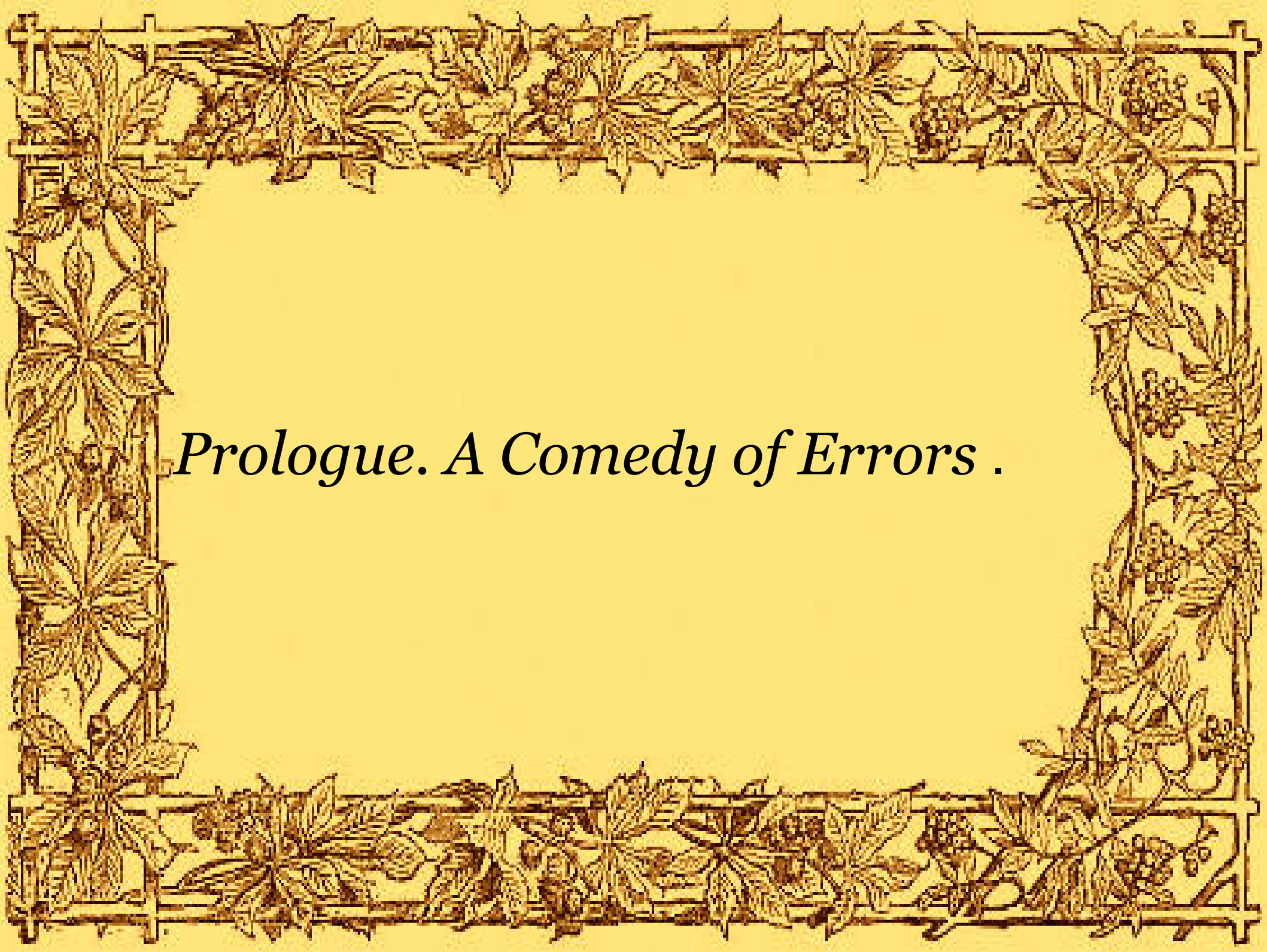


# *Prologue. A Comedy of Errors .*

Rear Admiral Grace Murray Hopper
"Amazing Grace" (1906 -- 1992)

9/9

0800 Arctan started

1000 " stopped - arctan ✓ { 1.2700 9.037 847 025
9.037 846 795 correct

13" dc (032) MP - MC ~~1.9821647000~~ ~~2.13047764715 (-3)~~ 4.615925059(-2)

(033) PRO 2 2.13047 6415

correct 2.13067 6415

Relays 6-2 in 033 failed special speed test
In relay " 11.00 test.

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Mult + Adder Test.

1545 

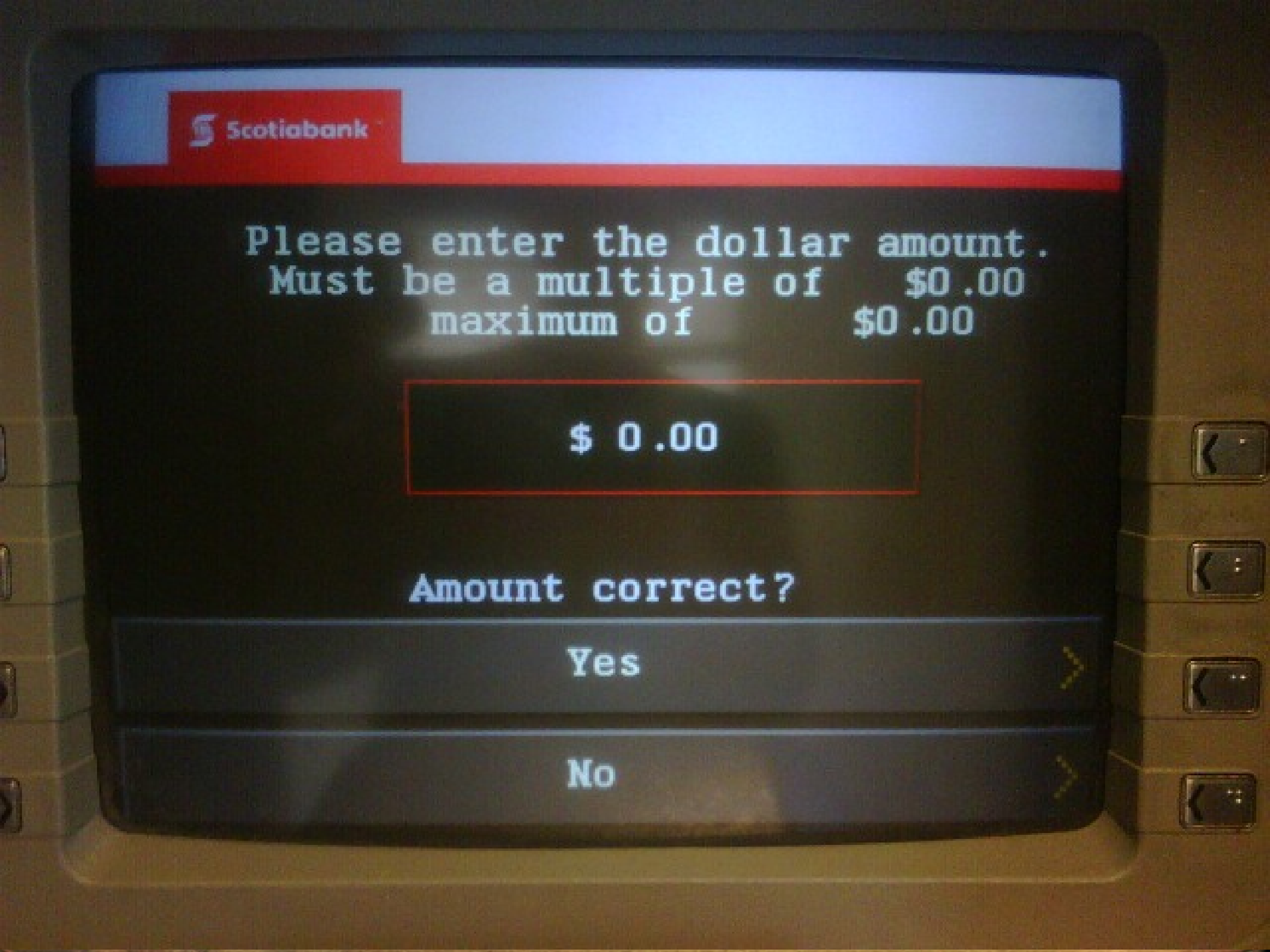
Relay #70 Panel F
(moth) in relay.

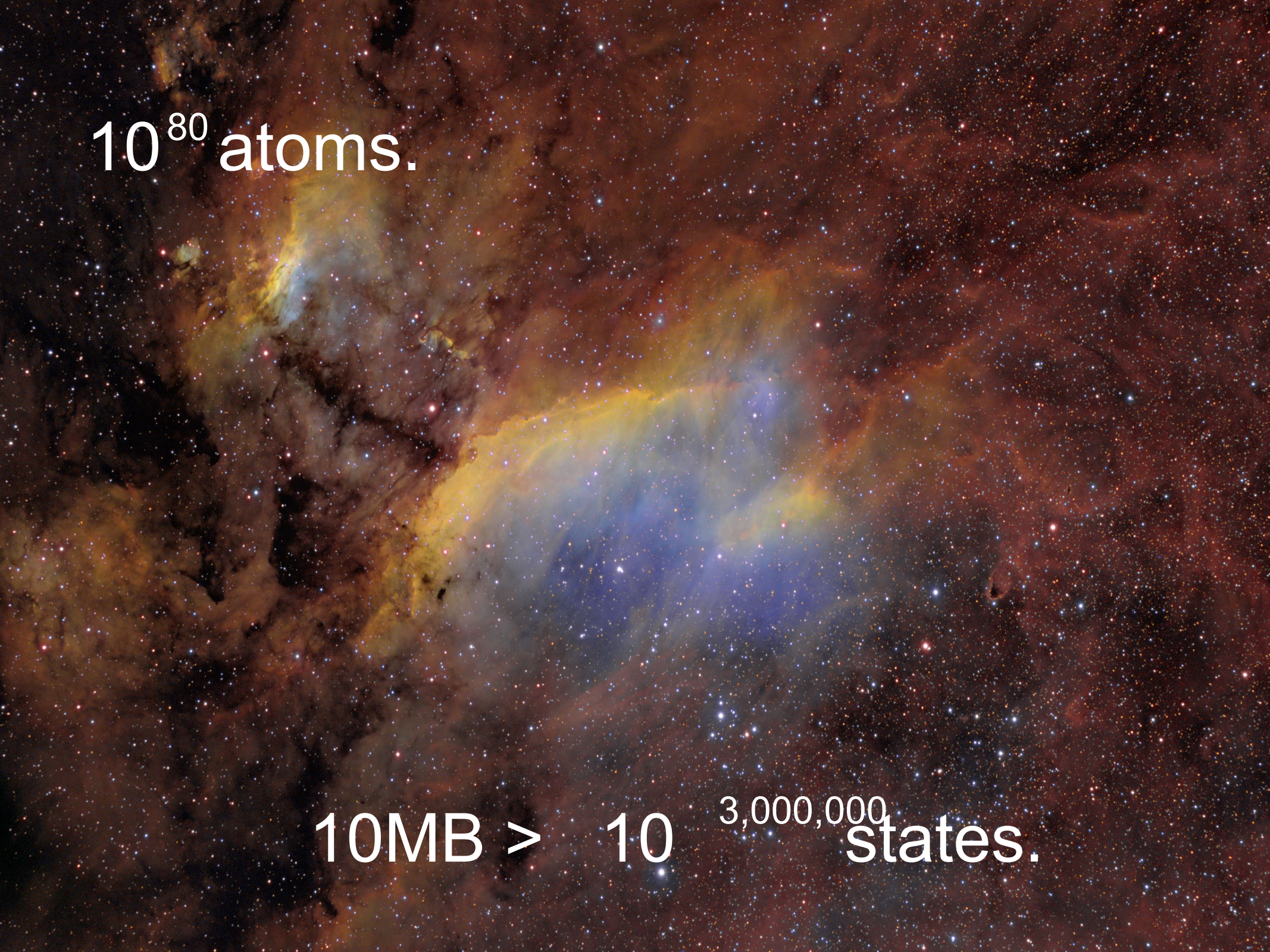
First actual case of bug being found.

