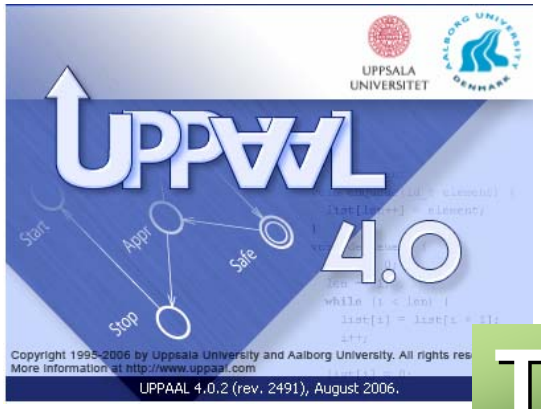


TRON

Real Time Testing

using UPPAAL

With

Mariius Mikucionis, Brian Nielsen, Arne Skou, Anders Hessel, Paul Pettersson

AALBORG UNIVERSITY DENMARK

BRICS
Basic Research
in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

Overview

- Introduction
- Conformance for Real-Time System

- Off-line Test Generation
  - Controllable Timed Automata CLASSIC CORA
  - Observable Timed Automata TIGA

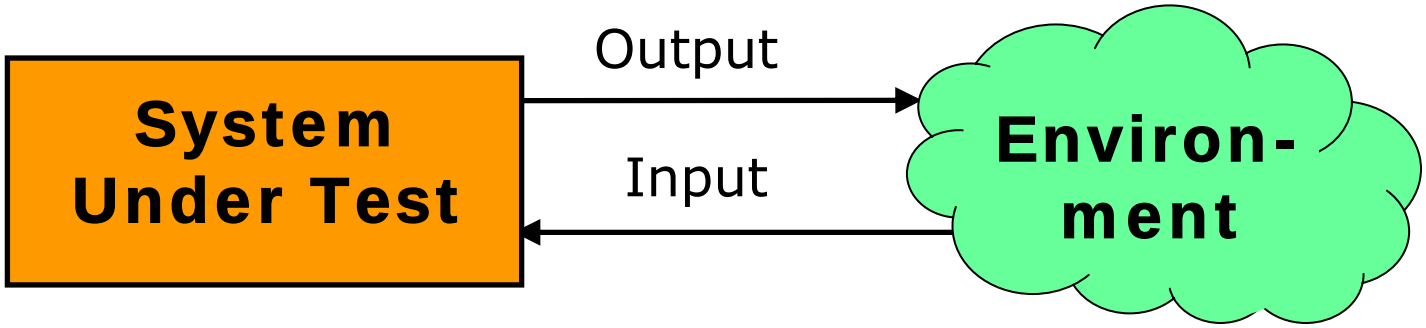
- On-line Test Generation TRON

- Conclusion and Future Work

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Testing

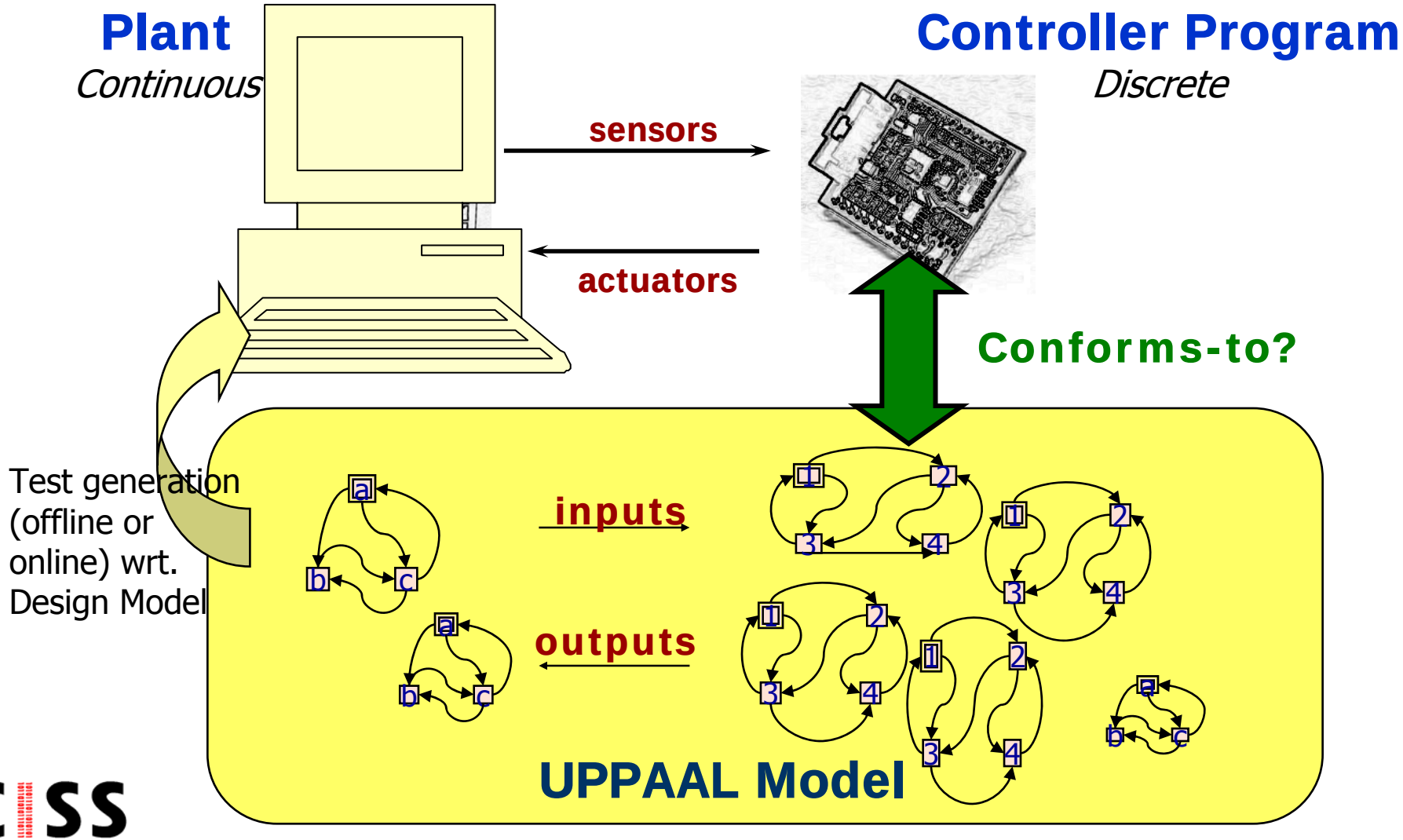
- Primary validation technique used in industry
  - In general avg. 10-20 errors per 1000 LOC
  - 30-50 % of development time and cost in embedded software
- To find errors
- To determine risk of release
- Part of system development life-cycle



- Expensive, error prone, time consuming (for Real-Time Systems)
- UPPAAL model can be used to generate test specifications

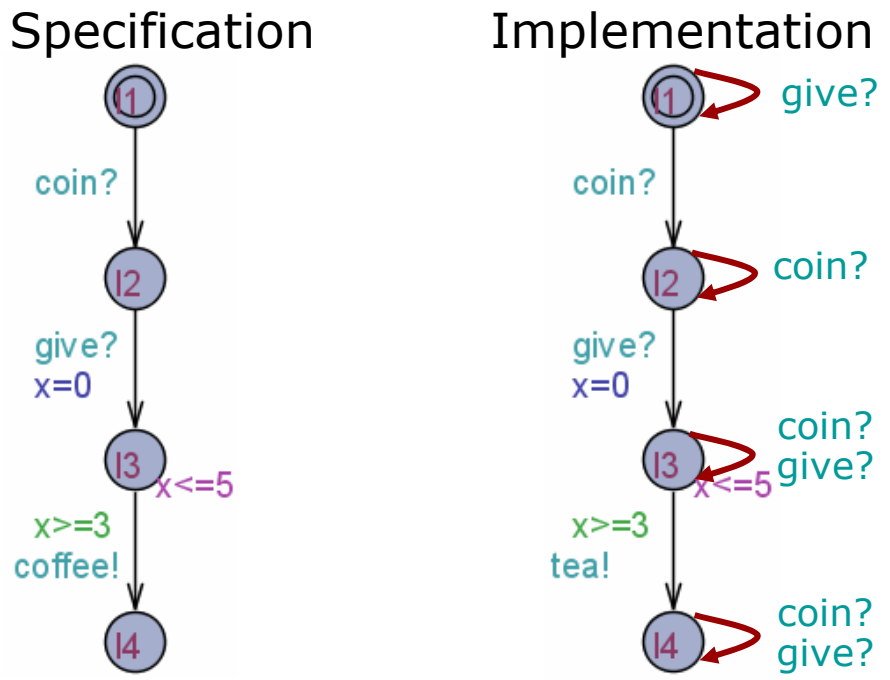
BRICS
Basic Research
in Computer Science

Real-time Model-Based Testing



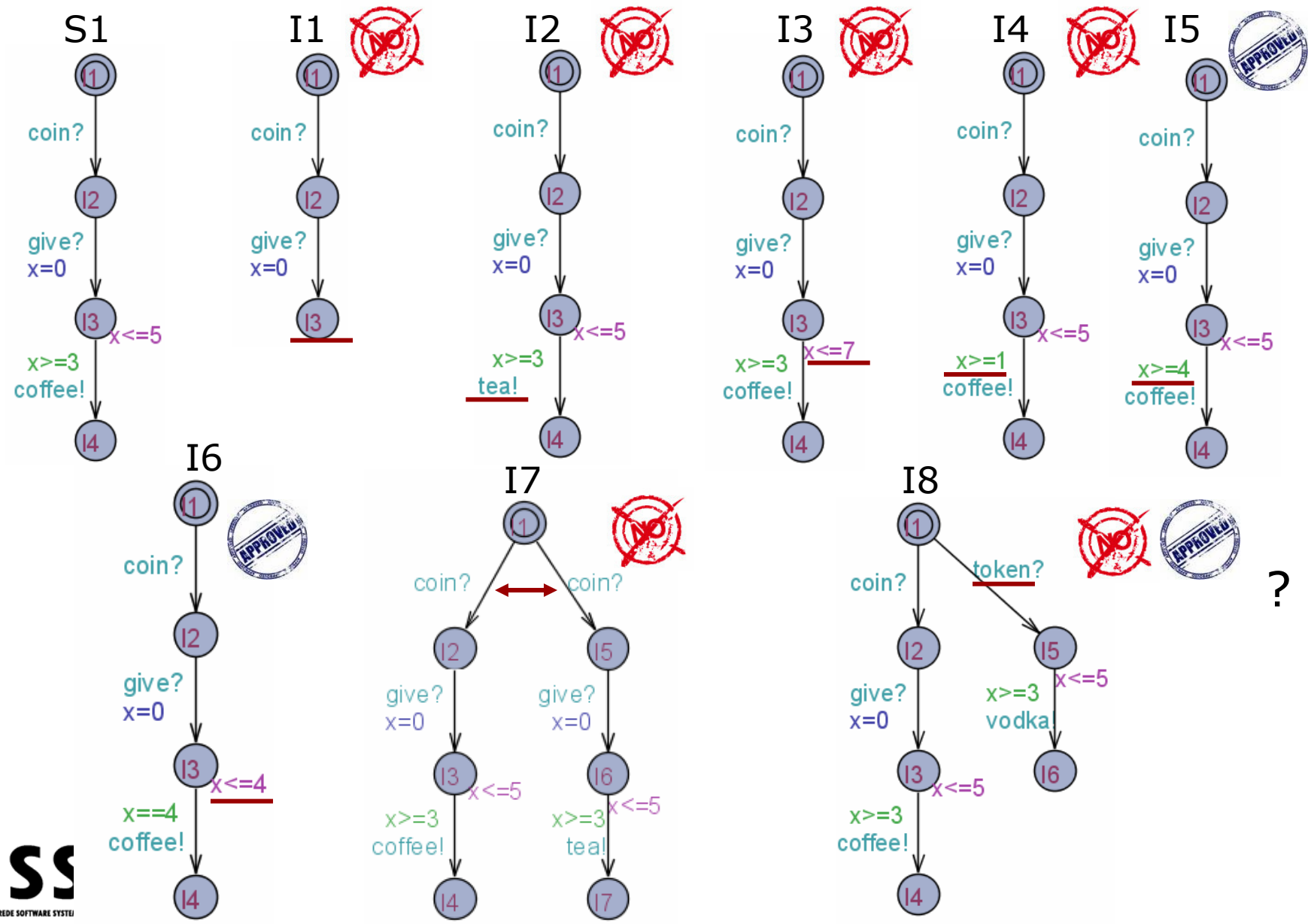
CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Conformance Relation

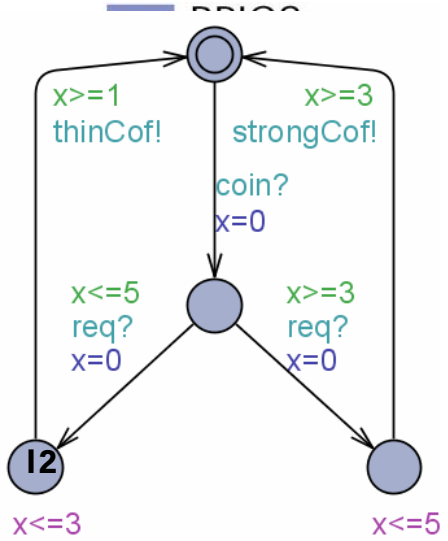


- Timed Automata with Timed-LTS semantics
- **Input** actions (?) are controlled by the environment
- **Output** actions (!) are controlled by the implementation
- Implementations are *input enabled*
- ***Testing hypothesis:*** IUT can be modeled by some (unknown) TA

Does I_n conform-to S_1 ?



Timed Conformance



- Derived from Tretman's IOCO

- Let **I, S** be timed I/O LTS, P a set of states
- **TTr**(P): the set of *timed traces* from P
  - eg.: σ = coin?.**5**.req?.**2**.thinCoffee!.**9**.coin?
- **Out**(P **after** σ) = possible ***outputs*** and ***delays*** after σ
  - eg. out ({l2,x=1}): {thinCoffee, **0...2**}

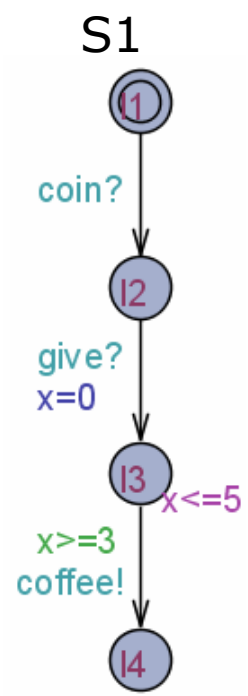
- **I rt-ioco S =def**
  - **∀σ ∈ TTr(S): Out(I after σ) ⊆ Out(S after σ)**
  - **TTr(I) ⊆ TTr(S)** *if s and I are input enabled*

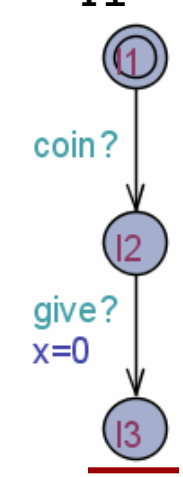
- **Intuition**
  - **no illegal output is produced and**
  - **required output is produced (at right time)**

See also [Krichen&Tripakis, Khoumsi]

CISS CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Does I_n conform-to S_1 ?



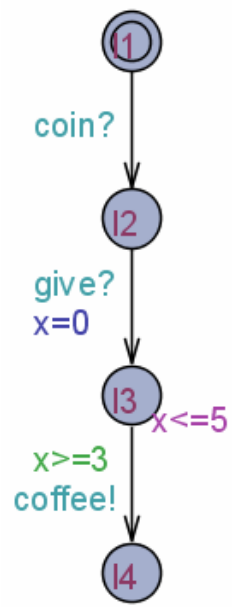


σ=coin.give.10
$\sigma \in$ **TTr**(I1), $\sigma \notin$ **TTr**(S1)

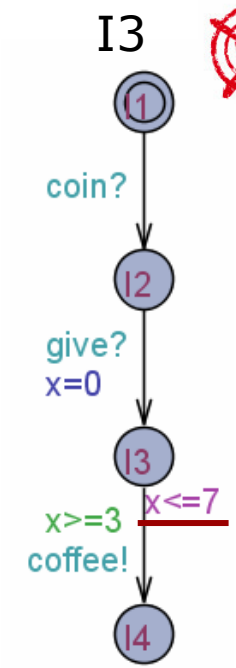
out(I1 **after** coin.give.3)={0... ∞}
$\not\subset$
out(S1 **after** coin.give.3)={coffee,0…2}

Does I_n conform-to S_1 ?

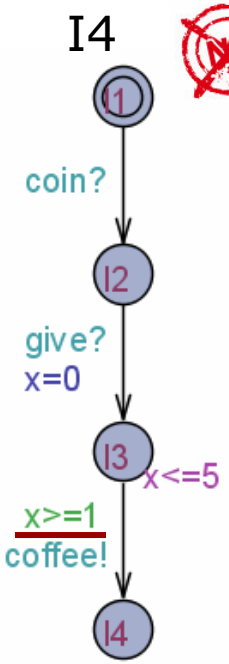
S1



I3



I4



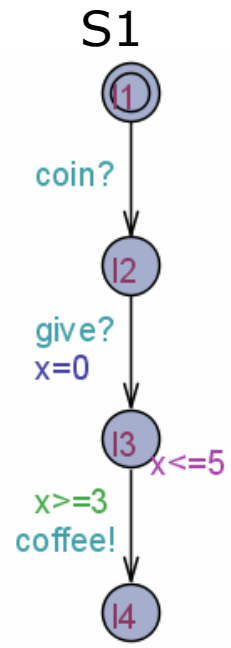
σ=coin.give.7.coffee
σ∈**TTr**(I3), σ ∉TTr(S1)

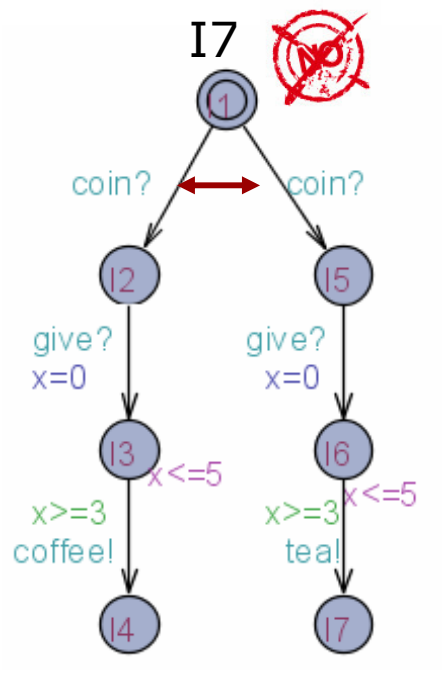
σ=coin.give.1.coffee
σ∈**TTr**(I4), σ ∉TTr(S1)

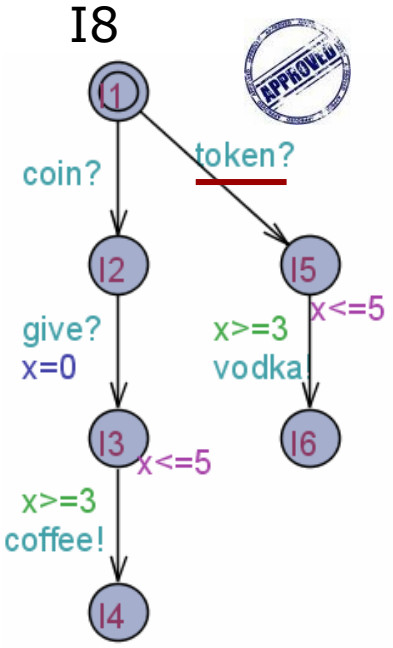
out(I3 after coin.give.7)={coffee,0}
⊄
out(S1 after coin.give.7)={}

out(I4 after coin.give.1)={coffee,0...4}
⊄
out(S1 after coin.give.1)={0...4}

Does I_n conform-to S_1 ?





