MFES – Métodos Formais em Engenharia de Software

Introduction

Ana Paiva

apaiva@fe.up.pt **www.fe.up.pt/~apaiva**

What are the Formal Methods?

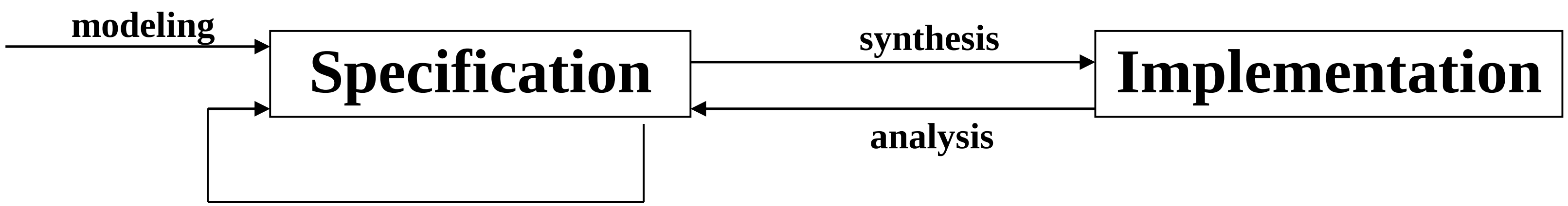
- "Techniques based on mathematics to describe properties of a system" [Wing 1994]
- Formal methods are based on mathematical techniques for the specification, construction (refinement) and verification of software systems and hardware, with the aim of achieving higher levels of quality
- They increase confidence in the correctness of the software through formal proofs, refinements (correction by construction) and test

What are the Formal Methods?

Formal Methods (FM) = Specification language + Formal Reasoning

- The techniques are supported by
  - Precise mathematical
  - Powerful analysis tools

- They are an accurate and effective way for modeling, synthesis and analysis systems



The Seven Myths of Formal Methods

- Formal methods can guarantee perfect software.
- Formal methods are only on verification of programs.
- Formal methods are only useful in critical systems.
- Formal methods involve complex mathematics.
- Formal methods increase the cost of development.
- Formal methods are incomprehensible to clients.
- Formal methods are not used in real systems of large dimension.

