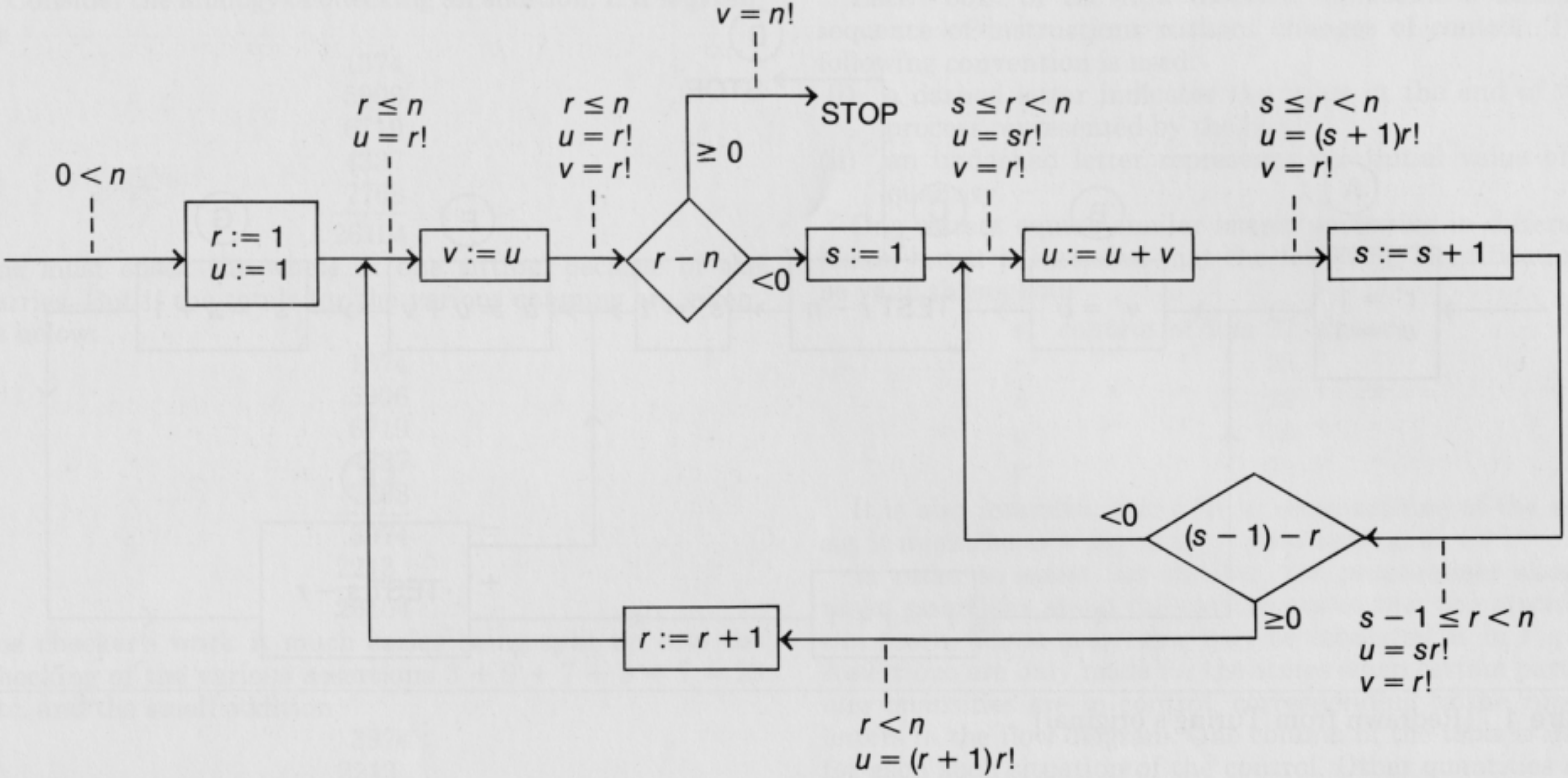
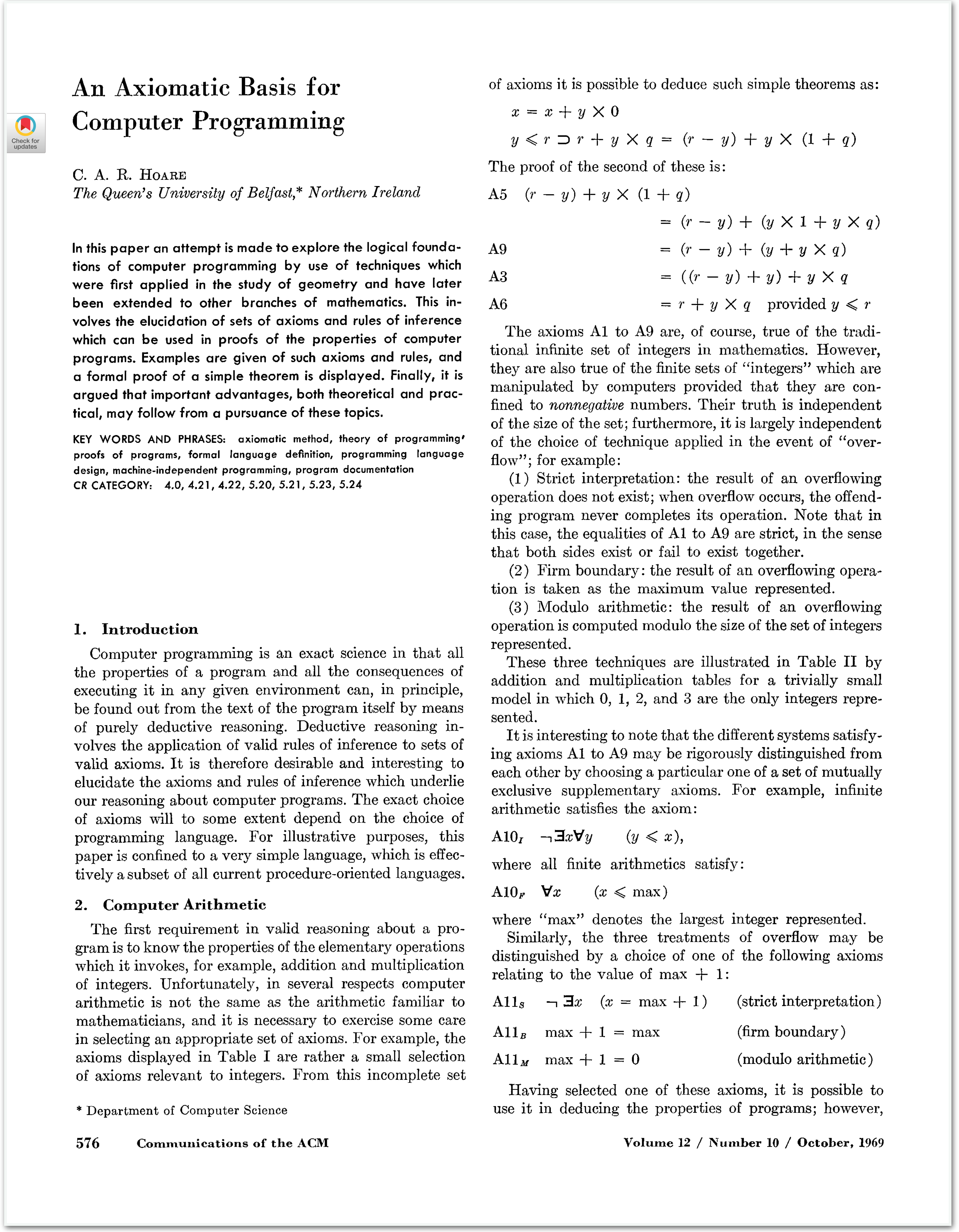



