Some Thoughts on Runtime Verification

Oded Maler

VERIMAG
CNRS and the University of Grenoble (UGA)
France

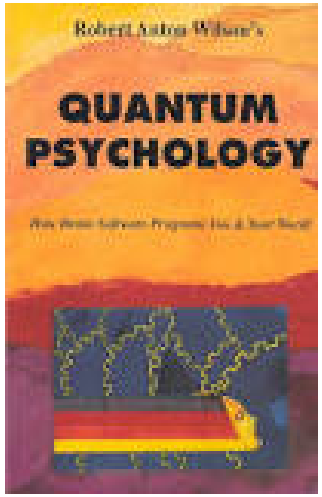
RV, September 2016
Madrid

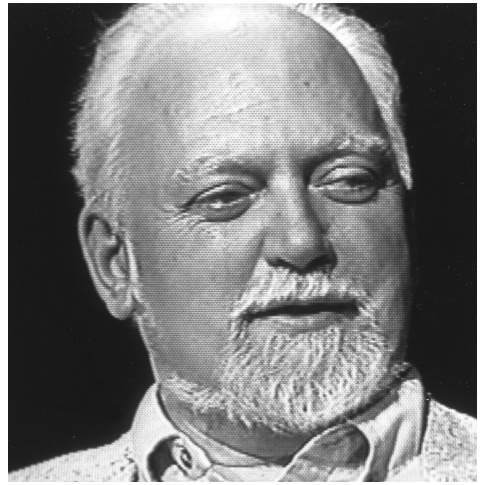
Before Dinner Speech

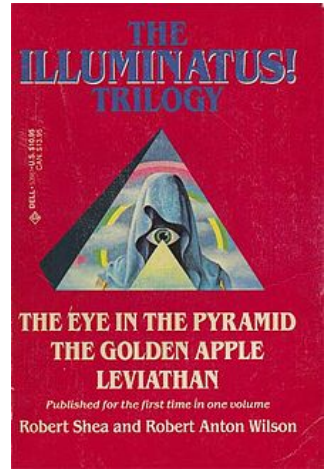
- I like long and general introductions in my papers and talks
- Not everybody does: recently someone protested that such ramblings belong in style to an **after dinner speech**
- But I will unfortunately miss the banquet
- So I allow myself to start my presentation with some reflection in this spirit on the **meaning of words**

What IS Runtime Verification?

- Robert Anton Wilson, (1932-2007), a writer and thinker



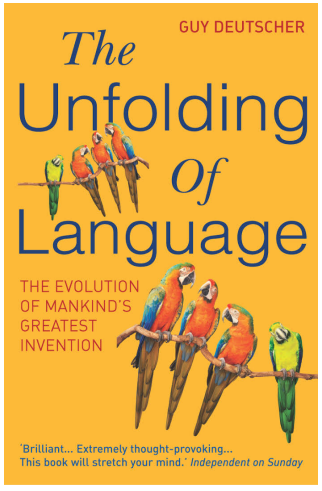


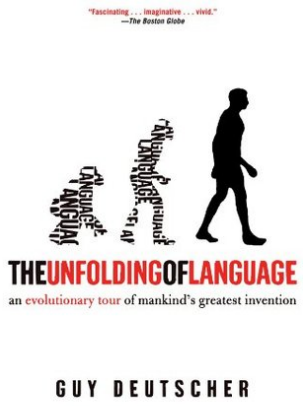


- *"Is", "is." "is" the idiocy of the word haunts me. If it were abolished, human thought might begin to make sense. I don't know what anything "is"; I only know how it seems to me at this moment.*

On The Meaning of Words

- Words do **not** have absolute meaning
- They are just tools to create the (very useful) illusion of common understanding between people
- Their meaning may be different for different individuals, communities and periods in time





- Analyzing a new word/expression, we should look at what **new distinctions** it makes with respect to **existing background**