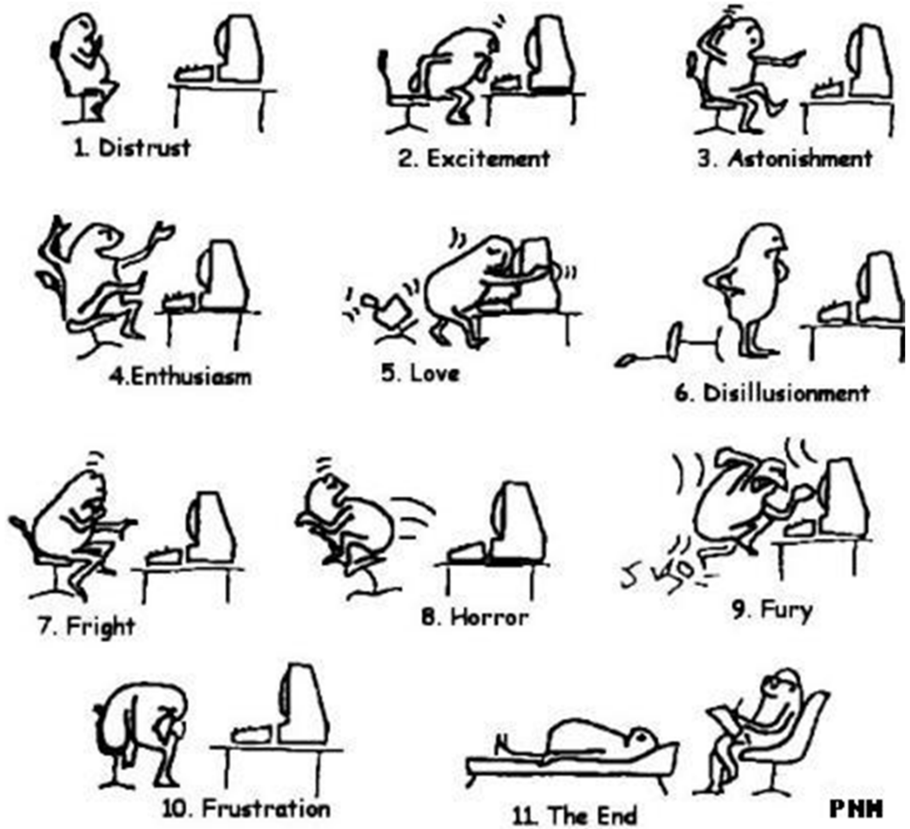
Formal Methods Europe

General reasons for choosing formality

- to improve quality and rigour of whole development process?
- to improve integrity, reliability or other characteristics of the system?
- to reduce specification errors?
- to improve requirements definition?
- to improve documentation and understanding of designs?
- to provide a firmer foundation for maintenance and enhancement?
- to explore the properties of a design architecture?
- to provide a more rational basis for choosing test data?
- to be as certain as possible that the design and implementation are error-free?
- to meet particular customer or standards requirements?

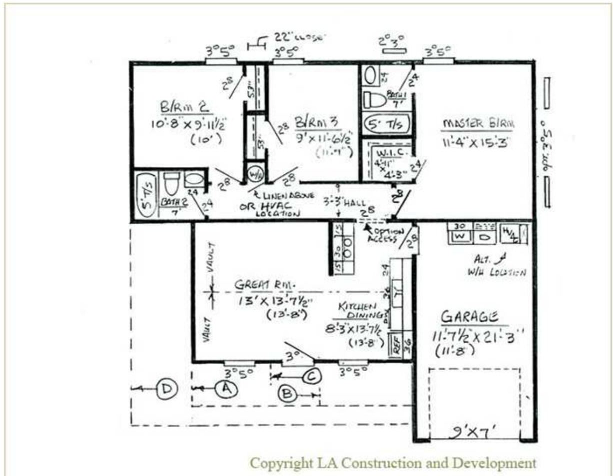
Formal methods through development process

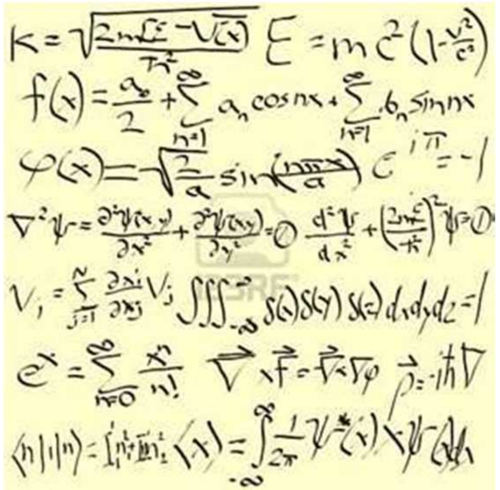
- Formal methods can be applied at various points through the <u>development process</u>.
  - Specification
  - Development
  - Verification



Formal methods through development process

- **Specification**
  - Formal methods may be used to give a description of the system to be developed, at whatever level(s) of detail desired. This formal description can be used to guide further development activities (see following sections); additionally, it can be used to verify that the requirements for the system being developed have been completely and accurately specified.





Formal methods through development process

- **Development:** Once a formal specification has been developed, the specification may be used as a guide while the concrete system is developed (i.e., realized in software and/or hardware).

  Examples:

  - If the formal specification is in an operational semantics, the observed behavior of the concrete system can be **compared** with the behavior of the specification (which itself should be executable or simulateable) (e.g., Spec Explorer). Additionally, the operational commands of the specification may be amenable to direct **translation** into executable code (e.g., VDMTools).

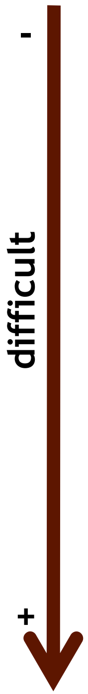
  - If the formal specification is in an axiomatic semantics, the preconditions and postconditions of the specification may become **assertions** in the executable code.

Formal methods through development process

- **Verification**
  - Once a formal specification has been developed, the specification may be used as the basis for proving properties of the specification (and hopefully by inference the developed system).
  - In contrast, there is increasing interest in producing proofs of correctness of such systems by automated means. Automated techniques fall into two general categories:
    - Automated theorem proving, in which a system attempts to produce a formal proof from scratch, given a description of the system, a set of logical axioms, and a set of inference rules.
    - Model checking, in which a system verifies certain properties by means of an exhaustive search of all possible states that a system could enter during its execution.