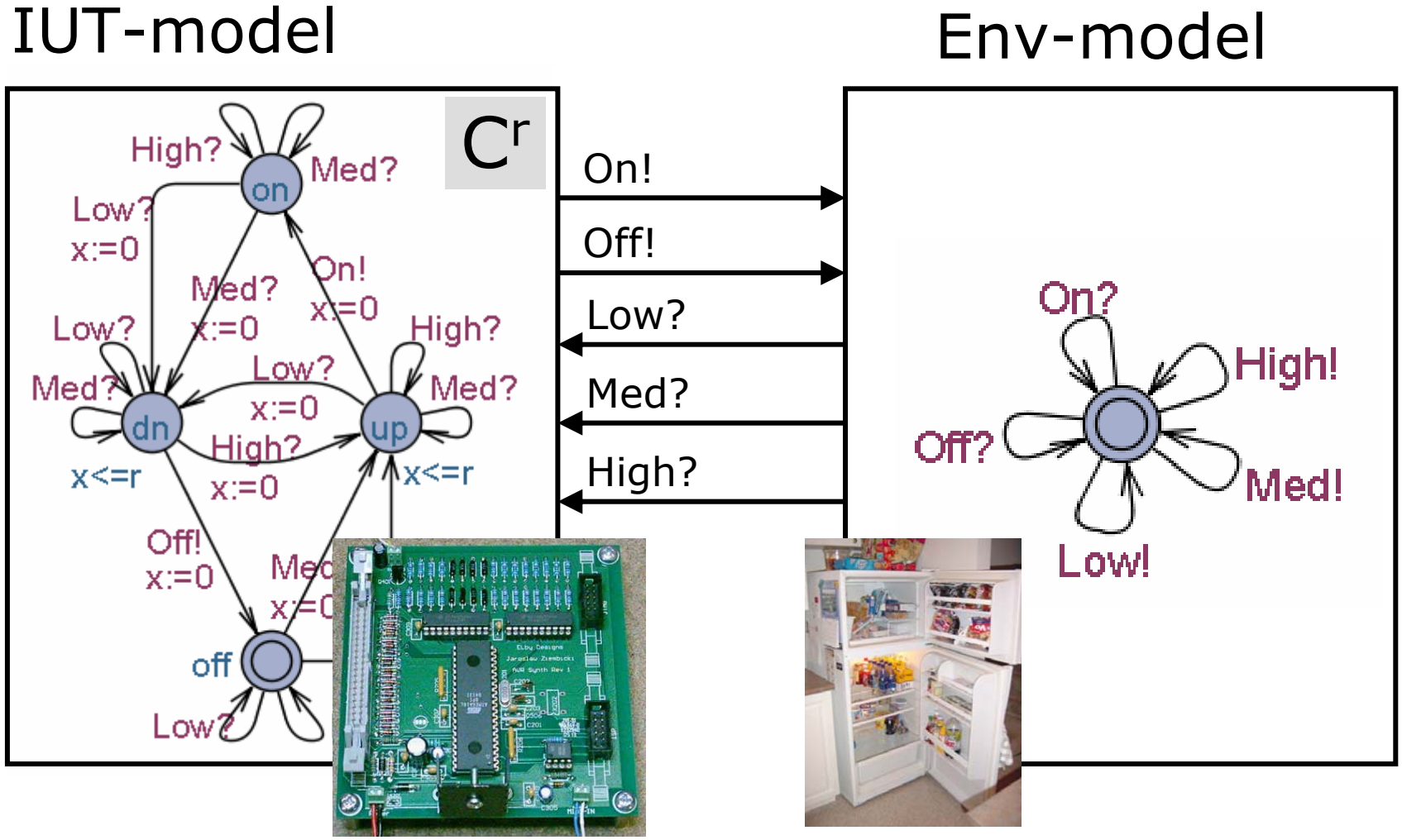


σ=coin.give.5.tea
σ∈**TTr**(I7), σ ∉**TTr**(S1)

σ=token.5.vodka
σ∈**TTr**(I8), σ ∉**TTr**(S1)
But σ was not specified

out(I7 **after** coin.give.5)={tea, coffee,0}
⊄
out(S1 **after** coin.give.5)={coffee,0}

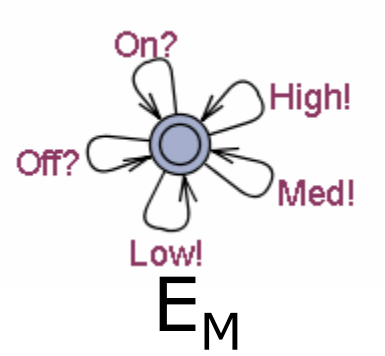
Sample Cooling Controller



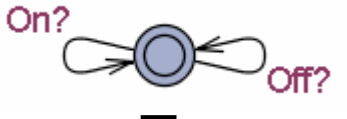
- When T is high (low) switch on (off) cooling within r secs.
- When T is medium cooling may be either on or off (impl freedom)

Environment Modeling

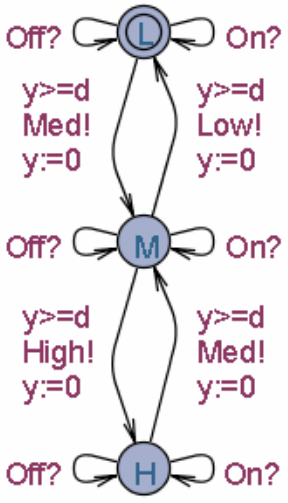
- E_M Any action possible at any time
- E_1 Only realistic temperature variations
- E_2 Temperature never increases when cooling
- E_L No inputs (completely passive)



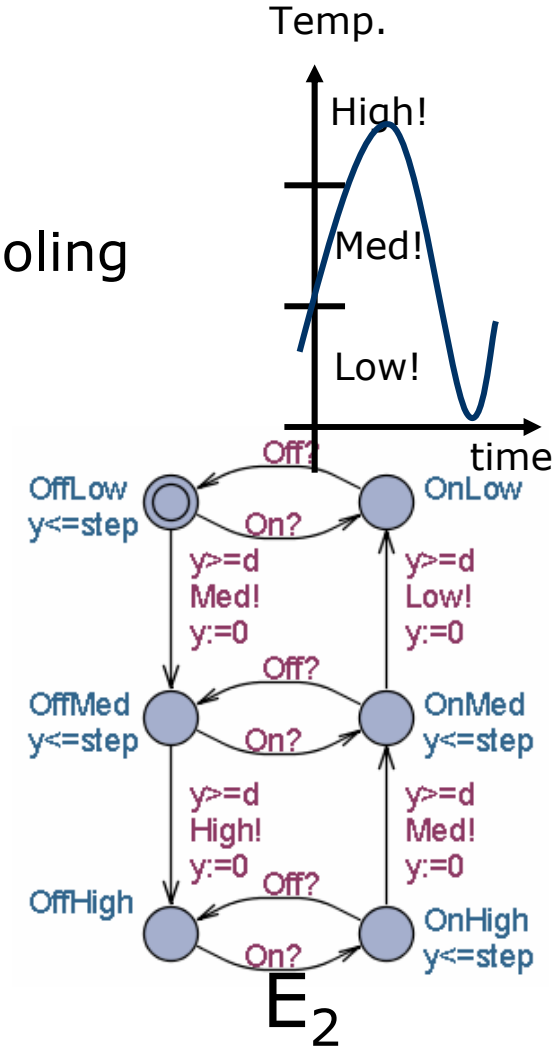
E_M



E_L



E_1

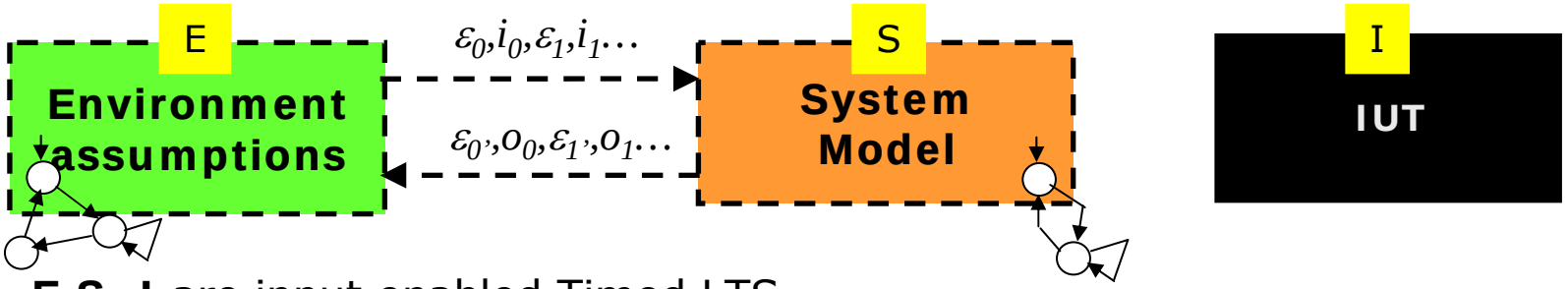


E_2

$$E_L \sqsubseteq E_2 \sqsubseteq E_1 \sqsubseteq E_M$$

Implementation relation

Relativized real-time io-conformance



- **E,S, I** are input enabled Timed LTS
- **Let** *P* **be a set of states**
- **TTr**(*P*): the set of *timed traces* from states in *P*
- *P* **after** σ = the set of states reachable after timed trace σ
- **Out**(*P*) = possible outputs and delays from states in *P*

- I rt-ioco$_E$ S $=_{def}$
  $\forall \sigma \in TTr(E)$: Out((E,I) after σ) $\subseteq$ Out((E,S) after σ)

- I rt-ioco$_E$ s iff $TTr(I) \cap TTr(E) \subseteq TTr(S) \cap TTr(E)$ *// input enabled*

- **Intuition, for all assumed environment behaviors, the IUT**
  - **never produces illegal output, and**
  - **always produces required output in time**

BRICS
Basic Research
in Computer Science

Re-use Testing Effort

- Given I, E, S
- Assume I rt-ioco$_E$ S

1. Given new (weaker) system specification S'

$$\text{If } S \sqsubseteq S' \text{ then I rt-ioco}_E\ S'$$

2. Given new (stronger) environment specification E'

$$\text{If } E' \sqsubseteq E \text{ then I rt-ioco}_{E'}\ S$$

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

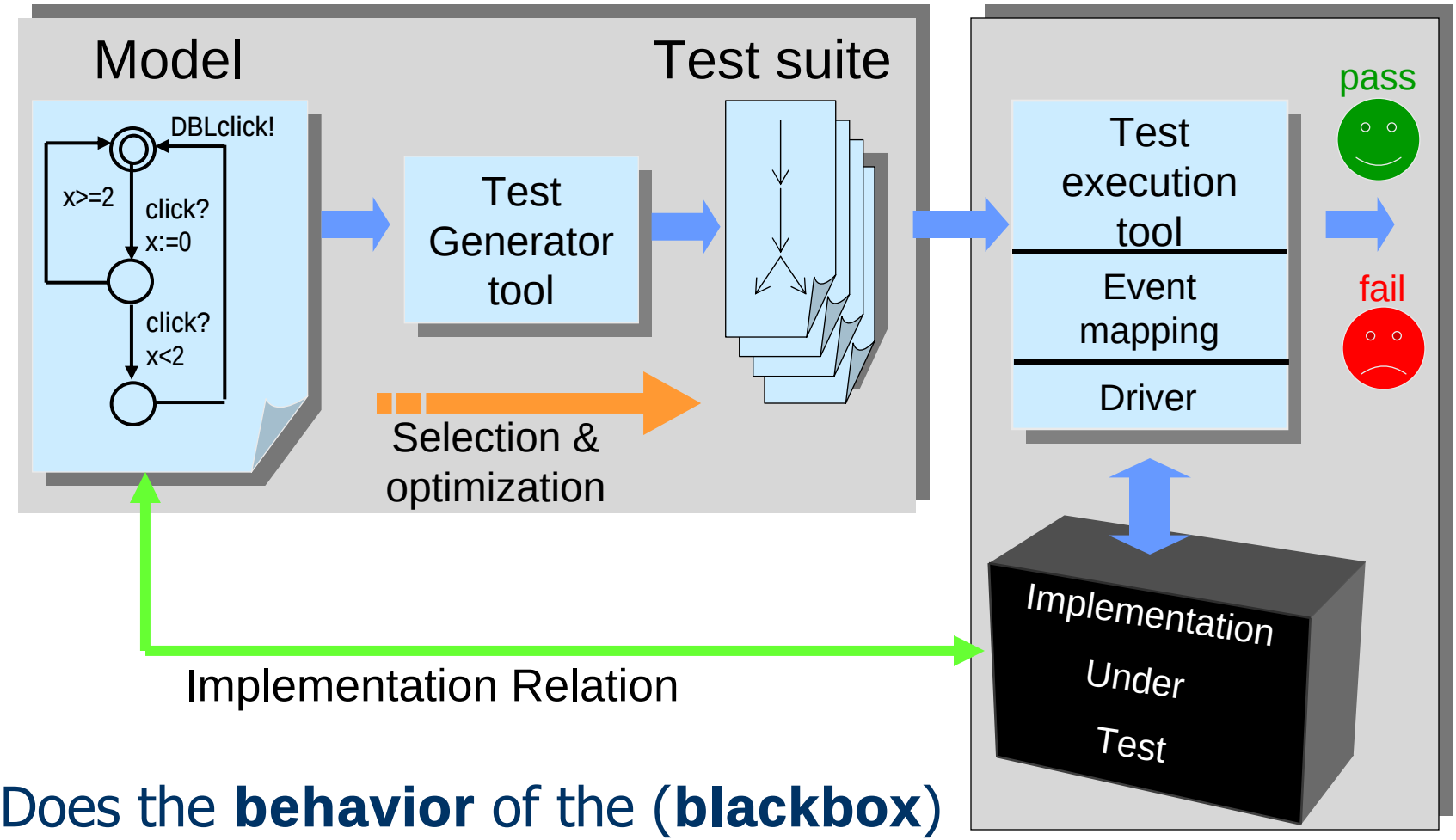
Off-Line Test Generation

Controllable Timed Automata

BRICS
Basic Research
in Computer Science

BRICS
Basic Research
in Computer Science

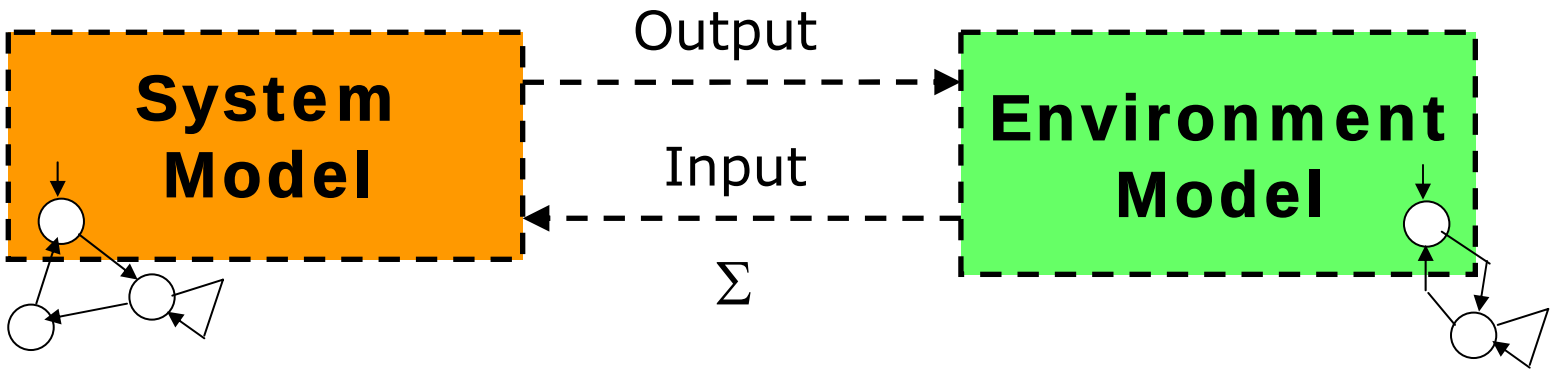
Model Based Conformance Testing



Does the **behavior** of the (**blackbox**) implementation ***comply*** to that of the specification?