~~1700~~ 1630 Arctangent started.

1700 closed down.

Relay 2145
Relay 3370


Scotiabank
Please enter the dollar amount.
Must be a multiple of $0.00
maximum of $0.00
$ 0.00
Amount correct?
Yes
No


$10^{80}$ atoms.
10MB > $10^{3,000,000}$ states.

Software is truly the most complex artifact we build routinely.

It is not surprising we rarely get it right.

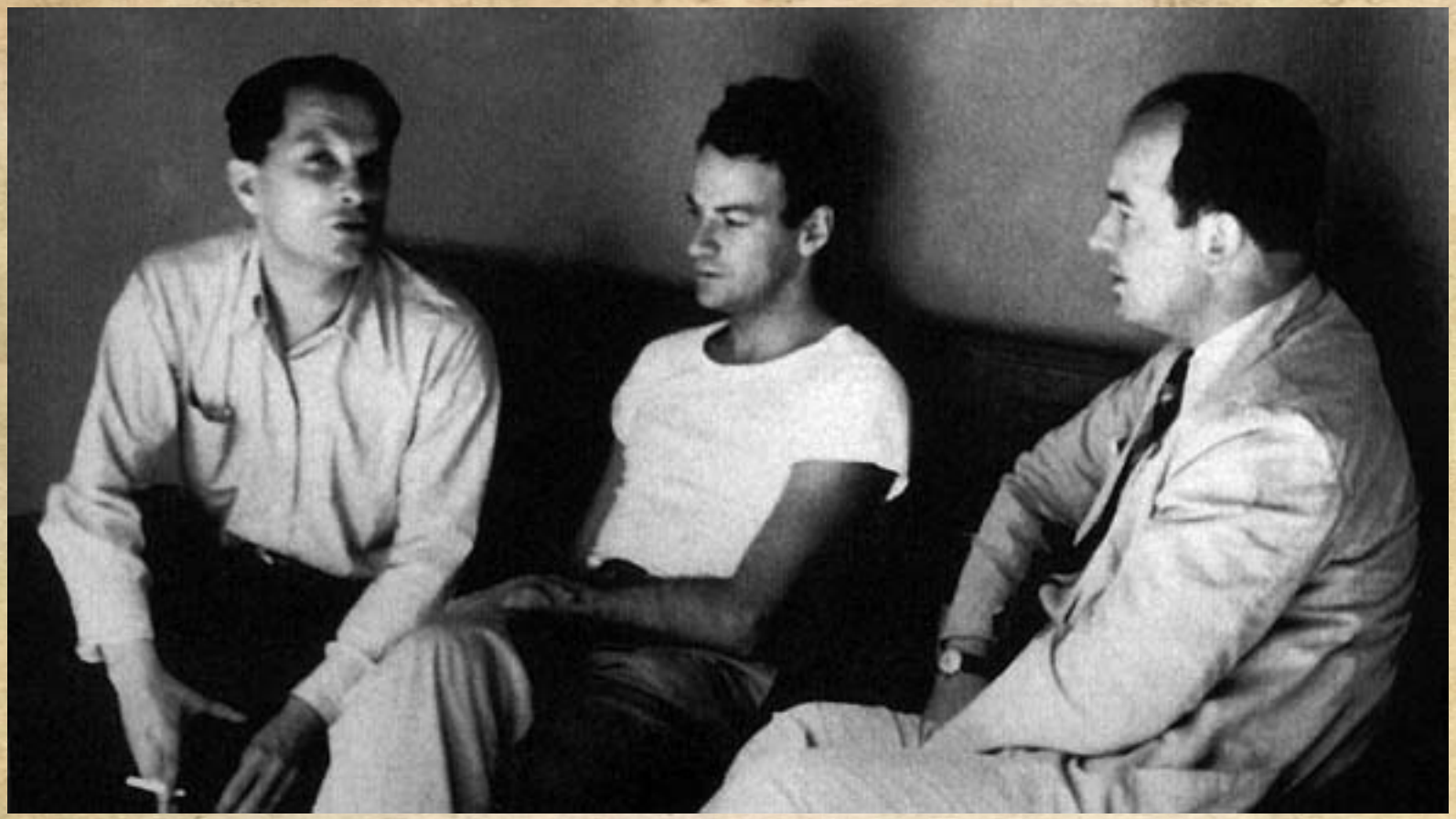
*---Thomas A. Henzinger, EPFL 2006*

*Could the God that plays dice trigger a nuclear holocaust by a random error in a military computer?*

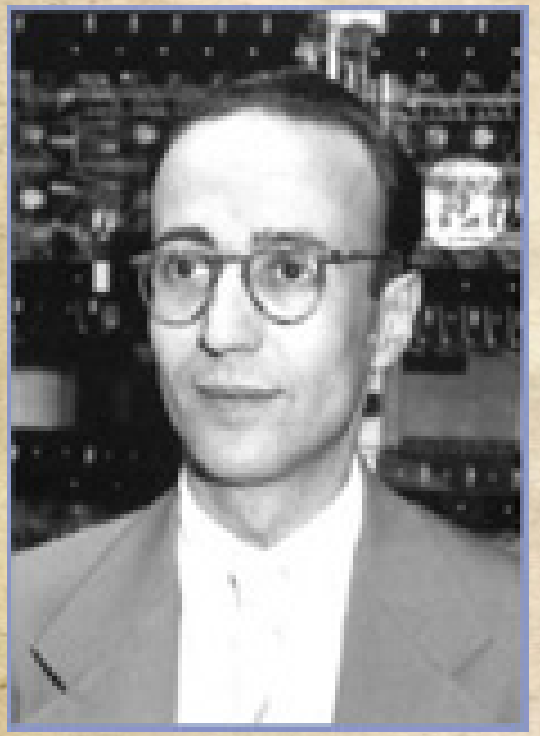
-- Heinz R. Pagels,  The Cosmic Code (1982)

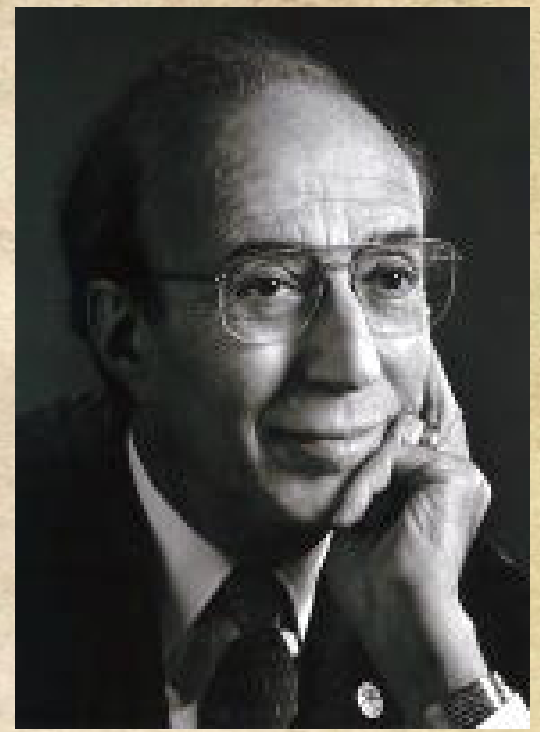
# *Chapter 1. Love's Labour Lost*

John von Neumann
(1903 -- 1957)



Stanislaw Ulam, Richard Feynman and John von Neumann





Herman Heine Goldstine
(1913 -- 2004)

*"... the whole atmosphere of our conversation changed from one of relaxed good humour to one more like the oral examination for the doctor's degree in mathematics."*

*--- Herman Goldstine, Biography*

# PLANNING AND CODING OF PROBLEMS
# FOR AN
# ELECTRONIC COMPUTING INSTRUMENT

BY

Herman H. Goldstine John von Neumann

Report on the Mathematical and Logical aspects of an
Electronic Computing Instrument
Part II, Volume I

