Amir Pnueli and the Dawn of Hybrid Systems

Oded Maler

CNRS - VERIMAG
Grenoble, France

2010

Preface

- **Amir Pnueli** (1941-2009) was one of the key figures in the verification of **discrete reactive systems**
- He received the **Turing Award** in 1996 for proposing **Temporal Logic** as a language for specifying acceptable behaviors of concurrent systems
- He participated in pioneering efforts to extend verification methodology to **timed** and **hybrid** systems
- He wrote one of the first CS papers on hybrid automata (**phase-transition systems**)
- He was among the founders of **HSCC** and a member of the steering committee (1998-2003)
- He was my **PhD advisor** (1986-1989) and a close collaborator and friend afterwards
- The talk is intended to tell about **himself**, his **work** and his contribution to **hybrid systems** research

Very Short Personal Notes

- Amir was a very nice person, universally **appreciated**, **admired** and **loved** by members of the communities he interacted with
- He earned his fame and status by the **quality** of his work and his **receptiveness** to **others** not because of being an **unbounded operator** (he was not)
- Unlike many of us he was **not** an ego-maniac (or managed to mask it perfectly by timidity, modesty and politeness)
- He knew **maths** of many sorts and did not shy away from **applications** and **software development**
- He had a broad **culture**, sense of **humor** and good appetite
- Much more on that will be said by many in the **Amir Pnueli Memorial Conference NYU, May 7-9, 2010** and other occasions related to his core community

Talk Overview

- **Part I: Verification for Dummies**
- A comprehensible introduction to the **context** of Amir's work, oriented toward outsiders (control persons, CS persons of other genres):
  - What is computer science?
  - What is verification?
  - What is temporal logic?
- **Part II: Historical Revisionism 101**:
- Sketch of (a personal projection of) the evolution of hybrid systems research in the CS side, roughly around (1988-1998):
  - Pre-history
  - Motivation
  - First models
  - Verification and controller synthesis
  - Temporal logic for continuous signals

What **Is** Computer Science ?

- Among other things computer science is: the (pure and applied) study of **discrete-event dynamical systems** (automata, transition systems)
- A natural point of view for the **reactive systems** parts of CS (hardware, protocols, real-time, stream processing)
- At least for people working on modeling and verification of such systems
- Sometimes obscured (intentionally or not) by **fancy formalisms**: Petri nets, process algebras, temporal logics..
- All honorable topics with intrinsic importance, beauty, etc.
- But sometimes should be distilled to their **essence** in order to make sense for potential users from other disciplines, rather than **intimidate** them

Digression: Greek vs. Egyptian/Babylonian Math

- Many computer scientists in the audience may not feel my definition faithfully describes their domain
- In many engineering disciplines there is a division between the amateurs of the **concrete** and the fans of the **abstract**
- In CS:
  - Concrete: systems, C or VHDL code, Makefiles,...
  - Abstract: automata, logic, semantics,...
- In control, EE, signal processing:
  - Concrete: Simulink blocks, filters, transistors, motors,...
  - Abstract: manifolds, differential equations, linear operators,...
- Abstract models are **not necessary nor sufficient** for building systems: software can be written in a programming language without thinking of the underlying abstract dynamical system
- But radically-new **insights** are easier to obtain in the abstract world

Dynamical System Models in General

- The following abstract features of dynamical systems are **common** to both **continuous** and **discrete** systems:
- **State variables** whose set of **valuations** determine the **state space**
- A **time domain** along which these values evolve
- A **dynamic law** which says **how** state variables evolve over time, possibly under the influence of **external** factors
- System **behaviors** are **progressions** of **states** in **time**
- Having such a model, knowing an initial state $x(0)$ one can **predict**, to some extent, the value of $x(t)$

Classical Dynamical Systems

- State variables: **real numbers** (location, velocity, energy, voltage, concentration)
- Time domain: the **real time axis** $\mathbb{R}$ or a discretization of it
- Dynamic law: **differential equations**

$$\dot{x} = f(x, u)$$

or their discrete-time approximations

$$x(t+1) = f(x(t), u(t))$$

- Behaviors: **trajectories** in the continuous state space
- Achievements: Apples, Stars, Missiles, Electricity, Heat, Chemical processes
- Theorems, Papers, Simulation tools

Automata as Dynamical Systems

- An **abstract discrete state space**, state variables need **not** have a numerical meaning
- A **logical time domain** defined by the **events** (order but not metric)
- Dynamics defined by **transition rules**: input event **a** takes the system from state **s** to state $\mathbf{s}'$
- Behaviors are **sequences** of **states** and/or **events**
- **Composition** of large systems from small ones, hierarchical structuring
- Different modes of **interaction**: synchronous/asynchronous, state-based/event-based
- Sometimes additional **syntax** may be required

Automata can Model many Phenomena and Devices

- Software, hardware,
- ATMs, user interfaces
- Administrative procedures
- Communication protocols
- Cooking recipes, Manufacturing instructions
- **Any** process that can be viewed as a **sequence** of steps

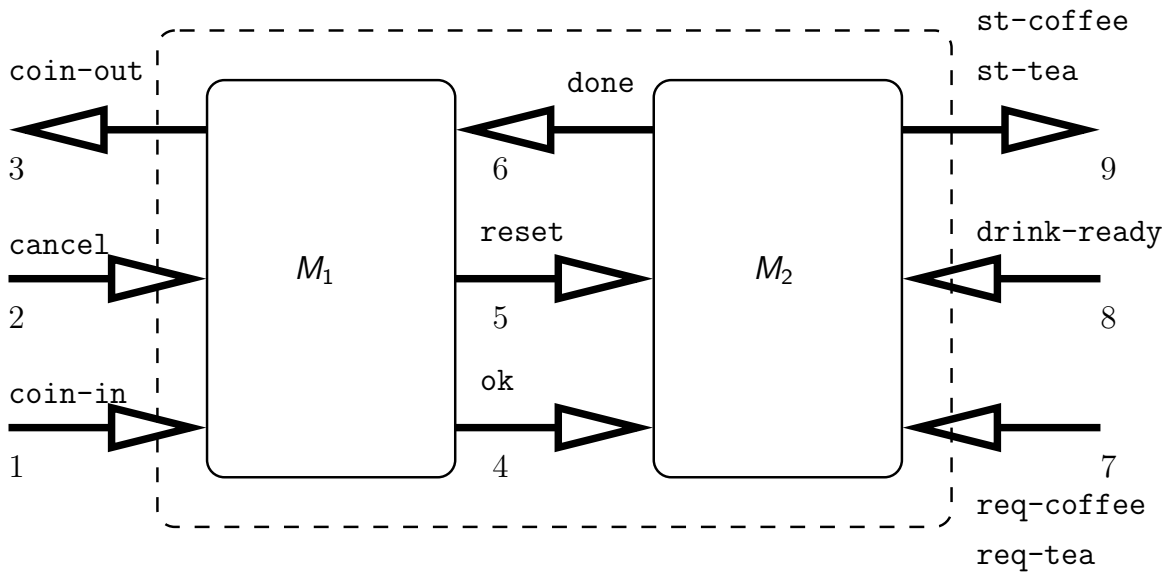
- But what can we **do** with these models?
- There are no easy-to-use **analytical tools** as in continuous systems
- We can **simulate** and sometimes do **formal verification**

What is Verification ?