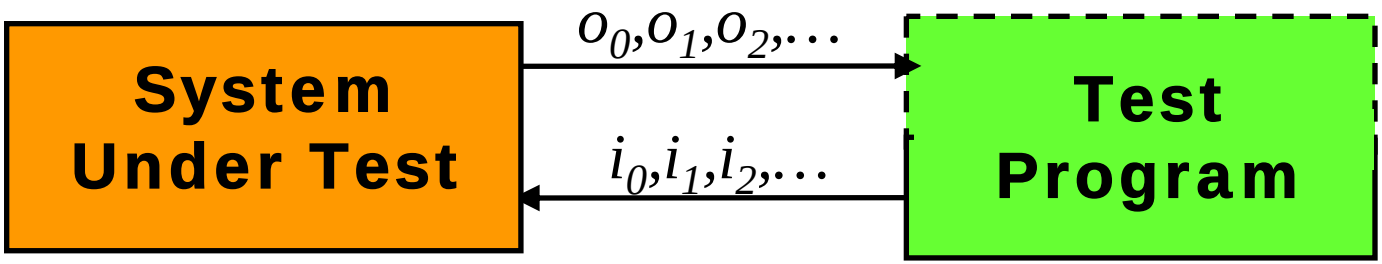
CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Model-Based Testing



- FSM model of system and environment
- Determinizable/deterministic models
- Test purpose $P \approx$ reachability property ϕ_P
- Test-case generation $\approx$ witness generation
- Test input sequence $\sigma_{\phi p} = i_0, i_1, i_2, \ldots$
- Test suite $T = \{\sigma_1, \ldots, \sigma_n\}$, minimized by excluding all σ_i substring of some other σ_j

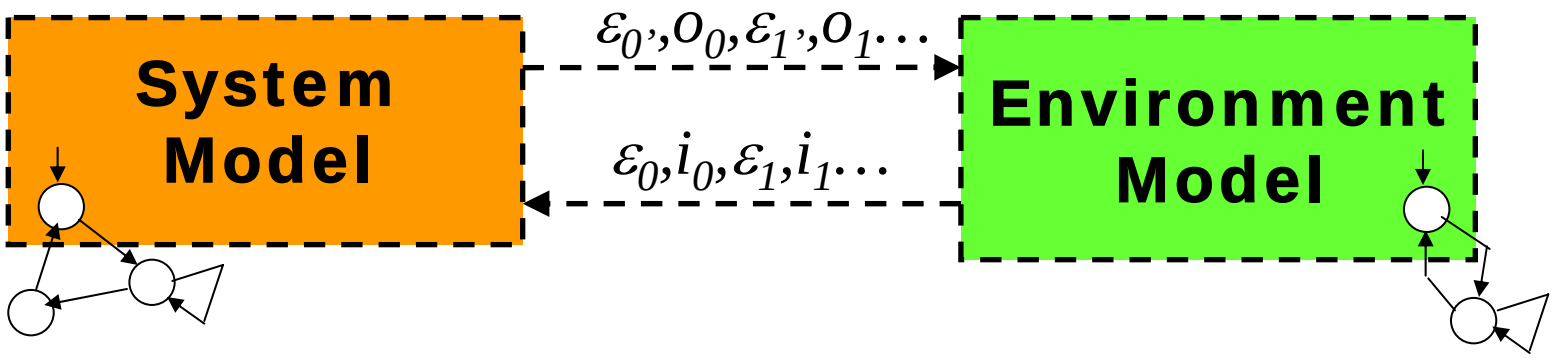
Testing Verdict



- Test program $\sigma_{\phi p} = i_0,i_1,i_2,\ldots$
- Test in/output $\delta_{\phi p} = i_0,o_0,i_1,o_1,i_2,i_3,\ldots$
- Test Verdict:
  - OK, if $\delta_{\phi p} = i_0,o_0,i_1,o_0,i_2,i_3,\ldots$ run of system model
  - NOK, otherwise

BRICS
Basic Research
in Computer Science

Testing Real-Time Systems



- Test input sequence $\sigma_{\phi p} = \varepsilon_0,i_0,\varepsilon_1,i_1,\varepsilon_2,i_2,\ldots$
- Test in/output $\delta_{\phi p} = \varepsilon_0,i_0,\varepsilon_1,o_0,\varepsilon_1,i_1,o_1,\ldots$
- Test Verdict:
  - OK, if $\delta_{\phi p} = \varepsilon_0,i_0,\varepsilon_1,o_0,\varepsilon_1,i_1,o_1,\ldots$run of system model
  - NOK, otherwise
- Timed Automata?

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

This work

- Test case generation from timed automata
  - by reachability analysis
  - implementation in UPPAAL
- Testing Criteria:
  - single test purpose
  - coverage criteria: location, branching, definition/use pairs, etc.
- Optimality:
  - Test Cases: $\sigma_{\phi p} = \varepsilon_0, i_0, \varepsilon_1, i_1, \varepsilon_2, i_2, \ldots$ with minimum cost
    e.g. $min(\varepsilon_0 + \varepsilon_1 + \ldots + \varepsilon_n)$
  - Test Suites: $T = \{\sigma_1, \ldots, \sigma_n\}$ with minimum cost

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

Controllable Timed Automata

Input Enabled:
all inputs can always be accepted.

Assumption about model of SUT

Output Urgent:
enabled outputs will occur immediately.

Determinism:
two transitions with same input/output leads to the same state.

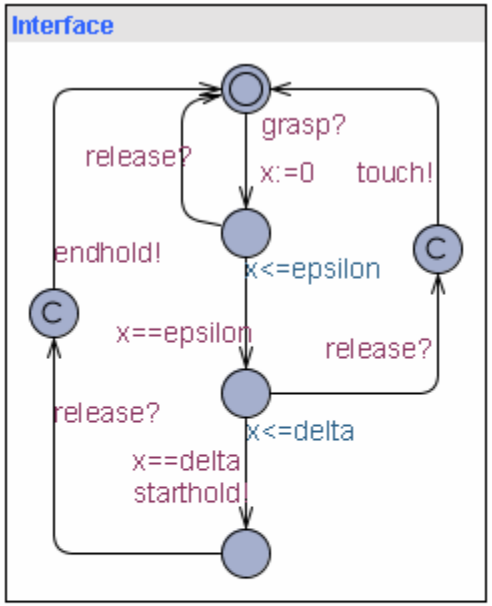
Isolated Outputs:
if an output is enabled, no other output is enabled.

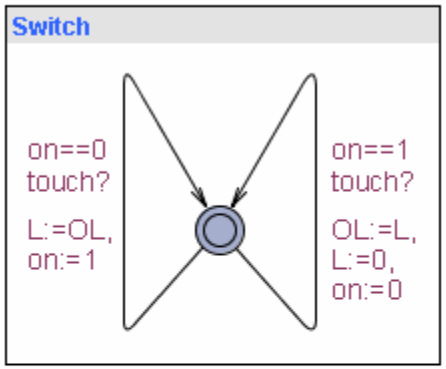
CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

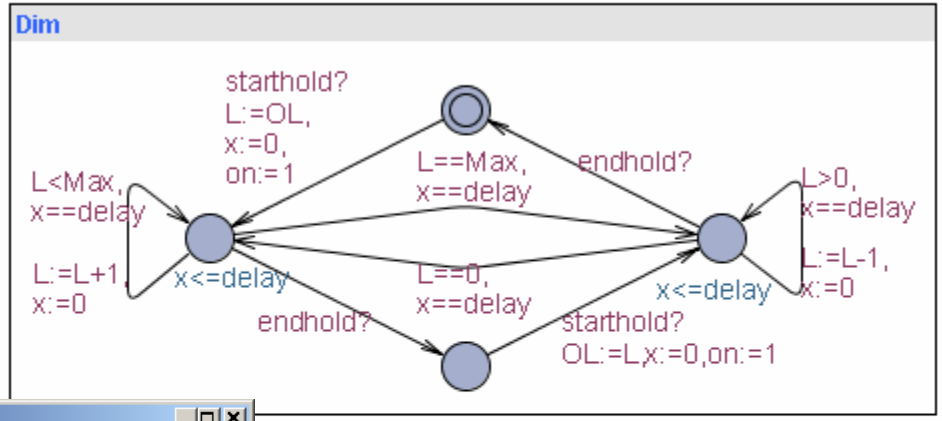
BRICS
Basic Research
in Computer Science

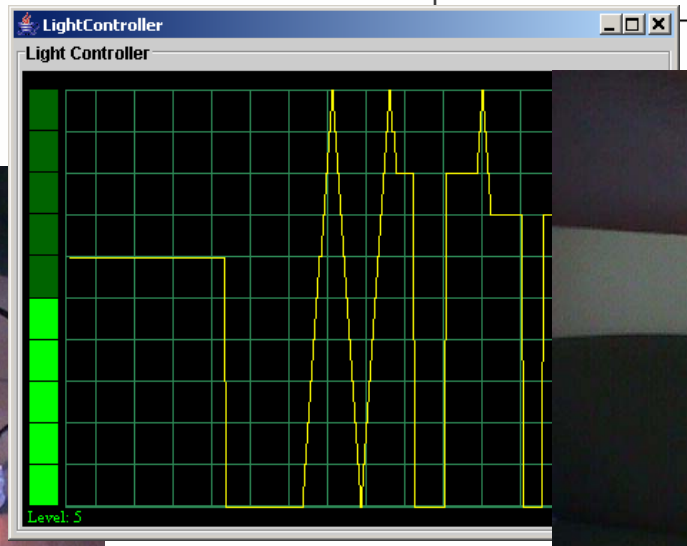
Example

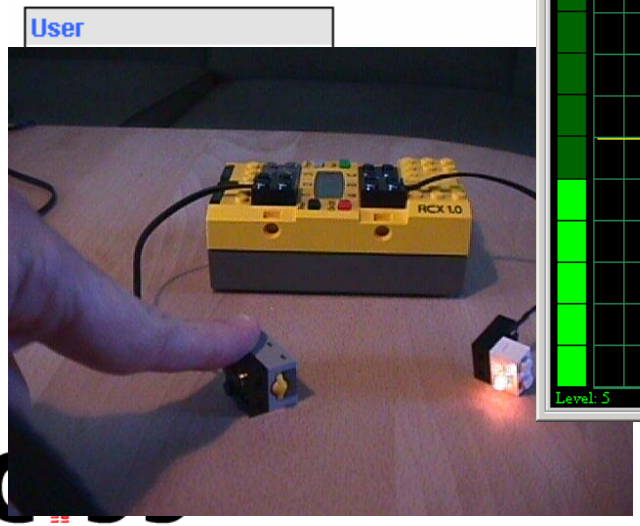
Light Controller







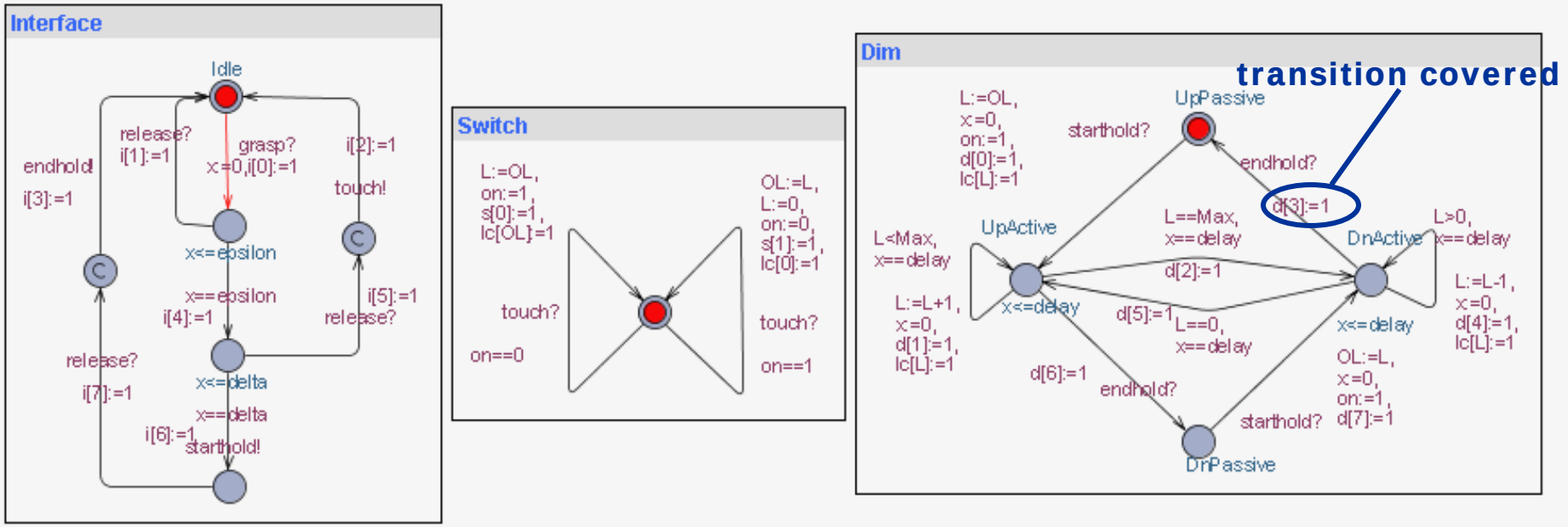




CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

Off-Line Testing = *Optimal Reachability*



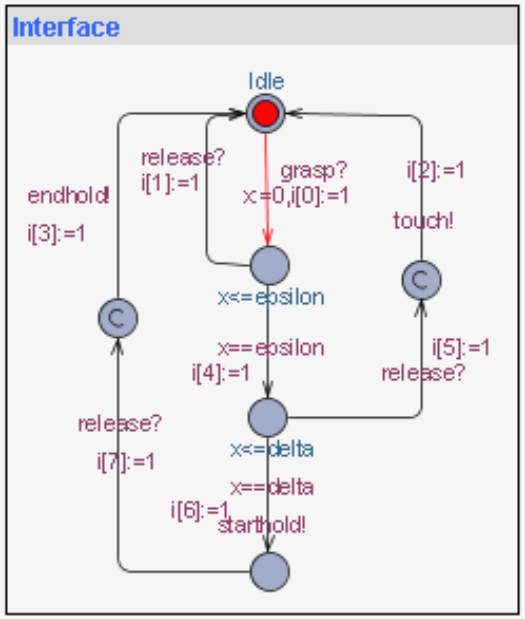
- Specific Test Purposes
- Model Coverage
- Optimal test-suites

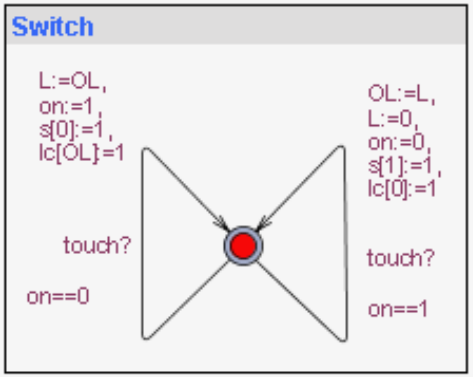
CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

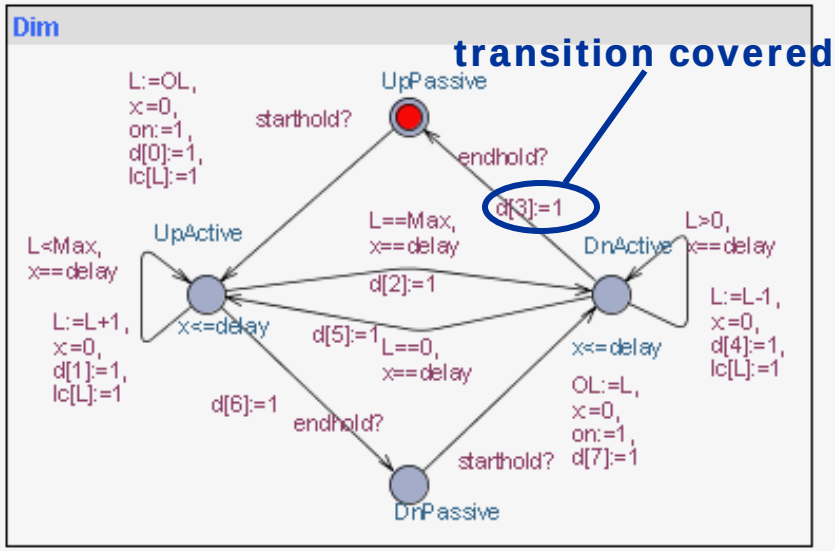
BRICS
Basic Research
in Computer Science

Off-Line Testing = *Optimal Reachability*

Fastest Transition Coverage =12600 ms








```
out(IGrasp);              //touch:switch light on
silence(200);
out(IRelease);
in(OSetLevel,0);

out(IGrasp); //@200        // touch: switch light off
silence(200);
out(IRelease);//touch
in(OSetLevel,0);

//9
out(IGrasp); //@400        //Bring dimmer from ActiveUp
silence(500); //hold       //To Passive DN (level=0)
in(OSetLevel,0);
out(IRelease);
```

Page 1

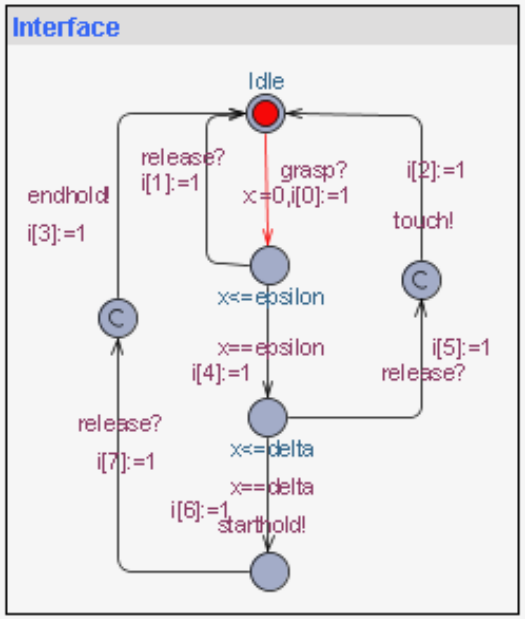
```
//13
out(IGrasp); //@900        // Bring dimmer  PassiveDn->ActiveDN->
silence(500);//hold        // ActiveUP+increase to level 10
silence(1000); in(OSetLevel,1);
silence(1000); in(OSetLevel,2);
silence(1000); in(OSetLevel,3);
silence(1000); in(OSetLevel,4);
silence(1000); in(OSetLevel,5);
silence(1000); in(OSetLevel,6);
silence(1000); in(OSetLevel,7);
silence(1000); in(OSetLevel,8);
silence(1000); in(OSetLevel,9);
silence(1000); in(OSetLevel,10
silence(1000);  in(OSetLevel,9); //bring dimm State to ActiveDN

out(IRelease);            //check release->grasp is ignored
out(IGrasp); //@12400
out(IRelease);
silence(dfTolerance);
```

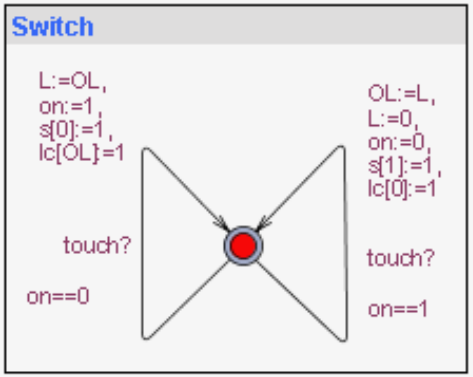
Page 2

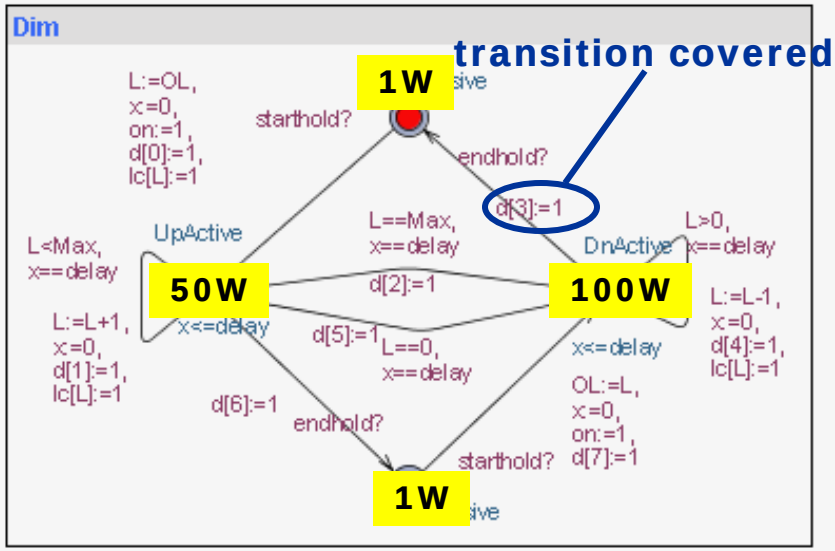
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