FIGURE 5. Algorithm to compute quotient $Q$ and remainder $R$ of $X \div Y$, for integers $X \geqq 0$, $Y > 0$

# Turing's proof in Floyd's notation



# Hoare Logic



## An Axiomatic Basis for Computer Programming

C. A. R. Hoare
*The Queen's University of Belfast,* Northern Ireland*

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and a formal proof of a simple theorem is displayed. Finally, it is argued that important advantages, both theoretical and practical, may follow from a pursuance of these topics.

KEY WORDS AND PHRASES: axiomatic method, theory of programming' proofs of programs, formal language definition, programming language design, machine-independent programming, program documentation
CR CATEGORY: 4.0, 4.21, 4.22, 5.20, 5.21, 5.23, 5.24

### 1. Introduction

Computer programming is an exact science in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely deductive reasoning. Deductive reasoning involves the application of valid rules of inference to sets of valid axioms. It is therefore desirable and interesting to elucidate the axioms and rules of inference which underlie our reasoning about computer programs. The exact choice of axioms will to some extent depend on the choice of programming language. For illustrative purposes, this paper is confined to a very simple language, which is effectively a subset of all current procedure-oriented languages.

### 2. Computer Arithmetic

The first requirement in valid reasoning about a program is to know the properties of the elementary operations which it invokes, for example, addition and multiplication of integers. Unfortunately, in several respects computer arithmetic is not the same as the arithmetic familiar to mathematicians, and it is necessary to exercise some care in selecting an appropriate set of axioms. For example, the axioms displayed in Table I are rather a small selection of axioms relevant to integers. From this incomplete set of axioms it is possible to deduce such simple theorems as:

$$x = x + y \times 0$$

$$y \leqslant r \supset r + y \times q = (r - y) + y \times (1 + q)$$

The proof of the second of these is:

$$\begin{array}{lr} \text{A5} & (r - y) + y \times (1 + q) \\ & = (r - y) + (y \times 1 + y \times q) \\ \text{A9} & = (r - y) + (y + y \times q) \\ \text{A3} & = ((r - y) + y) + y \times q \\ \text{A6} & = r + y \times q \quad \text{provided } y \leqslant r \end{array}$$

The axioms A1 to A9 are, of course, true of the traditional infinite set of integers in mathematics. However, they are also true of the finite sets of "integers" which are manipulated by computers provided that they are confined to *nonnegative* numbers. Their truth is independent of the size of the set; furthermore, it is largely independent of the choice of technique applied in the event of "overflow"; for example:

(1) Strict interpretation: the result of an overflowing operation does not exist; when overflow occurs, the offending program never completes its operation. Note that in this case, the equalities of A1 to A9 are strict, in the sense that both sides exist or fail to exist together.

(2) Firm boundary: the result of an overflowing operation is taken as the maximum value represented.

(3) Modulo arithmetic: the result of an overflowing operation is computed modulo the size of the set of integers represented.

These three techniques are illustrated in Table II by addition and multiplication tables for a trivially small model in which 0, 1, 2, and 3 are the only integers represented.

It is interesting to note that the different systems satisfying axioms A1 to A9 may be rigorously distinguished from each other by choosing a particular one of a set of mutually exclusive supplementary axioms. For example, infinite arithmetic satisfies the axiom:

$$\text{A10}_I \quad \neg \exists x \forall y \quad (y \leqslant x),$$

where all finite arithmetics satisfy:

$$\text{A10}_F \quad \forall x \quad (x \leqslant \max)$$

where "max" denotes the largest integer represented.