Example: Reactive Systems

On the Development of Reactive Systems*

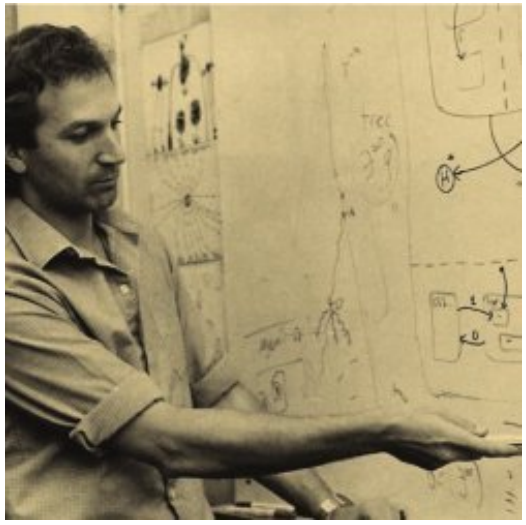
D. Harel and A. Pnueli

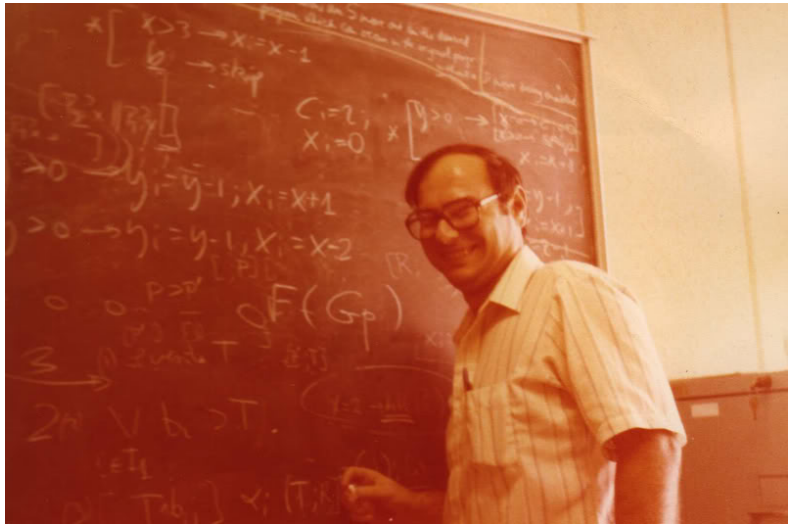
Department of Applied Mathematics
The Weizmann Institute of Science
Rehovot 76100, Israel

January, 1985

Abstract

Some observations are made concerning the process of developing complex systems. A broad class of systems, termed *reactive*, is singled out as being particularly problematic when it comes to finding satisfactory methods for behavioral description. In this paper we recommend the





Reactive Systems

- In this classical paper, more cited than read, reactive systems are defined as
- **Systems that maintain an ongoing interaction with their external environment**
- Real-time, embedded, cyber-physical ... in **contrast** with
- Programs that compute a static function from an input domain to an output domain without **being in time**
- Unlike classical theory of **computability**, **complexity** and **program semantics** dealing with static "autistic" computations[1]
- It is only against this background that the word **reactive** obtains its intended meaning

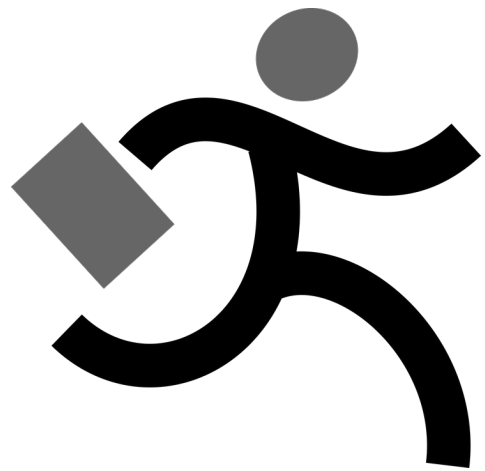
[1] See my pamphlet Hybrid Systems and Real-World Computations

Reactive Systems

- But if you say "reactive" to a **control engineer**, I am not sure he will understand what's the point
- All control systems are reactive by definition, implementing **feedback loops** against a **dynamic environment**
- And when you preach reactive systems to biologists, you really tell them to consider automata as an additional modeling tool
- To AI types exposed to cognitive science, reactive systems may sound like behaviorist **stimulus-response** psychology
-
- Another example: **reachability** (and **controllability**) are precise technical terms in linear control theory which were kidnapped to another meaning in hybrid systems research

So What is Runtime Verification?

