ARTIST2 - MOTIVES
Trento - Italy, February 19-23, 2007

Session: Testing and Runtime Verification

Formal Verification and Testing for Reactive Systems

Vlad Rusu

INRIA Rennes, France

Formal verification: proving correctness

- Verification by “paper/pencil”
- Algorithmic techniques
  - Model checking
  - Abstract interpretation
- Deductive techniques
  - Interactive theorem proving
- Various combinations of the above.

Testing: finding errors

- What is available for testing:
  - White box*:* Source code
  - Black box: Executable code
- What is tested:
  - Functional (against specification/oracle)
  - Structural (against coverage criteria)
  - Robustness, performance, real time…

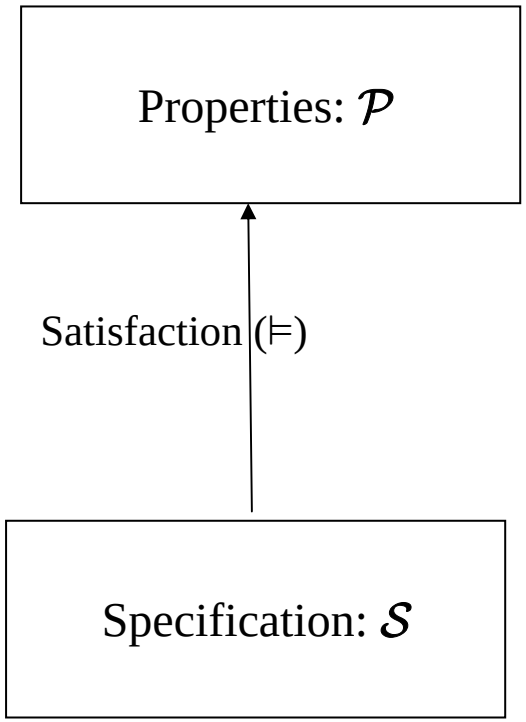
Combining verification and testing: best of both worlds?

- Testing using verification techniques
  - White box
    - Using a model checker to derive structural tests [Ammann][Heitmeyer]
    - Definition of coverage using temporal logic/observers [Lee][Jonsson]
    - Abstraction for structural testing: “predicate coverage” [Henzinger] “abstract path coverage” [Ball]
  - Black box
    - Test generation for conformance using model checking techniques [Jard, Jéron] [Brinksma, Tretmans]
    - Test generation for conformance using symbolic techniques [Le Gall] [Jéron, Rusu]
    - Test generation for temporal-logic properties using model checking [Fernandez]
- Combining verification and testing
  - The ESC/Java toolset

Outline

- A closer look at verification vs. conformance testing
- Verification: the many ways to reachability
- Conformance testing : the **ioco** relation and symbolic test generation
- Integrating verification and conformance testing
- Conclusion and perspectives.

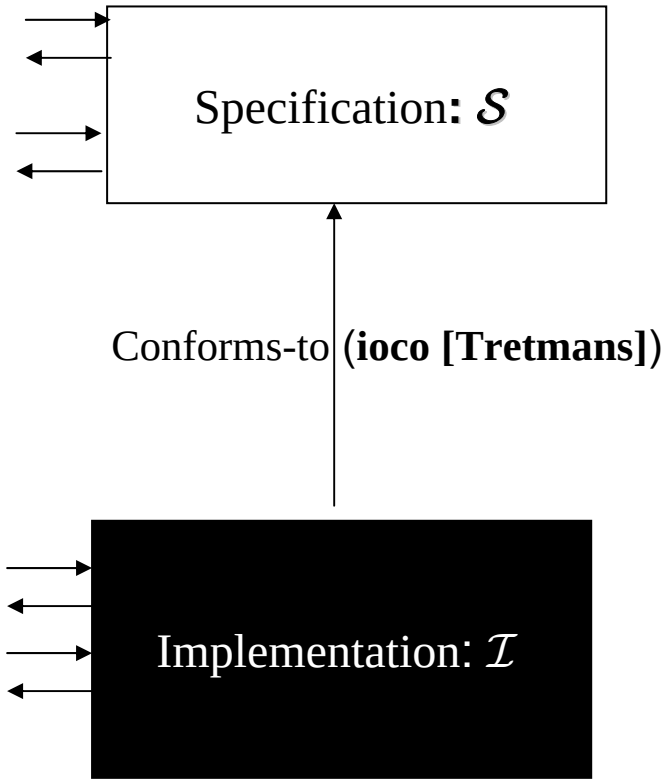
Formal Verification



Verification Problem: $\mathcal{S} \sqsubseteq \mathcal{P}$

- Can be reformulated as $\mathcal{S} \times \overline{\mathcal{P}} \nrightarrow$ 
- Basic operations involved:
  - Product
  - Complementation (determinisation)
  - **Reachability.**