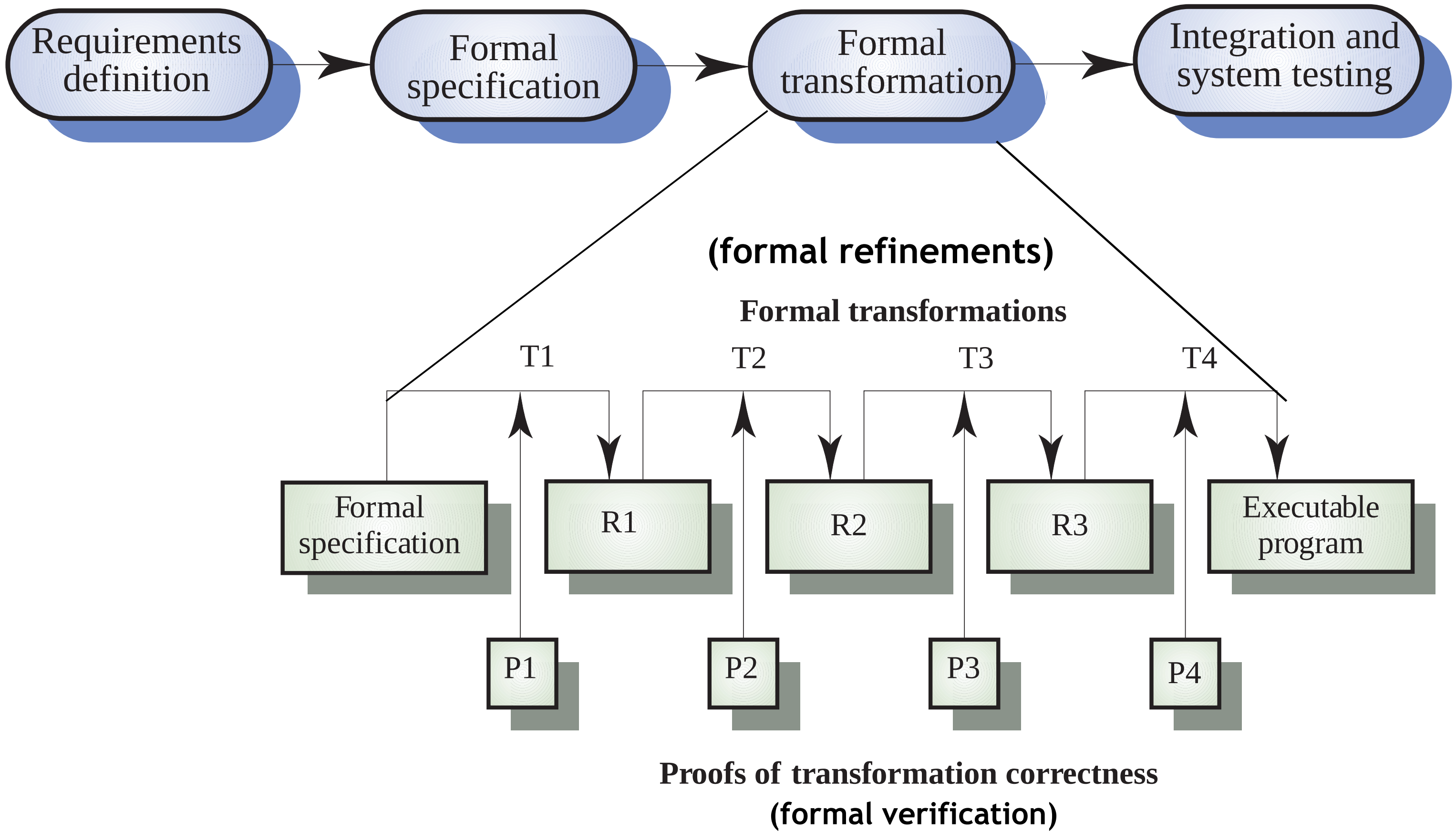
Formal methods

- Formal methods can be used at a number of levels:



  - **Level 0:** Formal specification may be undertaken and then a program developed from this informally. This has been dubbed *formal methods lite*. This may be the most cost-effective option in many cases.
  - **Level 1:** Formal development and formal verification may be used to produce a program in a more formal manner. For example, proofs of properties or refinement from the specification to a program may be undertaken. This may be most appropriate in high-integrity systems involving safety or security.
  - **Level 2:** Theorem provers may be used to undertake fully formal machine-checked proofs. This can be very expensive and is only practically worthwhile if the cost of mistakes is extremely high (e.g., in critical parts of microprocessor design).