- In this talk I will give three **interpretations** of what runtime verification is, in contrast with verification tout court

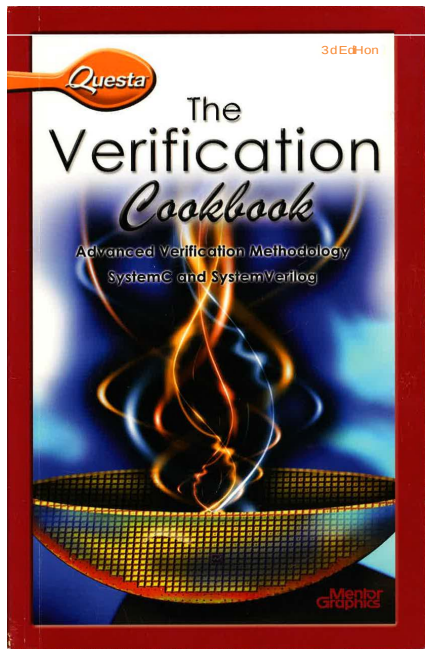


Outline

1. **So what is verification?**
2. RV as **lightweight verification**, non-exhaustive simulation (testing) plus formal specifications
3. RV as getting **closer** to **implementation**, away from abstract models
4. RV as checking systems **after** deployment while they are up and **running**
5. The **limitations** (if not impossibility) of classical formal verification in the cyber-physical world
6. **Qualitative** properties and **quantitative** measures

Verification

- The meaning of **verification** may also vary even among those who pretend to care about correct systems
- It may depend on whether you are a **theoretician** looking for an excuse for your math or a **practitioner** who needs to publish, or any linear combination of those
- I once got this industrial verification book and its intersection with the CAV literature was practically empty



My Version of Verification

- You have a system which is **open** (reactive)
- Each of its dynamic inputs may induce a **different** behavior
- Behaviors are viewed as **trajectories** in the state-space, typically the states of a **product of automata**
- You want to ensure that **all** those behaviors are **correct**, they comply with some restrictions on observed sequences
- These restrictions (specifications, requirements) are formulated either in a **declarative language** (temporal logic, regular expressions) or encoded directly into **observers**

My Version of Verification

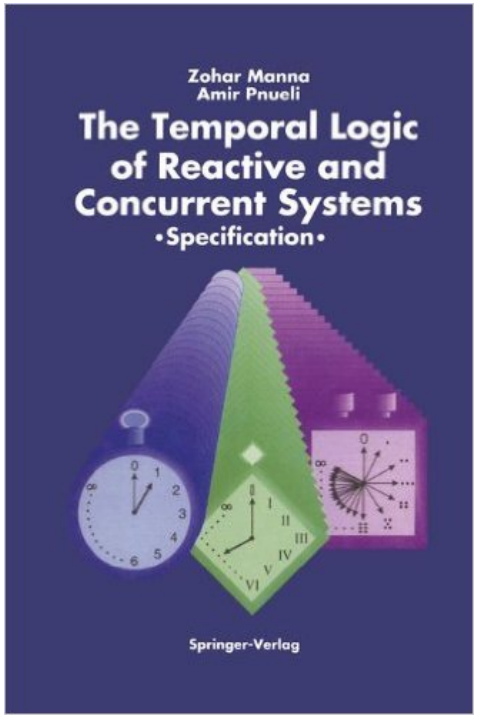
- Rather than stimulate the system with **all** admissible input sequences (exponential in the graph diameter)
- You use the **transition graph** and the **Bellman-Nerode principle** to explore possible behaviors more efficiently
- When systems are small enough you can explore all the paths
- Otherwise you either try to prove things analytically (**deductively**) or use **symbolic** techniques
- Run **set-based breadth-first** simulation while representing reachable states at time t by logic formulae, BDD, etc.
- And most of the rest is efficient implementation