Conformance Testing



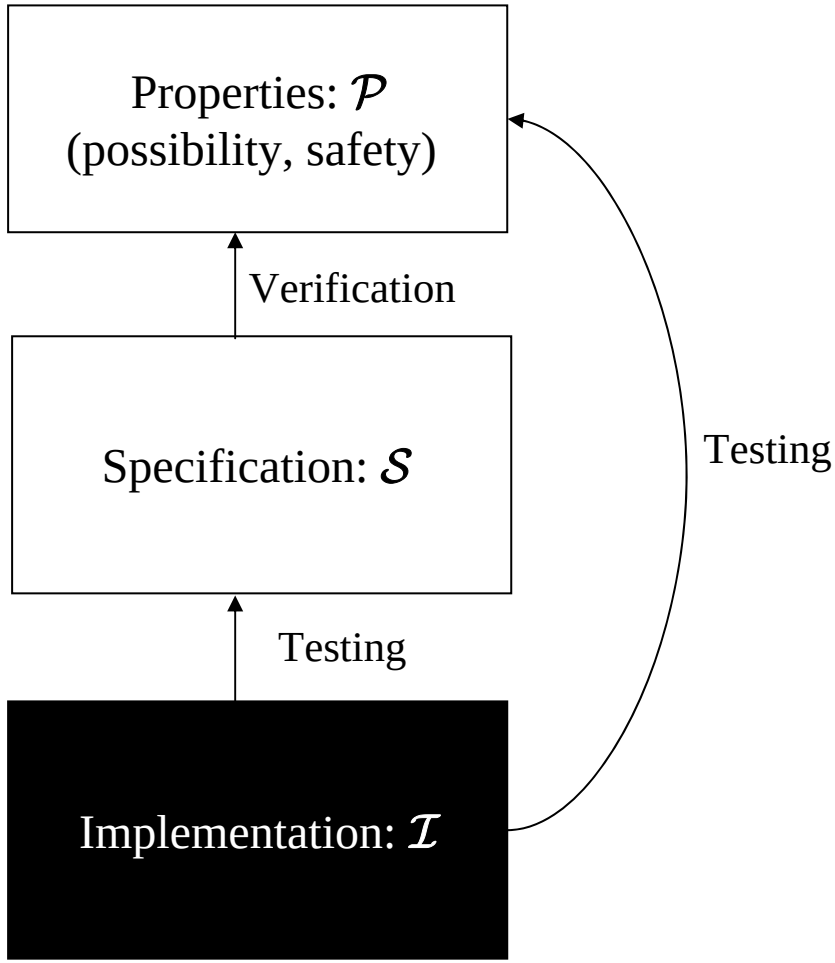
Conformance Testing Problem: $\mathcal{I}$ **ioco** $\mathcal{S}$

- Reformulated as $\mathcal{I} || test(\mathcal{S}) \not\rightarrow$ 
- Basic operations required :
  - Parallel composition
  - Complementation (determinisation)
  - **Reachability.**

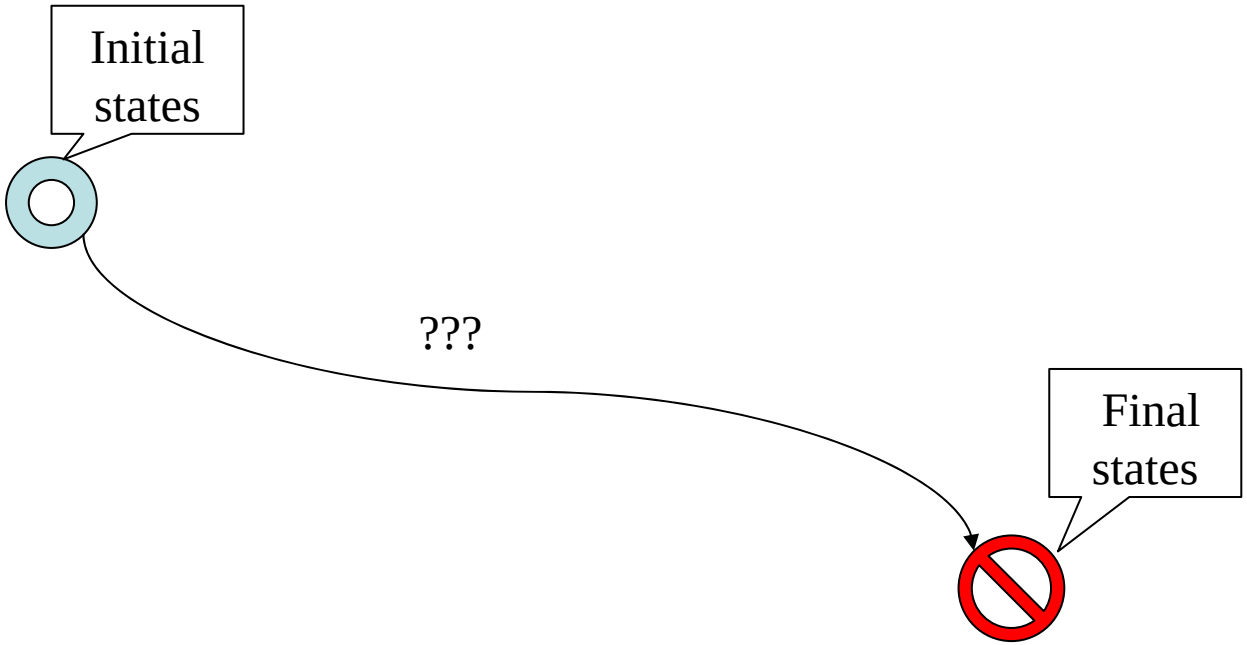
Verification vs. ConformanceTesting

- Same basic operations involved
- Verification: all formal models & reasoning
  - Can prove or disprove satisfaction relation
- Conformance testing: model of $\mathcal{I}$ unknown
  - Can only disprove conformance relation.