Formal methods: Refinement

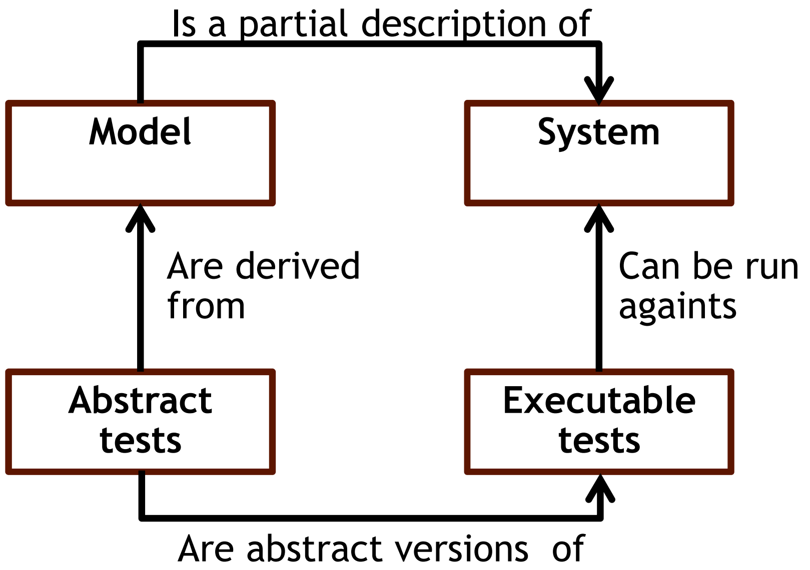


Formal verification techniques

- **Test**
  Check the execution of a software program for a specification and according to some coverage criterion (Model based testing)
- **Model checking**
  Use a tool to automatically check that a particular software program satisfies its specification
- **Theorem prooving**
  Use a logical formalism to prove (formally) that a software meets its specification