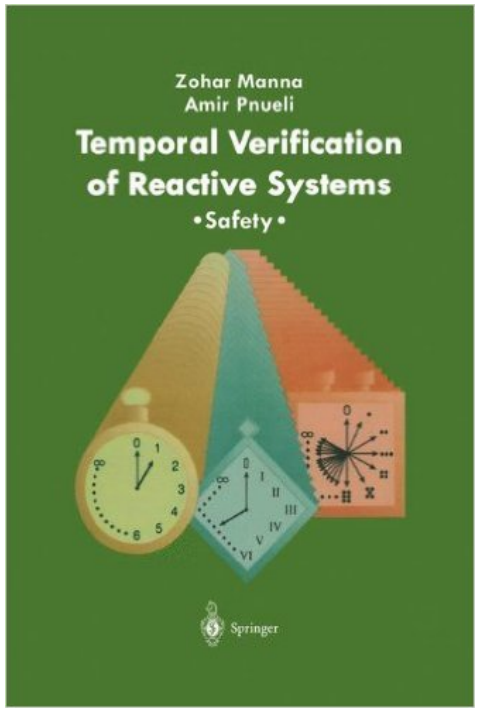
Another Linguistic Observation: Model Checking

- Algorithmic verification is known as **model checking**
- When you try to sell it to an outsider, say a biologist, she probably interprets it in the usual everyday sense:
- I have a **mathematical** model of my **physical** phenomenon and these guys help me to **check** if it makes sense internally
- The origin of MC has nothing to do with this sense of a model
- It comes from the technical notion of a **model** of a **logical theory**
- Verification checks algorithmically whether all system sequences are models of (satisfy) an LTL formula
- Or in branching time: whether the transition system is a model (Kripke structure) of a CTL formula

Another Linguistic Observation: Model Checking

- MC was coined as an alternative to **theorem proving**, where you prove **deductively** the logical specification based on axioms that include the system's description
- The deductive approach is described in these books:





Implicit Assumptions in the Verification Story

- Verification takes place during the **design** and **development** process **before** the system is up and running
- It is often done on an **abstract** model of the system
  - An **automaton** that abstracts from data and implementation details (actual code and platform)
  - The more abstract the model is, the **easier** it is to verify
  - But you need **syntax** to express the system and connect eventually to the real application
- The properties against which you verify are traditionally **qualitative**, providing a **yes/no** answer concerning correctness
- They clearly **partition** the set of global behaviors into acceptable and unacceptable ones
- Some of these assumptions will be dropped in the sequel

Outline

1. So what is verification?
2. **RV as lightweight verification, non-exhaustive simulation (testing) plus formal specifications**
3. RV as getting closer to implementation, away from abstract models
4. RV as checking systems after deployment while they are up and running
5. The limitations (if not impossibility) of cyber-physical formal verification
6. Qualitative Properties and Quantitative Measures

RV as Lightweight Verification (Monitoring)

- Verification is glorious and romantic but **practically impossible** beyond certain complexity
- **Simulation/testing** is here to stay with or without attempts to guarantee some coverage
- So let us add to this practice some **formal properties** and property **monitors** that check the simulation traces
- Instead of **language inclusion** $L_s \subseteq L_\varphi$ as in verification, we check **membership** $w \in L_\varphi$, one trace at a time
- Monitoring is less sensitive to system complexity
- I does not require a **mathematical model** of the system, a program or a black box is sufficient
- In fact, it does not care who generates the simulation traces, it could be **measurements** of a real physical process