Institute for Advanced Study
Princeton, New Jersey
1947

FIGURE 7.1

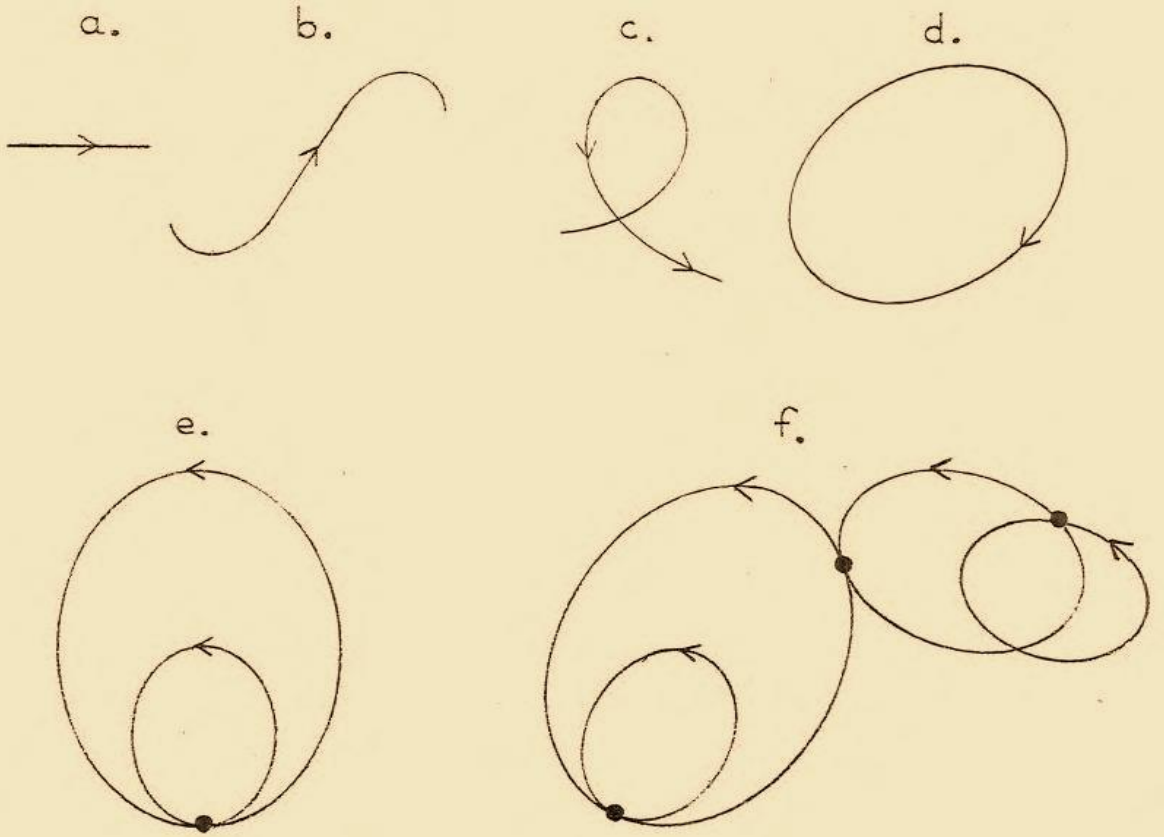
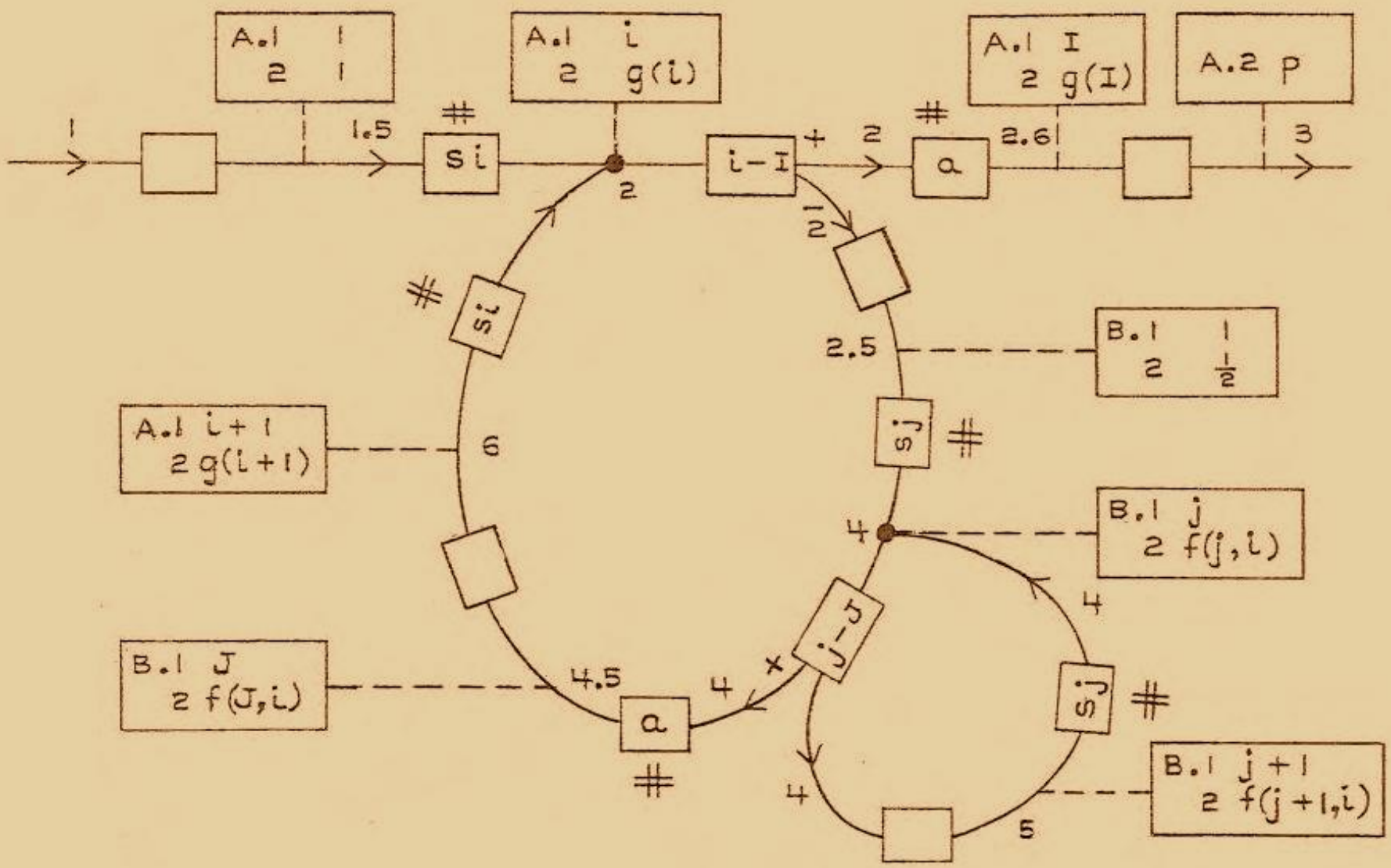
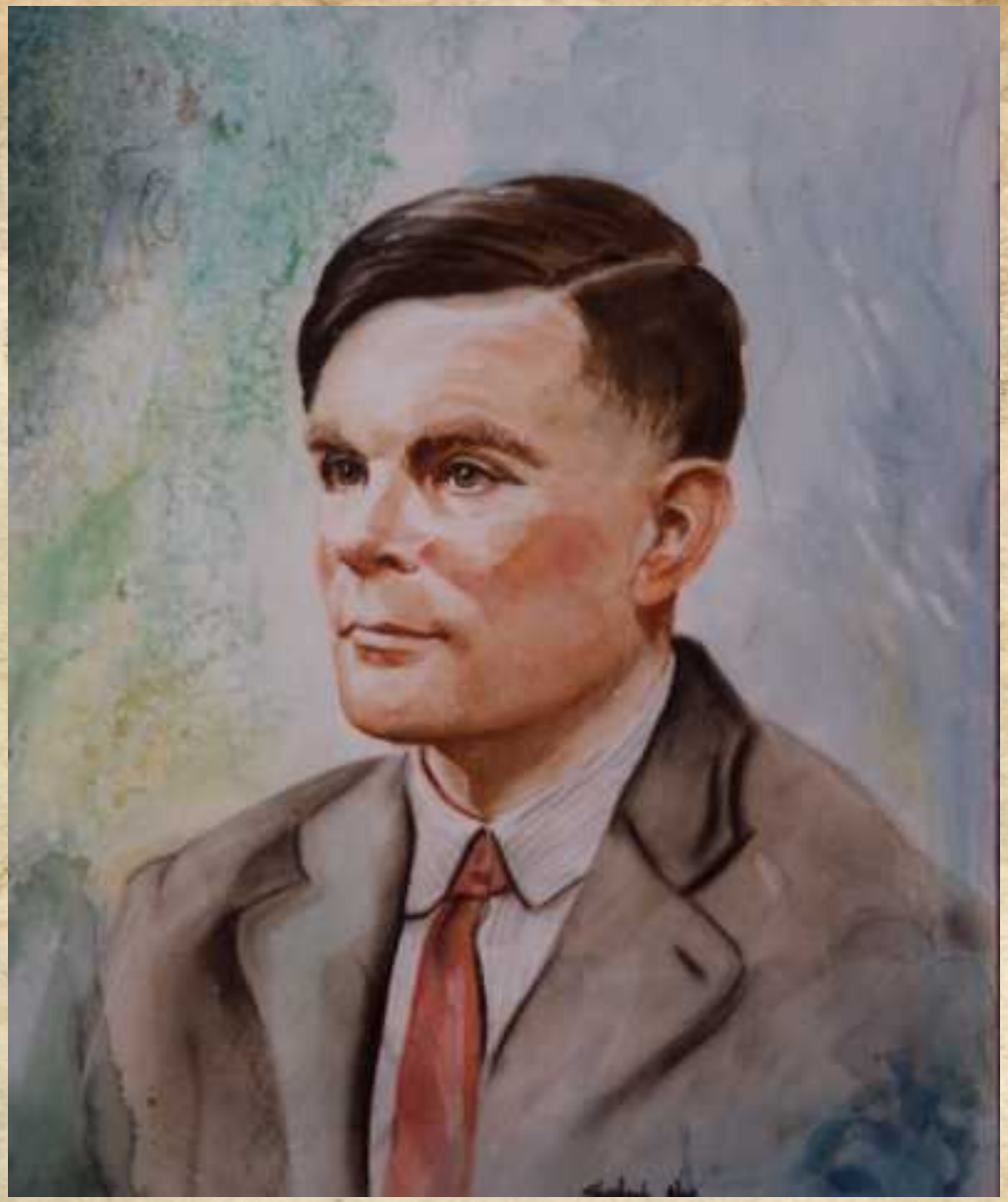


FIGURE 7.9


A.1 1
2 1
A.1 i
2 g(i)
A.1 I
2 g(I)
A.2 p
1
1.5
si
2
i-I
2
a
2.6
3
si
2.5
B.1 1
2 1/2
A.1 i+1
2 g(i+1)
6
sj
4
B.1 j
2 f(j,i)
4
j-J
x
4
B.1 J
2 f(J,i)
4.5
a
4
sj
B.1 j+1
2 f(j+1,i)
4
5

*Third:* The interval in question is immediately preceded by an assertion box: It must be demonstrable, that the expression of the field is, by virtue of the relations that are validated by this assertion box, equal to the expression which is valid in the field of the same storage position at the constancy interval immediately

preceding this assertion box. If this demonstration is not completely obvious, then it is desirable to give indications as to its nature: The main stages of the proof may be included as assertions in the assertion box, or some reference to the place where the proof can be found may be made either in the assertion box or in the field under consideration.