Off-Line Testing = *Optimal Reachability*







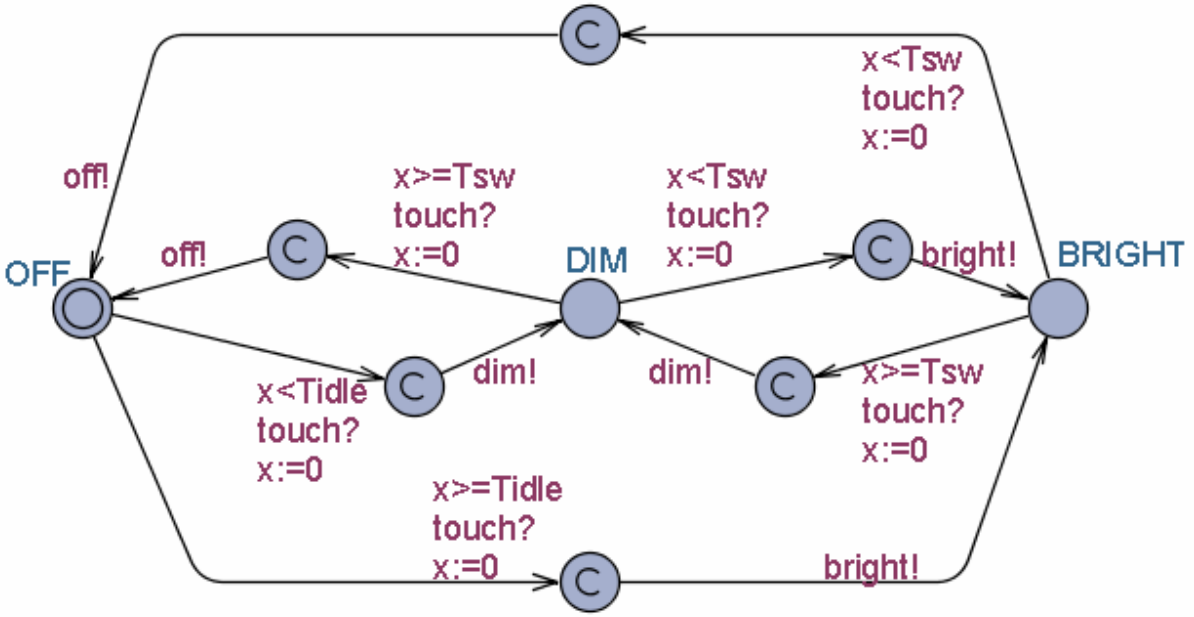
- Specific Test Purposes
- Model Coverage
- Optimal test-suites

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

Timed Automata

(E)FSM+clocks+guards+resets



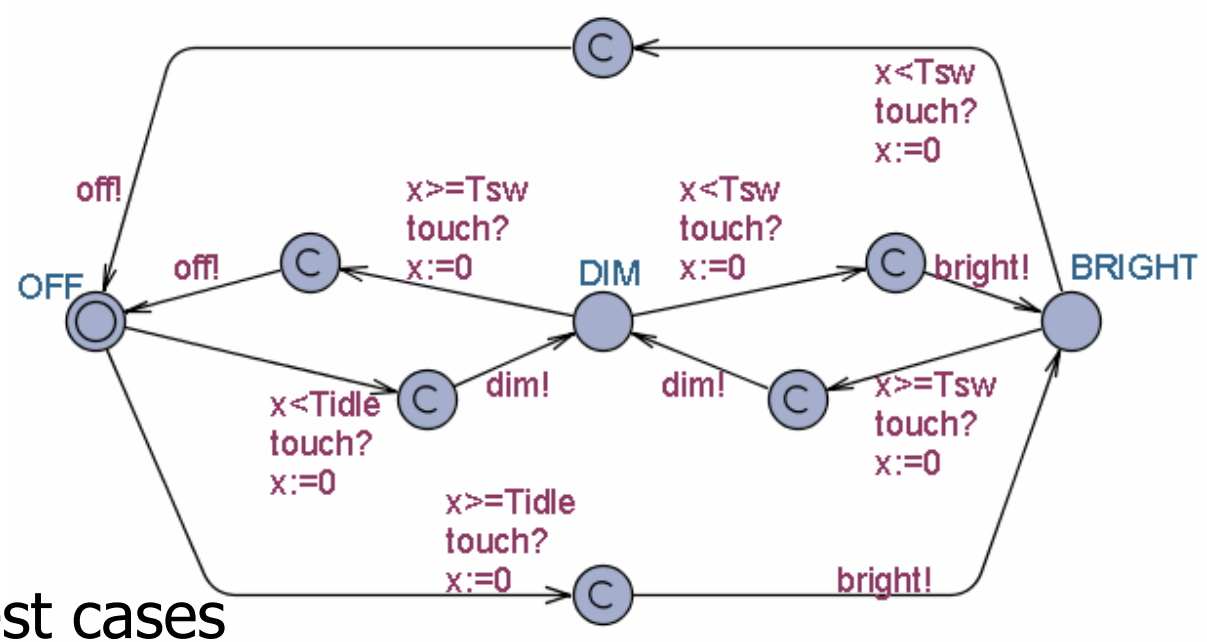

```
T_sw=4
T_idle=20
```

CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

WANT: if touch is issued twice quickly then the light will get brighter; otherwise the light is turned off.

Solution: Add real-valued clock **x**

Timed Tests



T_sw=4
T_idle=20

EXAMPLE test cases

0·touch!·0·dim?·2·touch!·0·bright?·2·touch!·off?·PASS

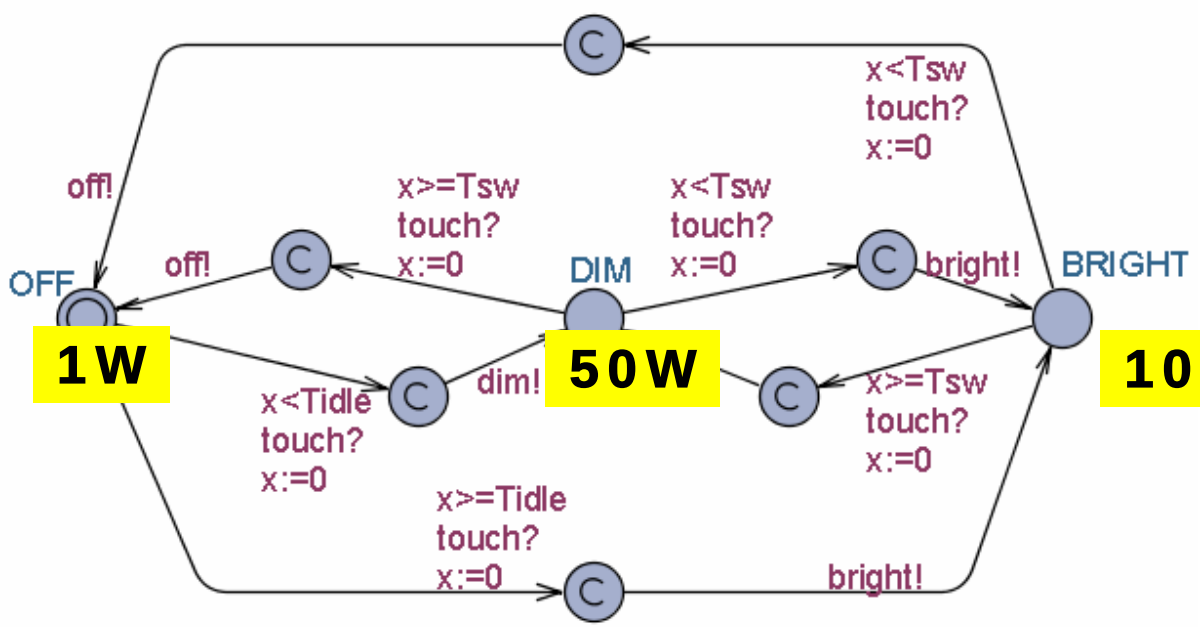
0·touch!·0.dim?·2½·touch!·0·bright?·3·touch!·off?·PASS

0·touch!·0·dim?·5touch!·0·off?·PASS

0·touch!·0·dim?·5·touch!·0·off?·50·touch!·0·bright?·6·touch!·0·dim?·PASS

INFINITELY MANY SEQUENCES!!!!!

Optimal Tests

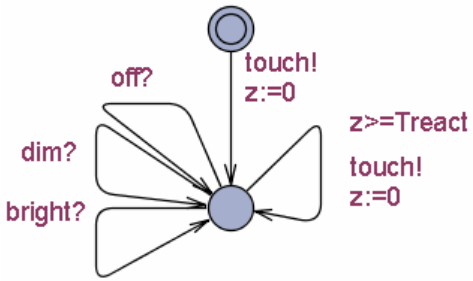



```
T_sw=4
T_idle=20
```

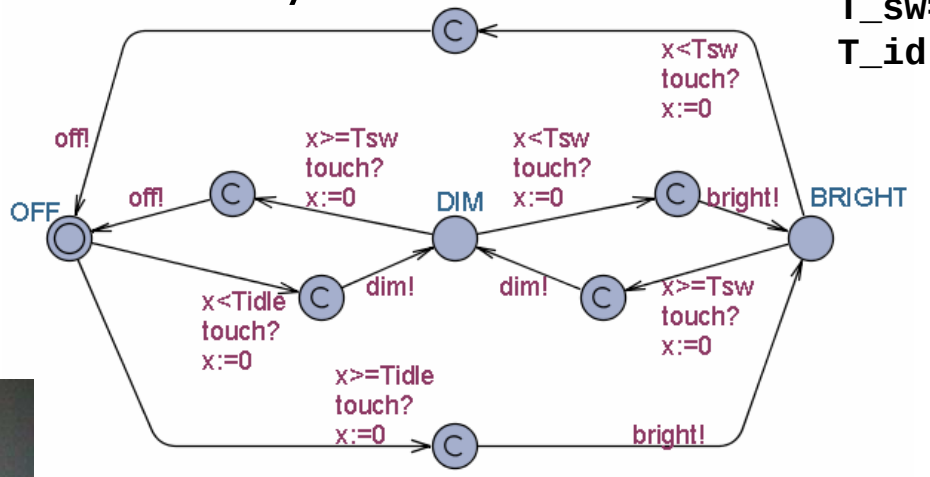
- ***Shortest*** test for bright light??
- ***Fastest*** test for bright light??
- ***Fastest*** edge-covering test suite??
- Least ***power*** consuming test??

Simple Light Controller

Environment model



System model

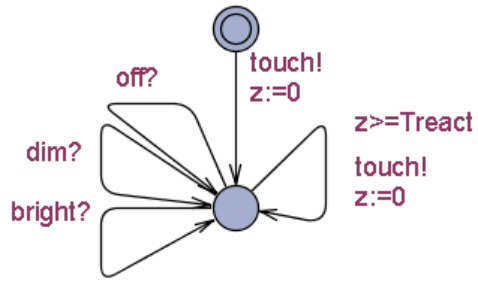



```
T_react=2
T_sw=4
T_idle=20
```

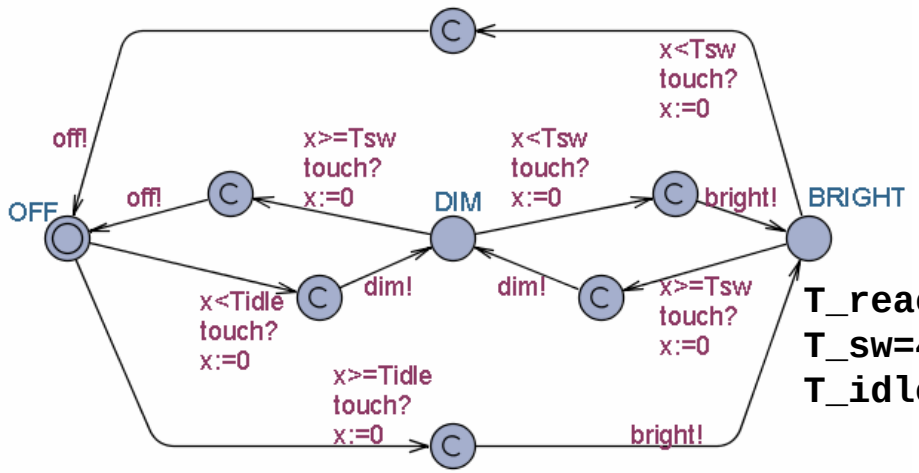
Test Purposes

A specific test objective (or observation) the tester wants to make on SUT

Environment model



System model




```
T_react=2
T_sw=4
T_idle=20
```

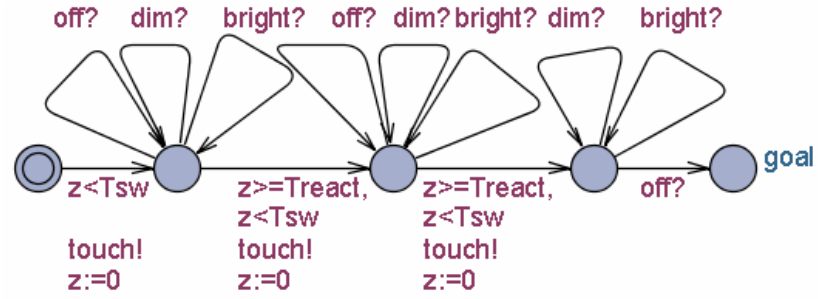
TP1: Check that the light can become bright:

`E<> LightController.bright`

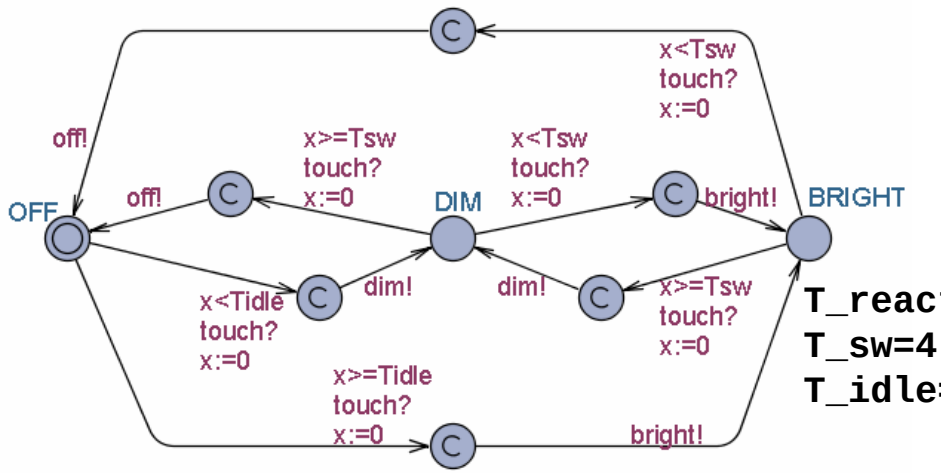
- *Shortest* Test: 20·touch!·0·bright?·PASS
- *Fastest* Test: 0·touch!·0·dim?·2·touch!·0·bright ?·PASS

Test Purposes 2

Environment model*TP2



System model



TP2: Check that the light switches off after three successive touches

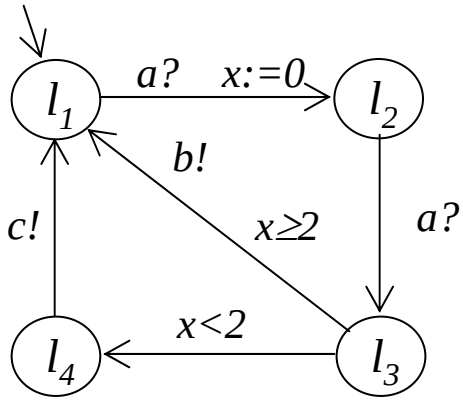
Use restricted environment and **`E<> tpEnv.goal`**

- The fastest test sequence is
0·touch!·0·dim?·2·touch!·0·bright?·2·touch!·0·off?·PASS

BRICS
Basic Research
in Computer Science

Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - Location coverage,
  - Edge coverage,
  - Definition/use pair coverage

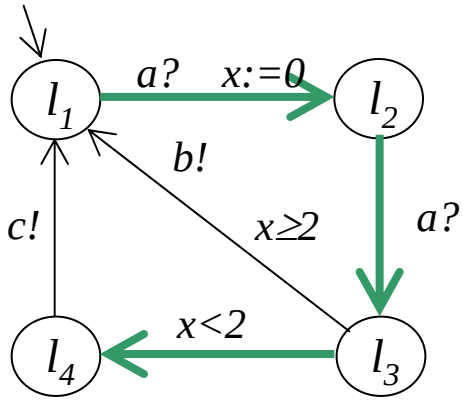


CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - Location coverage,
  - Edge coverage,
  - Definition/use pair coverage

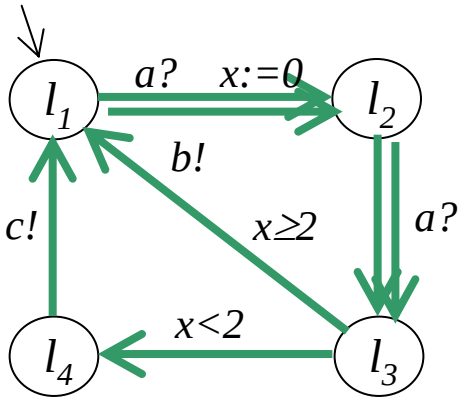


CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - Location coverage,
  - Edge coverage,
  - Definition/use pair coverage

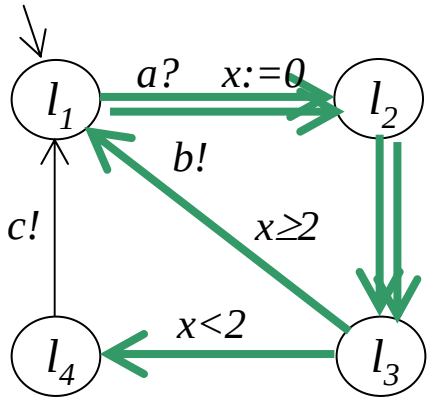


CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - Location coverage,
  - Edge coverage,
  - Definition/use pair coverage

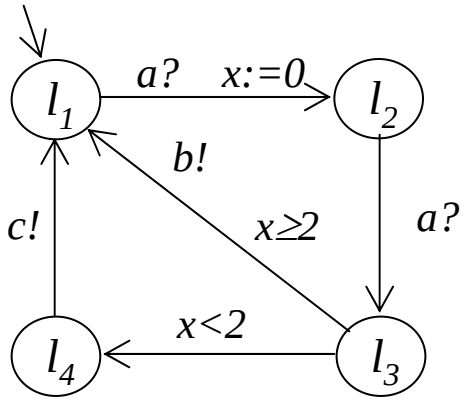


CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - Locations coverage,
  - Edge coverage,
  - Definition/use pair coverage
  - All Definition/Use pairs