Similarly, the three treatments of overflow may be distinguished by a choice of one of the following axioms relating to the value of max + 1:

$$\begin{array}{lll} \text{A11}_S & \neg \exists x \; (x = \max + 1) & \text{(strict interpretation)} \\ \text{A11}_B & \max + 1 = \max & \text{(firm boundary)} \\ \text{A11}_M & \max + 1 = 0 & \text{(modulo arithmetic)} \end{array}$$

Having selected one of these axioms, it is possible to use it in deducing the properties of programs; however,

* Department of Computer Science



"the purpose of this study is to provide a logical basis for proofs of the properties of a program"
C.A.R. Hoare (1969)

# Hoare's example

find the quotient $q$ and the remainder $r$ obtained on dividing $x$ by $y$

$$((r := x;\quad q := 0);\quad \textbf{while}\ y \leqslant r\ \textbf{do}\ (r := r - y;\quad q := 1 + q))$$

$$\neg y \leqslant r \wedge x = r + y \times q$$

TABLE III

| Line number | Formal proof | Justification |
|---|---|---|
| 1 | $\textbf{true} \supset x = x + y \times 0$ | Lemma 1 |
| 2 | $x = x + y \times 0 \{r := x\} x = r + y \times 0$ | D0 |
| 3 | $x = r + y \times 0\ \{q := 0\}\ x = r + y \times q$ | D0 |
| 4 | $\textbf{true}\ \{r := x\}\ x = r + y \times 0$ | D1 (1, 2) |
| 5 | $\textbf{true}\ \{r := x;\quad q := 0\}\ x = r + y \times q$ | D2 (4, 3) |
| 6 | $x = r + y \times q \wedge y \leqslant r \supset x = (r-y) + y \times (1+q)$ | Lemma 2 |
| 7 | $x = (r-y) + y \times (1+q) \{r := r-y\} x = r + y \times (1+q)$ | D0 |
| 8 | $x = r + y \times (1+q) \{q := 1+q\} x = r + y \times q$ | D0 |
| 9 | $x = (r-y) + y \times (1+q) \{r := r-y;\ q := 1+q\}\ x = r + y \times q$ | D2 (7, 8) |
| 10 | $x = r + y \times q \wedge y \leqslant r\ \{r := r-y;\ q := 1+q\}\ x = r + y \times q$ | D1 (6, 9) |
| 11 | $x = r + y \times q\ \{\textbf{while}\ y \leqslant r\ \textbf{do}\ (r := r-y;\quad q := 1+q)\}\ \neg y \leqslant r \wedge x = r + y \times q$ | D3 (10) |
| 12 | $\textbf{true}\ \{((r := x;\quad q := 0);\quad \textbf{while}\ y \leqslant r\ \textbf{do}\ (r := r-y;\quad q := 1+q))\}\ \neg y \leqslant r \wedge x = r + y \times q$ | D2 (5, 11) |

NOTES

1. The left hand column is used to number the lines, and the right hand column to justify each line, by appealing to an axiom, a lemma or a rule of inference applied to one or two previous lines, indicated in brackets. Neither of these columns is part of the formal proof. For example, line 2 is an instance of the axiom of assignment (D0); line 12 is obtained from lines 5 and 11 by application of the rule of composition (D2).
2. Lemma 1 may be proved from axioms A7 and A8.
3. Lemma 2 follows directly from the theorem proved in Sec. 2.

# Preliminaries

# A simple imperative language

command

integer variable

$$
\begin{aligned}
c ::= \;& x := a \\
\mid\;& \mathsf{skip} \\
\mid\;& c_1 ; c_2 \\
\mid\;& \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \\
\mid\;& \mathsf{while}\ b\ \mathsf{do}\ c
\end{aligned}
$$

arithmetic expression

Boolean expression

$$a ::= n \mid x \mid a_1 + a_2 \mid \ldots$$

$$b ::= a_1 \leq a_2 \mid b_1 \wedge b_2 \mid \ldots$$

# Concrete domain

state

set of variables

$$\sigma : X \to \mathbb{Z}$$

set of integers

set of all states

$$\Sigma \triangleq \{\sigma : X \to \mathbb{Z}\}$$

concrete domain

$$\wp(\Sigma) \triangleq \{P \mid P \subseteq \Sigma\}$$

state property

# Notation

state notation

$$[x \mapsto 1, y \mapsto 2]$$

the state where $x$ holds $1$, $y$ holds $2$ and any other variable holds $0$

state update

$$\sigma[x \mapsto n]$$

the state where $x$ holds $n$ and any other variable $y$ holds $\sigma(y)$

conjunction

property notation

$$(x = 1, y = 2)$$

the set of all states where $x$ holds $1$ **and** $y$ holds $2$

# Assertion language

assertion

$$
\begin{aligned}
P ::= \;& \mathsf{true} \mid \mathsf{false} \mid a_1 < a_2 \mid a_1 = a_2 \mid \ldots \\
\mid \;& \neg P \mid P_1 \wedge P_2 \mid \exists x\,.\,P \mid \ldots
\end{aligned}
$$

Boolean and classical assertions

# Notation

$\sigma \models P$ or also $\sigma \in P$

the state $\sigma$ satisfies the property $P$

$P \Rightarrow Q$ or also $P \subseteq Q$ or also $P \leq Q$

any state that satisfies $P$ satisfies $Q$

# Collecting semantics

concrete semantics

$$[\![c]\!] : \wp(\Sigma) \to \wp(\Sigma)$$

$[\![c]\!]P$

is the set of all and only states reachable from some state in $P$ after executing $c$

$[\![c]\!]\sigma$ as a shorthand for $[\![c]\!]\{\sigma\}$

additive: $[\![c]\!](P_1 \cup P_2) = ([\![c]\!]P_1) \cup ([\![c]\!]P_2)$

# Collecting semantics

concrete semantics

no errors are possible

$$[\![a]\!] : \Sigma \to \mathbb{Z}$$

$$[\![a]\!]\sigma$$

evaluates the arithmetic expression $a$ in the current state $\sigma$

e.g.