Alan Mathison Turing
(1912 -- 1954)

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.

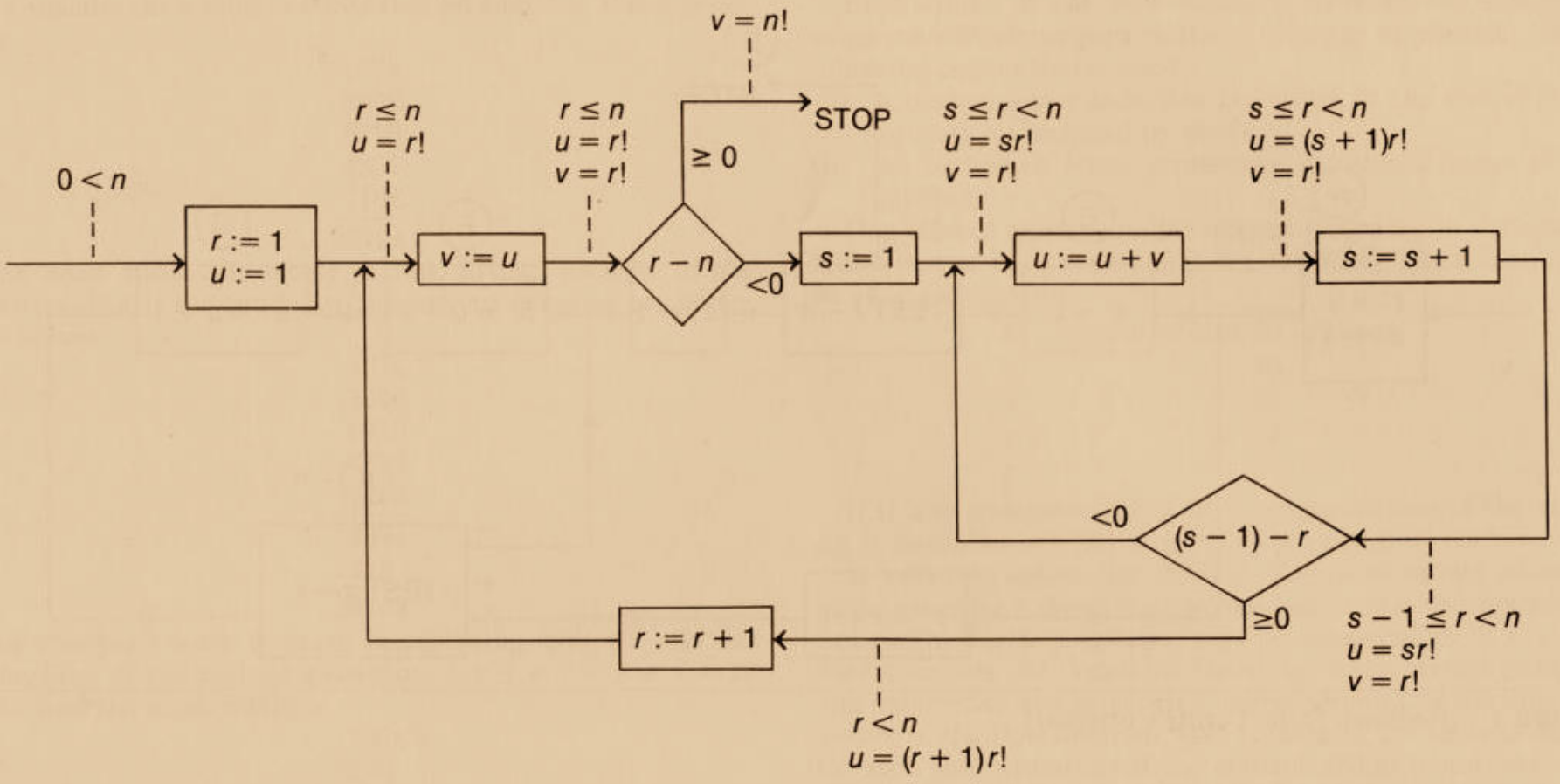
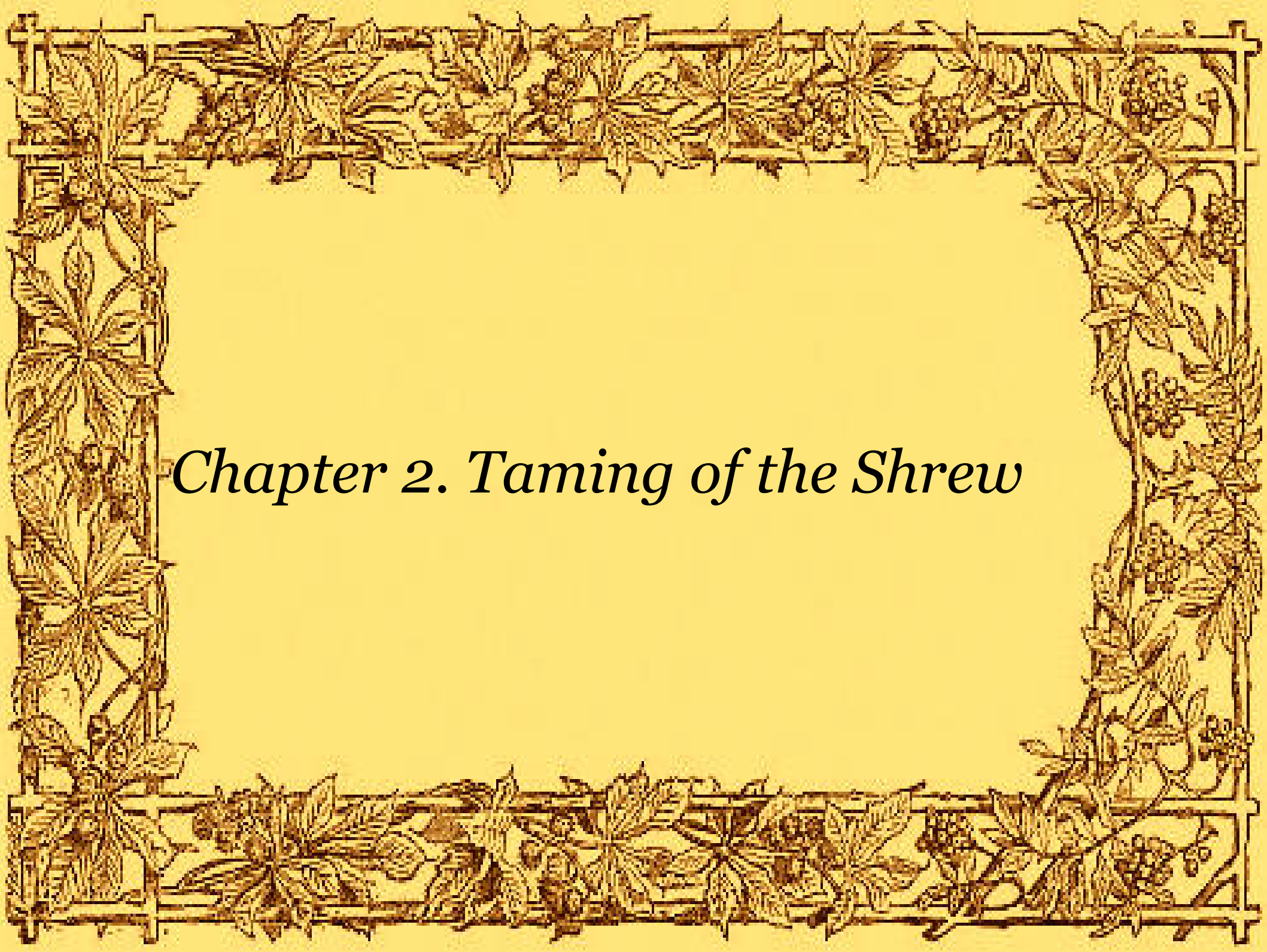



**Figure A**

*"The checker has to verify that the initial condition and the final condition agree with the claims that are made for the routine as a whole... He also has to verify that each assertion in the flow diagram is correct.* ***In doing this the [assertions] may be taken in any order and quite independently.*** *Finally the checker also has to verify that the process comes to an end."*

*"People would write code and make test runs, then find bugs and make patches, then find more bugs and make more patches, and so on until not being able to discover any further errors, yet always living in dread for fear that a new case would turn up on the next day and lead to a new type of failure.* ***We never realized that there might be a way to construct a rigorous proof of validity;****... The early treatises of Goldstine and von Neumann, which provided a glimpse of mathematical program development, had long been forgotten."*

--- Donald Knuth on programming in the 60s



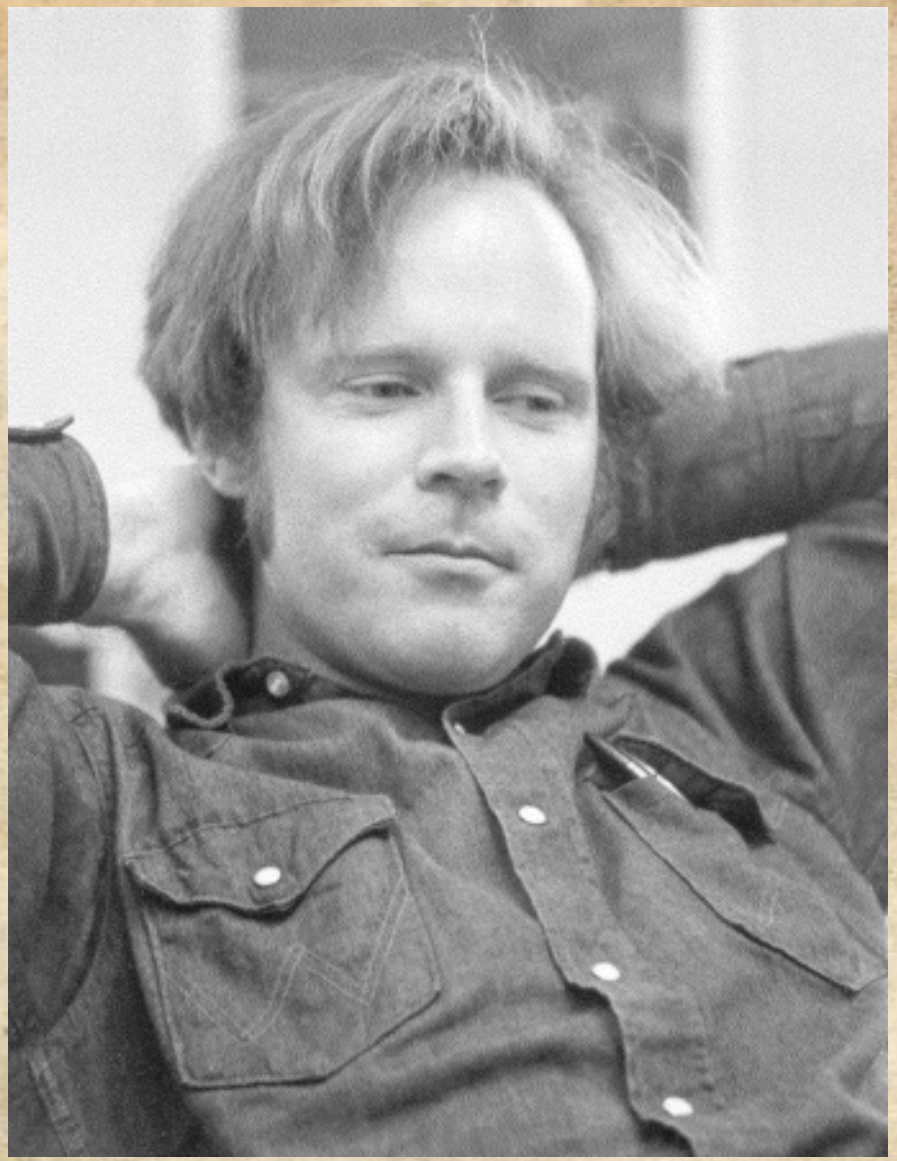
# *Chapter 2. Taming of the Shrew*

*"It is reasonable to hope that the relationship between computation and mathematical logic will be as fruitful in the next century as that between analysis and physics in the last. The development of this relationship demands a concern for both applications and for mathematical elegance."*