- The generic question:
- Given a complex discrete dynamical system with some **uncontrolled inputs** or unknown parameters
- Check whether **all** its **behaviors** satisfy some properties
- Properties:
  - Never reach some part of the state space
  - Always come eventually to some (equilibrium) state
  - Never exhibit some pattern of behavior
  - Quantitative versions of such properties..
- Existing verification tools can do this type of analysis for huge systems by sophisticated graph algorithms

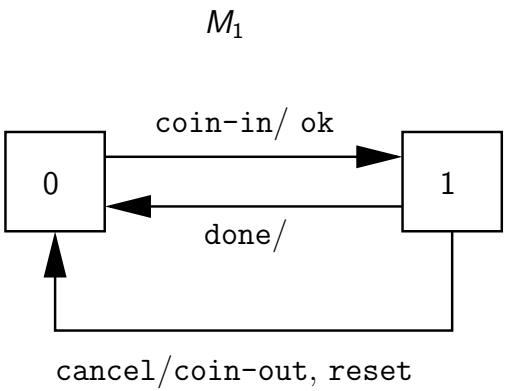
Illustration: The Coffee Machine

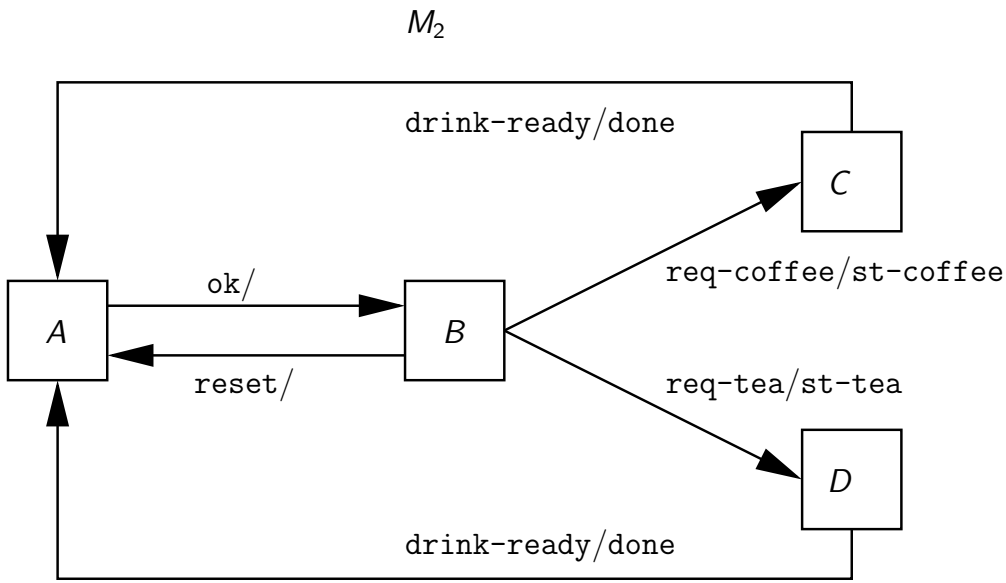
- Based on a first chapter of an unwritten book
- Consider a machine that takes **money** and distributes **drinks**
- The system is built from two **subsystems**, one that takes care of financial matters, and one which handles choice and preparation of drinks
- They communicate by sending messages



Automaton Models

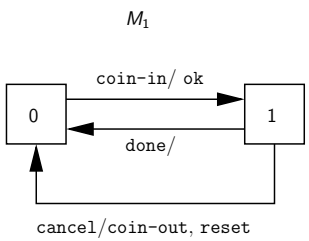
- The two systems are modeled as automata
- transitions are triggered by external events and events coming from the other subsystem