Model Based Testing

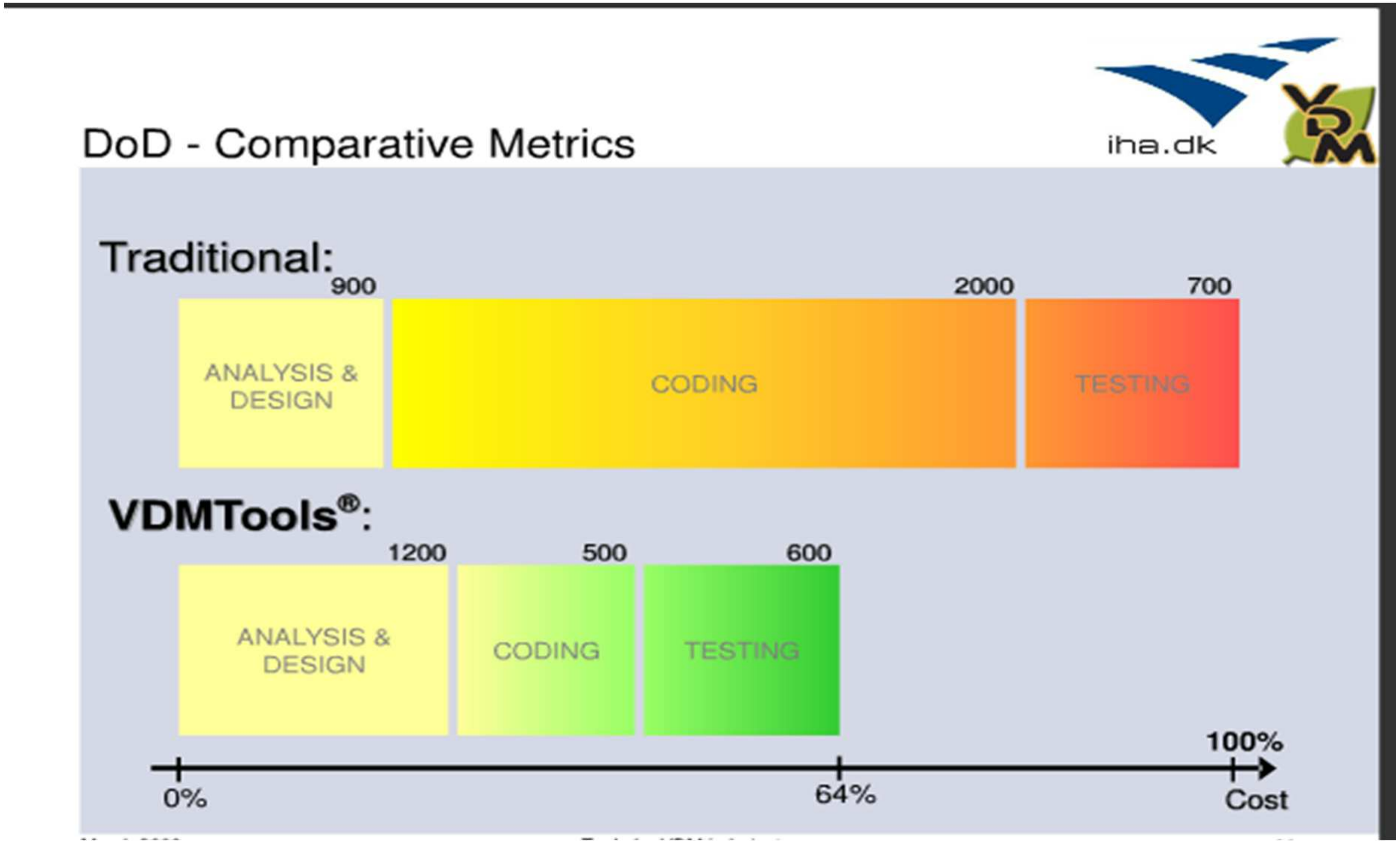
- Software testing does not prove the non-existence of errors
- So what is software testing?
  - Software testing is the process that runs a program with the aim of finding bugs
  - A successful test is a test that can find errors



Tools

- VDMTools
  - http://www.vdmtools.jp/en/
  - Design test cases based on the specification and use them afterwards to test the implementation. A test set that covers completely the specification may not cover completely the implementation
- Spec Explorer
  - http://research.microsoft.com/specexplorer/
  - Establishes a map between specification and implementation actions in order to run test steps in both levels (specification and implementation) and compare results obtained. If the results are different something wrong has to be fixed
- NModel
  - http://www.codeplex.com/NModel

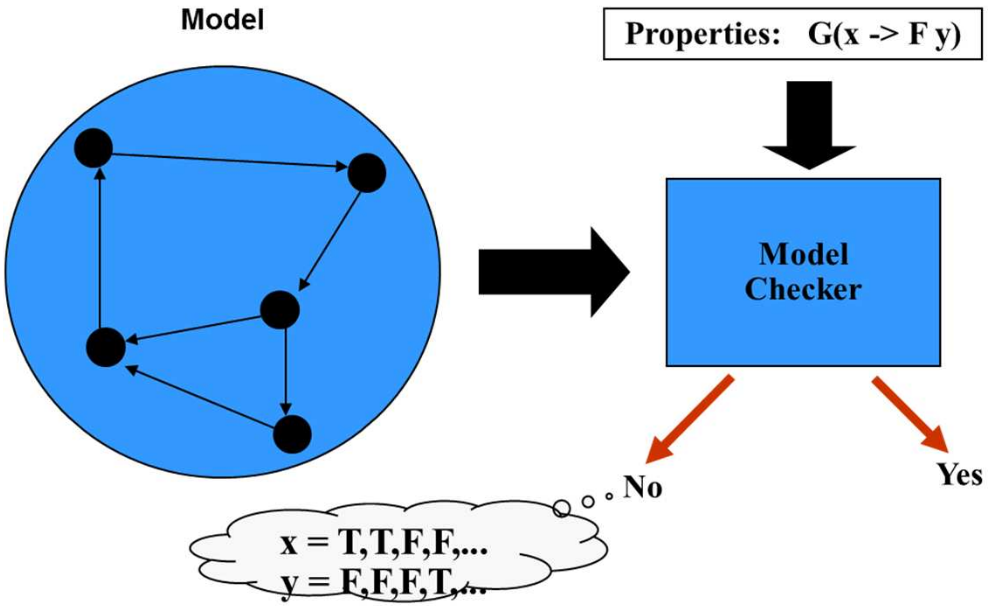
Development process



Model Checking

- An automatic technique that, given a **finite state model** of a system and a **logical property** (in temporal logic), systematically checks whether the property is true in that model by searching in all states

