KTH Computer Science
and Communication

Incremental Learning-Based Testing for Reactive Systems

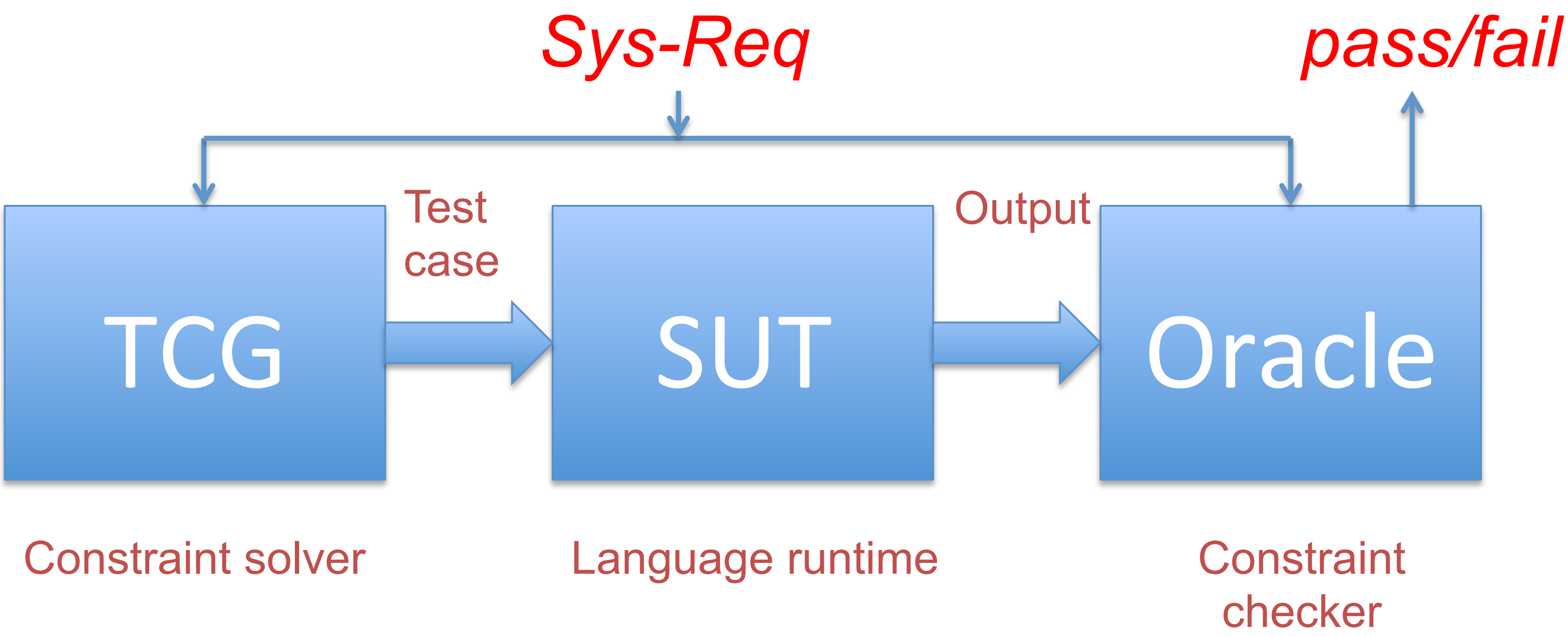
Karl Meinke, Muddassar Sindhu
Royal Institute of Technology (KTH)
Stockholm

0. Overview of Talk

1. Specification Based Black-box Testing
2. Learning Based Testing paradigm (LBT)
   - connections between learning and testing
   - testing as a search problem
   - testing as an identification problem
   - testing as a parameter inference problem
3. Chosen Framework: reactive systems
4. Results
5. Conclusions