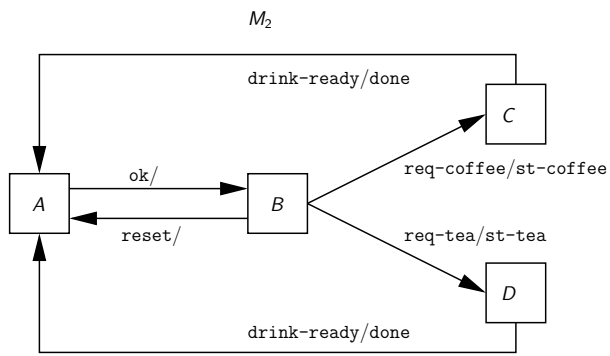


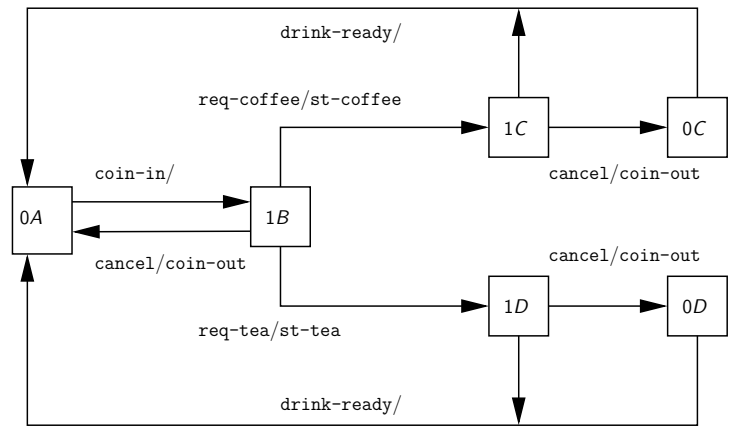


The Global Model

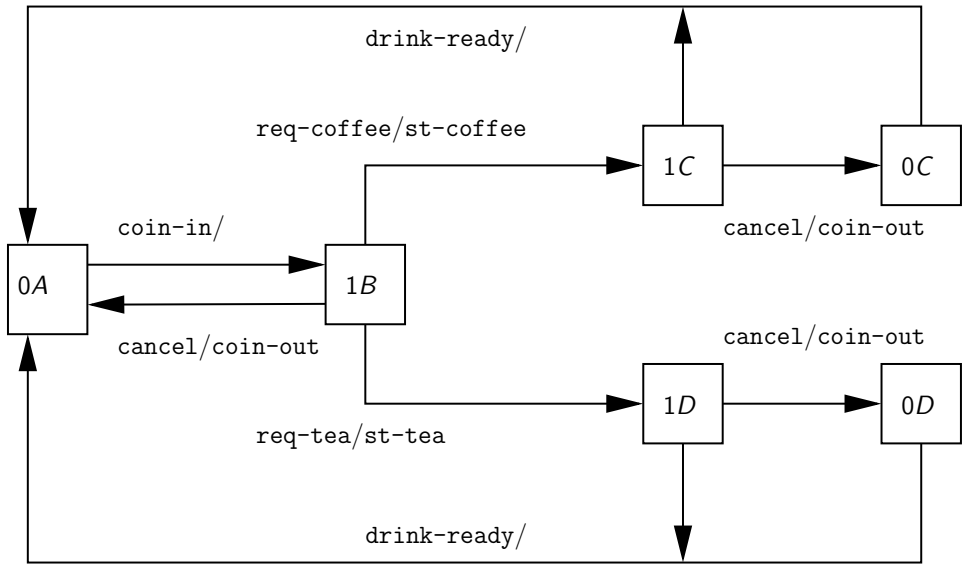
- The behavior of the whole system is captured by a **composition** (product) $M_1 \parallel M_2$ of the components
- States are elements of the **Cartesian product** of the respective sets of states, indicating the state of each component
- Some transitions are independent and some are **synchronized**, taken by the two components simultaneously
- Behaviors of the systems are **paths** in this transition graph







Normal Behaviors



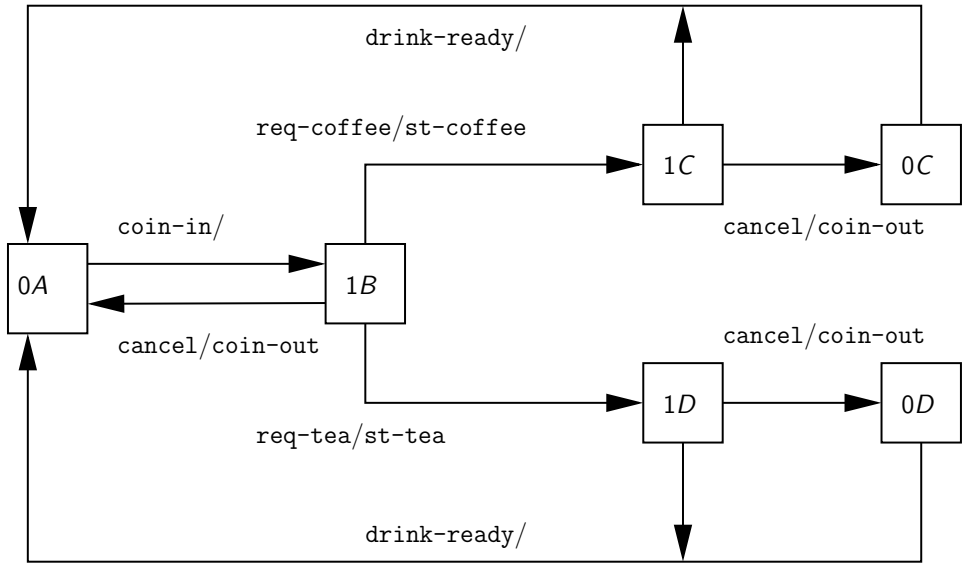
- Customer puts coin, then sees the bus arriving, cancels and gets the coin back

$$0A \texttt{ coin-in } 1B \texttt{ cancel coin-out } 0A$$

- Customer inserts coin, requests coffee, gets it and the systems returns to initial state

$$0A \texttt{ coin-in } 1B \texttt{ req-coffee st-coffee } 1C \texttt{ drink-ready } 0A$$

An Abnormal Behavior



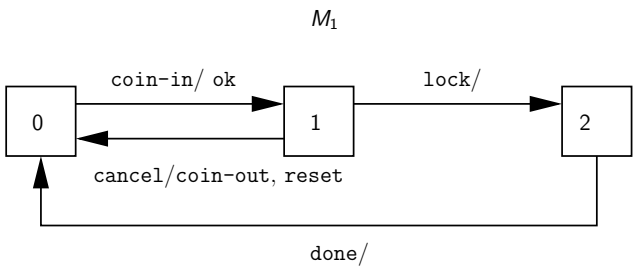
- Suppose the customer presses the cancel button *after* the coffee starts being prepared..

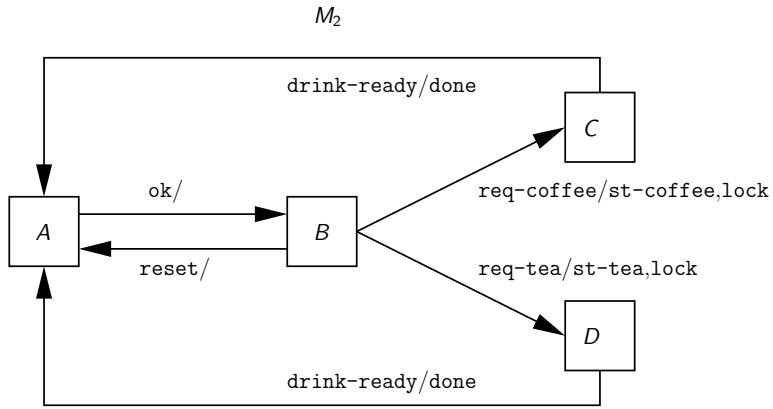
  $0A$ `coin-in` $1B$ `req-coffee` `st-coffee` $1C$ `cancel` `coin-out` $0C$ `drink-ready` $0A$

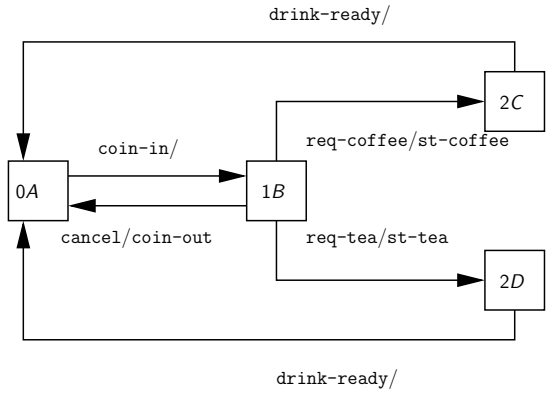
- Not an attractive feature for the owner of the machine

Fixing the Bug

- When M_2 starts preparing coffee it emits a `lock` signal
- When M_1 received this message it enters a new state where `cancel` is refused







The Moral of the Story

- Many complex systems can be modeled as a **composition** of **interacting automata**
- Behaviors of the system correspond to paths in the **global transition graph** of the system
- The size of this graph is **exponential** in the number of components (state explosion, curse of dimensionality)
- These paths are labeled by **input events** representing influences of the outside environment
- Each input sequence may generate a **different** behavior
- We want to make sure that a system responds correctly to all conceivable inputs, that it behaves properly in **any** environment (robustness)

So What is Verification, Then?