$$[\![x + 1]\!][x \mapsto 1, y \mapsto 2] = 2$$

# Collecting semantics

concrete semantics

$$[\![ b ]\!] : \wp(\Sigma) \to \wp(\Sigma)$$

$[\![ b ]\!] P$ (intuitively $b \wedge P$)

is the set of all and only states in $P$ that satisfy the condition $b$

e.g.

$$[\![ x < y ]\!] \{ [x \mapsto 1, y \mapsto 2], [x \mapsto 2, y \mapsto 1] \} = \{ [x \mapsto 1, y \mapsto 2] \}$$

$$[\![ x < y ]\!] [x \mapsto 2, y \mapsto 1] = \varnothing$$

# Collecting semantics: atomic commands

$$[\![\mathsf{skip}]\!]P \triangleq P$$

$$[\![x := a]\!]P \triangleq \{\sigma[x \mapsto [\![a]\!]\sigma] \mid \sigma \in P\}$$

e.g.

$$[\![r := x]\!][x \mapsto 5, y \mapsto 2] = \{[x \mapsto 5, y \mapsto 2, r \mapsto 5]\}$$

# Collecting semantics: sequence

$$[\![c_1; c_2]\!]P \triangleq [\![c_2]\!]([\![c_1]\!]P)$$

e.g.

$$[\![r := x; q := 0]\!][x \mapsto 5, y \mapsto 2] = \{[x \mapsto 5, y \mapsto 2, r \mapsto 5]\}$$

$q \mapsto 0$
implicit

# Collecting semantics: conditionals

$$[\![\text{if } b \text{ then } c_1 \text{ else } c_2]\!]P \triangleq [\![c_1]\!]([\![b]\!]P) \cup [\![c_2]\!]([\![\neg b]\!]P)$$

e.g.

$$\begin{aligned}
&[\![\text{if } x \geq 0 \text{ then skip else } x := -x]\!]\{[x \mapsto -1], [x \mapsto 1]\} \\
&\qquad \triangleq [\![\text{skip}]\!][x \mapsto 1] \cup [\![x := -x]\!][x \mapsto -1] \\
&\qquad \triangleq \{[x \mapsto 1]\}
\end{aligned}$$

# Collecting semantics: loops

least
fixed point

lambda
abstraction

$$[\![\text{while } b \text{ do } c]\!]P \triangleq [\![\neg b]\!]\ \text{fix}\ (\lambda S\,.\,P \cup [\![c]\!][\![b]\!]S)$$

# Kleene's fixpoint theorem

**Th.**

Let $f : \mathscr{C} \to \mathscr{C}$ be a continuous function on a CPO with bottom.

Then fix $f = \bigcup_{k=0}^{\infty} f^k(\perp)$.

e.g.

$$f \triangleq (\lambda S\,.\,P \cup [\![c]\!][\![b]\!]S) : \wp(\Sigma) \to \wp(\Sigma)$$

$$f^0(\emptyset) = \emptyset, \quad f^1(\emptyset) = P, \quad f^2(\emptyset) = P \cup [\![c]\!][\![b]\!]P, \ldots$$

# Collecting semantics: loops

least fixed point

lambda abstraction

$$[\![\text{while } b \text{ do } c]\!]P \triangleq [\![\neg b]\!] \text{ fix } (\lambda S \,.\, P \cup [\![c]\!][\![b]\!]S)$$

$$= [\![\neg b]\!] \bigcup_{k=0}^{\infty} ([\![c]\!][\![b]\!])^k P$$

$$= \bigcup_{k=0}^{\infty} [\![\neg b]\!]([\![c]\!][\![b]\!])^k P$$

# Collecting semantics: loops

$$[\![\text{while } b \text{ do } c]\!]P \triangleq \bigcup_{k=0}^{\infty} [\![\neg b]\!]([\![c]\!][\![b]\!])^k P$$

e.g.

$$w \triangleq \text{while } y \leq r \text{ do}$$
$$\quad r := r - y;$$
$$\quad q := q + 1$$

$$\sigma \triangleq [x \mapsto 5, \; y \mapsto 2, \; r \mapsto 5]$$

$$\begin{aligned}
[\![w]\!]\sigma &= \cup_{k=0}^{\infty} [\![y > r]\!]([\![\ldots]\!][\![y \leq r]\!])^k \sigma \\
&= [\![y > r]\!]\,\sigma \;\; \cup_{k=1}^{\infty} [\![y > r]\!]([\![\ldots]\!][\![y \leq r]\!])^k \sigma \\
&= [\![y > r]\!](\ldots)\,\sigma \;\; \cup_{k=2}^{\infty} [\![y > r]\!](\ldots)^k \sigma \\
&= [\![y > r]\!]\,[x \mapsto 5, y \mapsto 2, r \mapsto 3, q \mapsto 1] \; \cup_{k=2}^{\infty} [\![y > r]\!](\ldots)^k \sigma \\
&= [\![y > r]\!](\ldots)^2\,\sigma \cup_{k=3}^{\infty} [\![y > r]\!](\ldots)^k \sigma \\
&= [\![y > r]\!]\,[x \mapsto 5, y \mapsto 2, r \mapsto 1, q \mapsto 2] \; \cup_{k=3}^{\infty} [\![y > r]\!](\ldots)^k \sigma \\
&= \{[x \mapsto 5, y \mapsto 2, r \mapsto 1, q \mapsto 2]\}
\end{aligned}$$

# Inference rules

premises

$$\frac{\phi_1 \;\; \phi_2 \;\; \cdots \;\; \phi_n}{\phi}$$

conclusion

if all premises hold, then the conclusion holds

axiom

$$\frac{}{\mathsf{pos}(1)} \qquad \frac{\mathsf{pos}(x) \;\; \mathsf{pos}(y)}{\mathsf{pos}(x+y)}$$