1. Specification Based Black-Box Testing

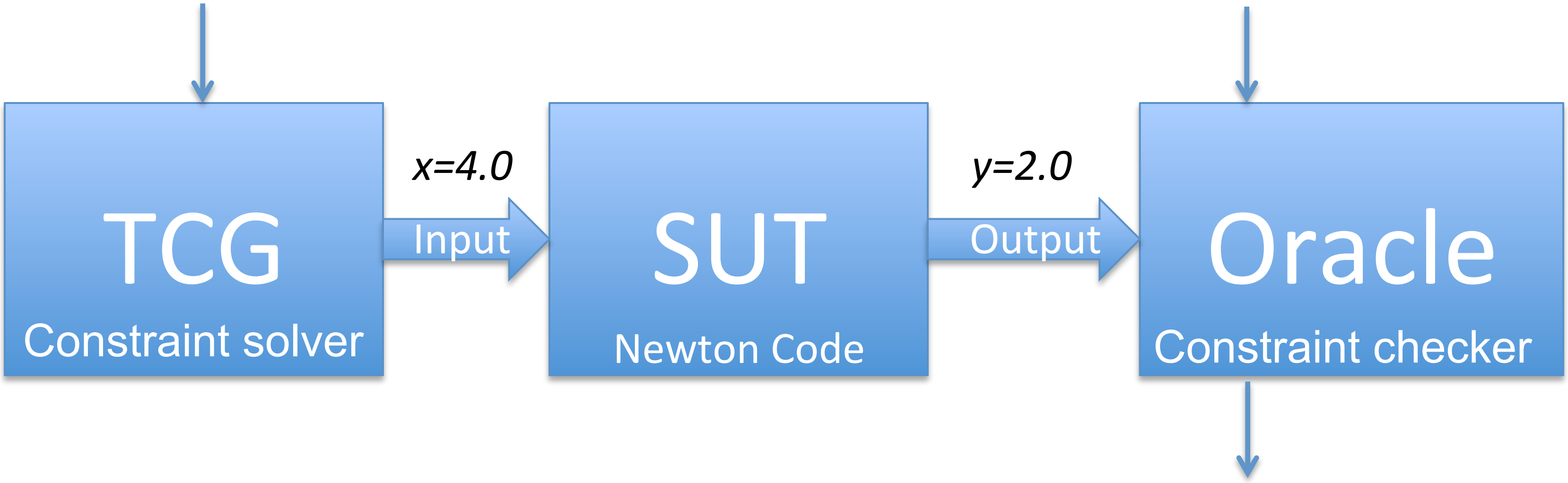
1. System requirement (*Sys-Req*)
2. System under Test (*SUT*)
3. Test verdict pass/fail (*Oracle step*)