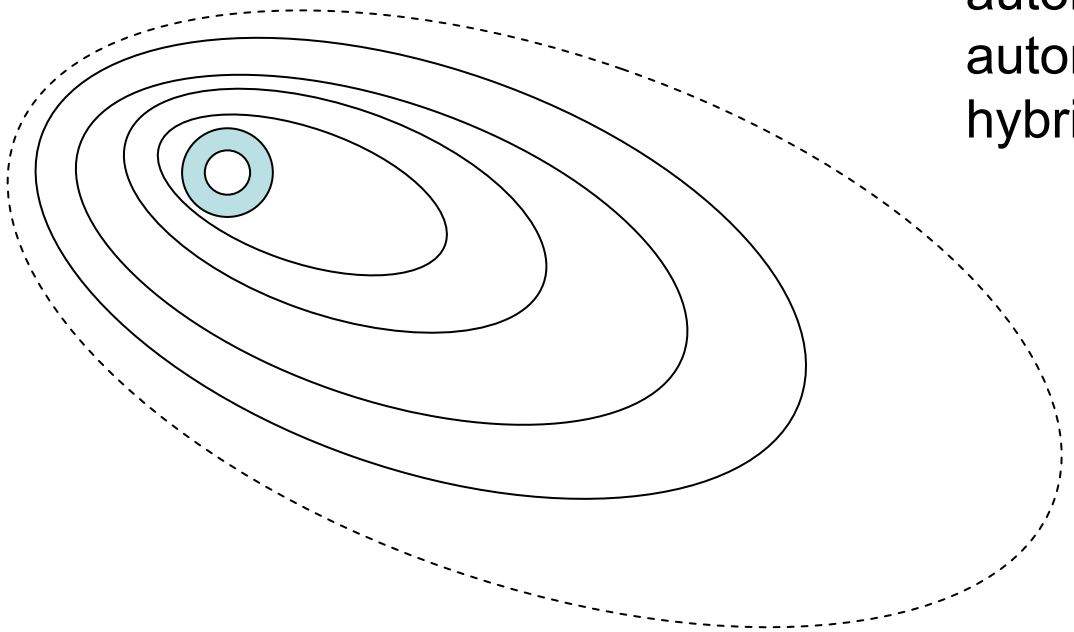
Verification *and* Conformance Testing



Verification: Reachability

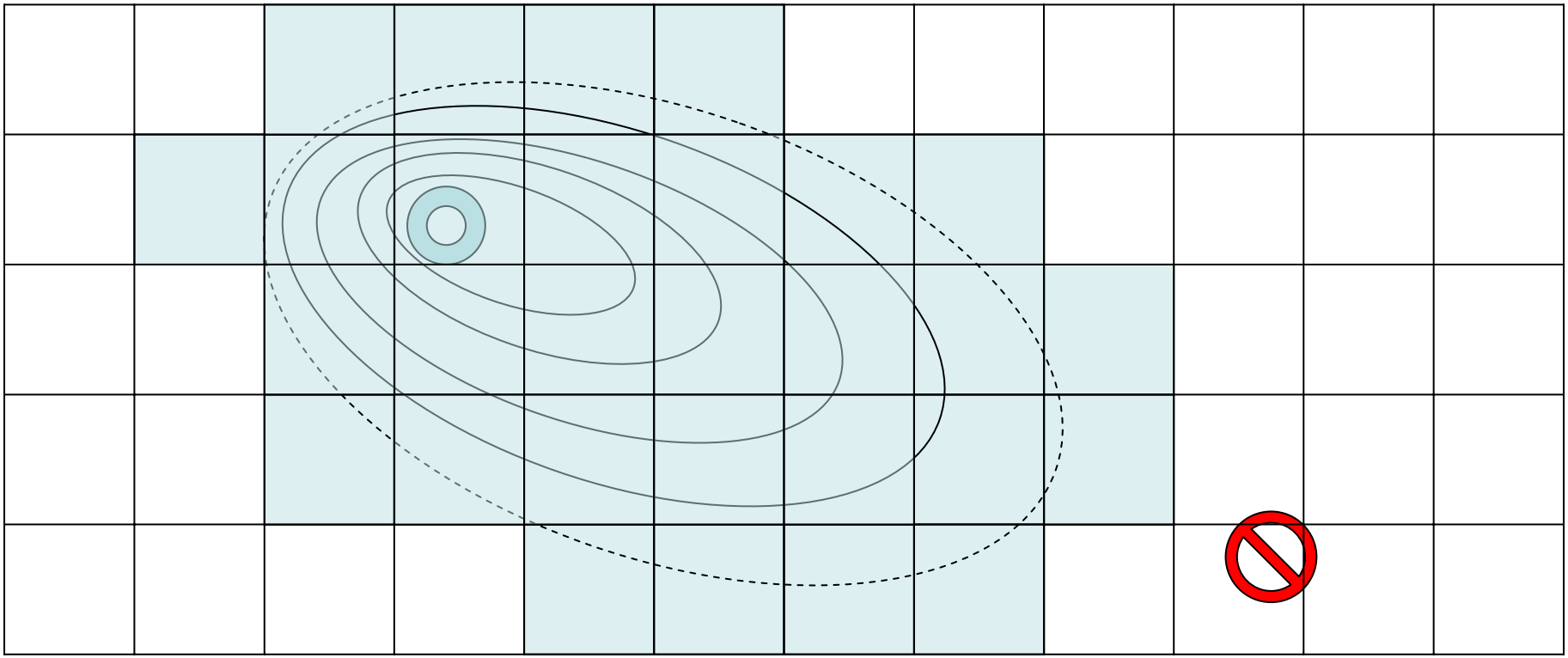


Computing sets of reachable states

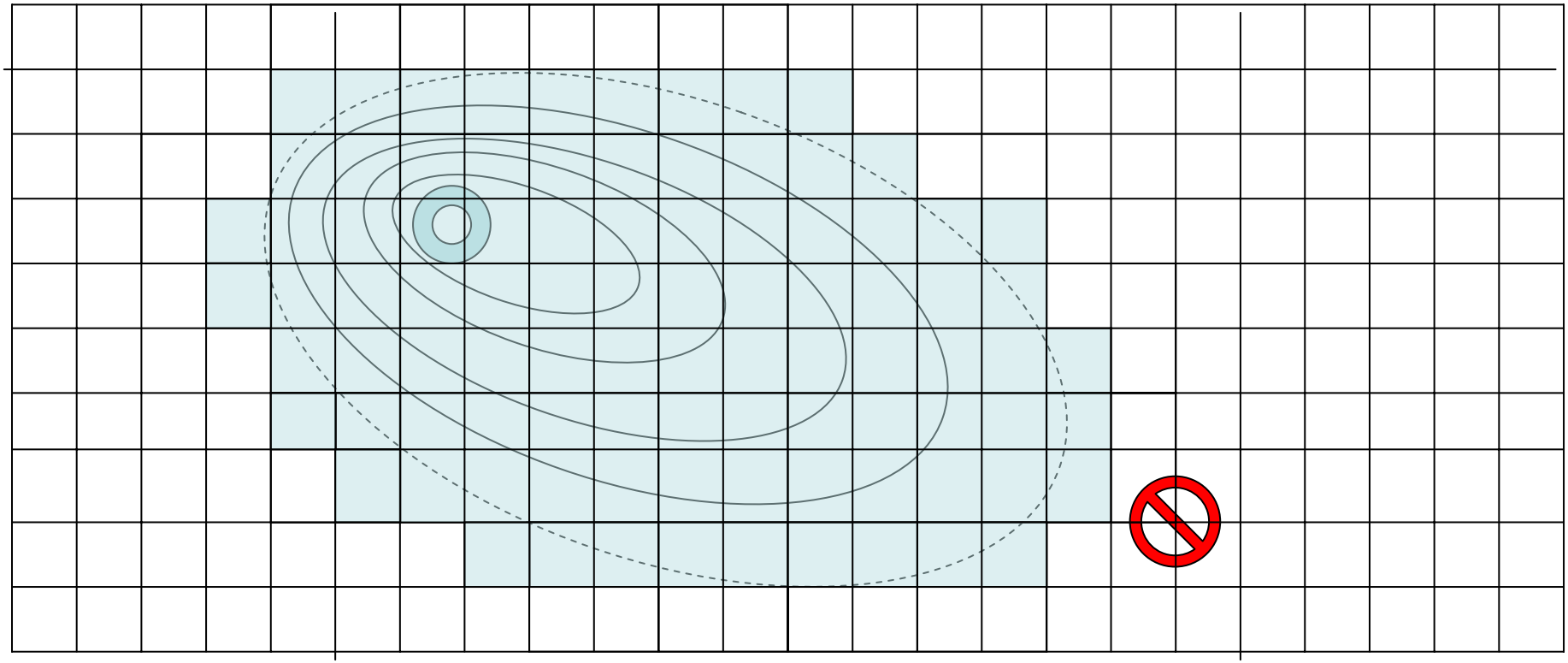


For certain classes of models (finite automata, timed automata, classes of hybrid automata…)

If exploration does not terminate..



Refine approximation



Success story in formal verification

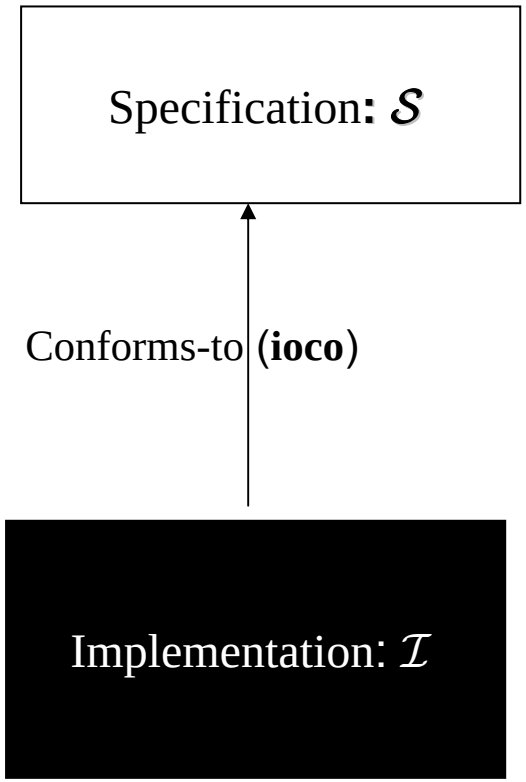
Cf. SLAM (Microsoft)