Monitoring Continuous and Hybrid Systems with STL

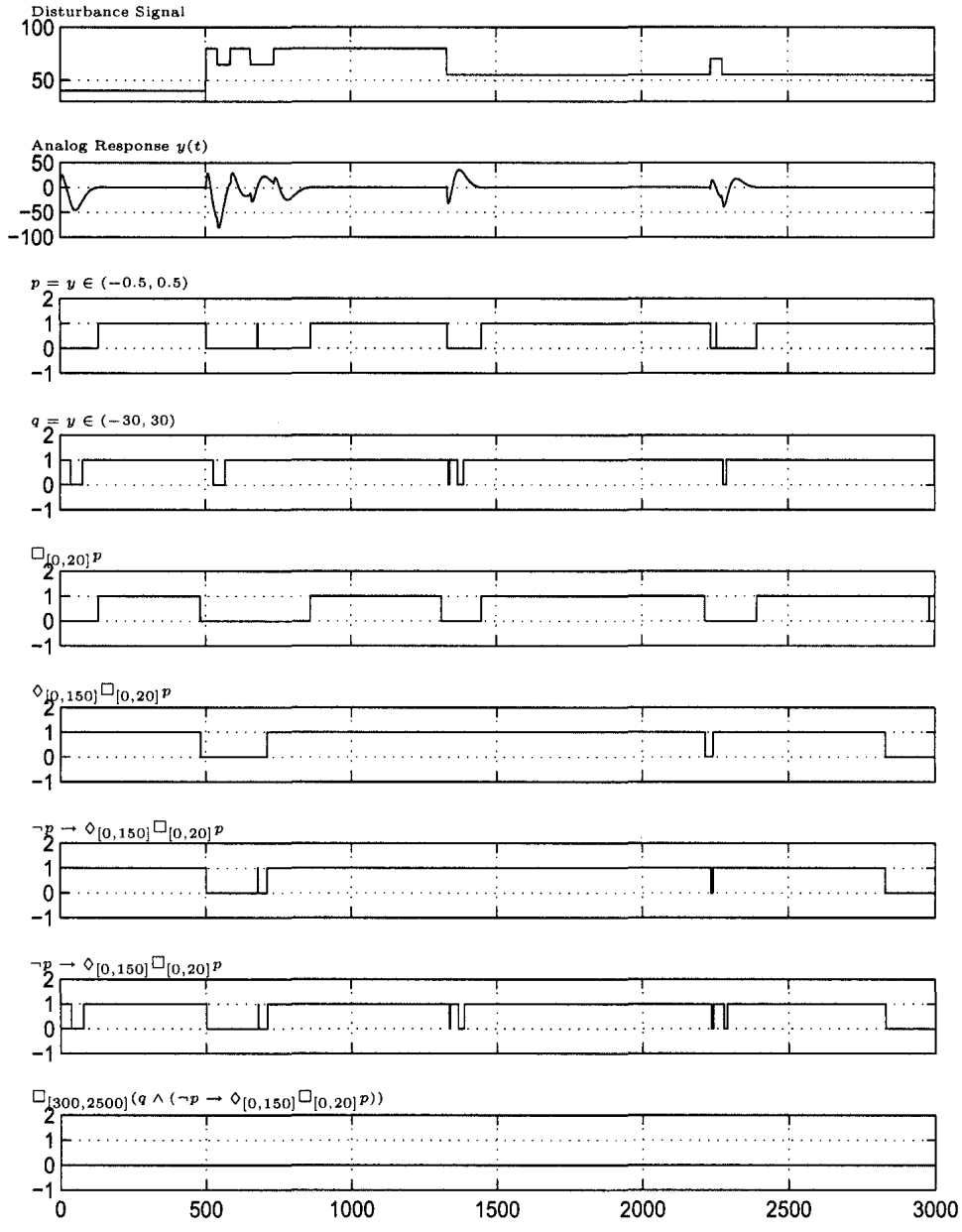
- In digital circuit verification, monitoring is called **dynamic verification** or **assertion checking**
- Motivated by analog and mixed-signal circuits, we extended LTL and MTL into **signal temporal logic** (STL)
- STL can express properties that speak of the **temporal distance** between **threshold-crossings** of continuous signals
- We developed novel monitoring techniques for this logic and implemented them into a tool called **AMT**
- It can **liberate** designers and verifiers from the need to **inspect** and analyze long simulation traces
- It remains an open question whether having a **clean declarative specification language** is a feature or a bug
- These issues were described in the summer school by Dejan Nickovic, a major contributor to this work

Example: Specifying Stabilization in STL

- 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 **remain** 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



The Success of STL

- This is **not** rocket science, much **simpler** than our heroic attempts to scale-up timed and hybrid **verification**
- But it turned out to be very **useful** or, at least, **popular** and also led to a better understanding of real-time logics
- There was **industrial** interest, including a thesis supported by **Mentor Graphics** on combining analog and digital simulators and design flows
- STL has been applied to **circuit** verification, **control** systems (verification, synthesis, falsification), **robotics** planning and **systems biology**
- So let us take a short publicity break