-- A Basis for a Mathematical Theory for Computation, 1963

*"Primarily, we would like to be able to prove that given procedures solve given problems… Instead of debugging a program, one should prove that it meets its specifications, and this proof should be checked by a computer program. For this to be possible, formal systems are required in which it is easy to write proofs."*

-- Towards a Mathematical Science of Computation, Proc. of Information Processing 1962



Robert W Floyd
(1936 -- 2001)

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

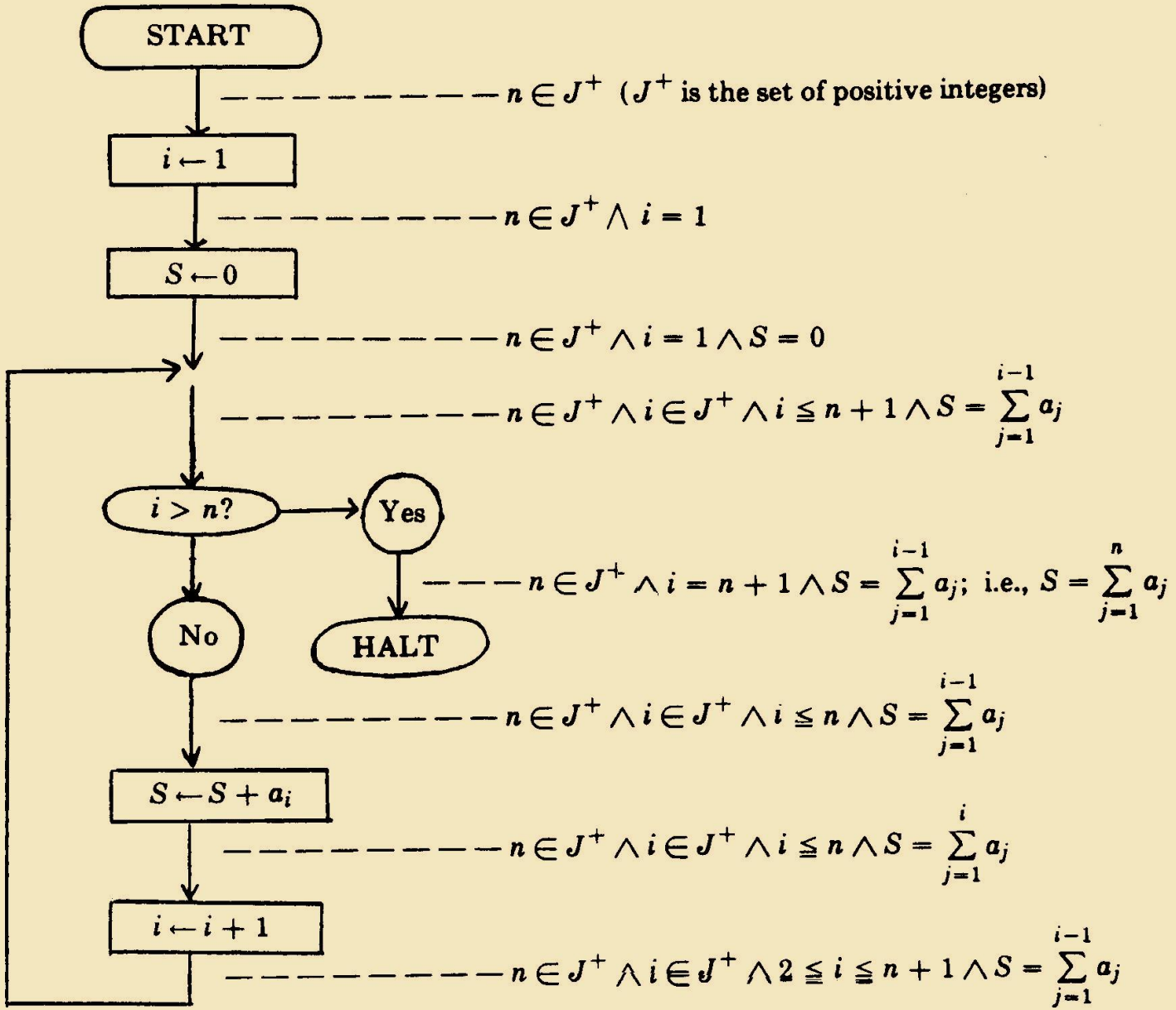
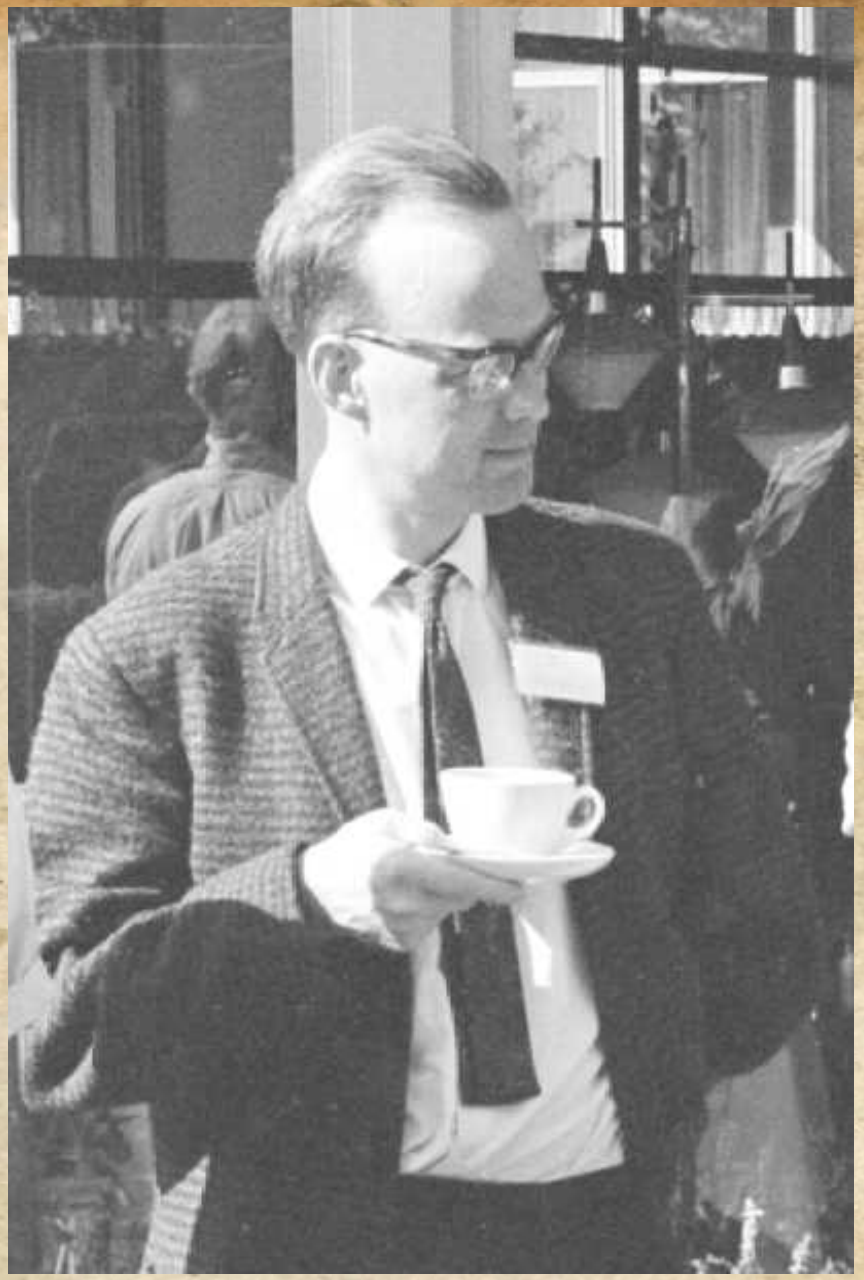



FIGURE 1. Flowchart of program to compute $S = \sum_{j=1}^{n} a_j$ $(n \geqq 0)$



Charles Anthony Richard Hoare
(1934 --  )

# The Axiomatic Basis of Computer Programming


## Summary

This paper attempts 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.


## 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 underly 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 confines itself to a very simple language, which is effectively a subset of all current procedure-oriented languages, and yet is theoretically as powerful as any of them, in the sense of being usable to program any computable function.