Still an active research domain

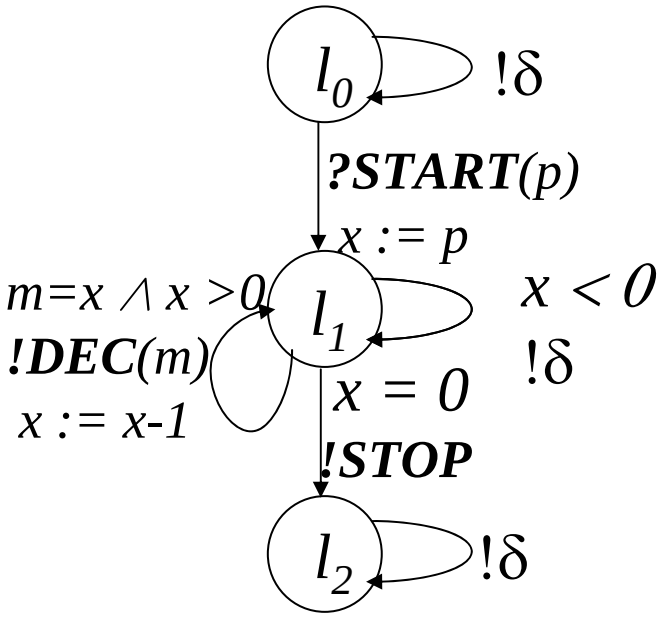
Verification by Theorem Proving: "Invariant Strengthening"

- Goal: find predicate Θ
  - *Invariant* (closed) under $\longrightarrow$
  - Includes 
  - Does not intersect 
- Start with Θ = 
  - Failed invariance proofs: *auxiliary predicates* $\mathcal{A}$
  - Continue with $\Theta := \neg \mathcal{A}$ until proof (or… too tired)
- Also with compositional reasoning, partial-order reduction: SSCOP protocol (3 months) [Computer Journal'06].