Annotated Bibliography I

- OM, D Nickovic, **Monitoring Temporal Properties of Continuous Signals**, FORMATS/FTRTFT 2004 (first paper)
- D Nickovic, OM, **AMT: A Property-based Monitoring Tool for Analog Systems**, FORMATS 2007 (tool)
- OM, D Nickovic, A Pnueli, **From MITL to Timed Automata**, FORMATS 2006 (theoretical byproduct)
- OM, D Nickovic, A Pnueli, **Checking Temporal Properties of Discrete, Timed and Continuous Behaviors**, Pillars of Computer Science, 2008 (good and long introduction)
- OM, D Nickovic, **Monitoring Properties of Analog and Mixed-Signal Designs**, STTT 2013 (more up to date)
- A Donze, OM, **Robust Satisfiability of Temporal Logic over Real-Valued Signals**, FORMATS 2010 (quantitative semantics)
- A Donze, T Ferrere, OM, **Efficient Robust Monitoring for STL**, CAV 2013 (improved algorithm)

Annotated Bibliography II

- E Asarin, A Donze, OM, D Nickovic, **Parametric Identification of Temporal Properties**, RV 2011 (inverse problem, learning)
- A Donze, OM, E Bartocci, D Nickovic, R Grosu, S Smolka, **On Temporal Logic and Signal Processing**, ATVA 2012 (preliminary extension to frequency domain)
- T Ferrere, OM, D Nickovic, **Trace Diagnostics using Temporal Implicants**, ATVA 2015 (minimal explanation for violation)
- D Ulus, T Ferrere, E Asarin, OM, **Timed Pattern Matching**, FORMATS 2014 (monitoring timed regular expressions)
- D Ulus, T Ferrere, E Asarin, OM, **Online Timed Pattern Matching using Derivatives**, TACAS 2016 (online monitring)
- T Ferrere, OM, D Nickovic, D Ulus, **Measuring with Timed Patterns**, CAV 2015 (a declarative measurement language)

Outline

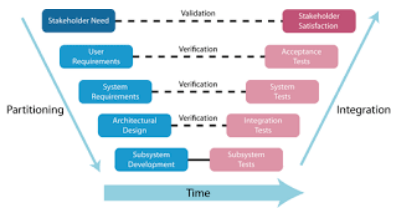
1. So what is verification?
2. RV as lightweight verification, non-exhaustive simulation (testing) plus formal specifications
3. **RV as getting closer to implementation, away from abstract models**
4. RV as checking systems after deployment while they are up and running
5. The limitations (if not impossibility) of cyber-physical formal verification
6. Qualitative Properties and Quantitative Measures

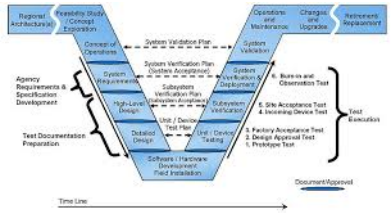
RV as Getting More real 

- Runtime can be interpreted as "while some **program** is running", so we have real piece of **code**
- Already generated from the abstract model or written directly without such a model
- Unlike abstract models, programs are **not** naturally amenable to set-based simulation
- You need to **instrument** the code to generate traces
- The program might (or not) run on the target platform
- There are many degrees of being closer to the final product

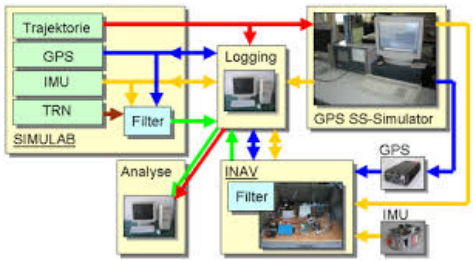
To V or not to V

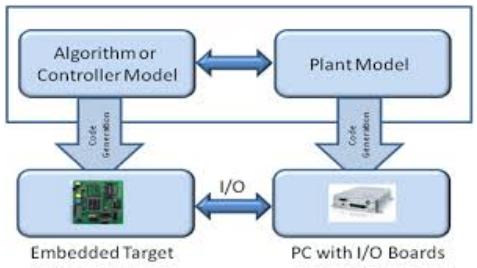
- CPS have heterogenous components, including the **external environment** which is **modeled** but **not** implemented
- The implemented system consists of software, hardware and physical components
- The development process follows some structure