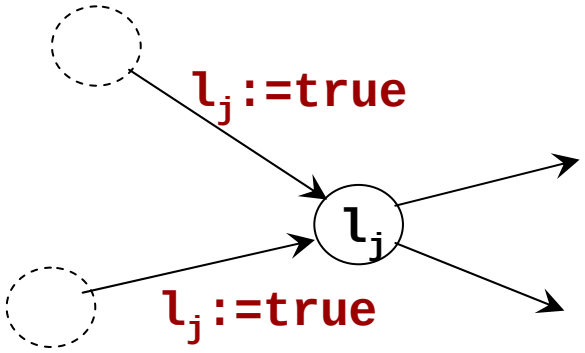


- Generated by min-cost reachability analysis of annotated graph

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

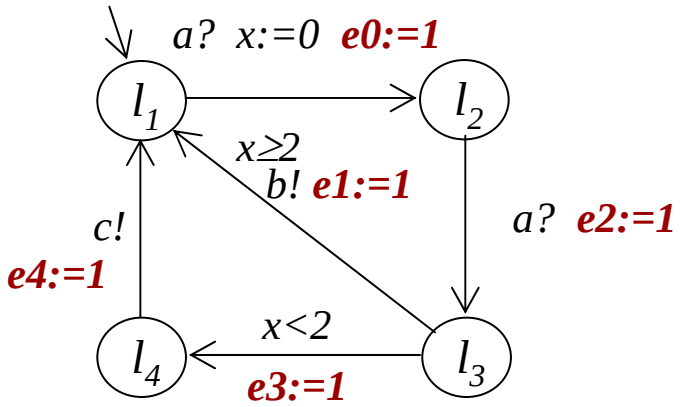
Location Coverage

- Test sequence traversing all locations
- Encoding:
  - Enumerate locations $l_0, \ldots, l_n$
  - Add an auxiliary variable l_i for each location
  - Label each ingoing edge to location i l_i:=true
  - Mark initial visited l_0:=true
- Check: **EF(l_0=true $\wedge$ … $\wedge$ l_n=true)**

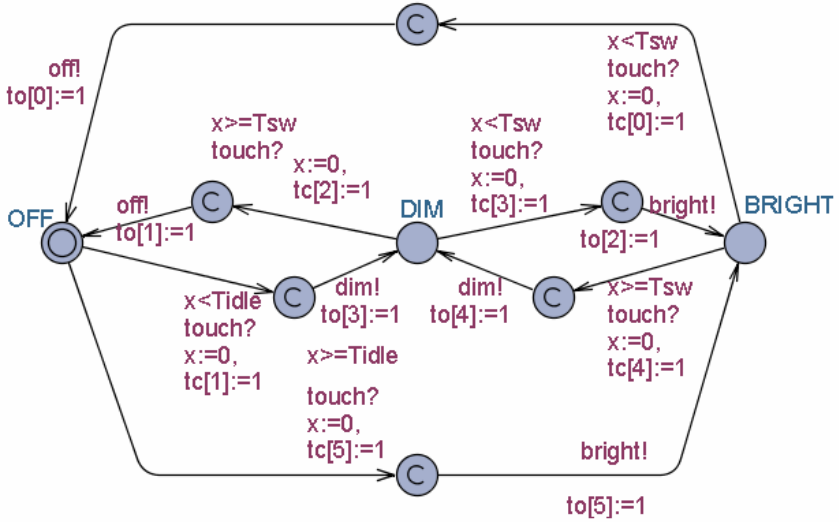


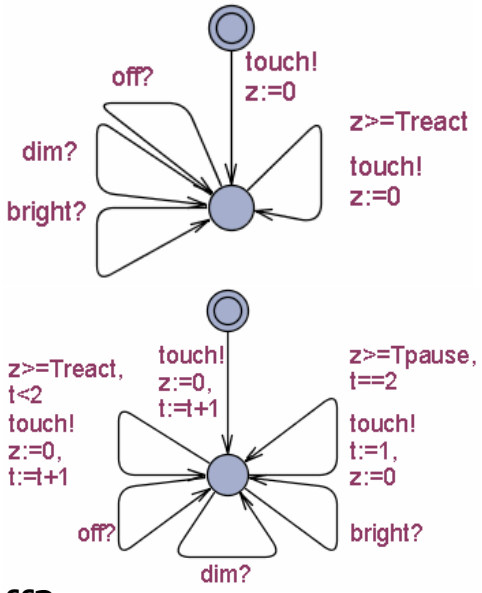
Edge Coverage

- Test sequence traversing all edges
- Encoding:
  - Enumerate edges $e_0, \ldots, e_n$
  - Add auxiliary variable e_i for each edge
  - Label each edge $e_i := \mathbf{true}$
- Check: $\mathbf{EF}(e_0 = \mathbf{true} \wedge \ldots \wedge e_n = \mathbf{true})$



Edge Coverage





EC: *T_react=0*

0·touch!·0·dim?·0·touch!·0·bright?·0·touch!·0·off?·
20·touch!·0·bright?·4·touch!·0·dim?·4·touch!·0·off?·PASS

Time=28

EC': *T_react=2*

0·touch!·0·dim?·4·touch!·0·off?· 20·touch!·0·bright?·
4·touch!·0·dim?·2·touch!·0·bright?·2·touch!·0·off?·PASS

Time=32

EC'': *pausing user T_react=2, T_pause=5*

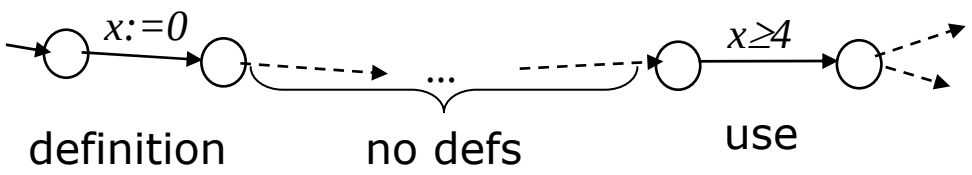
0·touch!·0·dim?·2·touch!·0·bright?·5·touch!·0·dim?·
4·touch!·0·off?·20·touch!·0·bright?·2·touch!·0·off?·PASS

Time=33

BRICS
Basic Research
in Computer Science

Definition/Use Pair Coverage

- Dataflow coverage technique
- Def/use pair of variable x:

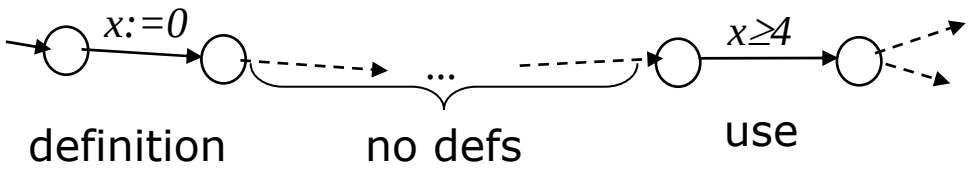


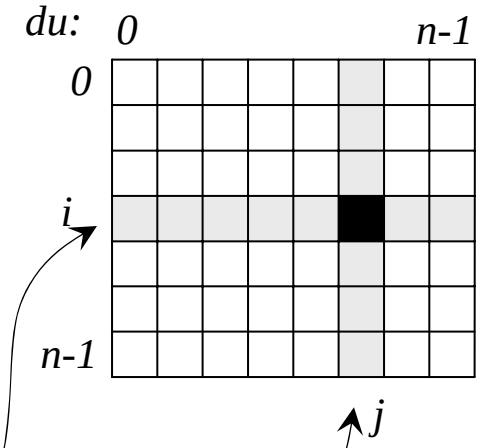
- Encoding:
  - $v_d \in \{ false \} \cup \{ e_0, \ldots, e_n \}$, initially false
  - Boolean array du of size $|E|$ x $|E|$
  - At definition on edge i: $v_d := e_i$
  - At use on edge j: if(v_d) then $du[v_d, e_j] := true$

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Definition / Use Pair Coverage

- Dataflow coverage technique
- Def/use pair of variable x:

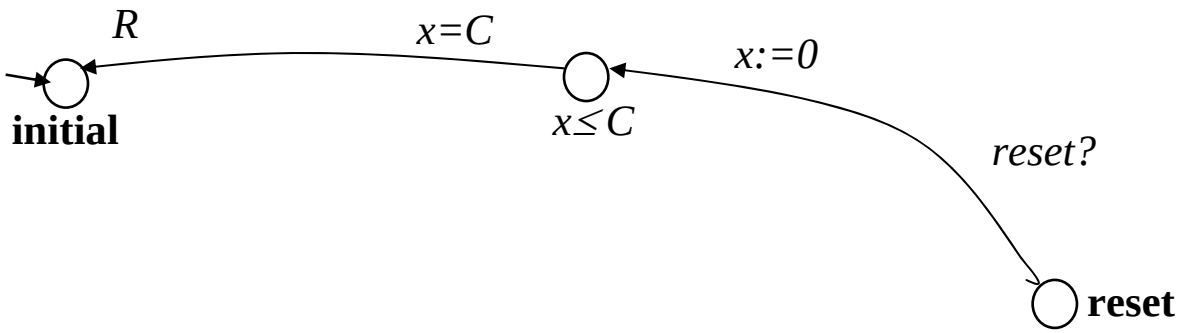




- Encoding:
  - $v_d \in \{ false \} \cup \{ e_0, \ldots, e_n \}$, initially false
  - Boolean array du of size $|E|$ x $|E|$
  - At definition on edge i: $v_d := e_i$
  - At use on edge j: if(v_d) then $du[v_d, e_j] := true$
- Check:
  - EF(all $du[i,j] = true$)

Test Suite Generation

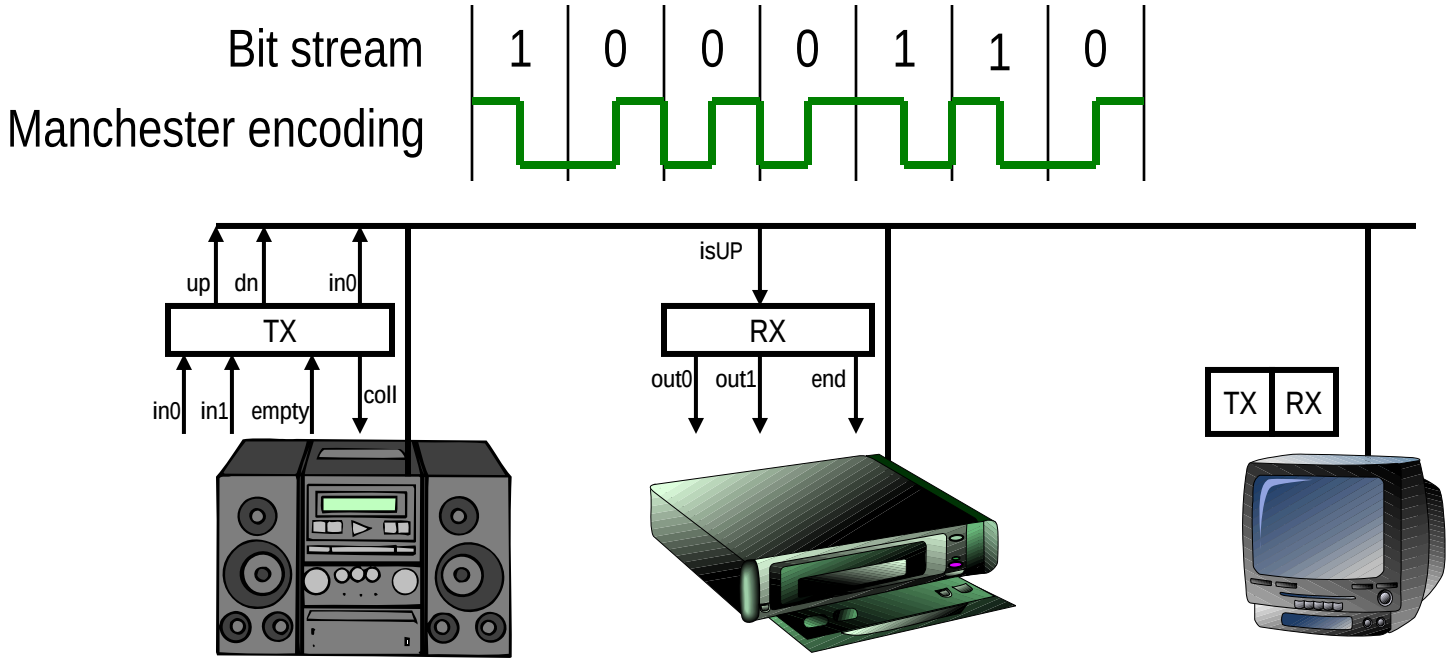
- In general a set of test cases is needed to cover a test criteria
- Add global reset of SUT and environment model and associate a cost (of system reset)



- Same encodings and min-cost reachability
- Test sequence $\sigma = \mathcal{E}_0, i_0, \ldots, \mathcal{E}_1, i_1, \textbf{reset}\ \underbrace{\mathcal{E}_2, i_2, \ldots, \mathcal{E}_0, i_0, \textbf{reset}}_{\sigma_i}, \mathcal{E}_1, i_1, \mathcal{E}_2, i_2, \ldots$
- Test suite $T = \{\sigma_1, \ldots, \sigma_n\}$ with minimum cost

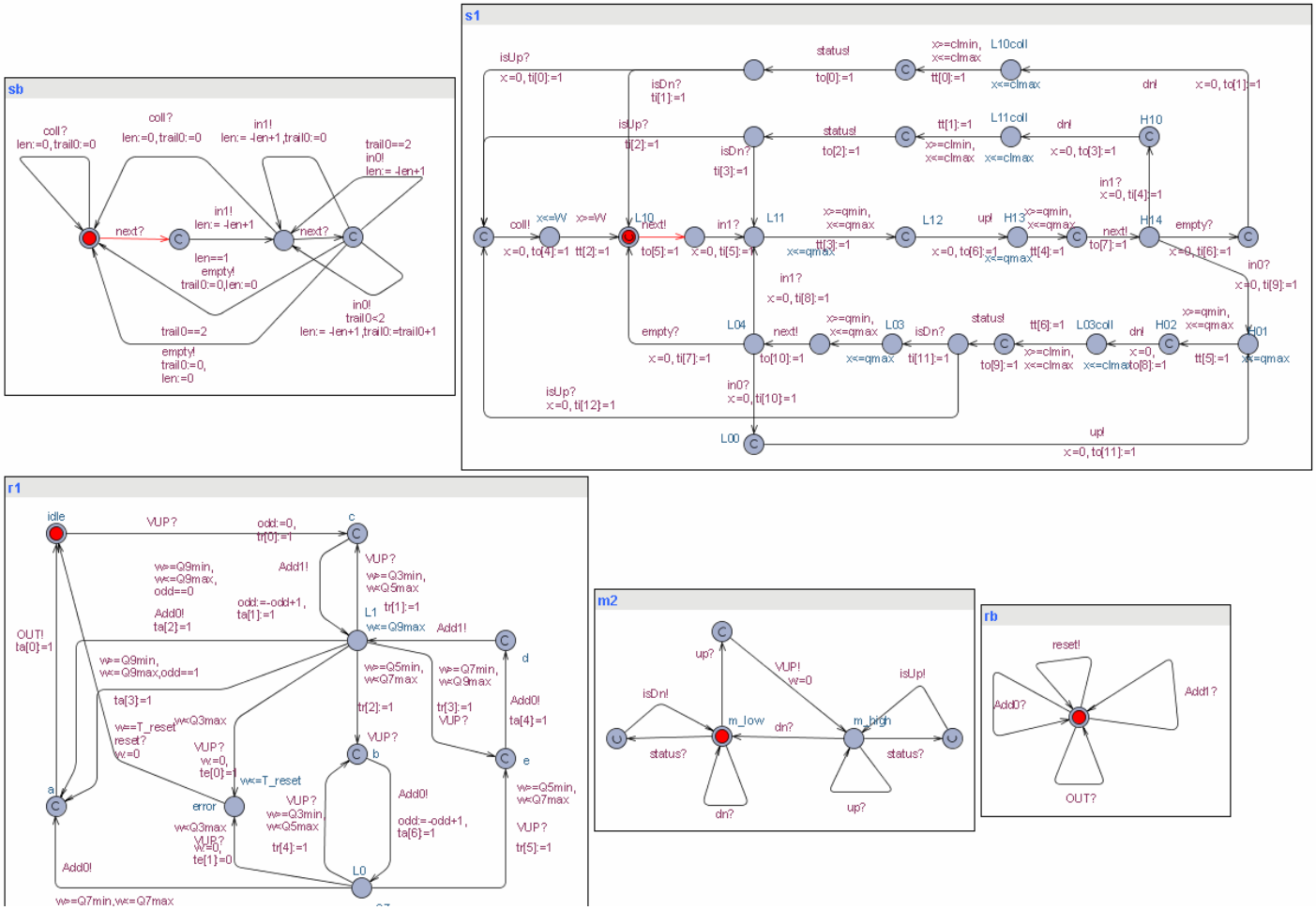
The Philips Audio Protocol

- A bus based protocol for exchanging control messages between audio components
  - Collisions
  - Tolerance on timing events



BRICS
Basic Research
in Computer Science

Philips Audio Protocol



CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

Benchmark Example

- Philips Audio Protocol

Coverage Criterion	Execution time (μs)	Generation time (s)	Memory usage (KB)
Edge_{Sender}	212350	2.2	9416
$\text{Edge}_{Receiver}$	18981	1.2	4984
$\text{Edge}_{Sender,Bus,Receiver}$	114227	129.0	331408