1.1. Procedural System Example:
Newton's Square Root Algorithm

Postcondition
$| y*y - x | \leq \varepsilon$

precondition $x \geq 0.0$



$x=4.0$ satisfies $x \geq 0.0$

Verdict
$x=4.0, y=2.0$ satisfies
$| y*y - x | \leq \varepsilon$

1.4. Reactive System Example: *Coffee Machine*

Sys-Req: always(in=$1 implies after(10, out=coffee))   *pass/fail*

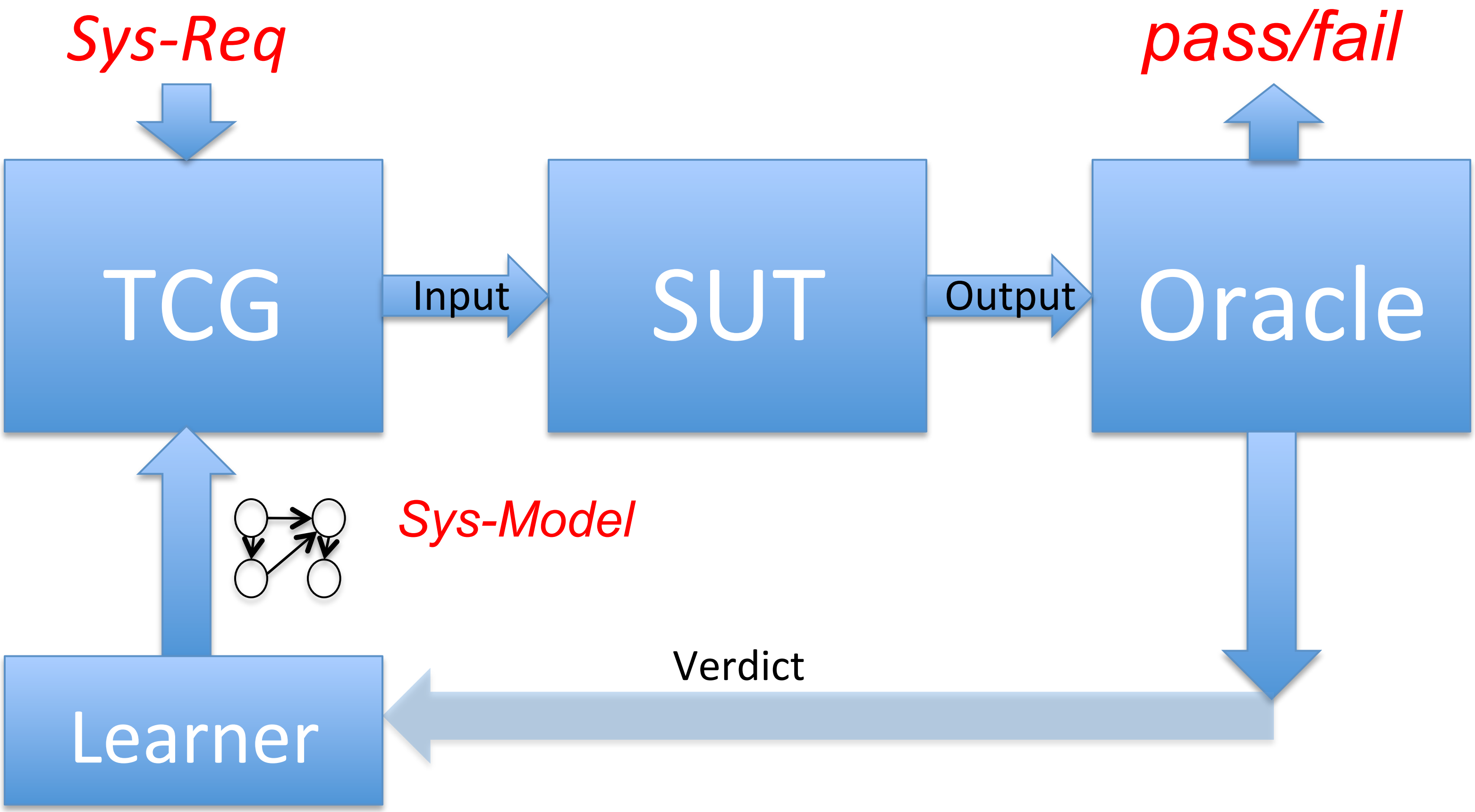
in_0=$1, out_{11}= coffee Satisfies
always(in=1$ implies after(10, out=coffee))

1.2. Key Problem: Feedback

Problem: How to modify this architecture to..

1. Improve next test case using previous test outcomes
2. Execute a large number of good quality tests?
3. Obtain good coverage?
4. Find bugs quickly?

2. Learning-Based Testing



"Model based testing without a model"

2.1. Basic Idea …

LBT is a search heuristic that:

1. Incrementally learns an SUT model
2. Uses generalisation to predict bugs
3. Uses best prediction as next test case
4. Refines model according to test outcome

2.2. Abstract LBT Algorithm

1. Use $(i_1, o_1), \ldots, (i_k, o_k)$ to learn model M_k
2. Model check M_k against *Sys-Req*
3. Choose "best counterexample" i_{k+1} from step 2
4. Execute i_{k+1} on SUT to produce o_{k+1}
5. Check if (i_{k+1}, o_{k+1}) satisfies *Sys-Req*
   a) Yes: terminate with i_{k+1} as a bug
   b) No: goto step 1

Difficulties lie in the technical details …

2.3. General Problems

Difficulty is to find combinations of

models, requirements languages and Sat algorithms

(*M, L, A*)