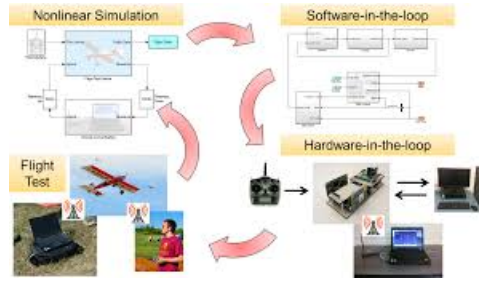




- Coming up from the bottom of the V, you integrate more real components (hardware in the loop, system in the loop)







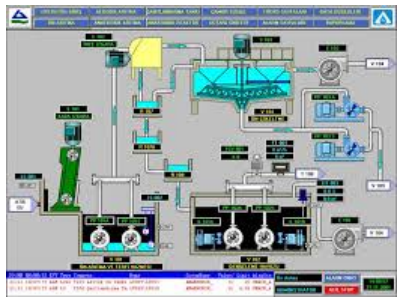
- Runtime can refer to the verification and testing of those

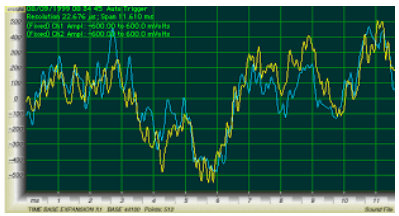
Outline

1. So what is verification?
2. RV as lightweight verification, non-exhaustive simulation (testing) plus formal specifications
3. RV as getting closer to implementation, away from abstract models
4. **RV as checking systems after deployment while they are up and running**
5. The limitations (if not impossibility) of cyber-physical formal verification
6. Qualitative Properties and Quantitative Measures

RV as Verifying Systems while they Run

- Monitoring real systems during their **normal** and **abnormal** execution is the most radical interpretation of RV
- Many systems are observed and **monitored** during execution
- Nuclear and industrial plants, airplanes and cars, medical patients, military control rooms, sound systems in rock concerts, stock markets, google analytics, traffic control ...





New Opprotunities

- A monitoring process which is **simultaneous** with the ongoing behavior of the systems offers new opportunities
- You can detect important **events** and **patterns** of activity in **real time**, almost as soon as they occur
- And **react** to them by **alerting** a **human operator** or triggering an **automatic action**
- These opportunities are new only in the context of verification
- Control panels, displays and alarms exist in low-tech ever since the electrical revolution
- In cars they range from **speed**, **fuel level** and **temperature** indicators to
- More modern ABS, **collision avoidance** systems and **airbags** that detect collisions if they are not avoided

Rethinking Specifications in this Context

- What are the **properties** against which we should monitor online in real time?
- To answer the question I will use the method of the naive **straw man**, a true believer in verification
- Well, he would say, let φ be the **complete specification** of the system, then we monitor for $\neg\varphi$ and shout when it occurs
- But anyway, this will **not happen** if we have **verified** the system (or synthesized the controller properly)
- To see what is wrong here we need to discuss the limitations of **verification** in the **physical** world

The Narrow Scope of Formal Verification

- The verification story depends on the following ingredients:
- 1) A very **faithful model** of the system under verification
- 2) Formal requirements that indeed trace the **boundary** between acceptable and unacceptable behaviors
- In addition, the system should be sufficiently **small** so that formal verification is computationally feasible
- For CPS, (1) and (2) above hold for a very small niche
- Some **hardware** and **software** components, analyzed for their **functional** properties, without physical aspects such as power consumption or timing

The Narrow Scope of Formal Verification

- Software is special, admitting a chain of **semantics preserving** models from **programs** down to **gates** and transistors
- **Nothing** like that exists in the **physical world** where models are just **useful approximations**
- The same holds for **specifications**: you can characterize the valid behaviors of a chip realizing a hardware **protocol**
- You can verify them on a faithful model of the chip and expect that it will indeed work correctly
- For physical systems there is never a comprehensive list of requirements that holds **globally** over the **whole state-space**, which is not part of the conceptual map of engineers
- You have domain-specific **intuitions** on the form of **response curves** but not an explicit formalized partition of behaviors in this huge state-space
- Airplanes fly, nevertheless, most of the time