Conformance Testing



$\mathcal{I}$ ioco $\mathcal{S}$: after all traces of $\delta(\mathcal{S})$, outputs of $\delta(\mathcal{I}) \subseteq$ outputs of $\delta(\mathcal{S})$



?START(0)

!STOP

?START(0)

?START(0)

!DEC(0)

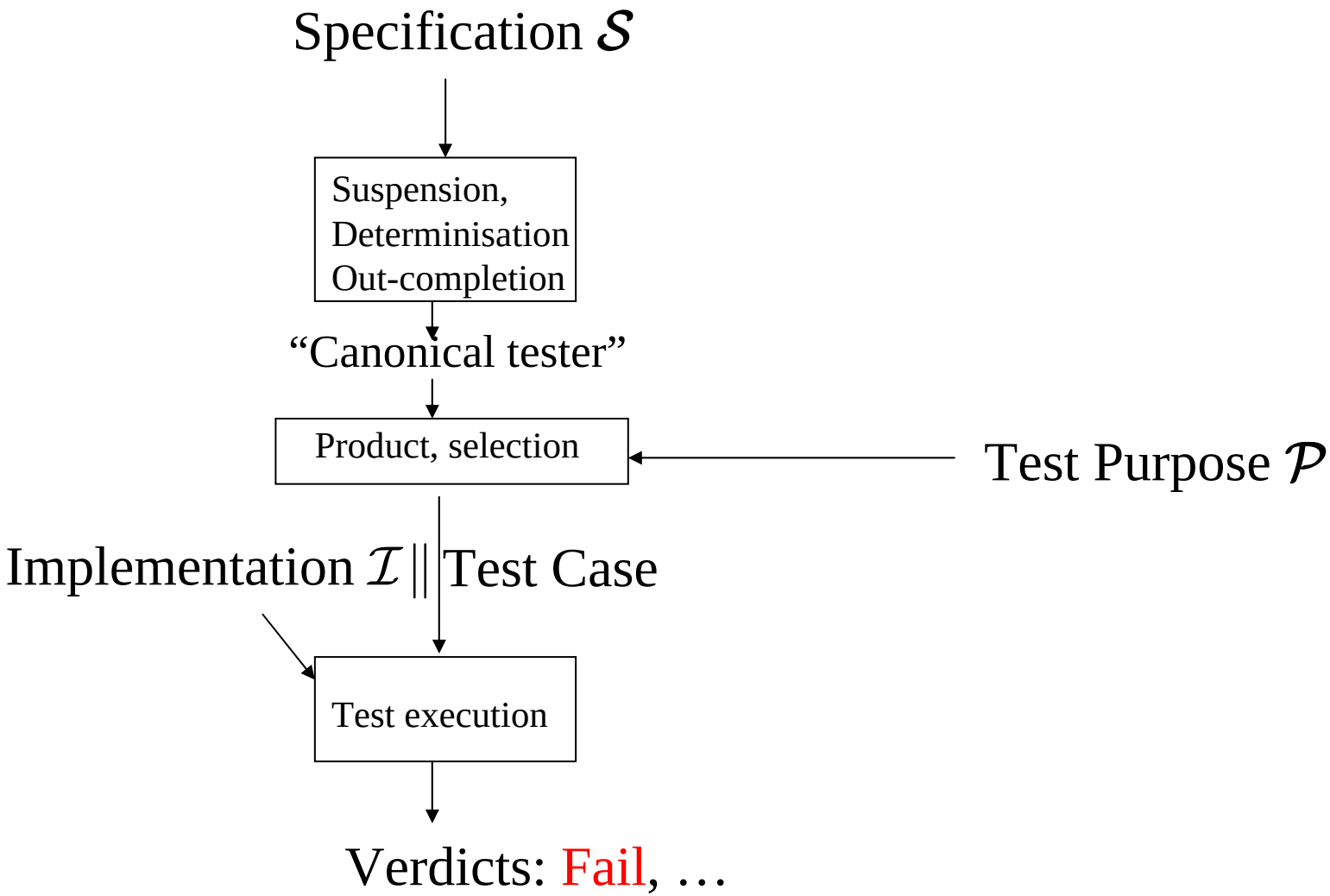
?START(0)

!DEC(0)

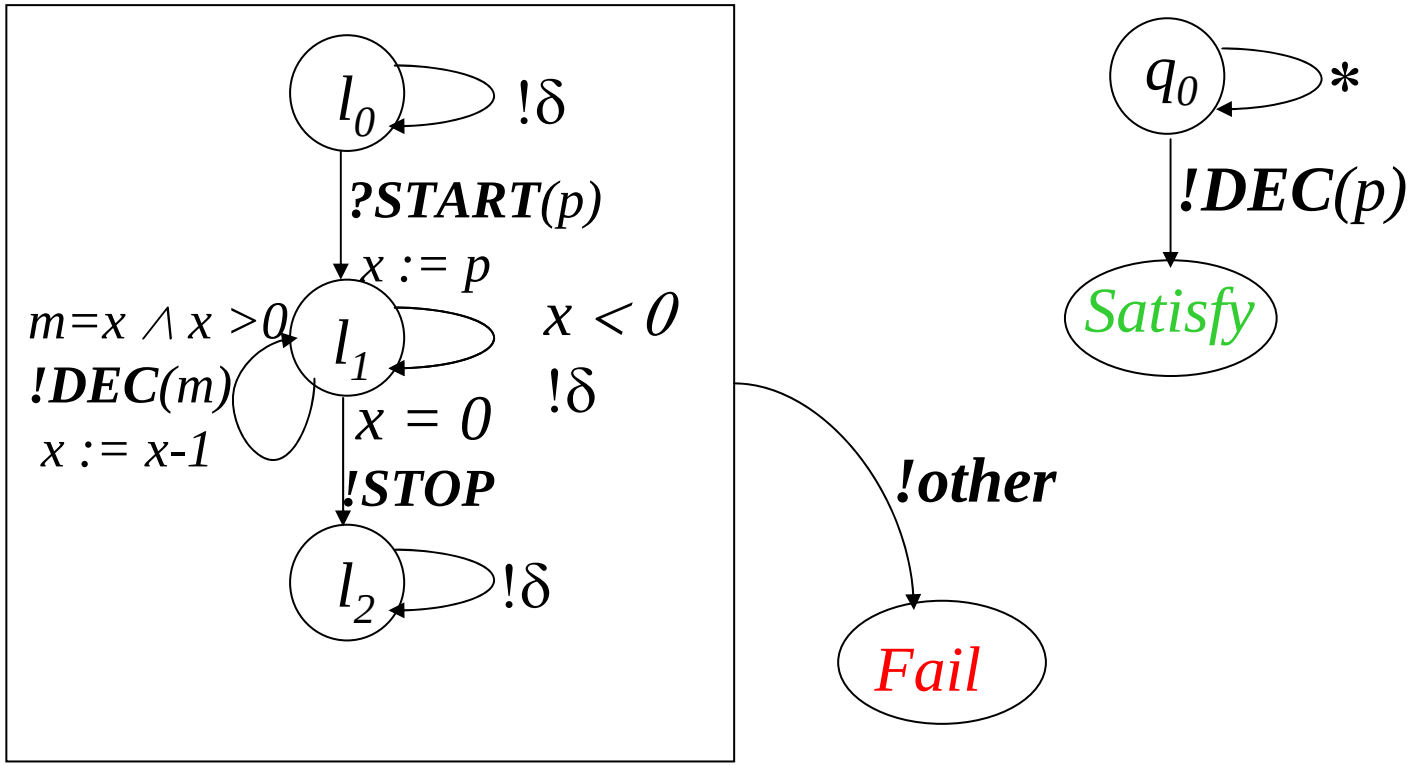
?START(0)