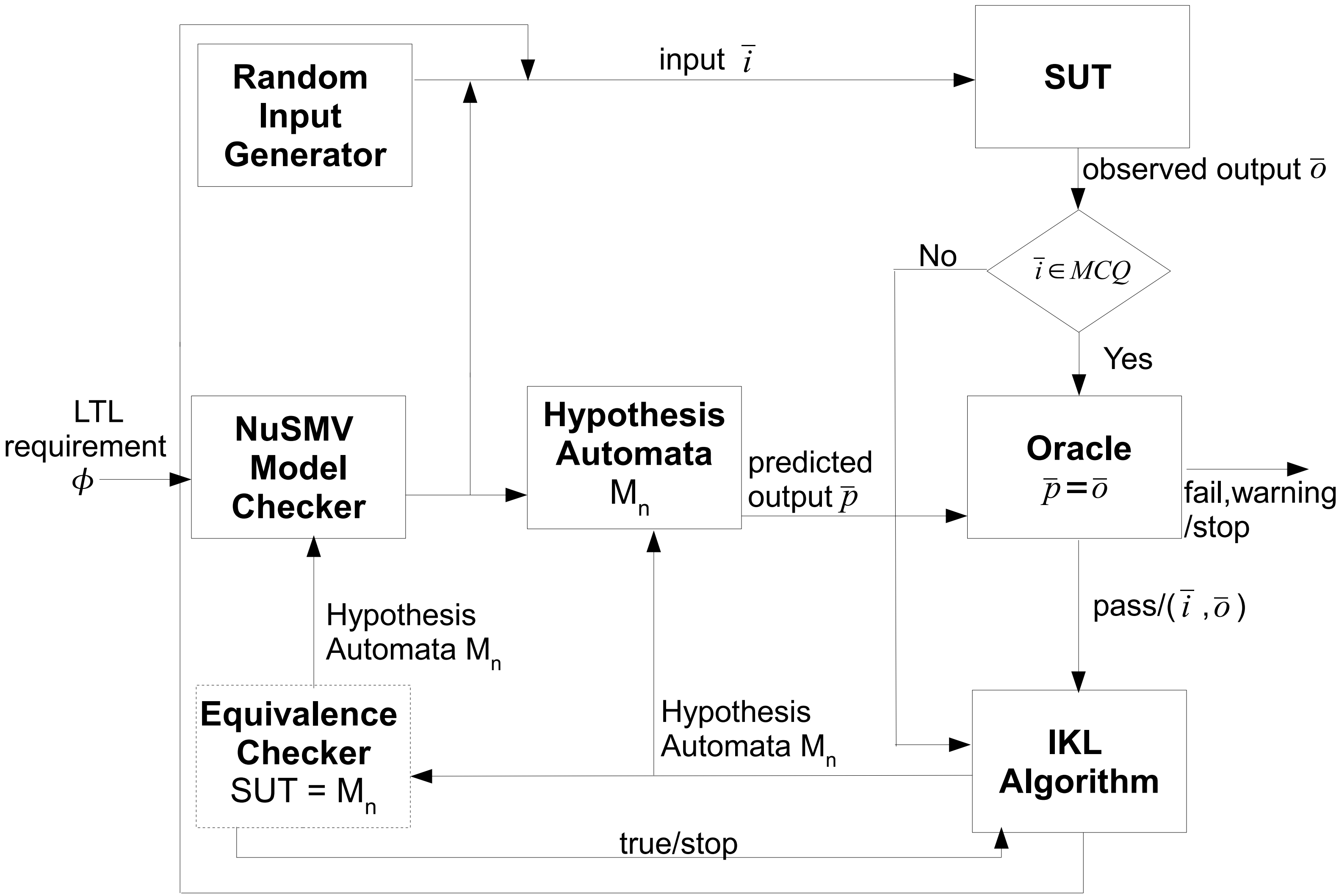
so that …

1. models M are:
   - expressive,
   - compact,
   - partial and/or local (an abstraction method)
   - easy to manipulate and learn
2. M and L are feasible to model check with A