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Off-Line Test Generation

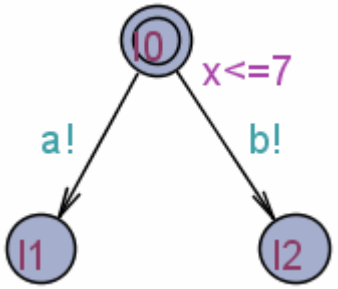
Observable Timed Automata

BRICS
Basic Research
in Computer Science

BRICS
Basic Research
in Computer Science

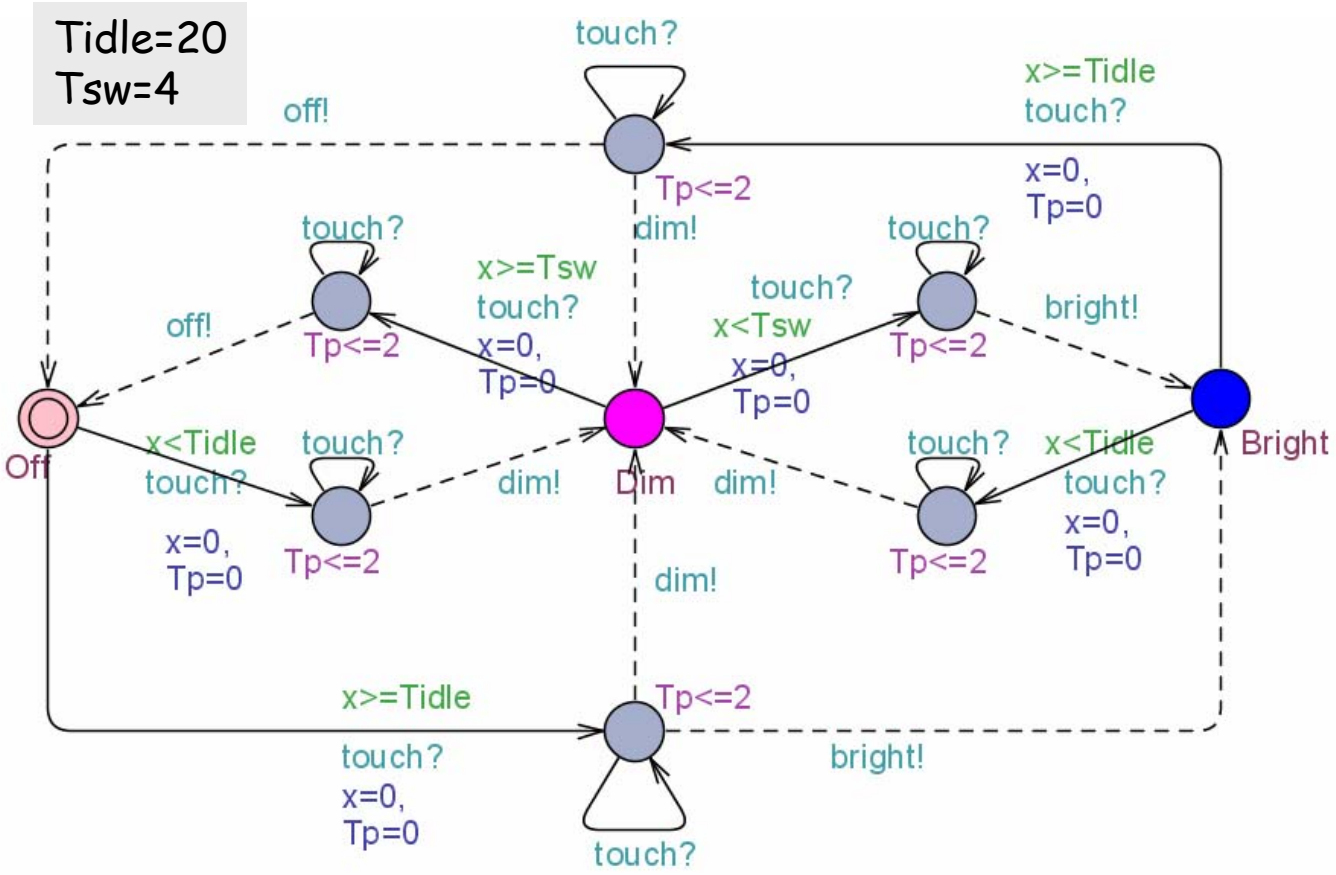
Observable Timed Automata

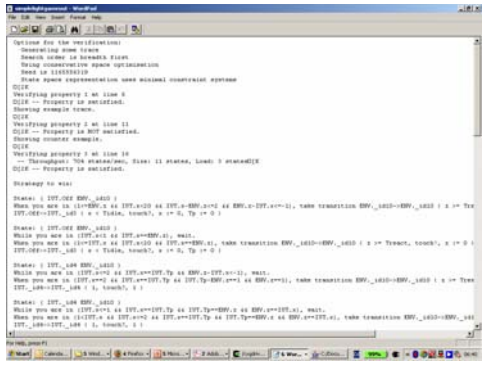
- **Determinism**:
  two transitions with same input/output leads to the same state
- **Input Enabled**:
  all inputs can always be accepted
- **Time Uncertainty of outputs**:
  timing of outputs uncontrollable by tester
- **Uncontrollable output**:
  IUT controls which enabled output will occur in what order



CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Timed Games and Test Generation

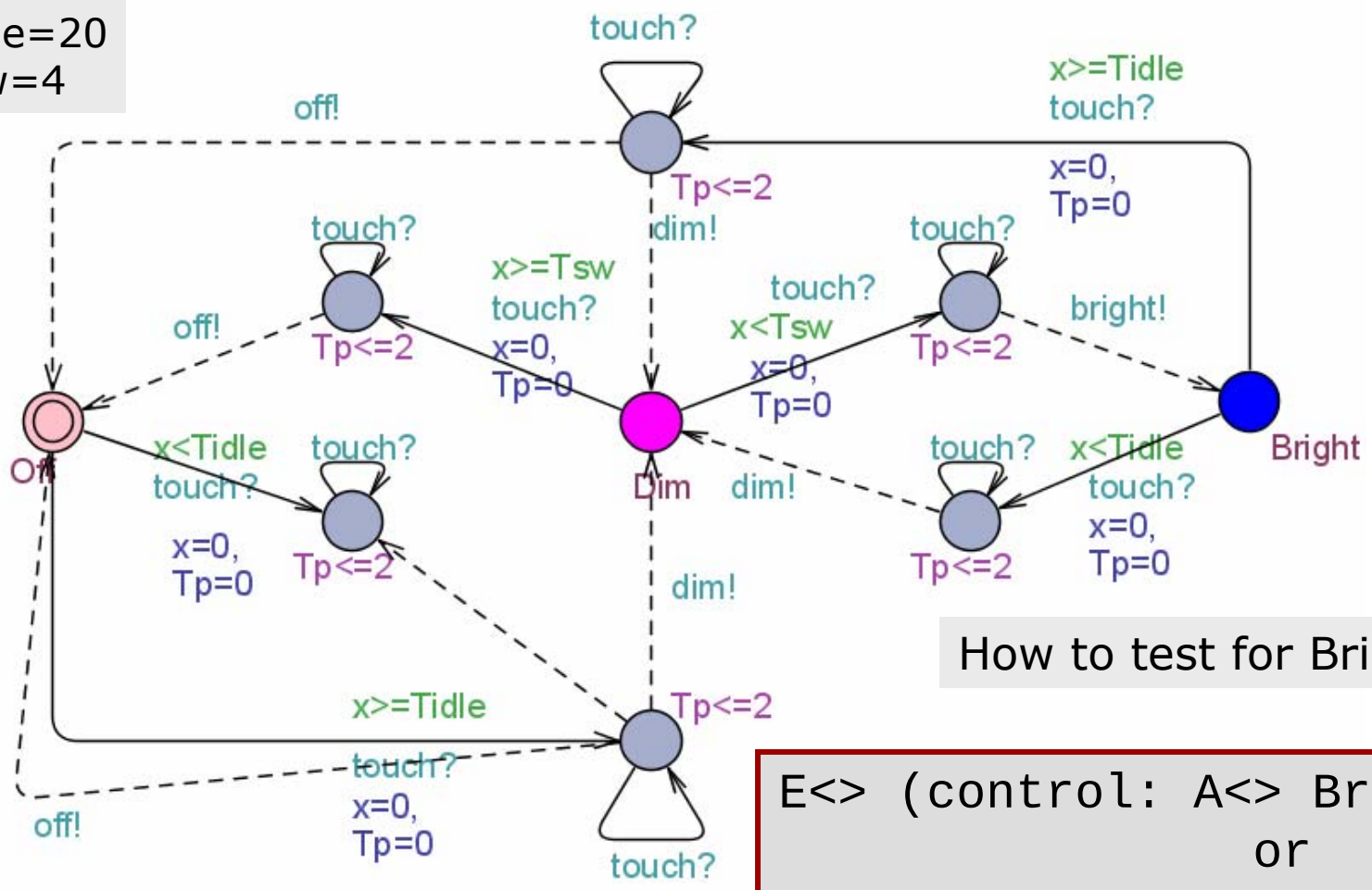




Off-line test-case generation =
Compute winning strategy for reaching **Bright**
Assign verdicts st. lost game means IUT not conforming

A trick light control

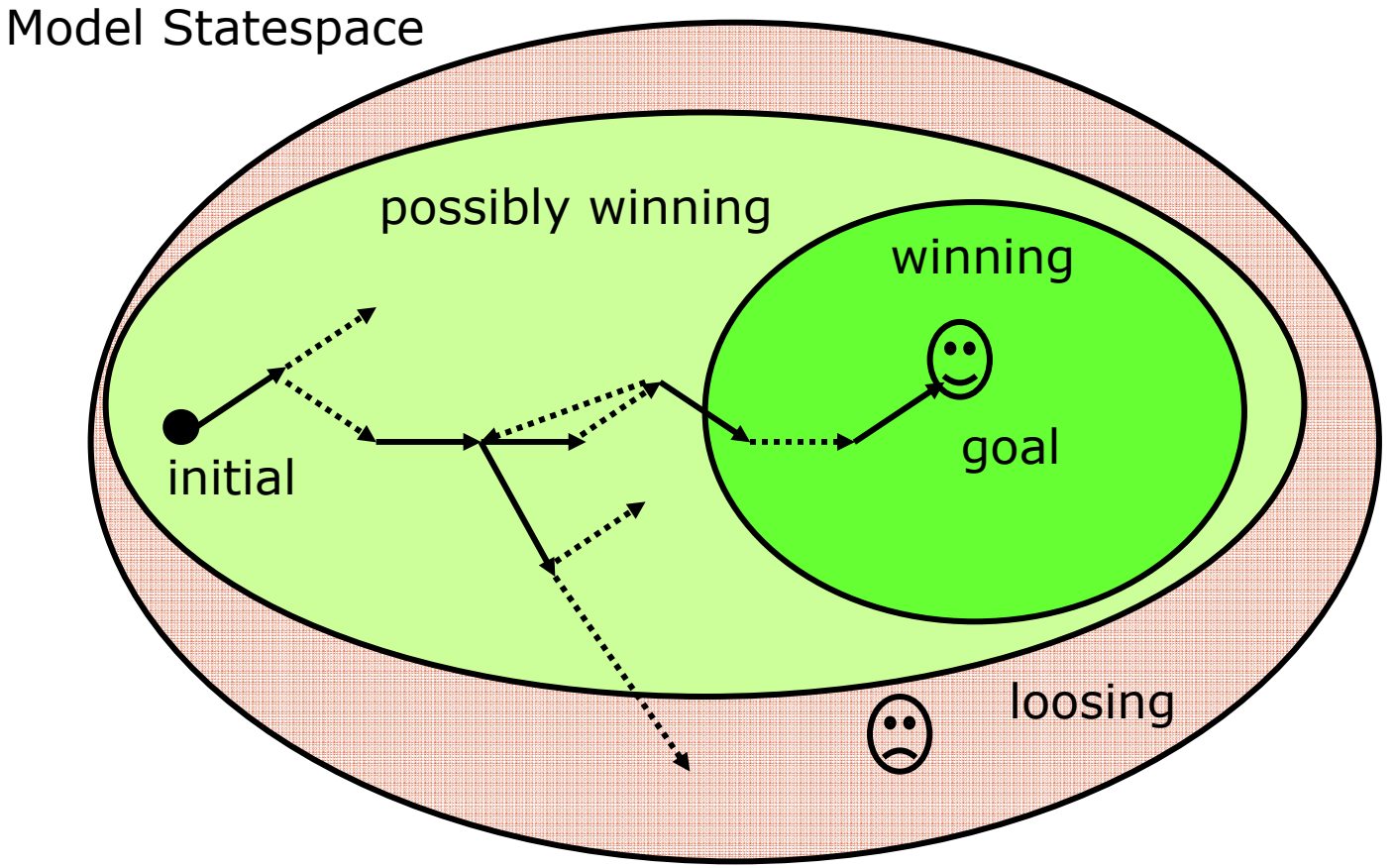
Tidle=20
Tsw=4



How to test for Bright ?

E<> (control: A<> Bright)
or
<<c,u>> ◇(<<c>> ◇ Bright)

Cooperative Strategies

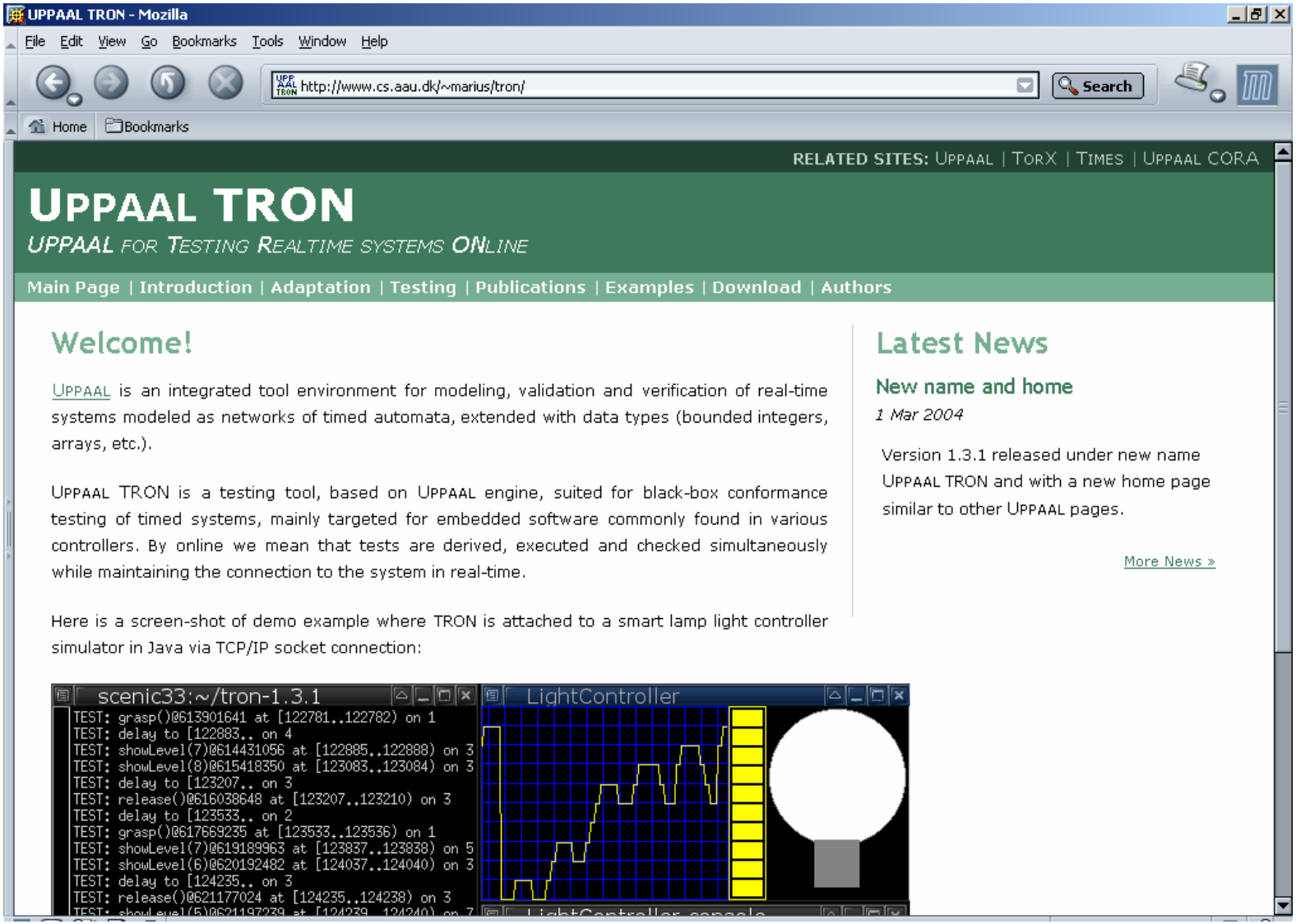


- Play the game (execute test) while time available or game is lost
- Possibly using ranomized online testing

On-Line Testing

BRICS
Basic Research
in Computer Science

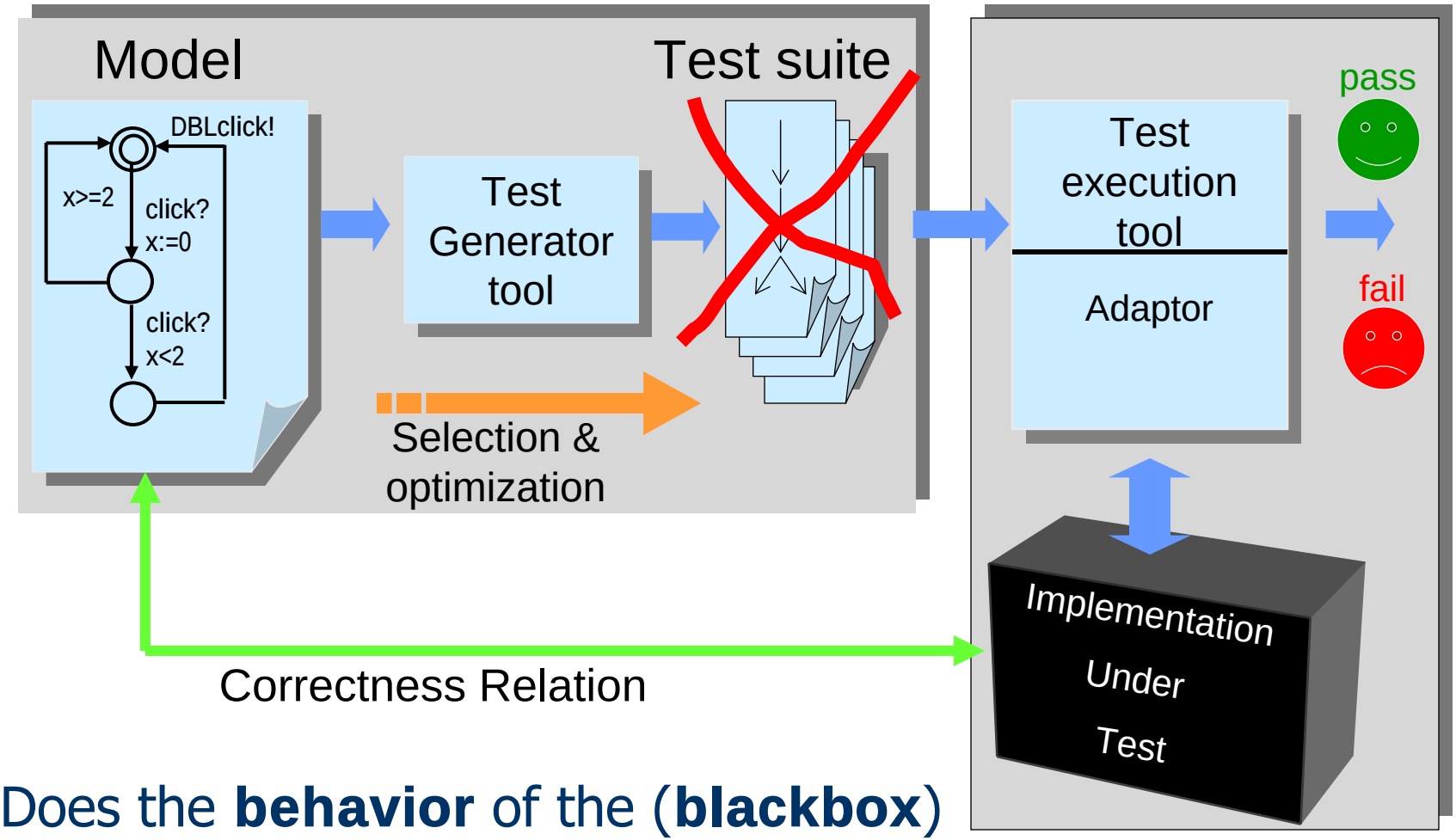
BRICS
Basic Research
in Computer Science



CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

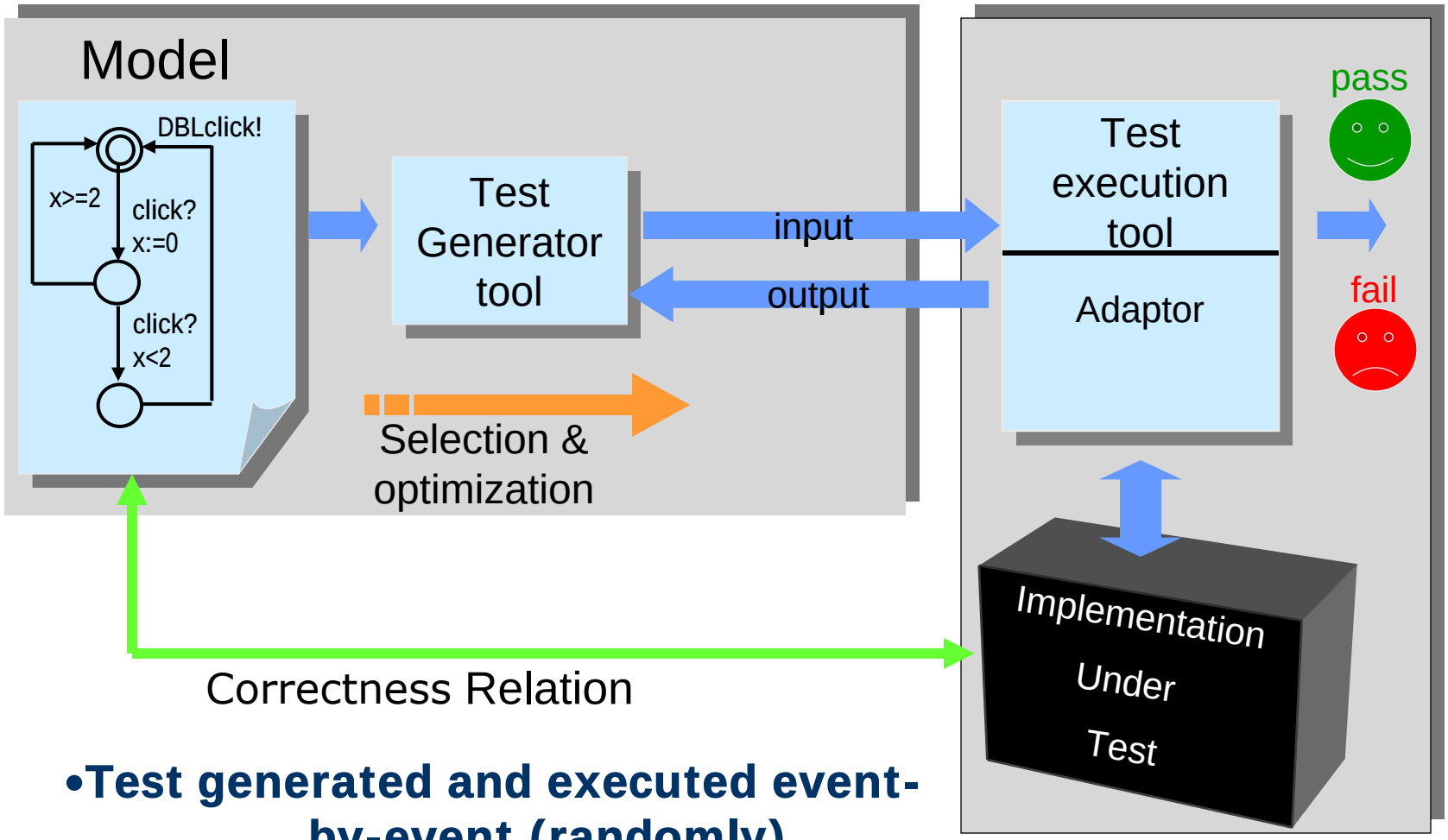
Automated Model Based Conformance Testing



Does the **behavior** of the (**blackbox**) implementation ***comply*** to that of the specification?

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Online Testing



- **Test generated and executed event-by-event (randomly)**
- **A.K.A on-the-fly testing**

An Algorithm

BRICS
Basic Research
in Computer Science

Algorithm Idea:

State-set tracking

- Dynamically compute all potential states that the model M can reach after the timed trace

  $\varepsilon_0, i_0, \varepsilon_1, o_1, \varepsilon_2, i_2, o_2, \ldots$ [Tripakis] Failure Diagnosis

- $Z = M$ ***after*** $(\varepsilon_0, i_0, \varepsilon_1, o_1, \varepsilon_2, i_2, o_2)$

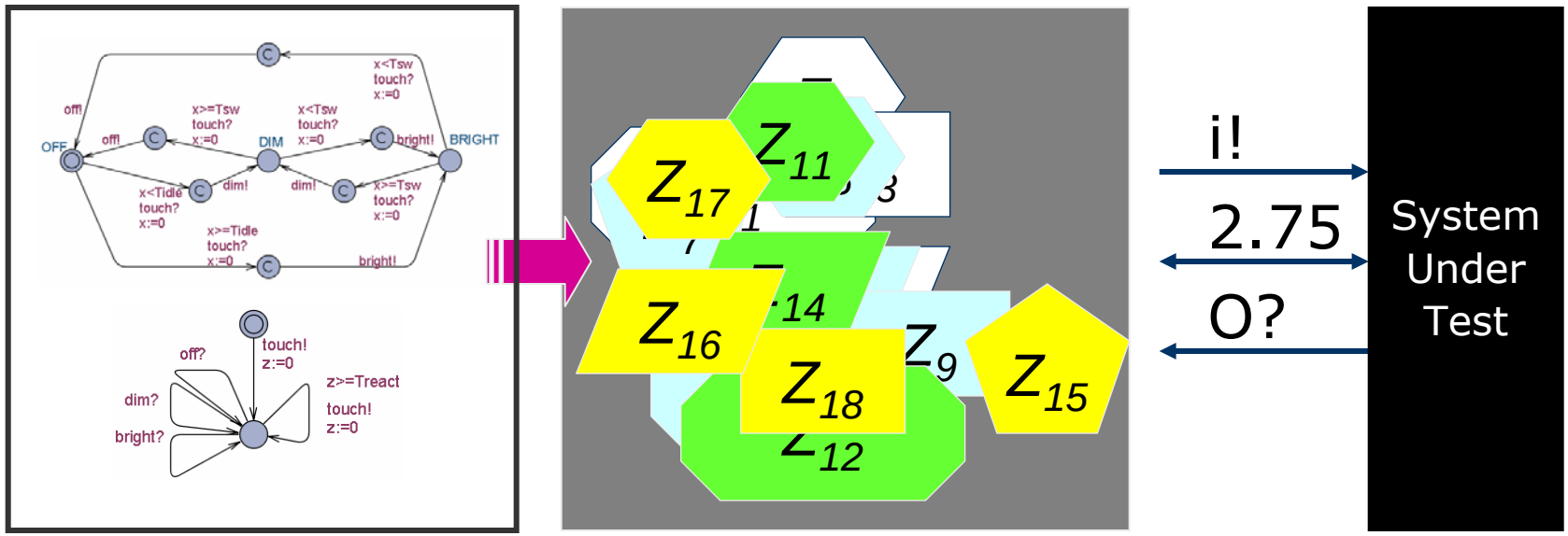
- *If* $Z = \emptyset$ *the IUT has made a computation not in model:* ***FAIL***

- *i is a relevant input in Env iff* $I \in EnvOutput(Z)$

Online State Estimation

Timed Automata Specification

State-set explorer:
maintain and analyse a *set* of *symbolic* states in real time!

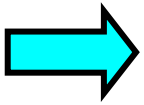


(Abstract) Online Algorithm

Algorithm *TestGenExe (S, E, IUT, T)* **returns** {**pass**, **fail**)
Z := {(s0, e0)}.

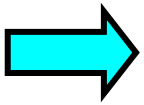
while Z ≠ ∅ **and** ♯iterations ≤ T **do either** randomly:

1. // offer an input

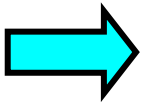


if *EnvOutput(Z)* ≠ ∅
randomly choose *i*∈ *EnvOutput(Z)*
send i to IUT
Z := Z *After* i

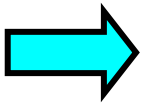
2. // wait d for an output



randomly choose d∈ *Delays(Z)*
wait (for d time units or output o at d′ ≤ d)



if o occurred **then**
Z := Z *After* d′
Z := Z *After* o // may become ∅ (⇒fail)



else
Z := Z *After* d // no output within d delay

3. *restart*:

Z := {(s0, e0)}, **reset** IUT //reset and restart

if Z = ∅ **then return fail else return pass**

(Abstract) Online Algorithm

Algorithm *TestGenExe (S, E, IUT, T)* **returns** {**pass**, **fail**)
Z := {(s0, e0)}.

while Z ≠ ∅ □ ♯iterations ≤ T **do either** randomly:

1. // offer an input

   if *EnvOutput(Z)* ≠ ∅
   randomly choose *i* *EnvOutput(Z)*
   send i to IUT
   Z := Z

2. // wait d fo
   randomly c
   wait (for d
   if o occurre
   Z := Z *A*
   Z := Z *After* o // may become ∅ (⇒fail)
   else
   Z := Z *After* d // no output within d delay

3. *restart*:
   Z := {(s0, e0)}, **reset** IUT //reset and restart

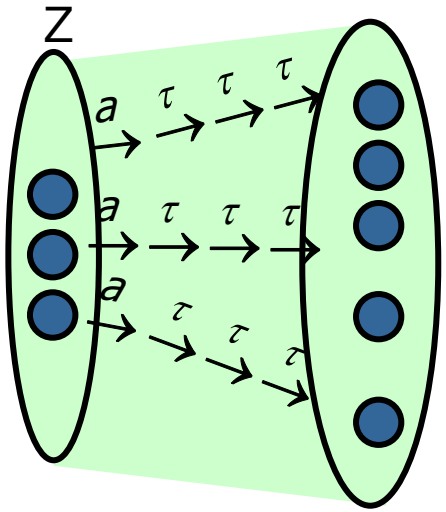
if Z = ∅ **then return fail else return pass**

- Sound
- Complete (as T → ∞)

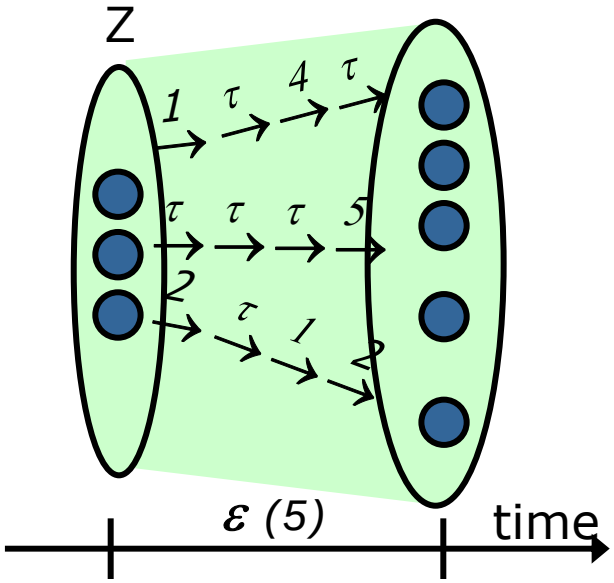
(Under some technical assumptions)

State-set Operations

Z* after *a: *possible states after* ***action a*** *(and* τ^**)*

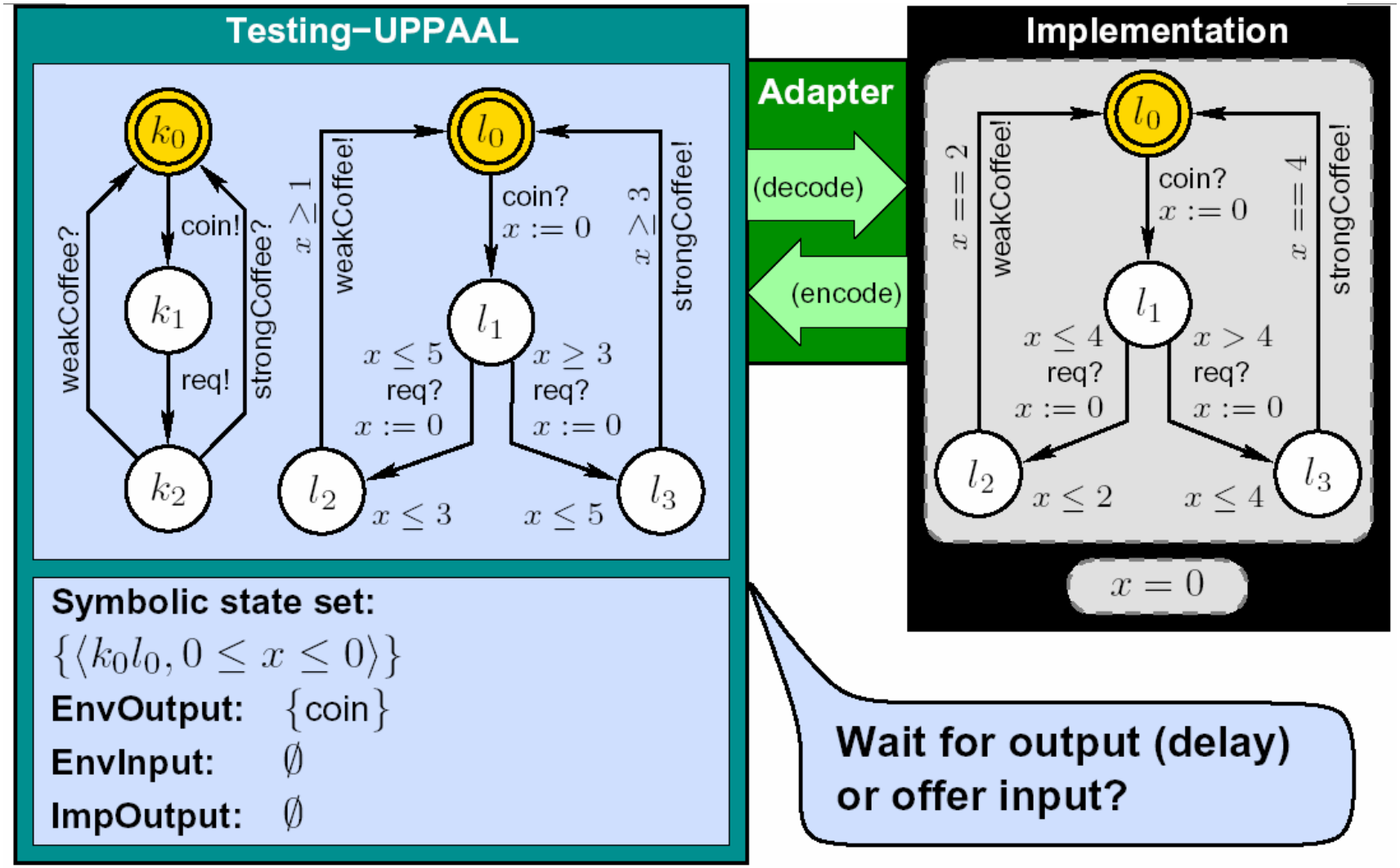


Z* after** ε :*possible states after* τ^* *and* ε_i *, totaling a* ***delay *of* ε

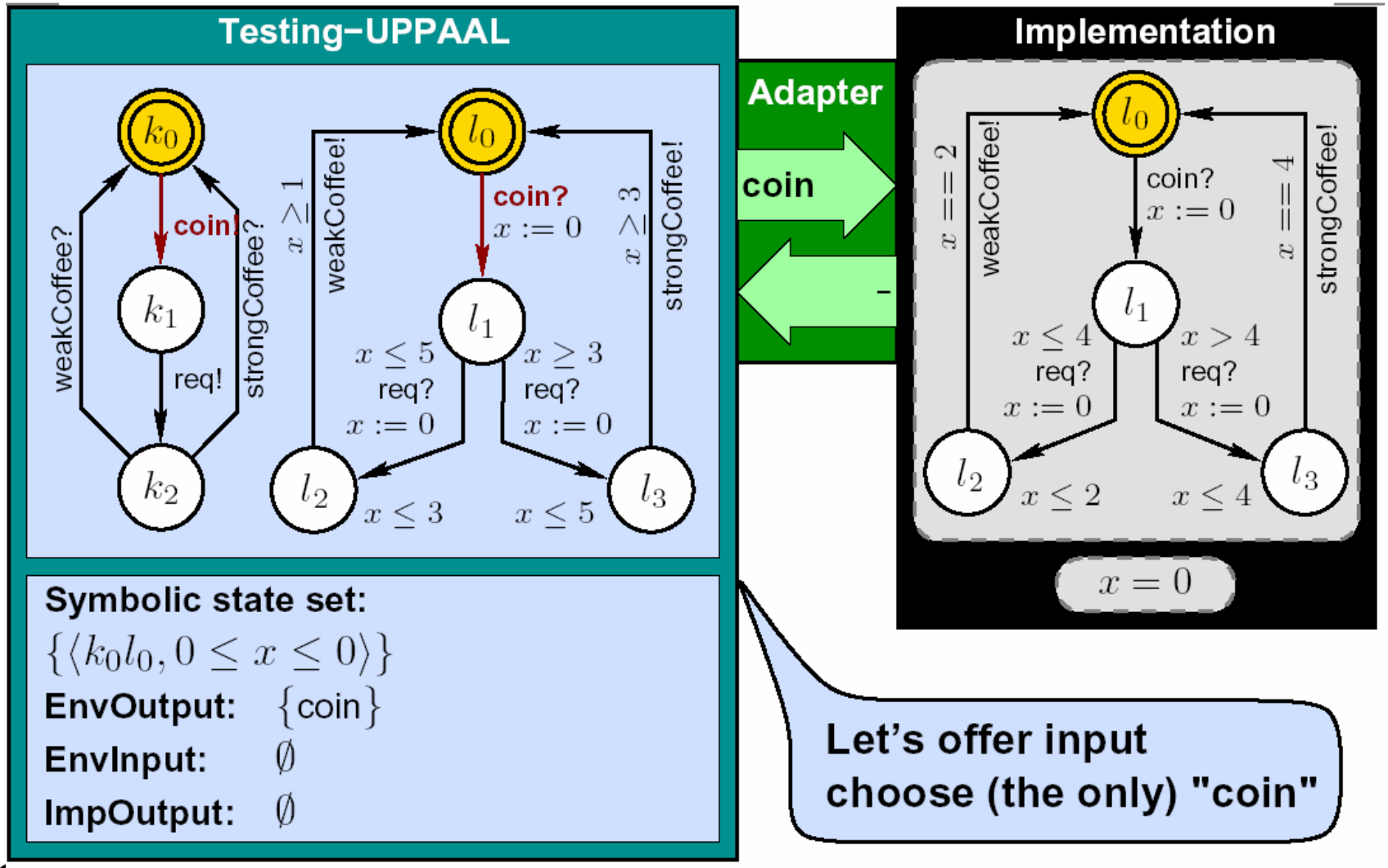


- Can be computed efficiently using the symbolic data structures and algorithms in Uppaal