# Proof systems

a set of inference rules

$$[\text{base}]\ \frac{}{\mathsf{pos}(1)} \qquad \frac{\mathsf{pos}(x) \quad \mathsf{pos}(y)}{\mathsf{pos}(x + y)}\ [\text{sum}]$$

# Proof tree

$$\dfrac{\text{[base]}\,\dfrac{}{\mathrm{pos}(1)} \qquad \dfrac{\text{[sum]}\,\dfrac{\text{[base]}\,\dfrac{}{\mathrm{pos}(1)} \quad \dfrac{}{\mathrm{pos}(1)}}{\mathrm{pos}(2)} \qquad \dfrac{\dfrac{}{\mathrm{pos}(1)} \quad \dfrac{}{\mathrm{pos}(1)}\,\text{[base]}}{\mathrm{pos}(2)}\,\text{[sum]}}{\mathrm{pos}(4)}\,\text{[sum]}}{\mathrm{pos}(5)}\,\text{[sum]}$$

# Hoare Logic
# (HL)

# Hoare's triples

original
paper

$$P\ \{c\}\ Q$$

pre
condition

since
then

$$\{P\}\ c\ \{Q\}$$

post
condition

when the precondition is met,
executing the command establishes the postcondition

$$[\![c]\!]P \subseteq Q$$

can include non
reachable states

over
approximation!

# An obvious axiom

$$\frac{}{\{P\}\ \mathsf{skip}\ \{P\}}$$

$$\{x > 0\}\ \mathsf{skip}\ \{x > 0\}$$

# Floyd's axiom for assignment

$$\overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}$$
$$\{P\}\ x := a\ \{\, \exists x'\,.\, \underbrace{P[x'/x]}_{\text{syntax replacement}} \wedge x = \underbrace{a[x'/x]}_{\text{syntax replacement}} \,\}$$

$$\{\mathsf{true}\}\ r := x\ \{\, \exists r'\,.\ \mathsf{true}, r = x \,\} \equiv \{r = x\}$$

$$\begin{aligned} \{x = r + qy\}\ r := r - y\ &\{\, \exists r'\,.\ x = r' + qy, r = r' - y \,\} \\ &\equiv \{\, \exists r'\,.\ x = r + y + qy, r' = r + y \,\} \\ &\equiv \{x = r + (q+1)y\} \end{aligned}$$

# Hoare's axiom for assignment

$$\frac{}{\{Q[a/x]\}\ x := a\ \{Q\}}$$

syntax replacement

$$\{\mathsf{true}\} \equiv \{x = x + 0y\}\ r := x\ \{x = r + 0y\}$$

$$\{x = r\} \equiv \{x = r + 0y\}\ q := 0\ \{x = r + qy\}$$

$$\{x = r + qy\} \equiv$$
$$\{x = r - y + (q+1)y\}\ r := r - y \{x = r + (q+1)y\}$$

# An observation

forward oriented

[Floyd's]

$$\overline{\{P\}\ x := a\ \{\exists x'.\, P[x'/x] \land x = a[x'/x]\}}$$

backward oriented

$$\overline{\{Q[a/x]\}\ x := a\ \{Q\}}$$

[Hoare's]

# Composition rule

$$\frac{\{P\}\ c_1\ \{R\} \quad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\ \{Q\}}$$

forward / backward

$$\frac{\{x = r + qy\}\ r := r - y\ \{x = r + (q+1)y\} \qquad \{x = r + (q+1)y\}\ q := q + 1\ \{x = r + qy\}}{\{x = r + qy\}\ r := r - y; q := q + 1\ \{x = r + qy\}}$$

# Inlining assertions

$$\dfrac{\{x = r + qy\}\ r := r - y\ \{x = r + (q+1)y\} \qquad \{x = r + (q+1)y\}\ q := q + 1\ \{x = r + qy\}}{\{x = r + qy\}\ r := r - y; q := q + 1\ \{x = r + qy\}}$$

$\{x = r + qy\}$

$r := r - y;$

$\{x = r + (q+1)y\}$

$q := q + 1$

$\{x = r + qy\}$

# While rule

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

loop invariant

$\{x \geq 0\}$

while $x > 0$ do

$\{x \geq 0 \wedge x > 0\} \equiv \{x > 0\} \equiv \{x \geq 1\} \equiv \{x - 1 \geq 0\}$

$x := x - 1;$

$\{x \geq 0\}$

$\{x \geq 0 \wedge x \leq 0\} \equiv \{x = 0\}$

# Invariants as pre-fixed points

loop invariant

$$\frac{\{I \wedge b\}\ c\ \{I\}}{\{I\}\ \text{while } b \text{ do } c\ \{I \wedge \neg b\}}$$

$\{I \wedge b\}\ c\ \{I\}$ means that $[\![c]\!](I \wedge b) \subseteq I$

i.e. that $[\![c]\!][\![b]\!]I \subseteq I$

i.e. that $I$ is a pre-fixed point of $[\![c]\!] \circ [\![b]\!]$

whenever $P \subseteq I$, by definition of fix:

$[\![\text{while } b \text{ do } c]\!]P \triangleq [\![\neg b]\!]\ \text{fix}(\lambda S\,.\,P \cup [\![c]\!][\![b]\!]S) \subseteq [\![\neg b]\!]I$

# Consequence rule

$$\frac{P \Rightarrow P' \quad \{P'\}\ r\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ r\ \{Q\}}$$

forward / backward

$$\{x \geq 0 \land y > 0\} \Rightarrow$$
$$\{-y < 0 \land x \geq 0 \land y \geq 0\} \Rightarrow$$
$$\{x - y < x \land x + y \geq 0\}$$
$$n := x - y;$$
$$\{n < x \land x + y \geq 0\}$$