- The question: how to ensure that a system behaves properly in the presence of **all** conceivable external inputs and parameters?
- For every **individual** input sequence or parameter value we can **simulate** the reaction of the system. But we **cannot** do it exhaustively
- Verification is a collection of **automatic** and **semi-automatic** methods that allow us to say something about **all** the behaviors (paths in the graph, trajectories, runs)
- Verification complements other, more traditional, validation techniques based on **testing/simulation**

The Ingredients of a Verification Methodology

- A **specification language**: a formalism for describing the desired properties of the system. In other words a criterion for classifying event sequences as good or bad
- A **computational model**: a formalism for describing the designed system (automata, transition systems, programs)
- A **verification technique**: a method to show that the system satisfies the desired properties: all the behaviors generated by the system are those accepted by the specification
- Verification techniques come in two major flavors:
  - Deductive/Analytic
  - Algorithmic (Model Checking)
  - Plug and Pray

Deductive, Analytic Verification

- An intelligent user (an engineer) **proves** that all behaviors never reach a bad state (safety) or eventually reach a good state (progress) **without** actually **computing** all these behaviors
- The proof is based on specific features of the **dynamics** (the **program**, in the discrete case, the **vector field** in the continuous case)
- For safety, you find an **invariant**, a subset of the state space, **closed** under the dynamics (preserved by the transitions). In the continuous domain it is called a **barrier** certificate
- For liveness you find some **progress measure** (ranking function) which always **decreases** along the execution. In the continuous domain it is called a **Liapunov function**
- The need for an **intelligent user** is a major drawback

Algorithmic Verification

- Also known as **Model Checking** (Turing Award 2007, **Clarke, Emerson, Sifakis**)
- Best understood (at least by me) as a **model-aware exhaustive simulation**
- Rather than stimulating a **black box** with **all input sequences**, use **graph algorithms** to explore all the **paths** in the system
- **Symbolic** version (to cope with state explosion): compute sets of reachable states, represented by logical formulae, in a **breadth first** manner
- Compute all states reachable by **all inputs** of **length** $k+1$ from states reachable by all inputs of length k
- In other words, a qualitative version of dynamic programming
- Although algorithmic, it is **not** a **push-button** (nor click-mouse) activity

Temporal Properties in a General Context

- Any system is evaluated according to the **observable behaviors** (trajectories, signals, runs) it produces
- This is done according to **performance measures** defined on these behaviors
- In Control such measures are typically **quantitative**: the average cost/energy, the distance from a reference signal, quadratic norms,...
- Properties are **qualitative** yes/no measures. They classify behaviors as good or bad according to property satisfaction
- Temporal logic is good at specifying properties of **sequences** based on the occurrence of certain **events** in certain **orders**
- **Regular expressions** constitute an alternative formalism of a different flavor for specifying sets of sequences

Why **Temporal** Logic

- Consider the statement: **one day the sun will shine again**
- In **first-order predicate logic**: there is some t such that $t >$ **now** and the sun will shine at t
- Temporal logic, like human language, has special **concise constructs** to talk about **time**
- Sooner and later, before and after, until and since, next and previous
- A typical property: every customer will be (eventually) served
- **Always (request $\rightarrow$ eventually grant)** $\quad \Box(r \rightarrow \Diamond g)$
- Compare to first-order: $\forall t(r[t] \rightarrow (\exists t' > t\ g[t']))$
- "Manna and Pnueli try to do everything with one hand tied behind their back" (J. McCarthy)

How to use Properties in Verification

- From a temporal logic formula one can build a **property tester (observer)**, an automaton or a program that accepts exactly the sequences satisfying the property
- The property tester can be used to check behaviors produced by simulators (**monitoring, runtime verification**)
- It can be **composed** with the **system model** and integrated in the model-checking process to see whether states representing property violations are reachable
- Verification aside, the very process of **writing down** and debugging the specifications helps enormously in **understanding** the system requirements in a **non-ambiguous** manner

Part Two: Hybrid Systems, a Personal Perspective

Information and Control

- In 1957 the journal **Information and Control** has been founded
- I am not **that** old to **remember** this event
- The editorial board included figures such as Noam Chomsky, Peter Elias, Benoit Mandelbrot, Claude Shannon, and Norbert Wiener and many others
- These were **the** days of **Cybernetics** where **information, communication, control, computation, linguistics** and **psychology** were all mixed
- **Computer Science** as a distinct discipline did not exist

A Sample of Articles from the First Volumes

L. Brillouin: Mathematics, Physics, and Information (An Editorial)
C.E. Shannon: Certain Results in Coding Theory for Noisy Channels
N. Chomsky, G.A. Miller: Finite State Languages
M.-P. Schutzenberger: On the Quantization of Finite Dimensional Messages
R. Bellman: Dynamic Programming and Stochastic Control Processes
M. Eden: A Note on Error Detection in Noisy Logical Computers
J. Hartmanis: Symbolic Analysis of a Decomposition of Information Processing Machines
R.M. Karp: A Note on the Application of Graph Theory to Digital Computer Programming
I.J. Good, K.C. Doog: A Paradox Concerning Rate of Information
W.H. Burge: Sorting, Trees, and Measures of Order
L-H. Zetterberg: Detection of Moving Radar Targets in Clutter
C.A. Desoer: The Bang Bang Servo Problem Treated by Variational Techniques
C.W. Merriam: An Optimization Theory for Feedback Control System Design
H. Jacobson: The Informational Content of Mechanisms and Circuits

One Generation Later

- From the journal web site:
- "The journal was one of the earliest to publish extensively on formal language and automata theory, computability, inductive inference, and complexity theory, **as well as** on its titular subjects
- The present editor-in-chief took over in 1982 and recruited to the editorial board a distinguished international group of scholars focusing primarily on **theoretical Computer Science**
- The journal was renamed **Information and Computation** in 1987, reflecting its **new focus**."
- The **divorce** between **computation** and **control** was formalized during the period of my thesis under Pnueli
- A similar episode happened with another journal from the same period, **Mathematical Systems Theory**

