Verification Engineering: A Future Profession

Amir Pnueli
Weizmann Institute of Sciences

An A.M. Turing Award Lecture
PODC, San Diego, 23.8.97

1997 Celebrates the

20th anniversary of the introduction of the temporal logic approach to the specification and verification of reactive systems [Pnu77].

30th anniversary of formal verification [Floyd67], [Hoare69].

In this talk about FV (Formal Verification), we discuss the problem and the goals, but cannot indicate a single outstanding solution.

Perhaps this is the main message that, even with the impressive power and progress of FV, there exists no purely scientific solution to the system correctness problem.

We should concentrate on an engineering approach: assemblimg many tools and methods, among which FV should be a major player.

Formal Verification

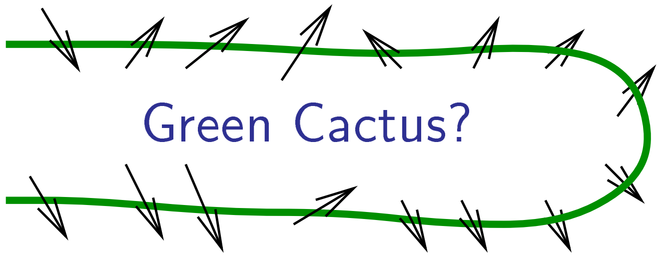
Started with sequential program verification which, so far, has not been universally embraced.

It then expanded into the are of reactive system verification, where it has a more visible impact and greater success. Why?

Distinguish between [HP85]

- Transformational systems (sequential): Run in order to produce a final result on termination. Can be modeled as a black box. Specified in terms of their Input/Output relations.

- Reactive systems, whose role is to maintain an ongoing interaction with their environment.



Such systems must be specified and verified in terms of their behaviors.

In Simpler Terms

We are all familiar with the equation

$$\text{Program} \quad = \quad \text{Data} \quad + \quad \text{Control}$$

The reactive view of systems emphasizes the control part.

Formal Verification began to flourish as soon as efforts were focused on the reactive view.

- More returns fro the same investment.
- A lot can be accomplished on the propositional/finite-state level.

The Essence of FV

- Consider two objects:

  S – A specification. The what of a system description.

  I – An implementation. The how of a system description.

- Establish that I conforms with S.

Considered conformance relations are:

Equivalence, various simulation relations, satisfaction (if S is expressed in a logic), implication (if both S and I are expressed logically), refinement, implementation-relations, etc.

Originally,

Formal verification was associated with the application of axiomatic or deductive techniques to proofs of correctness.

Things having to do with logic.

Since the early 80's [CE81], it also includes model-checking and other algorithmic approaches, which can be viewed as exhaustive simulation or exhaustive testing.

A first step towards engineerization of the field!