System ⊨ Property



Model Checker

- To implement a “Model Checker”:

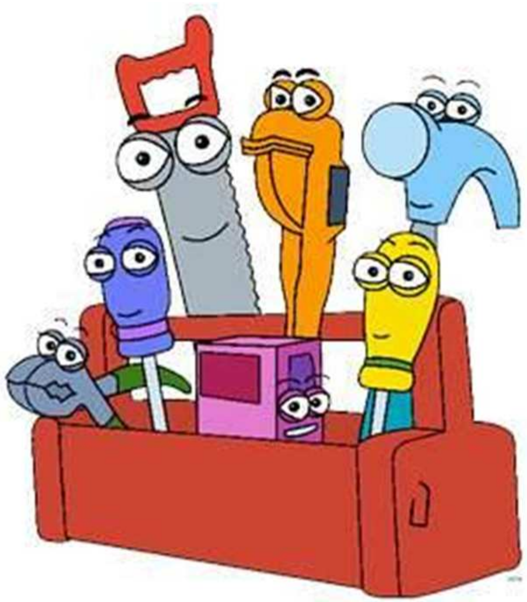
1. Building the automaton of S: A_S
2. Building the automaton of ~P: $A_{\sim P}$
3. Calculate $A_S \cap A_{\sim P}$
   - If Ø then P is true.
   - If ≠ Ø then the sequence of transitions obtained is a counterexample of P.

Challenges

- Explosion of states
- Difficulty in expressing certain types of properties of systems
- Need to learn two modeling notations: one for modeling the system and another to describe the properties of this system

Tools

- SPIN,
  - http://netlib.bell-labs.com/netlib/spin/whatisspin.html
- SMV (Symbolic Model Verifier)
  - http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/
- HYTECH (Linear Hybrid Systems)
  - http://www.eecs.berkeley.edu/~tah/HyTech
- UPAAL (Real-Time Systems)
  - http://www.upaal.com)
- Kronos (Real-Time Systems)
  - http://www-verimag.imag.fr/TEMPORISE/kronos
- "Alloy" (model finder)
  - http://alloy.mit.edu/



Theorem prooving

- A formal logical system consists of:
  - Notation (syntax)
  - A set of axioms
  - A set of rules of inference
  - A formal proof is a sequence of statements / phrases. Each statement is built from the application of one or more rules of inference to the previous statement(s)
  - A purely syntactic mechanism that does not care about the meaning of the claims but only to their construction

Statement	Reason
1) c is midpoint of $\overline{BX}$	1) Given
2) c is midpoint of $\overline{AY}$	2) Given
3) $\overline{BC} = \overline{XC}$	3) definition of midpoint
4) $\overline{CA} = \overline{CY}$	4) definition of midpoint
5) $\angle BCA \cong \angle XCY$	5) Vertical angles are congruent
$\Delta BCA \cong \Delta XCY$	by SAS

Theorem prooving

- Prove that an implementation (I) satisfies the specification (S) by means of mathematical reasoning.

$$I \rightarrow S$$

$$I \equiv S$$

- The implementation and specification are expressed by logical formulas.
- The (logical equivalence / logical implication) required is described as a theorem that has to be proved.
- A proof system provides a set of axioms and inference rules (simplification, rewriting, induction, etc.).

Challenges

- Interactive approach that ensures that the proof is correct but does not provide guidelines for building the same proof
- Consuming activity
- Requires knowledge and practice
- Scalability

Tools

- PVS (Specification and Verification Systems)
  - http://www-step.stanford.edu/
- STeP
  - http://pvs.csl.sri.com/
- HOL (Higher order logic)
  - http://www.cl.cam.ac.uk/Research/HVG/HOL
- The Logics Workbench
  - http://www.lwb.unibe.ch/
- Coq
  - http://coq.inria.fr/
- Jape