!δ

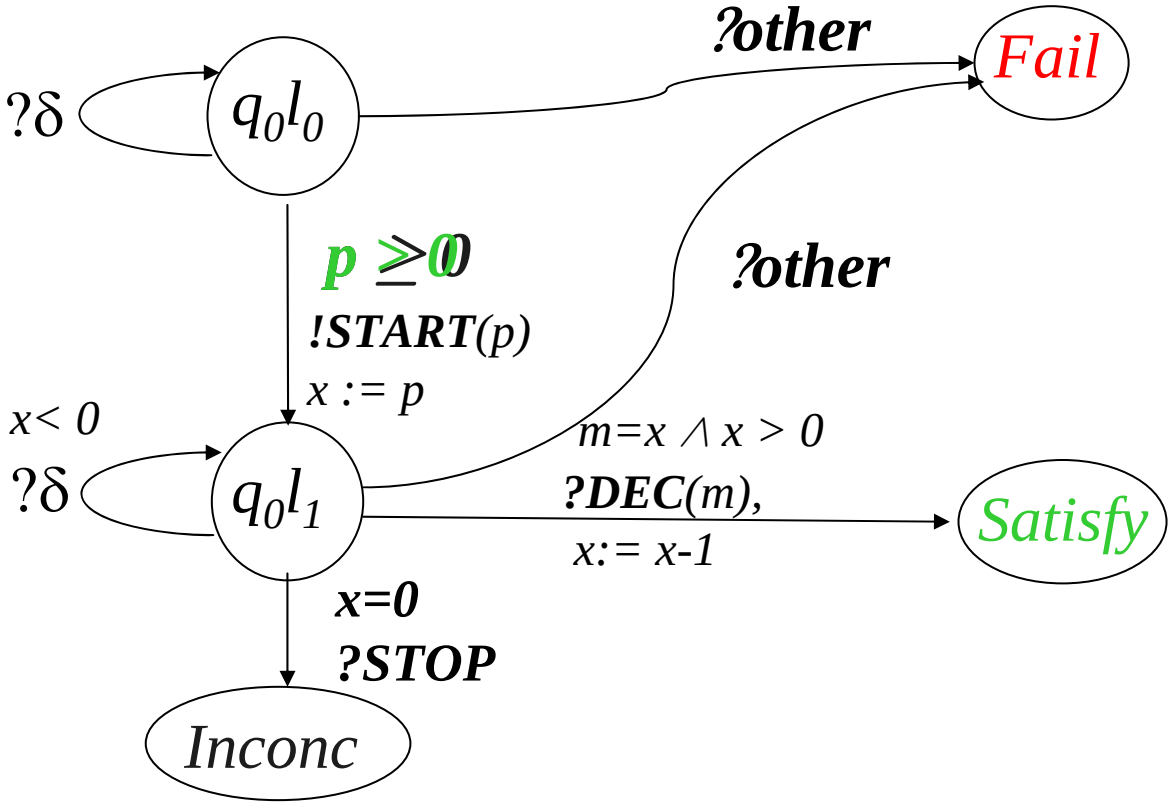
Symbolic Test Generation



Back to example: specification, test purpose



Resulting Symbolic Test Case



Papers on Symbolic Test Generation

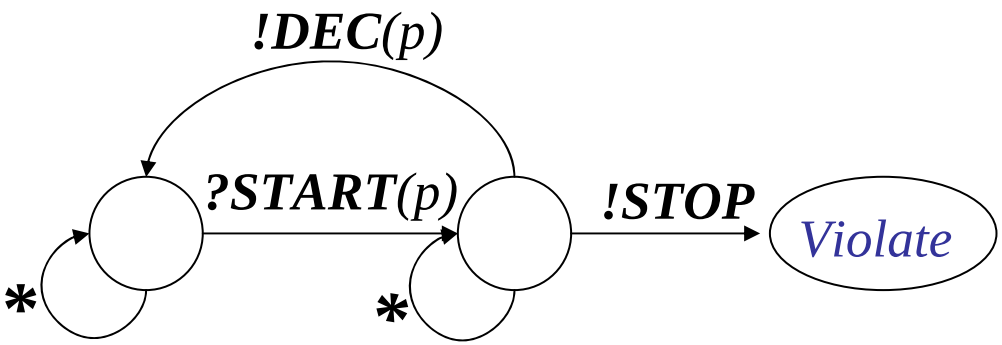
- Theory: [Integrated Formal Methods'00, Tacas'05, IFIP/TCS'06]
- STG tool [Tacas'02]
- Main case study: Electronic purse [e-smart'01]