Two Styles

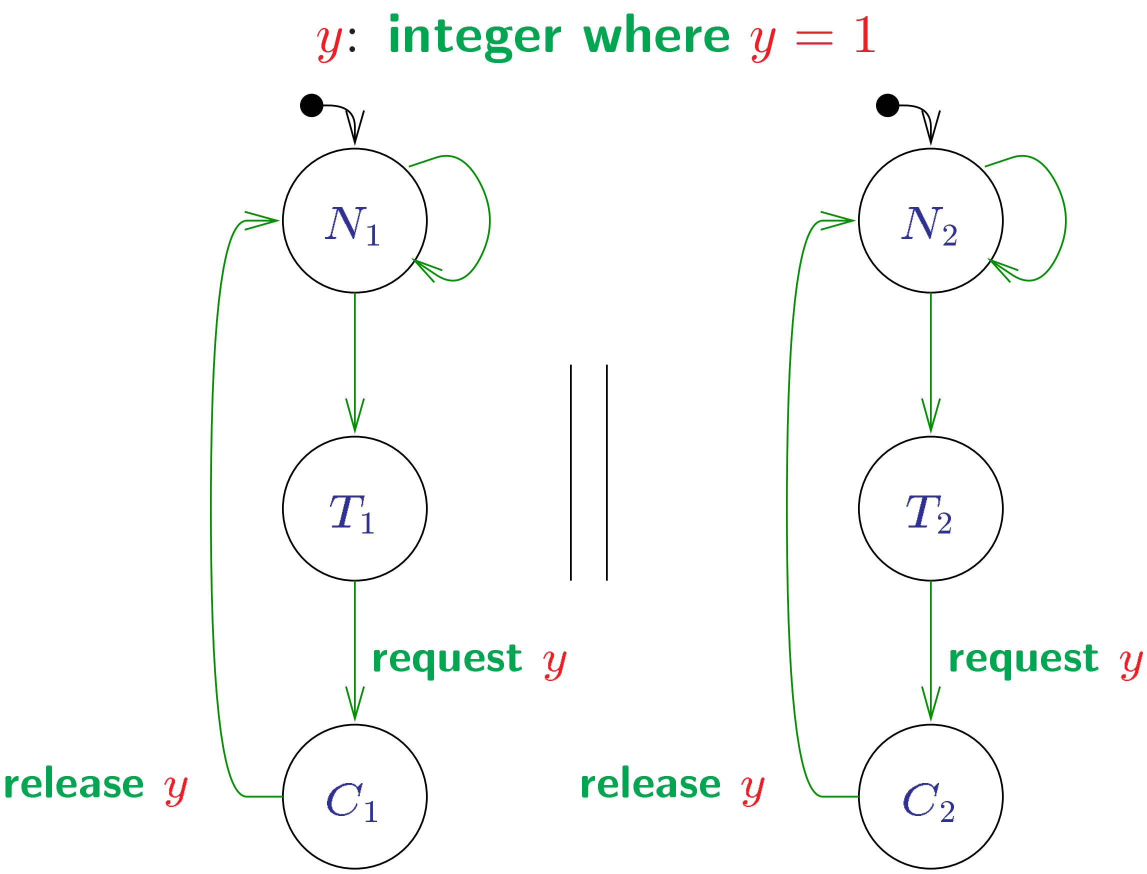
of specification, and corresponding verification have been proposed:

- Specification by property lists, and property verification. TL has been designed for this style of specification. — Requirement Specification.

- Specification by a reference model / high-level (abstract) program. Verification by simulation (refinement). — System Specification.

Example: Mutual Exclusion by Semaphores

Two processes coordinating access to their critical sections by Semaphores —



The semaphore instructions **request** y and **release** y stand for

$$\langle \textbf{await } y > 0\,;\; y := y - 1 \rangle \quad \text{and} \quad y := y + 1.$$

Specification of MUTEX by a Property List

- Safety:

$$\square\ \neg(C_1 \wedge C_2)$$

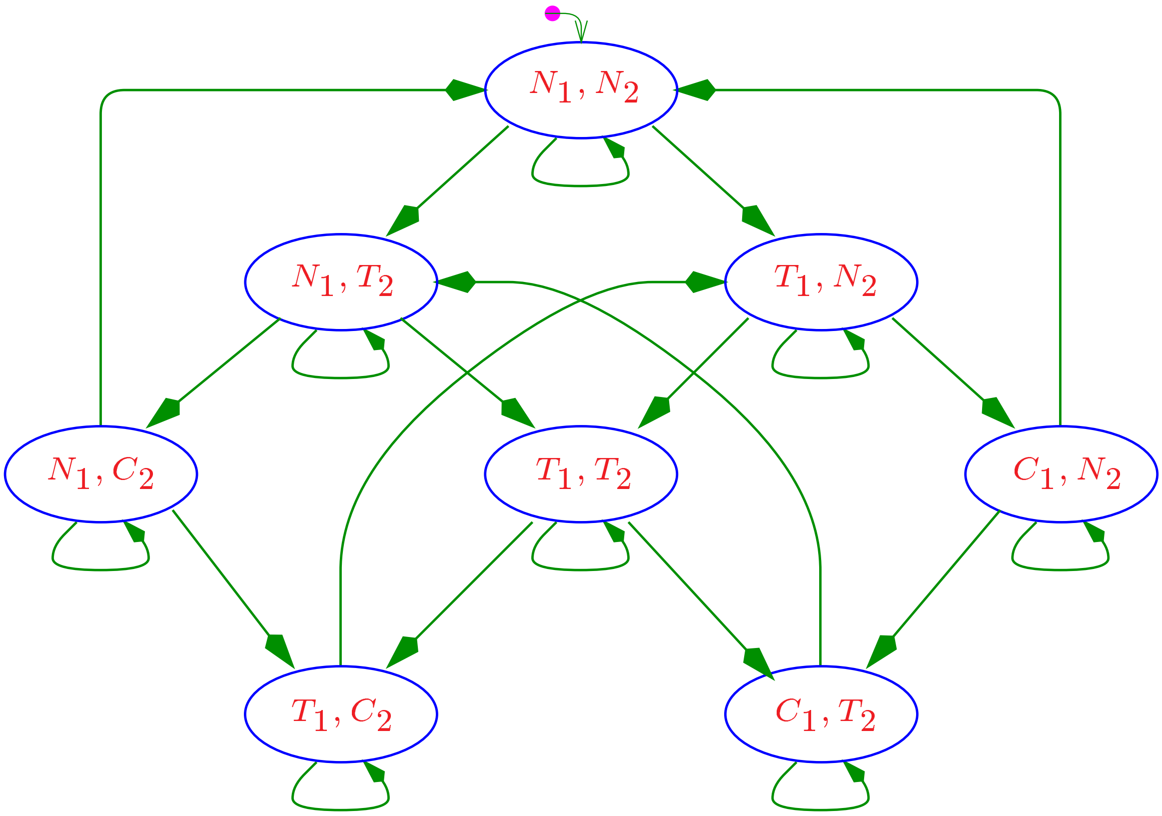
The two processes can never visit their respective critical sections at the same time.

- Liveness:

$$T_1 \Longrightarrow \diamondsuit\ C_1 \qquad T_2 \Longrightarrow \diamondsuit\ C_2$$

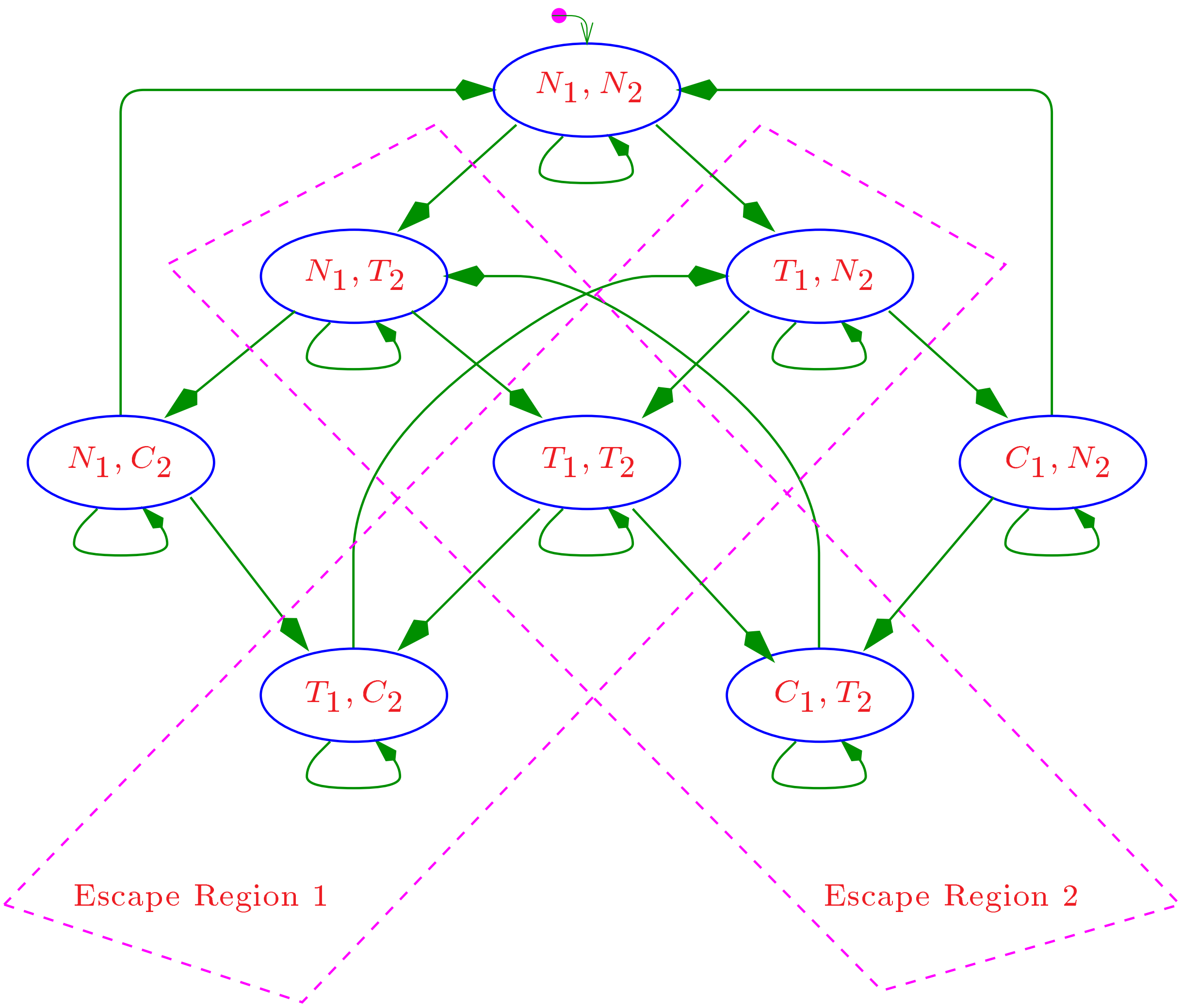
Every visit of a process to its trying section is followed by a visit to the critical section of the same process.