# Hoare's proof

$\{\text{true}\} \equiv \{x = x + 0y\}$

$r := x$

$\{x = r + 0y\}$

$q := 0;$

$\{x = r + qy\}$

loop invariant

while $y \leq r$ do

$\{x = r + qy \wedge y \leq r\} \Rightarrow \{x = (r - y) + (q + 1)y\}$

consequence rule

$r := r - y;$

$\{x = r + (q + 1)y\}$

$q := q + 1$

$\{x = r + qy\}$

$\{x = r + qy \wedge y > r\}$

$$\frac{\{P \wedge b\}\; c\; \{P\}}{\{P\}\,\text{while } b \text{ do } c\; \{P \wedge \neg b\}}$$

# Wait a moment…

$\{\text{true}\} \equiv \{x = x + 0y\}$

$r := x$

$\{x = r + 0y\}$

$q := 0;$

$\{x = r + qy\}$

while $y \leq r$ do

$\{x = r + qy \wedge y \leq r\} \;\Rightarrow\; \{x = (r - y) + (q + 1)y\}$

$r := r - y;$

$\{x = r + (q + 1)y\}$

$q := q + 1$

$\{x = r + qy\}$

$\{x = r + qy \wedge y > r\}$

$[\![c]\!][x \mapsto 5, y \mapsto -2] = \ldots = \varnothing$

# Wait a moment…

$\{\text{true}\} \equiv \{x = x + 0y\}$

$r := x$

$\{x = r + 0y\}$

$q := 0;$

$\{x = r + qy\}$

while $z = 0$ do

$\{x = r + qy \land z = 0\} \Rightarrow \{x = (r - y) + (q + 1)y\}$

$r := r - y;$

$\{x = r + (q + 1)y\}$

$q := q + 1$

$\{x = r + qy\}$

$\{x = r + qy \land z \neq 0\}$

$[\![c]\!][x \mapsto 5, y \mapsto 2, z \mapsto 0] = \ldots = \varnothing$

# No guarantee of termination

$\llbracket c \rrbracket [x \mapsto 5] = \ldots = \varnothing$

$\{x \geq 0\}$

while $x > 0$ do

$\{x \geq 0 \wedge x > 0\} \equiv \{x + 1 \geq 0\}$

$x := x + 1;$

$\{x \geq 0\}$

$\{x \geq 0 \wedge x \leq 0\} \equiv \{x = 0\}$

# False positive

$\{x = 1\}$ while $x > 0$ do $x := x + 1$ $\{x = 0\}$

not a possible output!

complete the proof below

$$
\begin{array}{l}
\quad \{x = 1\} \Rightarrow \{\ ?\ \} \\
\text{while } x > 0 \text{ do} \\
\qquad \{\ ?\ \wedge x > 0\} \\
\quad x := x + 1; \\
\qquad \{\ ?\ \} \\
\quad \{\ ?\ \wedge x \leq 0\} \Rightarrow \{x = 0\}
\end{array}
$$

# Partial vs total correctness

when the precondition is met,
executing the command and
establishes the postcondition

partial

$$\{P\}\ c\ \{Q\}$$

when the precondition is met,
executing the command **terminates**
and establishes the postcondition

total

**total correctness = partial correctness + termination**

# Rule for total correctness

proof obligations for termination

variant = termination function

fresh variable

precedence relation

bounded decrease

$$\frac{\{P \wedge b\}\ c\ \{P\} \qquad \{P \wedge b \wedge t = z\}\ c\ \{t \prec z\} \quad P \Rightarrow t \geq 0}{\{P\}\ \text{while}\ b\ \text{do}\ c\ \{P \wedge \neg b\}}$$

# Total correctness proof

$\{x \geq 0\}$ take $t \triangleq x$

while $x > 0$ do

$\{x \geq 0 \wedge x > 0\} \equiv \{x - 1 \geq 0\}$

$x := x - 1;$

$\{x \geq 0\}$

$\{x \geq 0 \wedge x \leq 0\} \equiv \{x = 0\}$

proof obligations

$P \Rightarrow t \geq 0$

$x \geq 0 \Rightarrow x \geq 0$

$\{P \wedge b \wedge t = z\}\ c\ \{t \prec z\}$

$\{x \geq 0 \wedge x > 0 \wedge x = z\} \Rightarrow$

$\{x = z\} \Rightarrow$

$\{x < z + 1\} \equiv$

$\{x - 1 < z\} \Rightarrow$

$x := x - 1$

$\{x < z\}$

# Total correctness proof

$\{x \geq 0 \wedge y > 0\} \equiv \{x \geq 0 \wedge y > 0 \wedge x = x + 0y\}$

$r := x$

$\{x \geq 0 \wedge y > 0 \wedge x = r + 0y\} \equiv \{r \geq 0 \wedge y > 0 \wedge x = r + 0y\}$

$q := 0;$

$\{r \geq 0 \wedge y > 0 \wedge x = r + qy\}$ take $t \triangleq r$

while $y \leq r$ do

$\{r \geq y > 0 \wedge x = r + qy\} \Rightarrow \{r - y \geq 0 \wedge y > 0 \wedge x = r - y + (q+1)y\}$

$r := r - y;$

$\{r \geq 0 \wedge y > 0 \wedge x = r + (q+1)y\}$

$q := q + 1$

$\{r \geq 0 \wedge y > 0 \wedge x = r + qy\}$

$\{y > r \geq 0 \wedge x = r + qy\}$

# Proof obligations

$$P \Rightarrow t \geq 0$$

$$(r \geq 0 \wedge y > 0 \wedge x = r + qy) \ \Rightarrow r \geq 0$$

$$\{P \wedge b \wedge t = z\}\ c\ \{t \prec z\}$$

$$\{r \geq y > 0 \wedge \cdots \wedge r = z\} \Rightarrow \{r \geq 0 \wedge y > 0 \wedge \cdots \wedge r - y < z\}$$