# Hoare logic

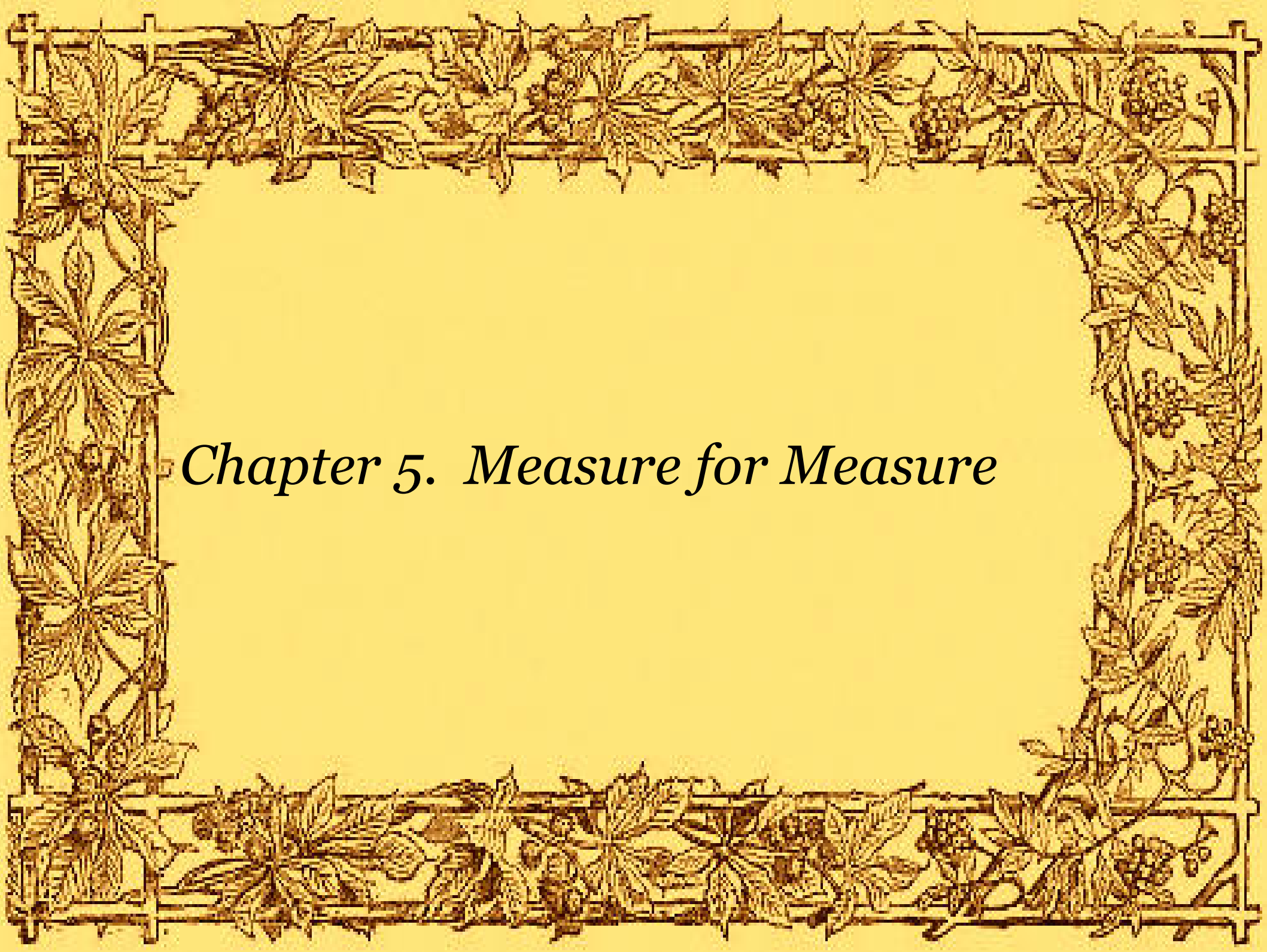
- $\{P[\mathtt{x} := \mathtt{e}]\}\ \mathtt{x{:=}e}\ \{P\}$ assignment axiom (1)

- $\dfrac{\{P\}C_1\{R\},\quad \{R\}C_2\{Q\}}{\{P\}C_1;C_2\{Q\}}$ composition rule (2)

- $\dfrac{\{P \wedge b\}C_1\{Q\},\quad \{P \wedge \neg b\}C_2\{Q\}}{\{P\}\ \mathtt{if}\ b\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2\ \mathtt{fi}\ \{Q\}}$ if-the-else rule (3)

- $\dfrac{\{P \wedge b\}C\{P\}}{\{P\}\ \mathtt{while}\ b\ \mathtt{do}\ C\ \mathtt{od}\ \{P \wedge \neg b\}}$ while rule (4)

- $\dfrac{(P \Longrightarrow P'),\quad \{P'\}C\{Q'\},\quad (Q' \Longrightarrow Q)}{\{P\}C\{Q\}}$ consequence rule (5)



# *Chapter 5. Measure for Measure*

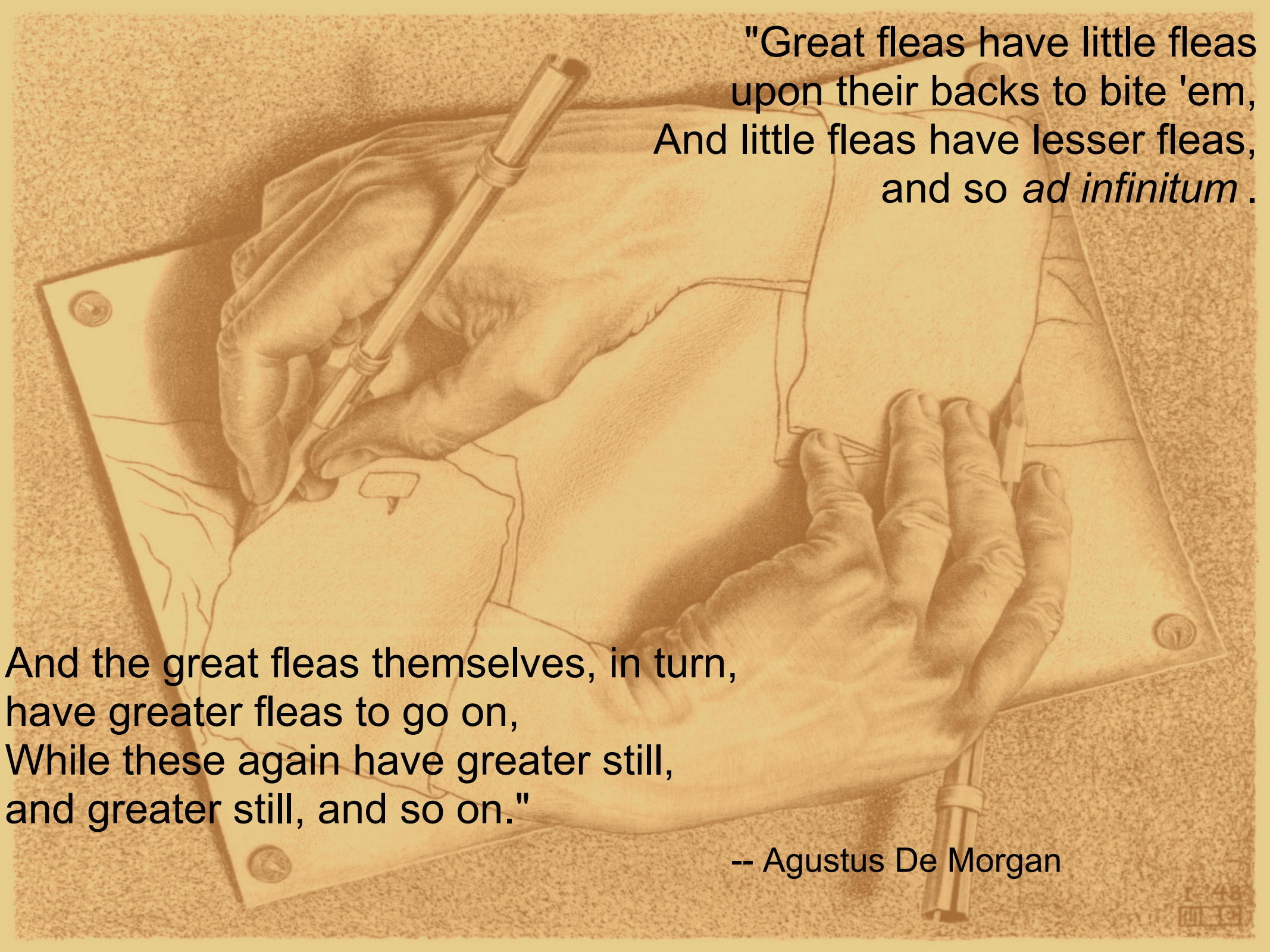
*"There has been nothing else in logic remotely comparable."*

-- Anil Nerode, on the 1957 Summer Institute on Symbolic Logic

*"Rosser gave a talk .. on Turing machines and actual computers; Church .. on logical synthesis of switching circuits for computer hardware, Abraham Robinson spoke on theorem proving. Among the younger contributors Michael Rabin and Dana Scott spoke about finite automata, Martin Davis talked about his implementation of [Presburger arithmetic]"*

-- Solomon Feferman, *Alfred Tarski and a watershed meeting in logic*


Alfred Tarski (nee Taijtelbaum)
(1901 -- 1983)


"Great fleas have little fleas
upon their backs to bite 'em,
And little fleas have lesser fleas,
and so *ad infinitum* .
And the great fleas themselves, in turn,
have greater fleas to go on,
While these again have greater still,
and greater still, and so on."
-- Agustus De Morgan

Dana Stewart Scott
(1932 --  )

*"There was a story that went the rounds. If Church said it's obvious, then everybody saw it a half hour ago. If Weyl says it's obvious, von Neumann can prove it. If Lefschetz says it's obvious, it's false."*

--- Barkley Rosser and Stephen Cole Kleene

Christopher Strachey
(1916 -- 1975)

*"Honey Dear,*

*My sympathetic affection  beautifully attracts your affectionate enthusiasm. You are my loving adoration: my breathless adoration. My fellow feeling breathlessly hopes for your dear eagerness. My lovesick adoration cherishes your avid ardour.*

*Yours Wistfully,*
*[Manchester University Computer]*

*"*

--- Strachey, Encounter Magazine, 1954

# OUTLINE OF A MATHEMATICAL THEORY OF COMPUTATION

by

Dana Scott
Princeton University

The motivation for trying to formulate a mathematical theory of computation is to give mathematical semantics for high-level computer languages. The word "mathematical" is to be contrasted in this context with some such term as "operational." Thus the mathematical meaning of a procedure ought to be the function from elements of the data type of the input variables to elements of the data type of the output. On the other hand, the operational meaning will generally provide a trace of the whole history of the computation following the sequencing stipulated in the stated procedure definition and will involve an explicit finitary choice of representations of data -- eventually in something close to bit patterns. The point is that, mathematically speaking, functions are independent of their means of computation and hence are "simpler" than the explicitly generated, step-by-step evolved sequences of operations on representations. In giving precise definitions of operational semantics there are always to be made more or less arbitrary choices of schemes for cataloging partial results and the links between phases of the calculation (cf. the formal definitions of such languages as PL/I and ALGOL 68), and to a great extent these choices are irrelevant for a true "understanding" of a program. Mathematical semantics tries to avoid these irrelevancies and should be more suitable to a study of such problems as the equivalence of programs.