A Sample of CS Articles from 1987

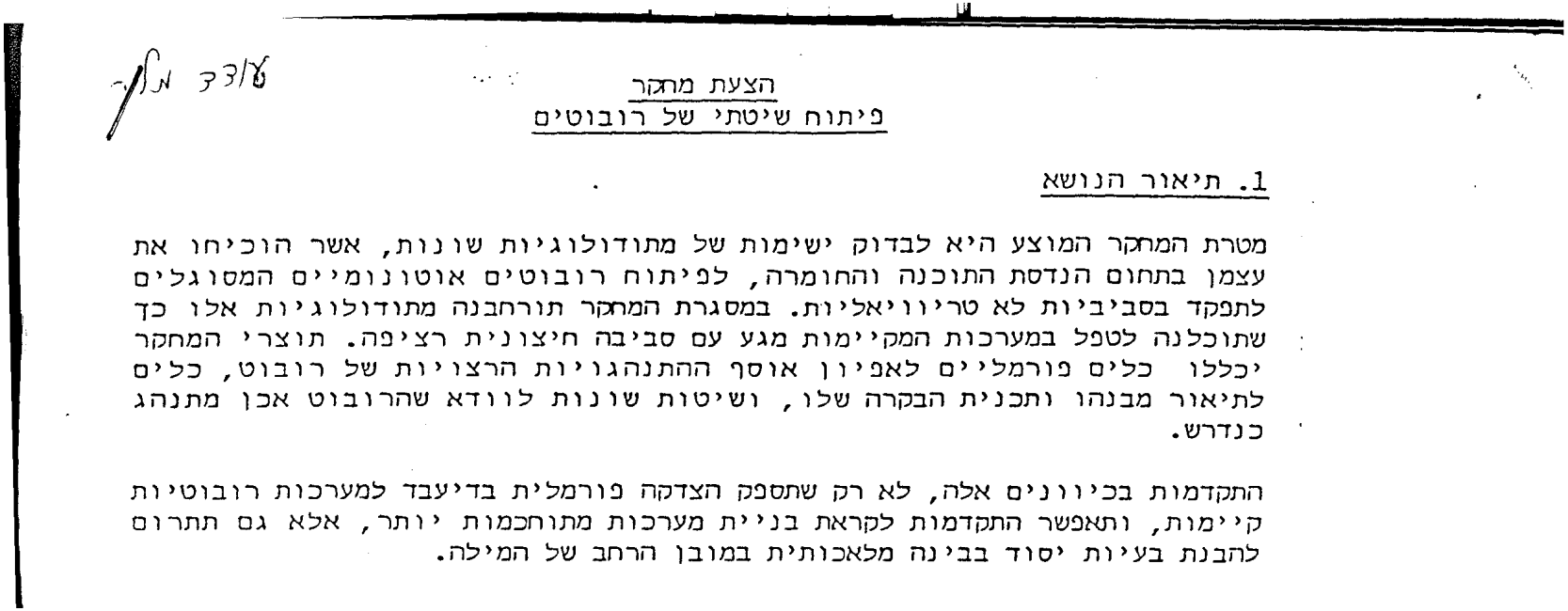
D. Peleg, B. Simons: On Fault Tolerant Routings in General Networks
A. Marron, K-I. Ko: Identification of Pattern Languages from Examples and Queries
J. Sakarovitch: Easy Multiplications. I. The Realm of Kleene's Theorem
R.B. Boppana, J.C. Lagarias: One-Way Functions and Circuit Complexity
A.Z. Broder, D. Dolev, M.J. Fischer, B. Simons: Efficient Fault-Tolerant Routings in Networks
G. Bracha: Asynchronous Byzantine Agreement Protocols
Y. Mansour, S. Zaks: On the Bit Complexity of Distributed Computations in a Ring with a Leader
A. Amir, D.M. Gabbay: Preservation of Expressive Completeness in Temporal Models
G. Winskel: Petri Nets, Algebras, Morphisms, and Compositionality
D. Angluin: Learning Regular Sets from Queries and Counterexamples

Personal Motivation

- Although my thesis was purely mathematical (automata theory) I felt from time to time as an **captive poet**
- Consequently I kept looking from time to time at **softer sciences** such as AI
- One day I found technical reports from MIT AI lab by **R. Brooks**, advocating a **behavior-based approach** to robotics and AI
- His proposed architecture was a sort of a block diagram
- Some blocks represented **sensors**, some were realized by **finite-state machines** or programs, some were **timers**
- I asked Amir: how can one **specify** and **verify** the correct behavior of these creatures?
- Amir was very responsive and called me immediately to his office. He has been thinking about the **physical environment** of the reactive system for a long time

A Research Proposal

- After discussing the issue we decided to write a research proposal on the topic, intended to support my post-doc
- The title was **Systematic Development of Robots**



הצעת מחקר
פיתוח שיטתי של רובוטים

1. תיאור הנושא

מטרת המחקר המוצע היא לבדוק ישימות של מתודולוגיות שונות, אשר הוכיחו את עצמן בתחום הנדסת התוכנה והחומרה, לפיתוח רובוטים אוטונומיים המסוגלים לתפקד בסביביות לא טריוויאליות. במסגרת המחקר תורחבנה מתודולוגיות אלו כך שתוכלנה לטפל במערכות המקיימות מגע עם סביבה חיצונית רציפה. תוצרי המחקר יכללו כלים פורמליים לאפיון אוסף ההתנהגויות הרצויות של רובוט, כלים לתיאור מבנהו ותכנית הבקרה שלו, ושיטות שונות לוודא שהרובוט אכן מתנהג כנדרש.

התקדמות בכיוונים אלה, לא רק שתספק הצדקה פורמלית בדיעבד למערכות רובוטיות קיימות, ותאפשר התקדמות לקראת בניית מערכות מתוחכמות יותר, אלא גם תתרום להבנת בעיות יסוד בבינה מלאכותית במובן הרחב של המילה.

- In other words: we propose to **export the practical failure of program verification toward new domains** ☺
- The proposal did not pass and I moved to France

From Timed to Hybrid Systems I

- Amir was already involved in extending discrete system model with **timing** information
- In this an extremely-important level of abstraction
- Can be used to specify (and verify) not only that every customer is (eventually) served, but also that he is served **at most** 5 **minutes after the request**
- Semantically speaking, in such systems the behaviors are **Boolean signals** rather than just sequences
- He already worked on **real-time** extensions of **temporal logic** and on **timed transition systems**, similar to timed automata
- In 1990 he proposed the model of **phase-transition systems**, a kind of hybrid automaton

From Timed to Hybrid Systems II

From Timed to Hybrid Systems *

Oded Maler
INRIA/IRISA†

Zohar Manna
Stanford University‡and Weizmann Institute of Science§

Amir Pnueli
Weizmann Institute of Science§

Abstract. We propose a framework for the formal specification and verification of *timed* and *hybrid* systems. For timed systems we propose a specification language that refers to time only through *age* functions which measure the length of the most recent time interval in which a given formula has been continuously true.

We then consider hybrid systems, which are systems consisting of a non-trivial mixture of discrete and continuous components, such as a digital controller that controls a continuous environment. The proposed framework extends the temporal logic approach which has proven useful for the formal analysis of discrete systems such as reactive programs. The new framework consists of a semantic model for hybrid time, the notion of *phase transition systems*, which extends the formalism of discrete transition systems, an extended version of Statecharts for the specification of hybrid behaviors, and an extended version of temporal logic that enables reasoning about continuous change.

From Timed to Hybrid Systems III

The discrete event approach is justified by an assumption that the environment, similar to the system itself, can be faithfully modeled as a digital (discrete) process. This assumption is very useful, since it allows a completely symmetrical treatment of the system and its environment and encourages modular analysis of systems, where what is considered an environment in one stage of the analysis may be considered a component of the system in the next stage.