Monitoring and Supervisory Control

- We want to use some formalism to express observable **conditions** and temporal **patterns** that trigger some response:

  if *some pattern is observed* **then** *do the right thing*

- When the reacting entity is a **human operator**, we should create an **alarm** to bring the situation to her attention
- If the action is **automatic**, this is another instance of feed-back control, appropriate for **high-level supervisory control** where discrete **decisions** are to be taken
- Intuitively, low level is likes controlling torques and velocities in cars or robots (continuous processes)
- Higher levels **decide** whether to bypass an obstacle from right or left or cancel the trip after observing traffic jams
- Similar motivations led in the past to **hybrid systems**

Do Not Wait for the Last Minute

- If we want to **react**, the specified patterns need not be the **complete** negations of properties but **prefixes** of those
- For property like $\square(x < c)$ we should raise a flag when x gets **too close** to c and try to **steer** the system in the opposite direction to **enforce** the property
- If every request should be granted within d time, a useful monitor will detect customers that wait for some $d' < d$ time, while there is a chance to serve them on time
- Note that monitoring is not immediately associated an **error**: fuel level in cars is displayed **continuously** and only when it crosses some **threshold** it is Booleanized into an **alarm**

Some Technicalities of Online Monitoring

- Offline monitoring can go **back and forth** on the simulation trace which has already been computed
- Going backwards is natural for **future** temporal logic which is **acausal**: truth at t depends on values at $t' > t$
- For real systems we cannot wait until the **end of time** and need to adopt **forward** techniques
- **Past** temporal logic is causal and can be monitored forward
- One may argue that unbounded liveness is useless, while **bounded liveness** (safety), translates to past TL and can be monitored causally
- In verification you consider behaviors starting at $t = 0$;
- For online monitoring you look for **segments** of the behavior that **match** some pattern
- **Regular expressions** (timed and hybrid) are more appropriate

Outline

1. So what is verification?
2. RV as lightweight verification, non-exhaustive simulation (testing) plus formal specifications
3. RV as getting closer to implementation, away from abstract models
4. RV as checking systems after deployment while they are up and running
5. The limitations (if not impossibility) of cyber-physical formal verification
6. **Qualitative Properties and Quantitative Measures**

Quantitative Semantics, Robustness

- Properties map behaviors (sequences, signals) into $\{0, 1\}$ according to **satisfaction** or **violation**
- Sometimes we want more refined information about the **robustness** of the answer
- For a behavior satisfying $\Box(x < c)$, the distance $c - \max_t x(t)$ tells us how **close** we were to violation
- When we violate $\Box(e \rightarrow \Diamond_{[0,a]} e')$, the maximal temporal distance between e and e' defines the severity of violation
- The quantitative (robustness) semantics of STL returns a real value $\rho = \rho(\varphi, w)$ satisfying:

$$\rho(\varphi, w) > 0 \leftrightarrow w \models \varphi \text{ and } \forall w' d(w, w') < \rho \rightarrow (w \models \varphi \leftrightarrow w' \models \varphi)$$

- This number is used in optimization/search procedures for finding bad behaviors (falsification)
- Implemented in tools such as **S-Taliro** (Fainekos and Sankaranarayanan) and **Breach** (Donze)

Unifying Properties and Performance Measures

- Robustness gives more information but it still suffers from the **extremal** nature of logic
- The quantitative semantics is obtained by replacing Boolean predicates such as $x < c$ by numbers like $c - x$ and then replacing $\vee$, $\wedge$ and $\neg$ by max, min and $-$
- The value will always depend on the **worst case**, the largest value of x, the largest response time
- Some work is needed to reconcile STL with other **additive** (average) measures used elsewhere

Unifying Properties and Performance Measures

- Specification formalisms such as STL and TREG and their quantitative extensions should be viewed as
- Yet another family of **performance measures** which are good in terms of expressing **sequential** behaviors
- As nobody is perfect, they are **weak** in other aspects and should be inserted into the rich arsenal of measures existing already in **control**, **signal processing**, **statistics**, etc.
- A **unified declarative language** for qualitative properties and quantitative measures can be a useful contribution toward **monitoring complex systems**, simulated and real
- Thank you