It is all very well to aim for a more "abstract" and a "cleaner" approach to semantics, but if the plan is to be any good, the operational aspects cannot be completely ignored. The reason is obvious: in the end the program still must be run on a machine -- a machine which does not possess the benefit of "abstract" human understanding, a machine that must operate with finite configurations. Therefore, a mathematical semantics, which will represent the first major segment of the complete, rigourous definition of a programming language, must lead naturally to an operational simulation of the abstract entities, which -- if done properly -- will establish the practicality of the language, and which is necessary for a full presentation.

Thinking only of functions for the moment, it is clear that a mathematically defined function can be known to be computable without its being quite obvious how to compute the function in a practical sense -- just as it is possible to know that an infinite series is convergent without having a clear idea of its sum. Even though the abstract definition of the function is sufficient to determine it, we cannot really say that the function is known until the algorithm is revealed. (Even then our knowledge is somewhat "indirect" or "potential," but never mind.) The conclusion is, then, that an adequate theory of computation must not only provide the abstractions (what is computable) but also their "physical" realizations (how to compute them.)

What is new in the present theory is exactly these abstractions; whereas the means of realization, the techniques of implementation, have been known for some time, as the many, highly complex compilers that are presently in operation demonstrate. Of course, new concepts may require (or suggest) new methods of implementation, but that remains to be seen. However, notice this essential point: unless there is a prior, generally accepted mathematical definition of a language at hand, who is to say whether a proposed implementation is correct? Now it is often suggested that the meaning of the language resides in one particular compiler for it. But that idea is wrong: the "same" language can have many "different" compilers. The person who wrote one of these compilers obviously had a (hopefully) clear understanding of the language to guide him, and it is the purpose of mathematical semantics to make this understanding "visible." This visibility is to be achieved by abstracting the central ideas into mathematical entities, which can then be "manipulated" in the familiar mathematical manner. Even if the compiler-oriented approach (even compiled to run on an "abstract" machine) were transparent -- which it is not --

Edmund Melson Clarke
(1945 --  )

# PROGRAM INVARIANTS AS FIXED POINTS

(Preliminary Report)

Edmund Melson Clarke, Jr.
Department of Computer Science
Duke University
Durham, N. C. 27706

Abstract

We argue that relative soundness and completeness theorems for Floyd-Hoare Axiom Systems([6], [5], [18]) are really fixed point theorems. We give a characterization of program invariants as fixed points of functionals which may be obtained in a natural manner from the text of a program. We show that within the framework of this fixed point theory, relative soundness and completeness results have a particularly simple interpretation. Completeness of a Floyd-Hoare Axiom system is equivalent to the existence of a fixed point for an appropriate functional, and soundness follows from the maximality of this fixed point. The functionals associated with regular procedure declarations are similar to predicate transformers of Dijkstra; for non-regular recursions it is necessary to use a generalization of the predicate transformer concept which we call a relational transformer.

cedure parameters under various restrictions on scope of variables), and Cherniavsky ([1], loop languages). Incompleteness results for language features, such as call-by-name parameter passing, are given in Clarke [2] and in Lipton and Snyder [17].

1.2 New Results of this Paper. Completeness results appear to be an important tool in investigations of program correctness; however, many basic open questions remain about the derivation and interpretation of such results. Although proofs of soundness and completeness are often long and tedious, it seems that the underlying ideas are quite simple. Are there general theorems from which many different completeness results may be obtained as special cases? What is the relationship between Cook's definition of completeness and the definition of completeness used in the earlier work of Manna [18] and deBakker and Meertens [5]? Gorelick [13], for example, has shown that a set of three axioms gives completeness for a language with parameterless

# *Chapter 7. The Tempest*

# Rice's Theorem [1953]

A property is trivial if it is satisfied by all programs or by no programs.

Theorem: *There exists no algorithm for checking any non-trivial property of a Turing-complete programming language.*

Consequence: Our efforts are doomed!

# Programming Language Constructs for Which It Is Impossible To Obtain Good Hoare Axiom Systems

EDMUND MELSON CLARKE JR.

*Duke University, Durham, North Carolina*

ABSTRACT Hoare axiom systems for establishing partial correctness of programs may fail to be complete because of (a) incompleteness of the assertion language relative to the underlying interpretation or (b) inability of the assertion language to express the invariants of loops Cook has shown that if there is a complete proof system for the assertion language (i e all true formulas of the assertion language) and if the assertion language satisfies a natural *expressibility* condition then a sound and complete axiom system for a large subset of Algol may be devised We exhibit programming language constructs for which it is impossible to obtain sound and complete sets of Hoare axioms *even* in this special sense of Cook's These constructs include (i) recursive procedures with procedure parameters in a programming language which uses static scope of identifiers and (ii) coroutines in a language which allows parameterless recursive procedures Modifications of these constructs for which sound and complete systems of axioms may be obtained are also discussed

KEY WORDS AND PHRASES Hoare axioms, soundness, relative completeness, procedure parameters, coroutines

CR CATEGORIES 4 29, 5 24, 5 27

# Social Processes and Proofs of Theorems and Programs

Richard A. De Millo
Georgia Institute of Technology

Richard J. Lipton and Alan J. Perlis
Yale University

**It is argued that formal verifications of programs, no matter how obtained, will not play the same key role in the development of computer science and software engineering as proofs do in mathematics. Furthermore the absence of continuity, the inevitability of change, and the complexity of specification of significantly many real programs make the formal verification process difficult to justify and manage. It is felt that ease of formal verification should not dominate program language design.**

**Key Words and Phrases: formal mathematics, mathematical proofs, program verification, program**

> I should like to ask the same question that Descartes asked. You are proposing to give a precise definition of logical correctness which is to be the same as my vague intuitive feeling for logical correctness. How do you intend to show that they are the same? ... The average mathematician should not forget that intuition is the final authority.
>
> J. Barkley Rosser

Many people have argued that computer programming should strive to become more like mathematics. Maybe so, but not in the way they seem to think. The aim of program verification, an attempt to make pro-

# PROGRAM VERIFICATION: THE VERY IDEA

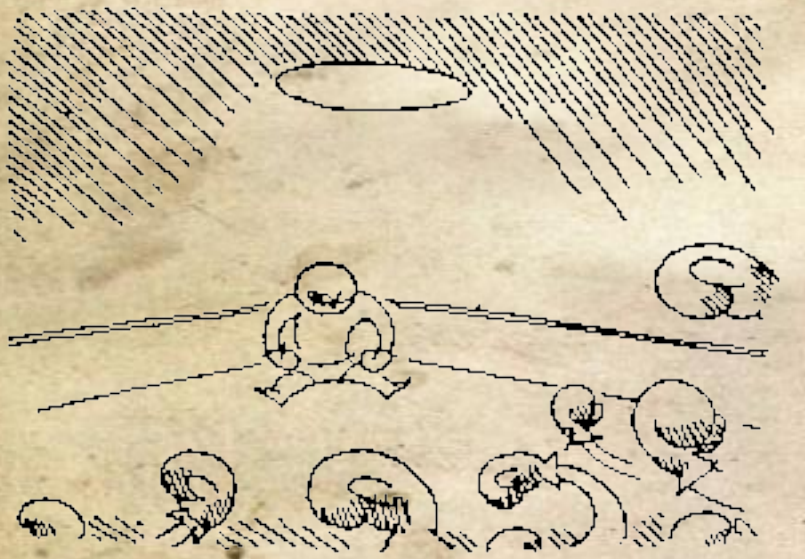
*The notion of program verification appears to trade upon an equivocation. Algorithms, as logical structures, are appropriate subjects for deductive verification. Programs, as causal models of those structures, are not. The success of program verification as a generally applicable and completely reliable method for guaranteeing program performance is not even a theoretical possibility.*

JAMES H. FETZER

*"Nobody is going to run into a friend's office with a program verification. Nobody is going to sketch a verification out on a paper napkin... One can feel ones eyes glaze over at the very thought ,"* -- dM, L, P

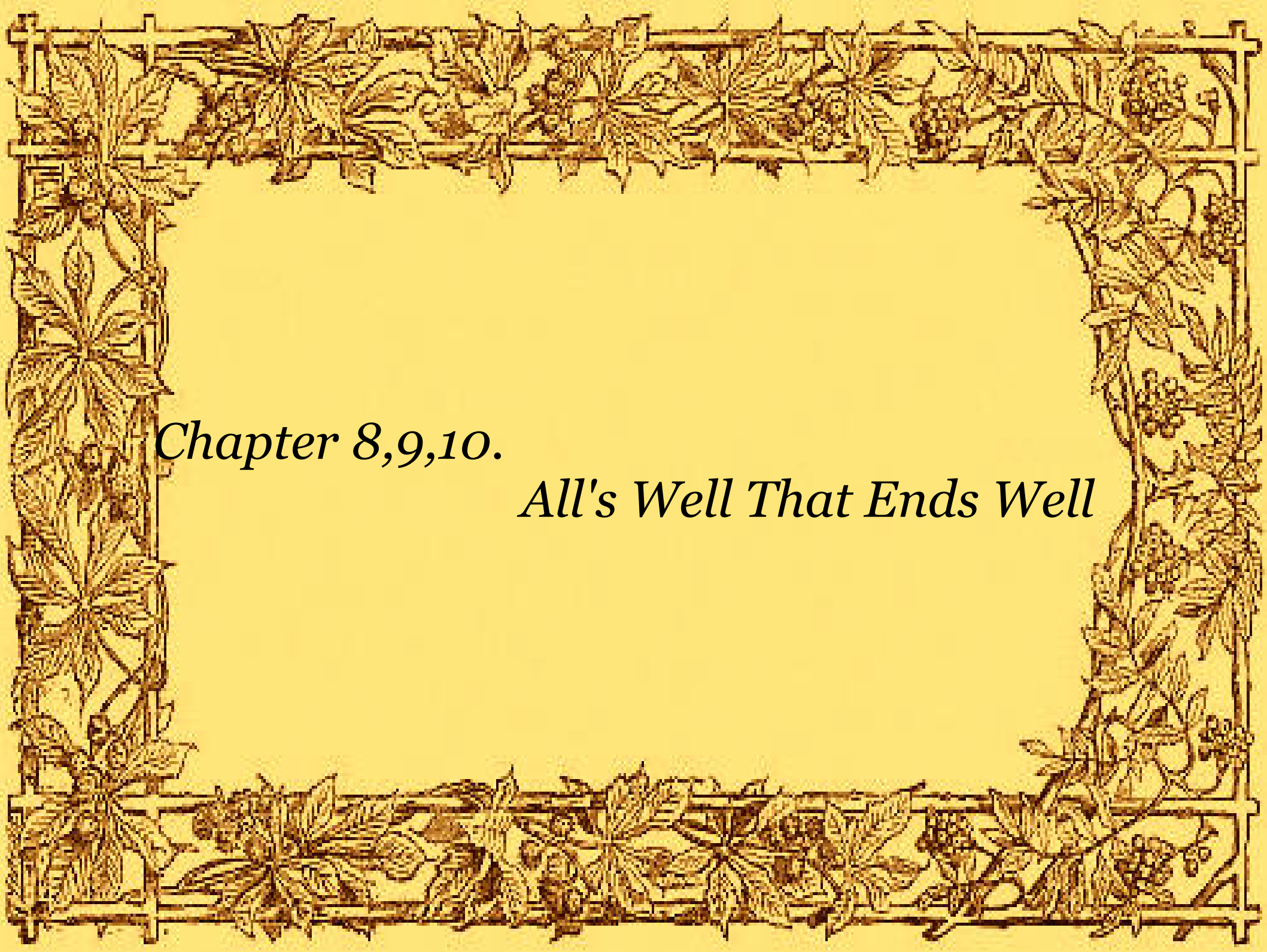
*"I don't believe that the correctness of a theorem is to be decided by a general election."* -- L. Lamport