While this assumption is justified for systems such as communication networks, where all members of the network are computers, there are certainly many important contexts in which modeling the environment as a discrete process greatly distorts reality, and may lead to unreliable conclusions. For example, a control program driving a robot within a maze or controlling a fast train must take into account that the environment with which it interacts follows continuous rules of change.

Phase Transition Systems

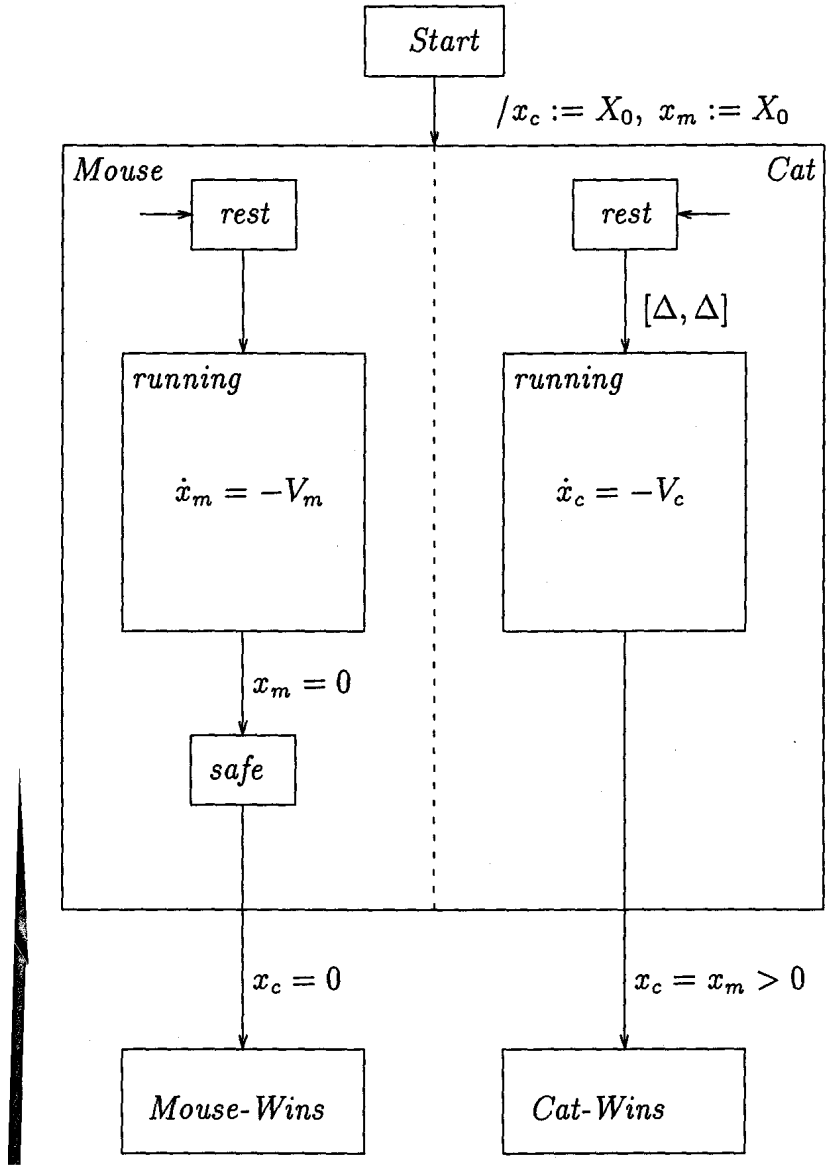
The generalization of a timed transition system to the hybrid domain is called a *phase transition system*. Phase transition systems allow an effective description of systems that can generate hybrid traces as previously described.

Before presenting the formal definition, we make the observation that changes in a phase transition system are governed by the dual constructs of *transitions* and *activities*. The table below compares some of the features of these two constructs.

	Transitions	*Activities*
Govern	Discrete Change	Continuous Change
Take	No Time	Positive Time
Execute	By Interleaving	In parallel
Defined by	Transition Relations	Differential Equations

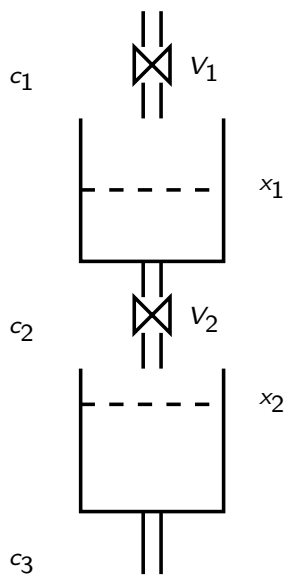
Transitions and activities interact. Transitions start and stop activities and modify the parameters on which the behavior of activities depends. Activities may generate events and conditions that enable or trigger transitions. A typical scenario is that a transition is triggered by the event $becomes(x \geq 0)$, which occurs precisely at the moment in which x switches from a negative value to a non-negative one. An immediate transition that depends on this event for its activation will interrupt the continuous change and execute at this precise time point.

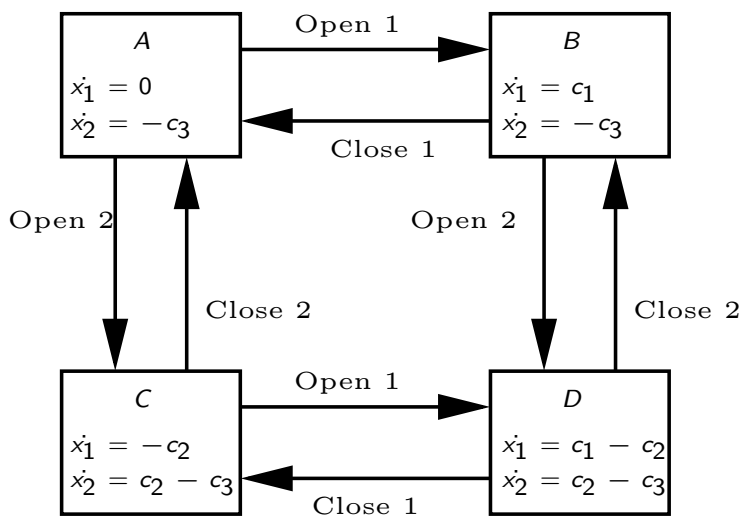
From Timed to Hybrid Systems IV



CS Research on Hybrid Systems

- The paper was accepted with enthusiasm and various groups started looking at the verification of hybrid systems
- At that time the **algorithmic** (model-checking) approach became more popular than the **deductive**
- Encouraged by the verification of **timed automata**, people started looking for similar algorithmic result on a class of **piecewise-trivial hybrid automata**
- Automata where in each state the **derivative** is **constant** and complexity comes from **switching**