Towards integrating verification and conformance testing

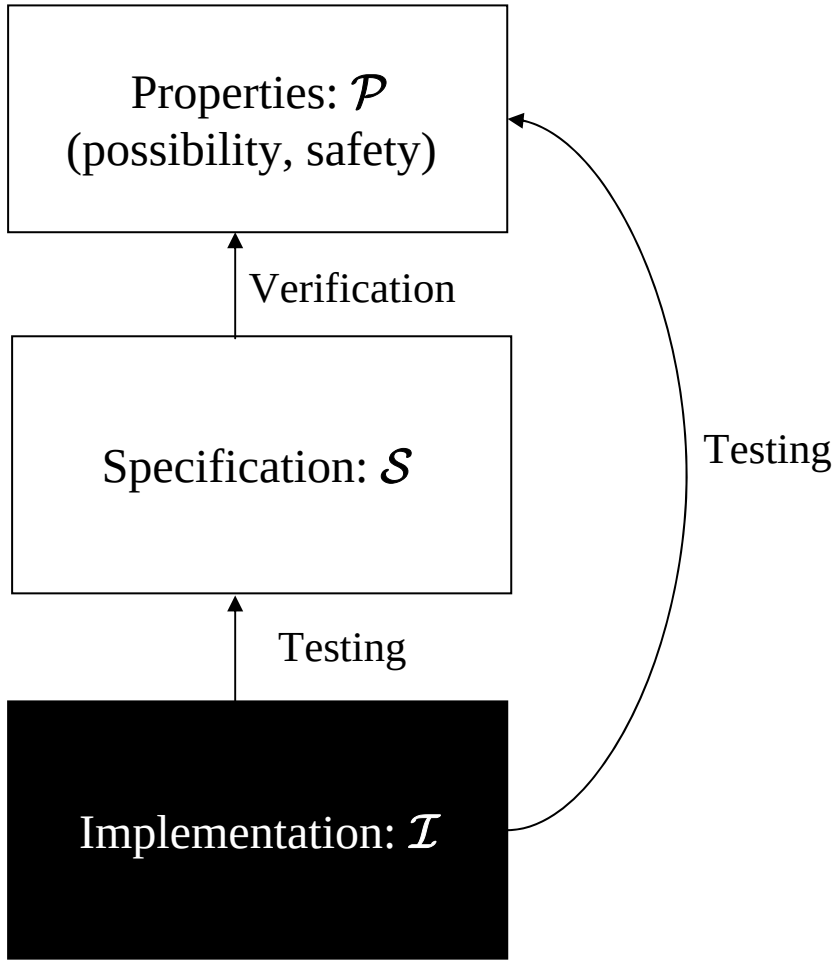
- "Test purpose": a *possibility* property of the specification: certain traces are *possible*
- More (most?) interesting properties: *safety*
- Different interpretation of *final locations*
- Observers: standard approach in verification.