$r := r - y;$

$$\{r \geq 0 \wedge y > 0 \wedge \cdots \wedge r < z\}$$

$q := q + 1$

$$\{r \geq 0 \wedge y > 0 \wedge \cdots \wedge r < z\} \Rightarrow \{r < z\}$$

# If rule

$$\frac{\{P \land b\}\ c_1\ \{Q\} \qquad \{P \land \neg b\}\ c_2\ \{Q\}}{\{P\}\ \text{if}\ b\ \text{then}\ c_1\ \text{else}\ c_2\ \{Q\}}$$

$\{\text{true}\}$

if $x \geq 0$ then

$\{x \geq 0\}$

skip

$\{x \geq 0\}$

else

$\{\neg(x \geq 0)\} \equiv \{-x > 0\}$

$x := -x$

$\{x > 0\} \Rightarrow \{x \geq 0\}$

$\{x \geq 0\}$

# Finding invariants

$\{\text{true}\}$

$k := 1;$

$r := x;$

while $k > 0$ do

    if $r > 100$ then

        $r := r - 10;$

        $k := k - 1$

    else

        $r := r + 11;$

        $k := k + 1$

$\{r = f(x)\}$

what is $f$?

# McCarthy's 91 function

$\{\text{true}\}$

$k := 1;$

$r := x;$

while $k > 0$ do

  if $r > 100$ then

    $r := r - 10;$

    $k := k - 1$

  else

    $r := r + 11;$

    $k := k + 1$

$\{r = f(x)\}$

$$f(x) \triangleq \begin{cases} 91 & x \leq 100 \\ x - 10 & \text{otherwise} \end{cases}$$

# Invariant for McCarthy's 91 function?

$\{\text{true}\}$

$k := 1;$

$r := x;$

$\{\ ?\ \}$

which invariant?

while $k > 0$ do

  if $r > 100$ then

    $r := r - 10;$

    $k := k - 1$

  else

    $r := r + 11;$

    $k := k + 1$

$\{\ ?\ \} \Rightarrow \{r = f(x)\}$

$$f(x) \triangleq \begin{cases} 91 & x \leq 100 \\ x - 10 & \text{otherwise} \end{cases}$$

# Invariant for McCarthy's 91 function

{true}

$k := 1;$

$r := x;$

$\{k \geq 0 \wedge f^k(r) = f(x)\}$

while $k > 0$ do

  if $r > 100$ then

    $r := r - 10;$

    $k := k - 1$

  else

    $r := r + 11;$

    $k := k + 1$

$\{k = 0 \wedge f^k(r) = f(x)\} \Rightarrow \{r = f(x)\}$

$$f(x) \triangleq \begin{cases} 91 & x \leq 100 \\ x - 10 & \text{otherwise} \end{cases}$$

# Variant for McCarthy's 91 function?

$\{\mathsf{true}\}$

$k := 1;$

$r := x;$

$\{k \geq 0 \wedge f^k(r) = f(x)\}$ $\quad t \triangleq ?$

which variant
(for termination)?

while $k > 0$ do

  if $r > 100$ then

    $r := r - 10;$

    $k := k - 1$

  else

    $r := r + 11;$

    $k := k + 1$

$\{k = 0 \wedge f^k(r) = f(x)\} \Rightarrow \{r = f(x)\}$

$$f(x) \triangleq \begin{cases} 91 & x \leq 100 \\ x - 10 & \text{otherwise} \end{cases}$$

# Finding invariants (McCarthy91)

{true}

$k := 1;$

$r := x;$

$\{k \geq 0 \land f^k(r) = f(x)\}$ $\quad t = (|101 - r + 10k|, k)$

lexicographic order

while $k > 0$ do

  if $r > 100$ then

    $r := r - 10;$

    $k := k - 1$

  else

    $r := r + 11;$

    $k := k + 1$

$\{k = 0 \land f^k(r) = f(x)\} \Rightarrow \{r = f(x)\}$

$$f(x) \triangleq \begin{cases} 91 & x \leq 100 \\ x - 10 & \text{otherwise} \end{cases}$$

# Validity, soundness, completeness

# Validity

A HL triple $\{P\}\ c\ \{Q\}$ is **valid** if $[\![c]\!]P \subseteq Q$

Is $\{x > 0\}\ x := 10x\ \{x > 10\}$ valid? ✗

Is $\{x > 0, y > 0\}\ x := yx\ \{x \geq 0\}$ valid? ✓

Is $\{\mathsf{false}\}\ c\ \{Q\}$ valid? ✓

Is $\{P\}\ c\ \{\mathsf{true}\}$ valid? ✓

# Correctness

**Th.** Any derivable HL triple is valid

**Proof.** By induction on the derivation tree

# Incompleteness I

**Conjecture** Any valid HL triple is derivable

**Counterexample:**

$\{\mathsf{true}\}\ c\ \{\mathsf{false}\}$ is valid when $c$ diverges

but halting problem is not r.e.

while the set of derivable HL triples is r.e.

# Incompleteness II

**Conjecture** Any valid HL triple is derivable

**Counterexample:**

$\{\mathsf{true}\}$ skip $\{Q\}$ is valid when $Q$ is a tautology
but Godel's Incompleteness Theorem (1939) tells us that there is no *effective* proof system such that its theorems coincide with all valid arithmetic assertions

# Relative completeness I

**Relative completeness:** suppose we can consult an oracle to check if an assertion $P \Rightarrow P'$ is valid or not, then HL is complete

In other words, we separate concerns about programs and reasoning about them from concerns to do with arithmetic and the incompleteness of any proof system for it

# Dijkstra's weakest precondition

Given a command $c$ and a postcondition $Q$ a **weakest liberal precondition** is a predicate $P$ such that for any precondition $R$

$$\{R\}\ c\ \{Q\} \text{ iff } R \Rightarrow P$$

i.e., $P$ is the least restrictive requirement that guarantees that $Q$ holds after executing $c$ (if it terminates)