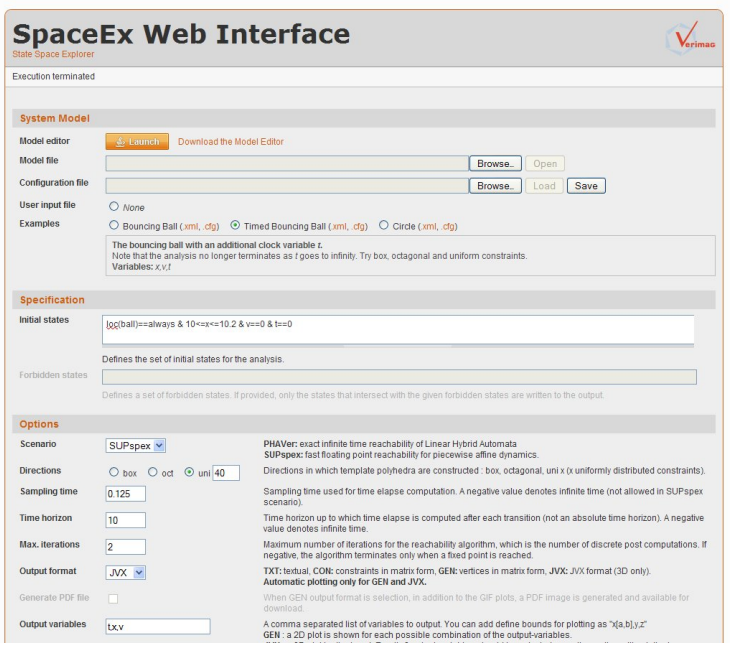


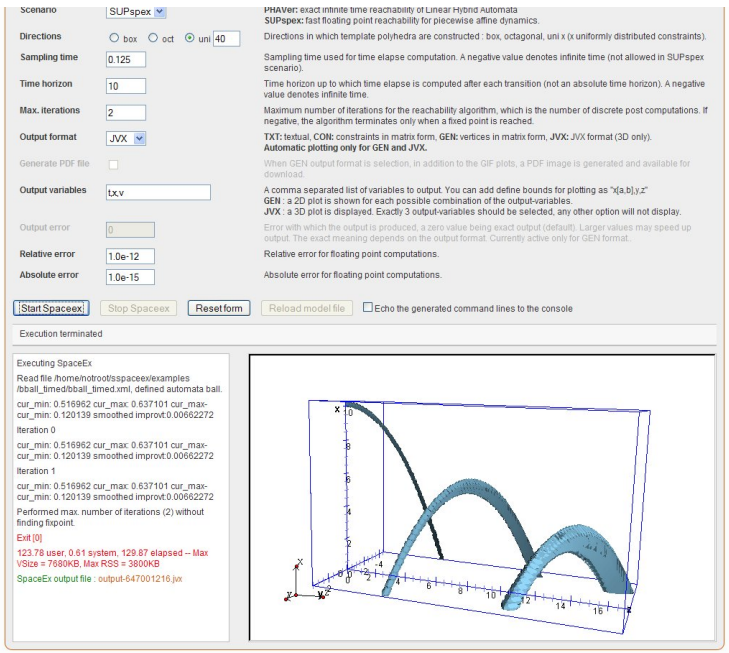
Some of Amir's Contributionsd to Hybrid Systems

- Extending StateCharts to **hybrid StateCharts** [Kesten]
- Contribution to **deductive verification** [Manna, Henzinger]
- Decidability and undecidability results on classes of **stopwatch automata**, timed automata with preemptible clocks, [Kesten, Sifakis, Yovine]
- Decidability (in 2 dimensions) and undecidabilty (in 3 dimensions) of verification for a class of systems with **piecewise-constant derivatives** [Asarin, M]
- Controller **synthesis** for timed automata [Asarin, M, Sifakis]
- Switching controller synthesis for **piecewise-linear** systems [Asarin, Bournez, Dang, M]
- The topic has evolved since, abandoning **exact** computations, replacing them with **approximation** of reachable states, illustrated in the coming commerical

Commercial I: SpaceEx

- Coming soon: SpaceEx the **state space** explorer (G. Frehse)
- A tool platform for developing hybrid verification tools
- Two tools will be released: PHAVer 2.0 for **linear hybrid automata** and a tool for **piecewise-linear differential equations** based on LeGuernic/Girard **support function** algorithms
- Web interface





Back to Temporal Logic

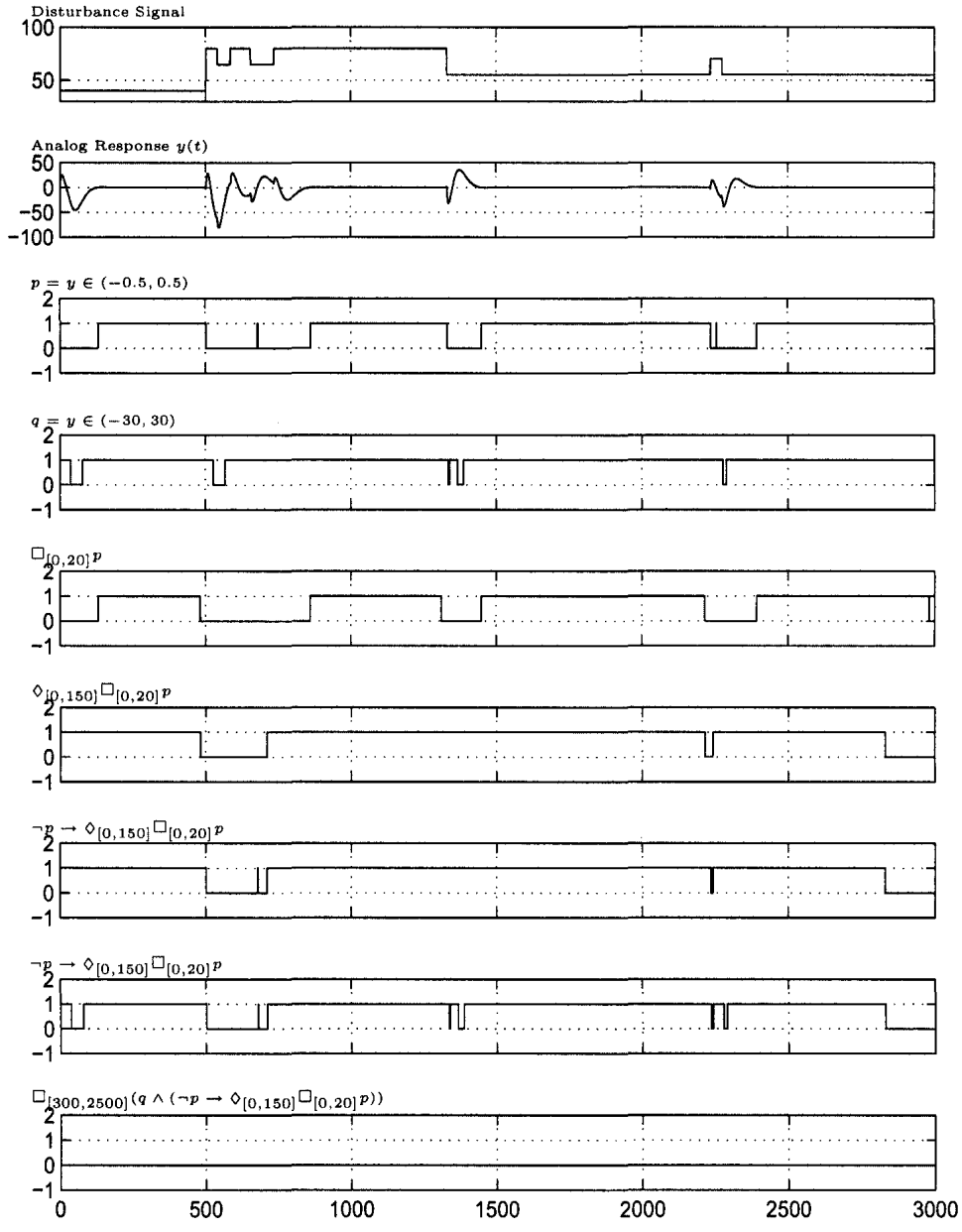
- In the last years Amir's focused more on new problems in **discrete verification** and **synthesis**
- Meanwhile some variants of **temporal logic** have been adopted as standards in the **semi-conductor industry**
- We participated together in the EU project **property-based systems design** (PROSYD) with IBM, ST and Infineon
- My part in the project was to extend the specification language to treat **analog** and **mixed** signals
- We called the logic STL (**signal temporal logic**), An extension of the real-time logic MITL with numerical predicates