*"Lamport and Maurer display an amazing inability to distinguish between algorithms and programs."* -- dM, L, P



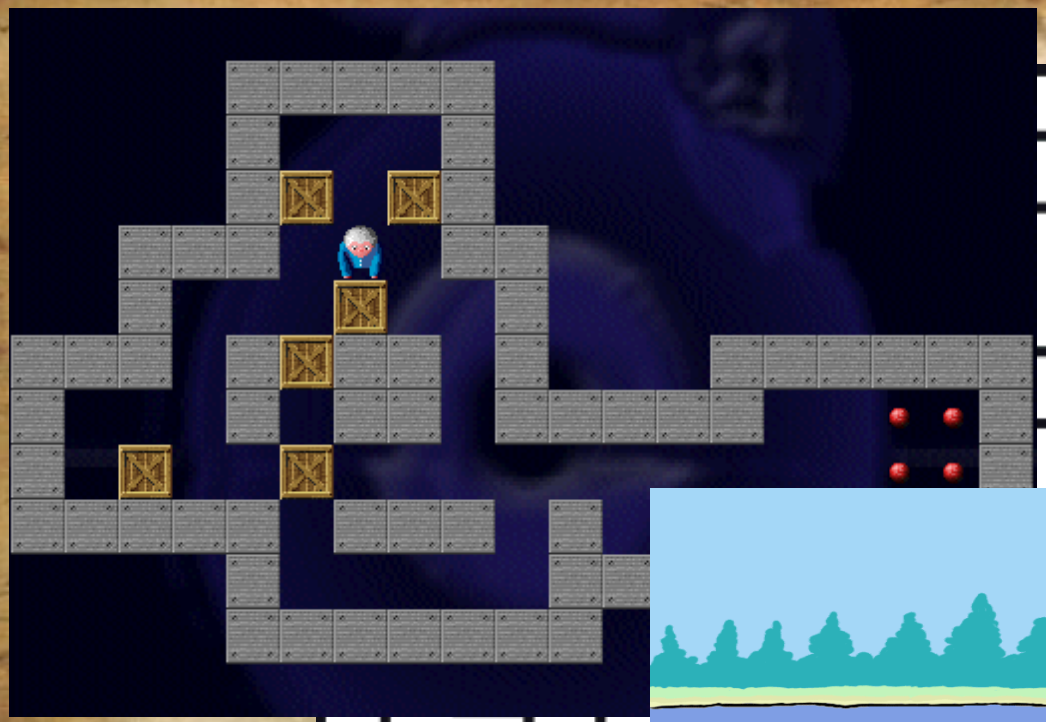
Problems worthy
of attack
prove their worth
by hitting back.

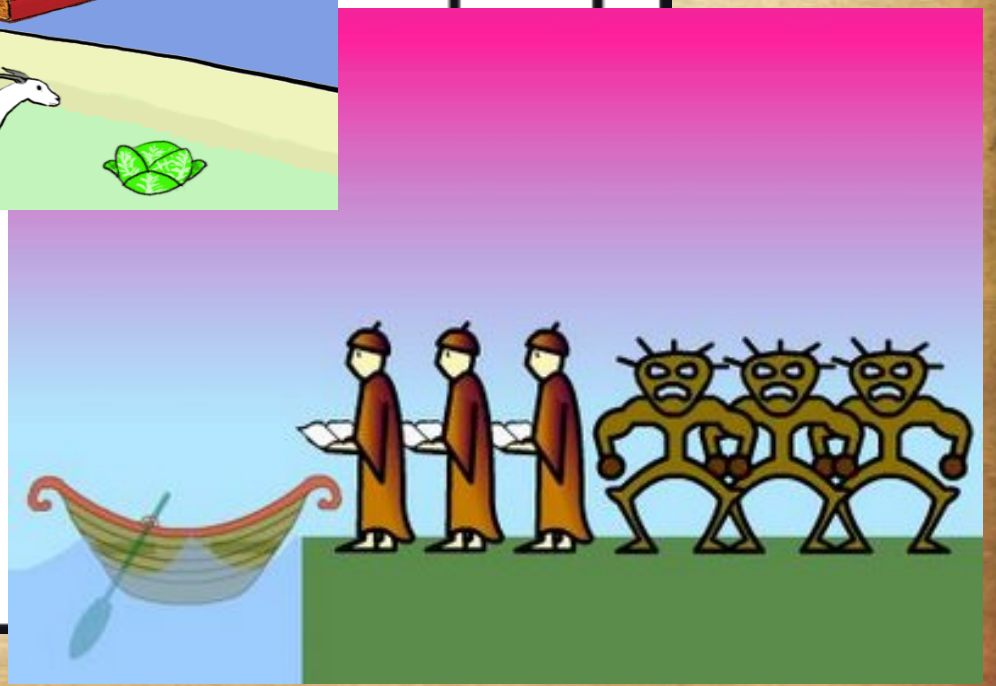
-- Piet Hein



# *Chapter 8,9,10.*

## *All's Well That Ends Well*



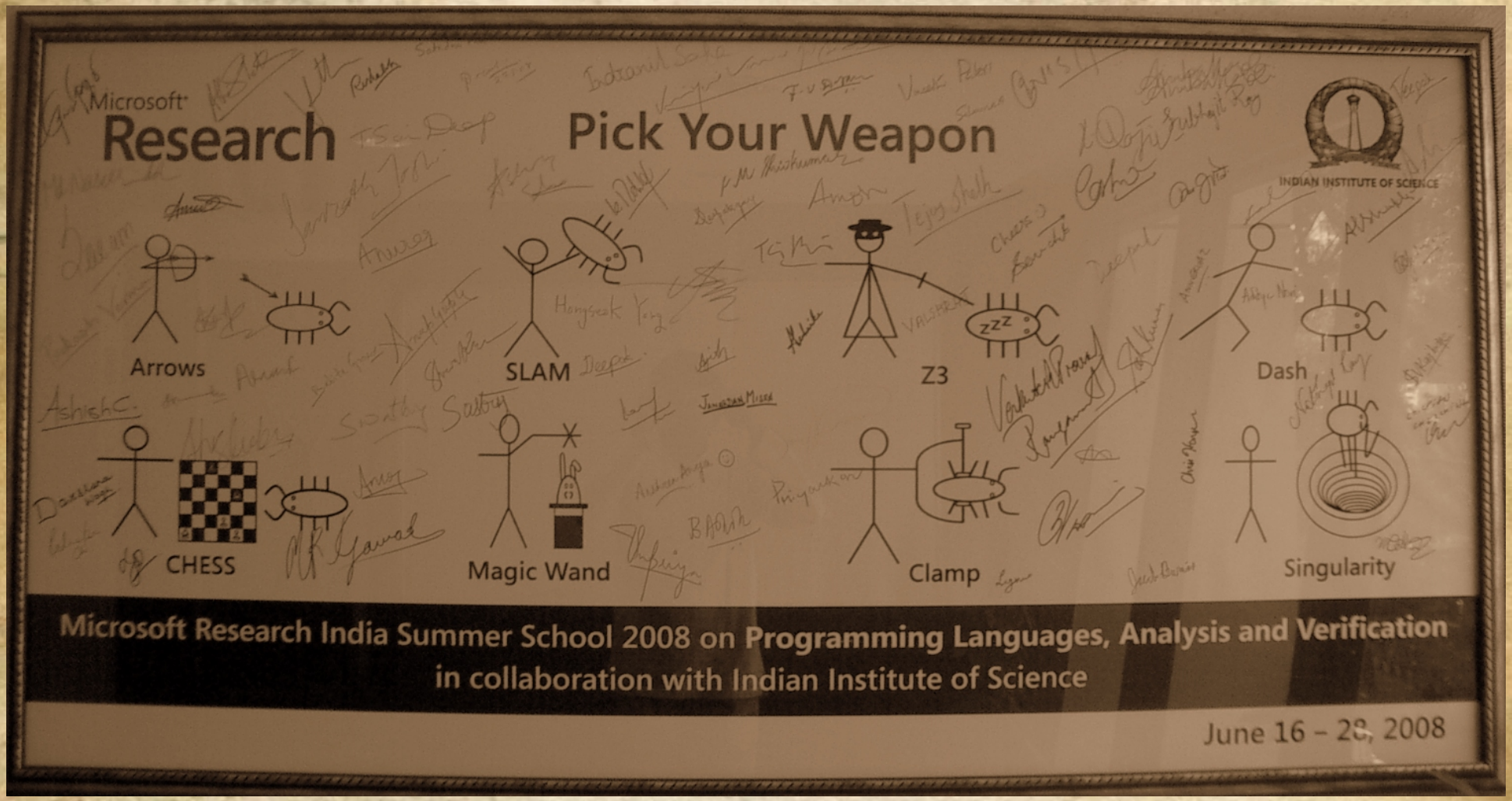


Patrick Cousot
(1948 --  )

# [Da Vinci, 1506-15] Does the Mona Lisa smile?

[Livingstone, 2004] Yes, but only in low spatial frequencies.

*Epilogue.  As You Like It!*


Microsoft Research
Pick Your Weapon
INDIAN INSTITUTE OF SCIENCE
Arrows
SLAM
Z3
Dash
CHESS
Magic Wand
Clamp
Singularity
Microsoft Research India Summer School 2008 on Programming Languages, Analysis and Verification
in collaboration with Indian Institute of Science
June 16 – 28, 2008