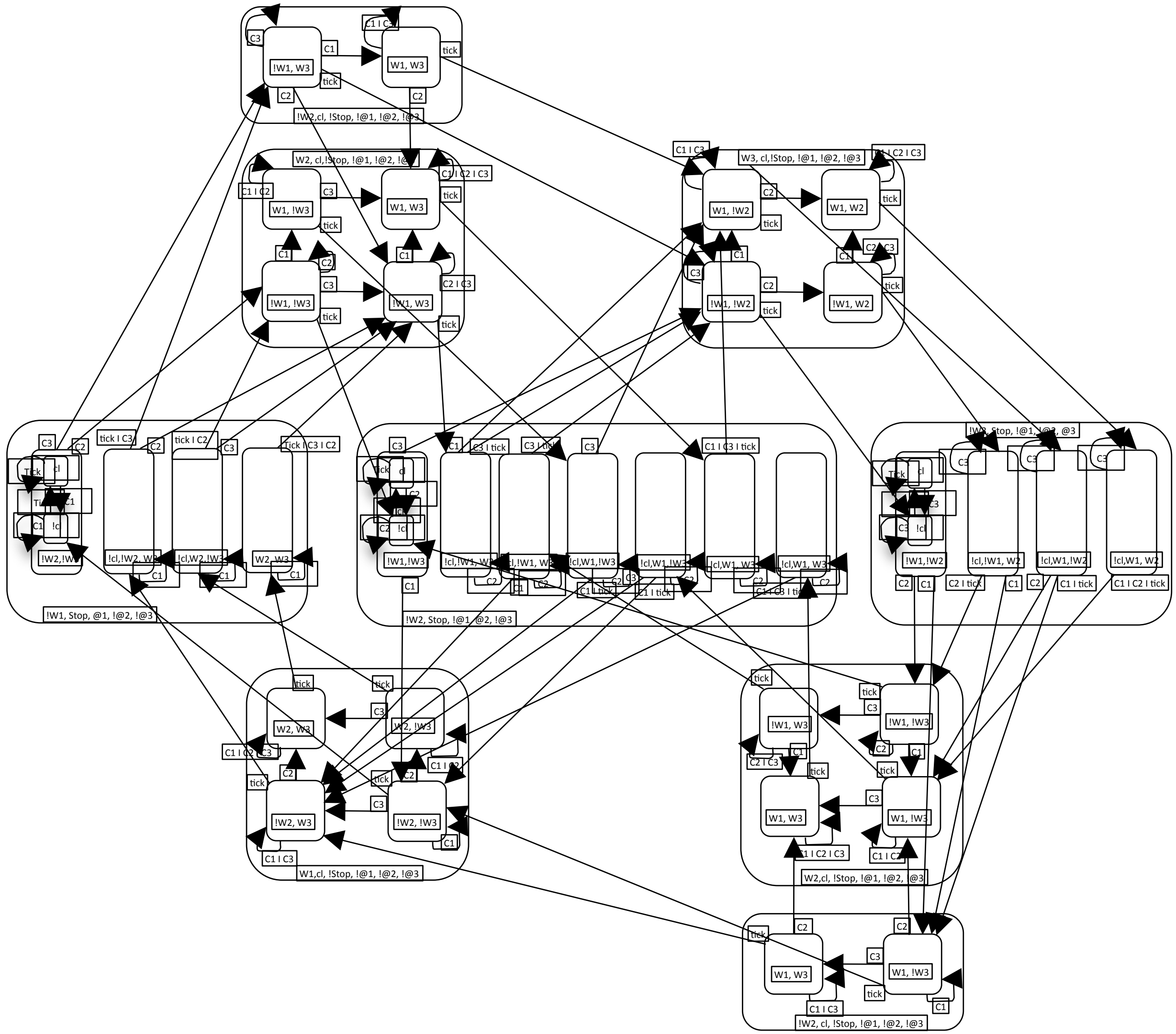
3. Chosen Framework for Study:

1. SUT = reactive system
2. Model = deterministic Kripke structure
3. Sys-Req Lang = linear temporal logic (LTL)
4. Learning = IKL incremental learning algorithm
5. Model Checker = NuSMV

LBT Architecture



A Case Study: Elevator Model



Elevator Results

Req	t $_{first}$ (sec)	t $_{total}$ (sec)	MCQ $_{first}$	MCQ $_{tot}$	PQ $_{first}$	PQ $_{tot}$	RQ $_{first}$	RQ $_{tot}$
Req 1	0.34	1301.3	1.9	81.7	1574	729570	1.9	89.5
Req 2	0.49	1146	3.9	99.6	2350	238311	2.9	98.6
Req 3	0.94	525	1.6	21.7	6475	172861	5.7	70.4
Req 4	0.052	1458	1.0	90.3	15	450233	0.0	91
Req 5	77.48	2275	1.2	78.3	79769	368721	20.5	100.3
Req 6	90.6	1301	2.0	60.9	129384	422462	26.1	85.4

5. Conclusions

- A promising approach …
- Flexible general heuristic,
  - many models and requirement languages seem possible
- Many SUT types might be testable
  - procedural, reactive, real-time etc.

Open Questions

- Benchmarking?
- Scalability? (abstraction, infinite state?)
- Efficiency? (model checking and learning?)