Specifying Stabilization in Temporal Logic

- A **water-level controller** for a **nuclear plant** should maintain a controlled variable y around a fixed level despite external disturbances x
- We want y to stay always in the interval $[-30, 30]$ except, possibly, for an initialization period of duration 300
- If, due to disturbances, y goes outside the interval $[-0.5, 0.5]$, it should return to it within 150 time units and stay there for at least 20 time units
- The property is expressed as

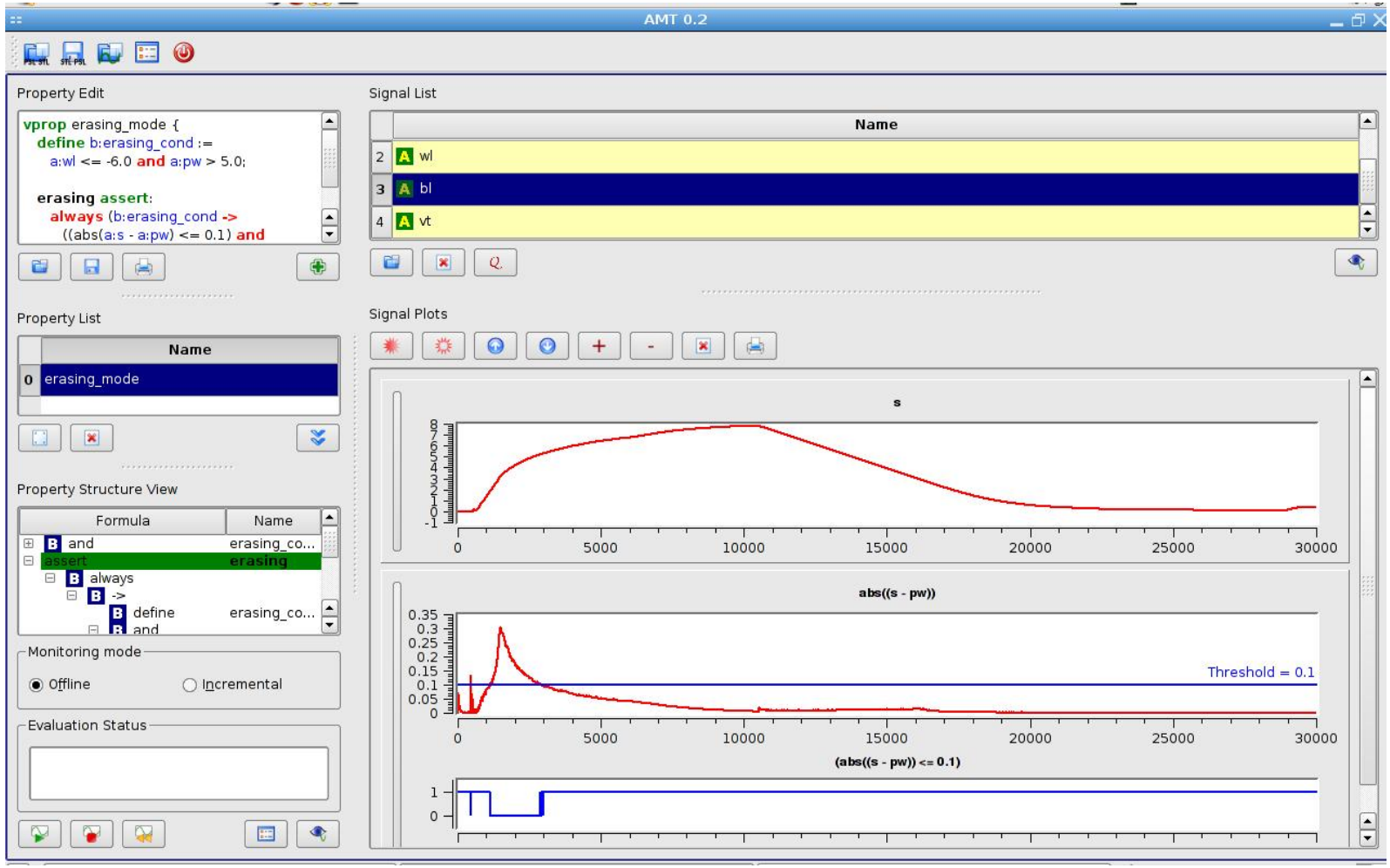
$$\Box_{[300,2500]}((|y| \leq 30) \wedge ((|y| > 0.5) \Rightarrow \Diamond_{[0,150]} \Box_{[0,20]} (|y| \leq 0.5)))$$

Monitoring Stabilization



Commercial II: AMT

- The **Analog Monitoring Tool** (D. Nickovic) is available for download



Concluding Remarks

- Hybrid extensions of Amir's **temporal** logic has a lot of potential applications:
- **Analog circuits**: it was used to check properties of DDR and Flash memories and is considered by some as an interface language for combining **digital** and **analog design flows**
- In **Biology** it can be used, non traditionally, to give a short descriptive model of experimental data
- In **Control** it can be used to define a new class of performance measures based on events
- In **Robotics** it has been used to specify goals for planners
- New studies on **quantitative semantics**: how robustly is a property satisfied
- Even in these domains, which were **not** his core domains, the **contributions** and **insights** of **Amir Pnueli** will occupy us for some time

Last Photos: CAV, June 2009, Grenoble

Last Photos: CAV, June 2009, Grenoble