Specification by an Abstract Model



The absence of the state $\langle C_1, C_2 \rangle$ implies mutual exclusion.

Adding Liveness



Liveness imposed by the specification of escape regions and requiring that no computation remains contained forever in any escape region.

Is a Verified System Always Correct?

The prefix formal in formal verification implies, and is often advertised as providing absolute confidence in the correctness of the program/design/system.

This is far from precise, and has been the source for multiple attacks on the discipline.

Formal verification guarantees beyond any doubt the correct relation between the mathematical objects I and S.

However, it does not establish in unequivocal terms the correctness of the real implementation described by I.

Possible Reasons for the Disparity

There are two types of reasons why the implementation described by I may be incorrect even though it has been formally verified with repsect to S:

1. Discrepancies between the mathematical models and reality.

The model I may fail to capture seom essential features of the real implementation, such as bounded word length, or precise timing information.

The model S may make wrong (too strong) assumptions about the environment.

2. Inadequate specifications.

The specification S may fail to capture the intentions of the user/customer.

It can be incomplete (missing property) or unsound (prescribe wrong behavior).

Dealing with Unfaithfulness of Models

The problem of discrepancy between models and reality is not unique to computerized systems.

It is a problem commonly faced in all branches of engineering.

Follow the standard solutions:

- Develop a hierarchy of models, each more realistic than its predecessor. E.g.,

Qualitative Model of Reactive Systems Fair Transition Systems

$$\cap$$

Add real time to obtain Clocked Transition Systems

$$\cap$$

Add Continuous Behavior described by diff. equations to get Hybrid Systems

- Study approaches to error analysis and damage containment.

Dealing with Deviant Specifications

Validate the specification S!

Validation of a model M: Making sure that M is correct.

No available method (including FV) can guarantee correctness. At best they can increase confidence in the correctness of M.

Informal (and universally practiced) approaches to validation include: simulation, testing, prototyping, walk-throughs, static analysis, etc.

The formal approach to validation of M consists of:

- Formulate a specification S.
- Verify that M conforms with S.

How do we know that S is correct?

Dealing with Deviant Specifications

Validate the specification S!

Validation of a model M: Making sure that M is correct.

No available method (including FV) can guarantee correctness. At best they can increase confidence in the correctness of M.

Informal (and universally practiced) approaches to validation include: simulation, testing, prototyping, walk-throughs, static analysis, etc.

The formal approach to validation of M consists of:

- Formulate a specification S.
- Verify that M conforms with S.

How do we know that S is correct?