Example: observer for a safety property

No *!STOP* between *?START* and *!DEC*



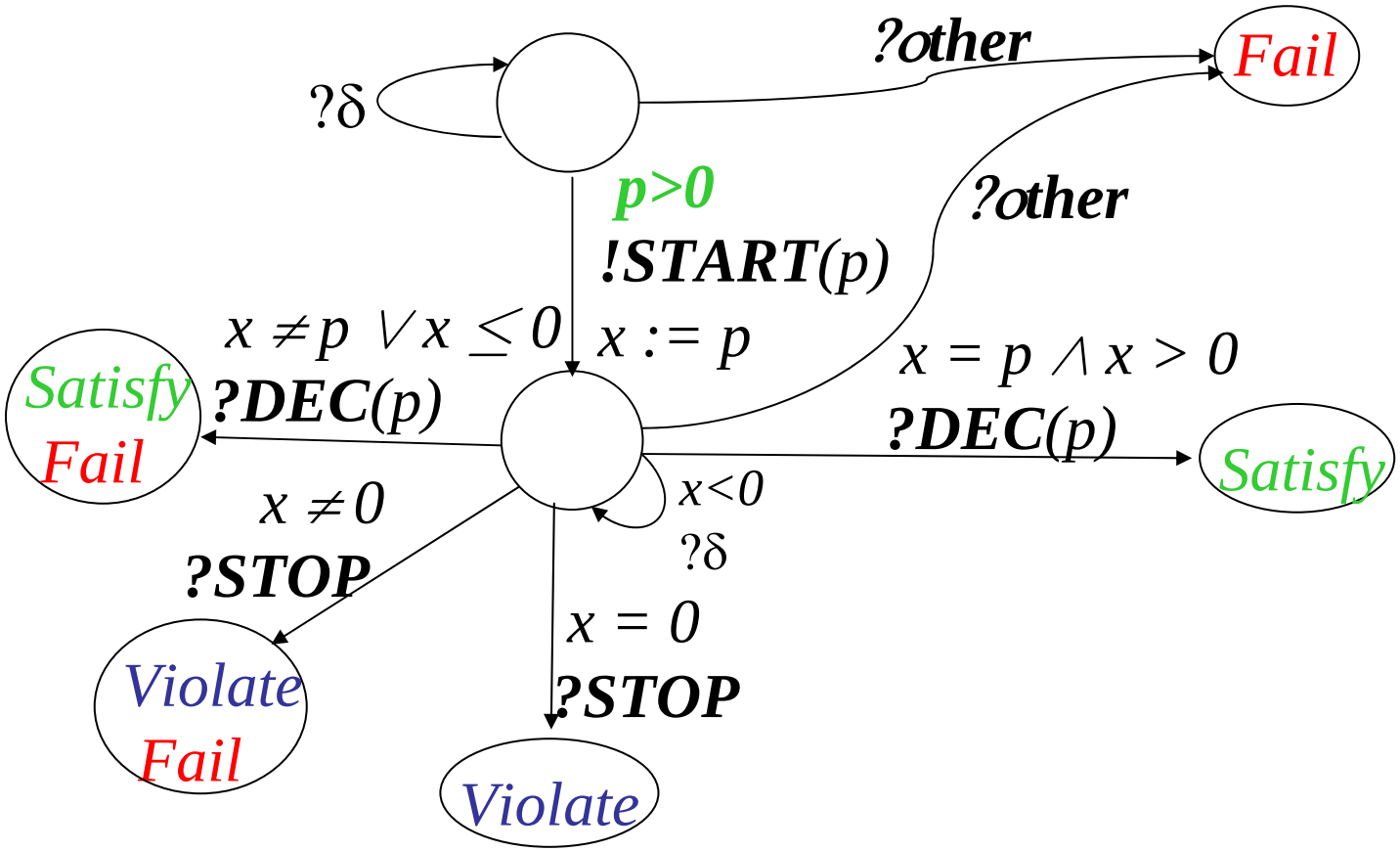
Verification *and* Conformance Testing



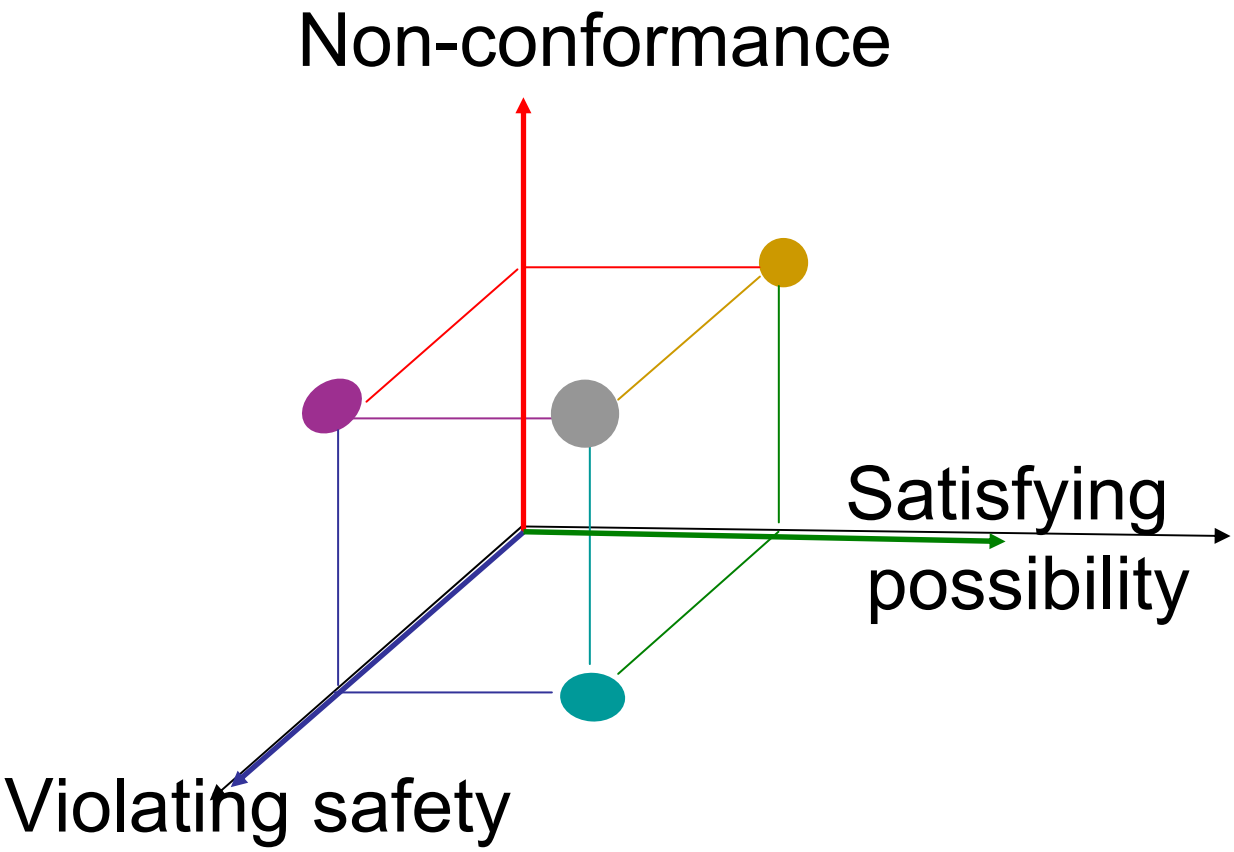
Methodology

- Verify $\mathcal{S}$ against (observers for) properties $\mathcal{P}$
  - Build their product $\times$, check reachability of final location(s)
  - Under-approximation (e.g. model checking) to prove reachability
  - Over-approximation (e.g. abstract interpretation) to disprove it
- Whether verification conclusive or not! test generation
  - Transform $\mathcal{S}$ into observer for nonconformance: “canonical tester”
    - Suspension, Determinisation, Output-completion
  - Product with observers for properties $\mathcal{P}$ **:** lots of verdicts!
- Test selection: choose among verdicts, compute co-reachability (abstraction interpretation again)
- Test execution: may complete verification.

Test generation: product, selection



Interpretation of verdicts



Summary: integrating verification and conformance testing…

- Establishes relative consistency between implementation, specification, properties
- Testing step does not depend on success of verification
  - Can even be done all at the same time
- [Formal Methods'05, Chap. 2 in Traité Hermès I2C, 2006]

Some General Conclusions

- Verification and testing are complementary
  - Operations, methodology
- Integration of methods is still the future
  - Also with control synthesis, fault diagnosis…
- Main issues to wider application
  - Complexity/limits of tools
  - Lack/incompleteness of formal specifications
  - But promising start in certain areas/industries.

Perspectives

- In conformance testing:
  - Coverage
  - More expressive models (time, recursion, …)
  - Compositionality
  - Testing and games
  - Target application: security
- In verification:
  - Build links with semi-formal methods, notations
  - “Invisible formal methods” [Rushby]
- Even more integration
  - To deal with incomplete/missing specifications: *learning.*