Typically, it is denoted by $wlp(c, Q) \triangleq \{\sigma \in \Sigma \mid [\![c]\!]\{\sigma\} \subseteq Q\}$

# Adjoints

$$P \Rightarrow wlp(c, Q)$$

iff

strongest postcondition

$$[\![c]\!]P \subseteq Q$$

iff

$$\{P\}\ c\ \{Q\}$$

# (Relative) Completeness

for any postcondition $Q$ expressible in the logic and for any command $c$, the precondition $wlp(c, Q)$ is also expressible in the logic

**Th.** If the logic language is *expressive enough*, then any valid HL triple can be derived.

**Proof.** Suppose $\{P\}\ c\ \{Q\}$ is valid (with $P$ and $Q$ expressible). By structural induction on $c$ we can build an assertion $R$ that is equivalent to $wlp(c, Q)$ and such that $\{R\}\ c\ \{Q\}$ is derivable. By applying the consequence rule we derive $\{P\}\ c\ \{Q\}$.

# Weakest liberal preconditions

$$wlp(\mathsf{skip}, Q) \triangleq Q$$

$$wlp(x := a, Q) \triangleq Q[x \mapsto a]$$

$$wlp(c_1; c_2, Q) \triangleq wlp(c_1, wlp(c_2, Q))$$

$$wlp(\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, Q) \triangleq (b \Rightarrow wlp(c_1, Q)) \wedge (\neg b \Rightarrow wlp(c_2, Q))$$

$$wlp(\mathsf{while}\ b\ \mathsf{do}\ c, Q) \triangleq \text{more complicated… but possible}$$

# Adding nondeterminism

# Regular commands

regular command

atomic command

choice

Kleene star

$$
\begin{array}{rcl}
r & ::= & e \\
 & | & r_1 ; r_2 \\
 & | & r_1 + r_2 \\
 & | & r^\star
\end{array}
$$

$$e ::= \mathsf{skip} \mid x := a \mid b? \mid \ldots$$

$$[\![ b? ]\!] P \triangleq [\![ b ]\!] P$$

$$[\![ r_1 + r_2 ]\!] P \triangleq [\![ r_1 ]\!] P \cup [\![ r_2 ]\!] P$$

$$[\![ r^\star ]\!] P \triangleq \bigcup_{k=0}^{\infty} [\![ r ]\!]^k P$$

# Encoding while commands

$$\text{if } b \text{ then } c_1 \text{ else } c_2 \triangleq (b?; c_1) + (\neg b?; c_2)$$

$$\text{while } b \text{ do } c \triangleq (b?; c)^{\star}; \neg b?$$

# Minimal set of rules

$$\frac{}{\{P\}\ e\ \{[\![e]\!]P\}}\ \{\text{atom}\}$$

$$\frac{\{P\}\ r_1\ \{R\} \quad \{R\}\ r_2\ \{Q\}}{\{P\}\ r_1; r_2\ \{Q\}}\ \{\text{seq}\}$$

$$\frac{\forall i \in \{1,2\} \quad \{P\}\ r_i\ \{Q\}}{\{P\}\ r_1 + r_2\ \{Q\}}\ \{\text{choice}\}$$

$$\frac{\{P\}\ r\ \{P\}}{\{P\}\ r^\star\ \{P\}}\ \{\text{iter}\}$$

$$\frac{P \Rightarrow P' \quad \{P'\}\ r\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ r\ \{Q\}}\ \{\text{cons}\}$$

# Auxiliary rules

$$\frac{\{P_1\}\ r\ \{Q_1\} \quad \{P_2\}\ r\ \{Q_2\}}{\{P_1 \vee P_2\}\ r\ \{Q_1 \vee Q_2\}}\ \{\text{disj}\}$$

assigned variables in $r$ are disjoint from free variables in $R$

$$\frac{\{P_1\}\ r\ \{Q_1\} \quad \{P_2\}\ r\ \{Q_2\}}{\{P_1 \wedge P_2\}\ r\ \{Q_1 \wedge Q_2\}}\ \{\text{conj}\} \qquad \frac{\{P\}\ r\ \{Q\}}{\{P \wedge R\}\ r\ \{Q \wedge R\}}\ \{\text{frame}\}$$

$$\frac{P \Rightarrow P' \quad \{P'\}\ r\ \{Q\}}{\{P\}\ r\ \{Q\}}\ \{\text{stren}\} \qquad \frac{\{P\}\ r\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ r\ \{Q\}}\ \{\text{weak}\}$$

# Questions

# Question 1

Is $\neg b$ an obvious invariant?

$$\frac{\{P \land b\}\ c\ \{P\}}{\{P\}\ \text{while } b \text{ do } c\ \{P \land \neg b\}}$$

$$\frac{\{\text{false}\}\ c\ \{\neg b\}}{\{\neg b\}\ \text{while } b \text{ do } c\ \{\neg b\}}$$

# Question 2

Find a derivation for the HL triple

$\{\text{true}\}$ if $x \geq y$ then $z := x$ else $z := y$ $\{z = \max(x, y)\}$

$\{\text{true}\}$
if $x \geq y$ then
$\{x \geq y\}$
$z := x$
$\{z = x \geq y\} \Rightarrow \{z = \max(x, y)\}$
else
$\{x < y\}$
$z := y$
$\{z = y > x\} \Rightarrow \{z = \max(x, y)\}$
$\{z = \max(x, y)\}$

# * Exam 1

Prove that rule {conj} is sound

$$\frac{\{P_1\}\ r\ \{Q_1\} \quad \{P_2\}\ r\ \{Q_2\}}{\{P_1 \land P_2\}\ r\ \{Q_1 \land Q_2\}}\ \text{\{conj\}}$$

# * Exam 2

Show that the following rule for assignment is not sound

$$\frac{}{\{P\}\ x := a\ \{P[a/x]\}}$$

syntax replacement