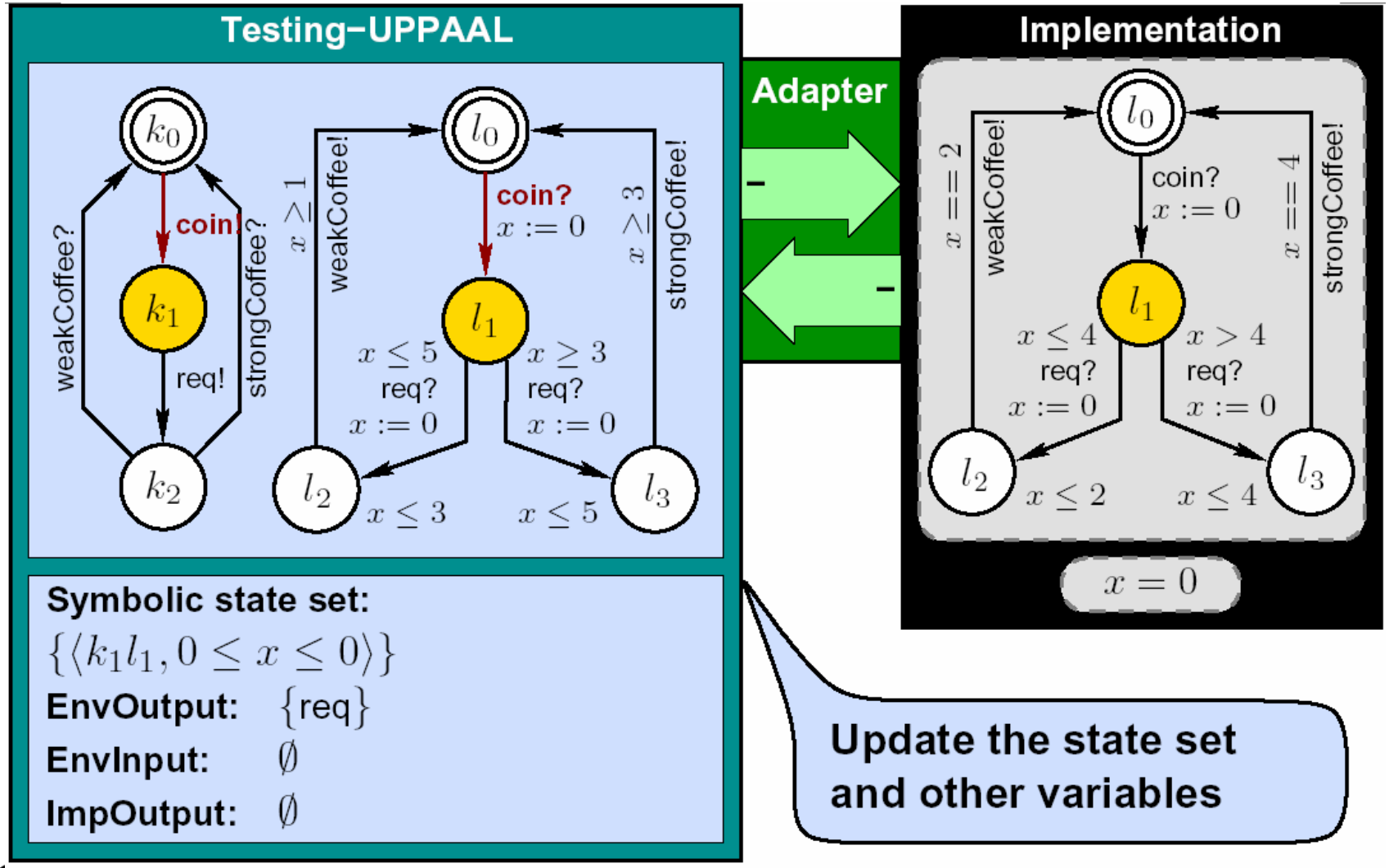
Online Testing Example



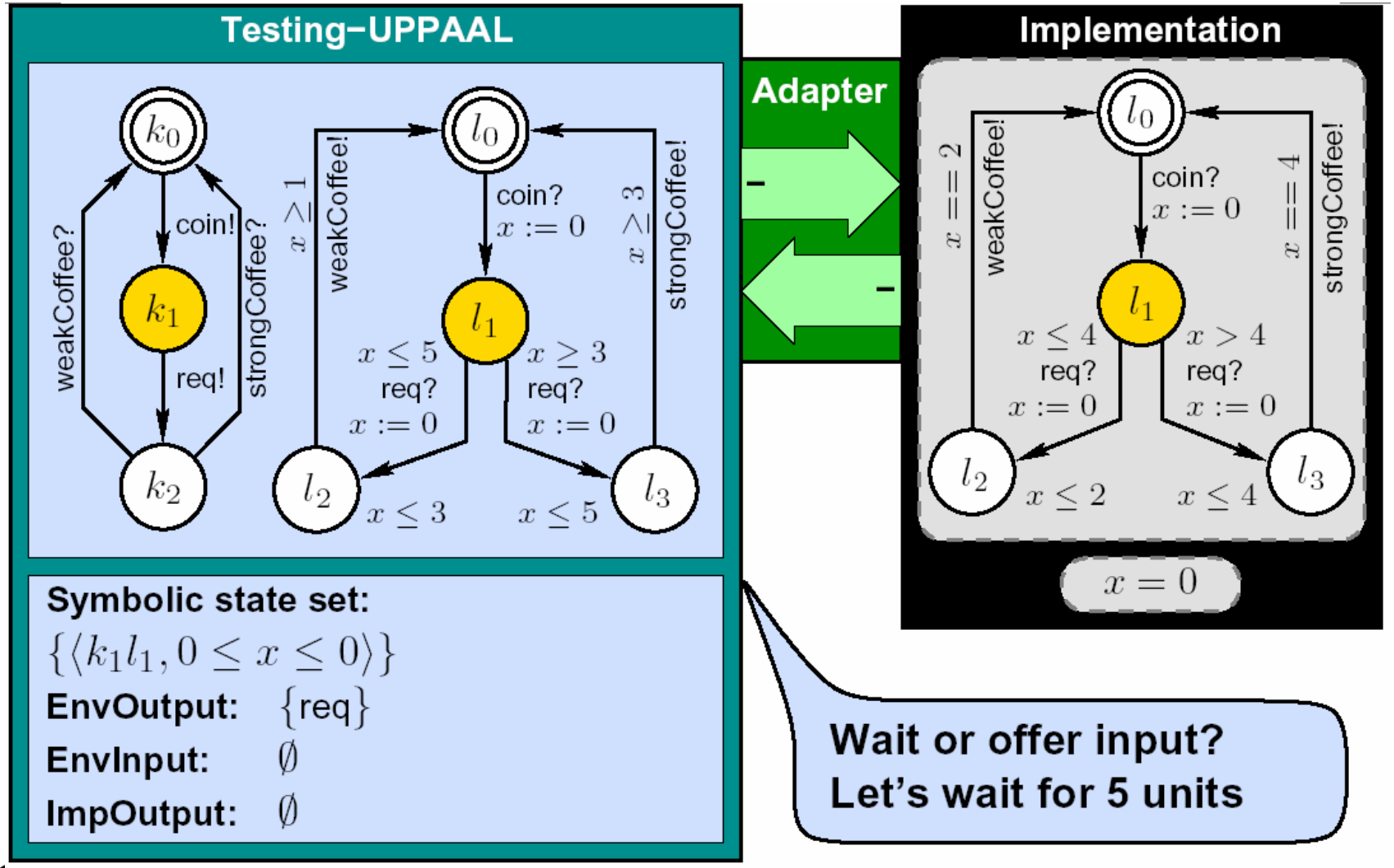
Online Testing



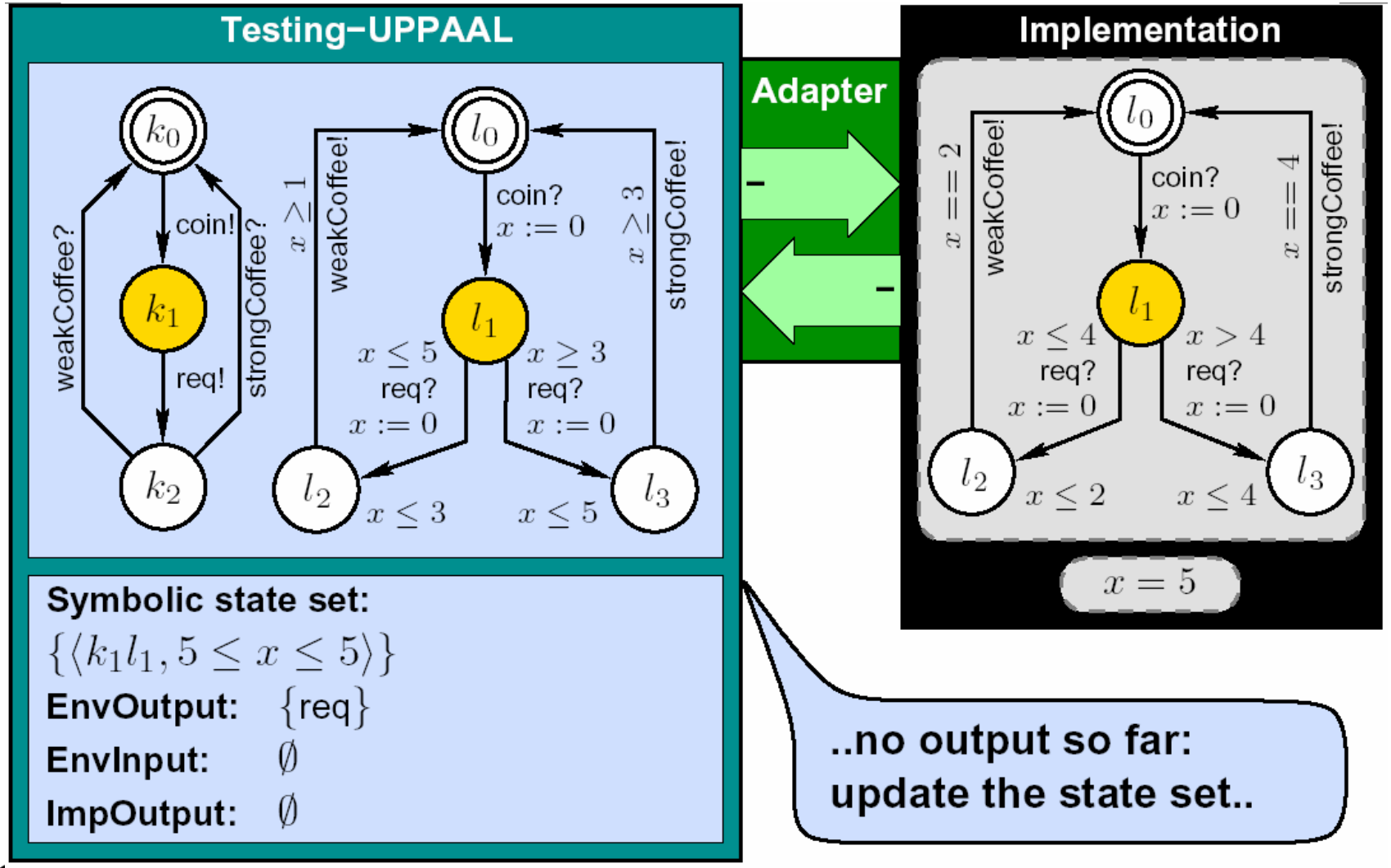
Online Testing



Online Testing

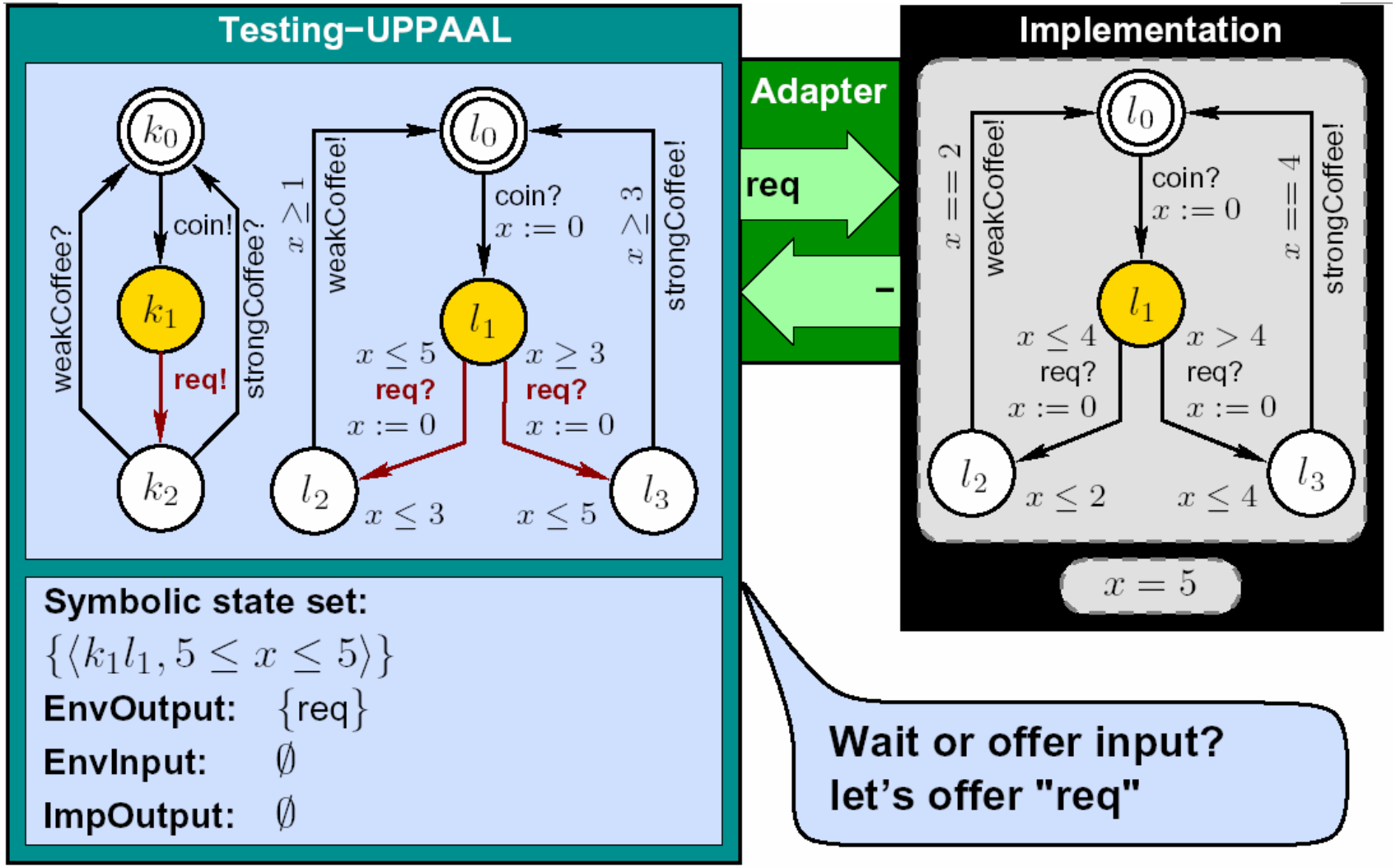


Online Testing



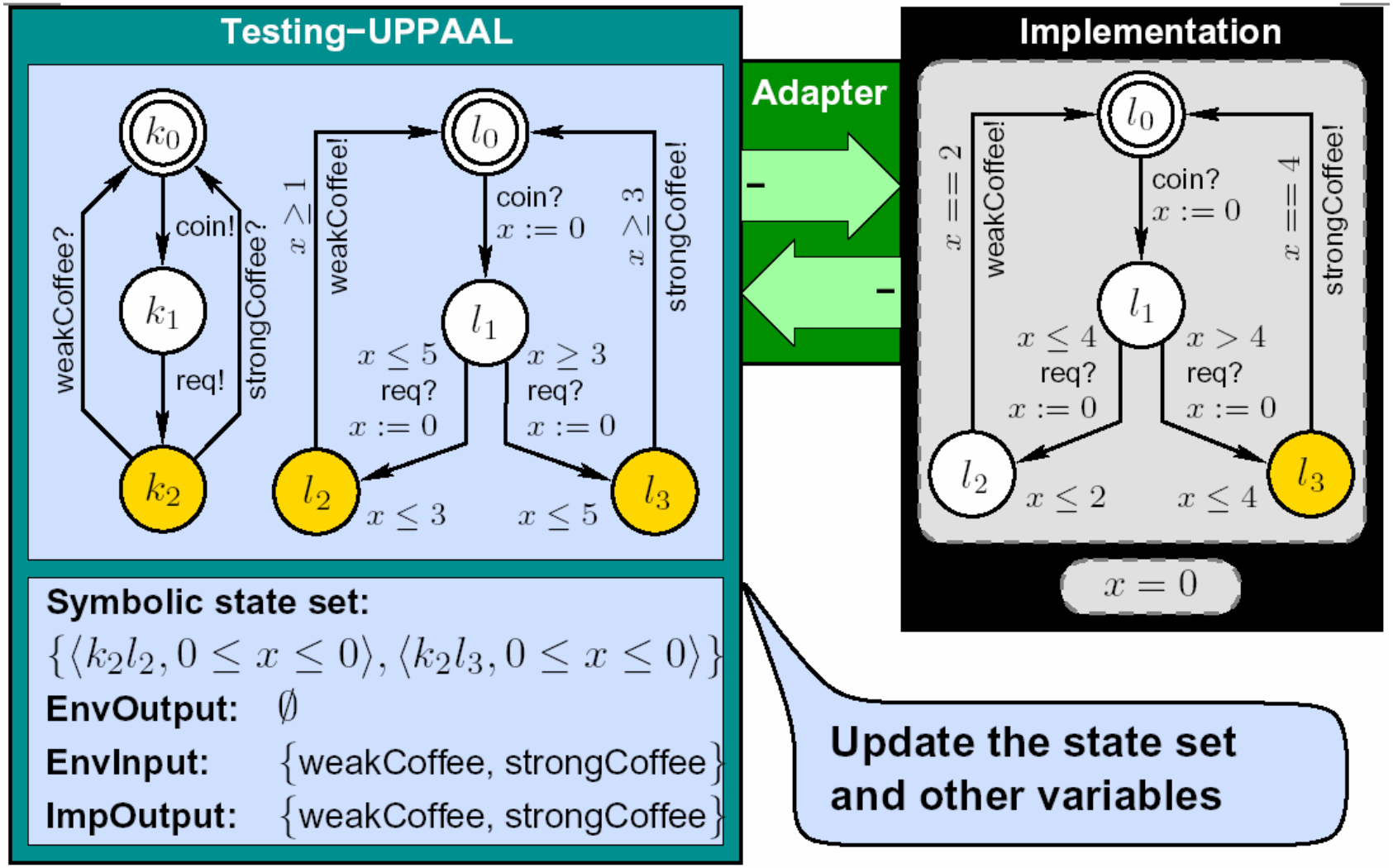
BRICS
Basic Research
in Computer Science

Online Testing



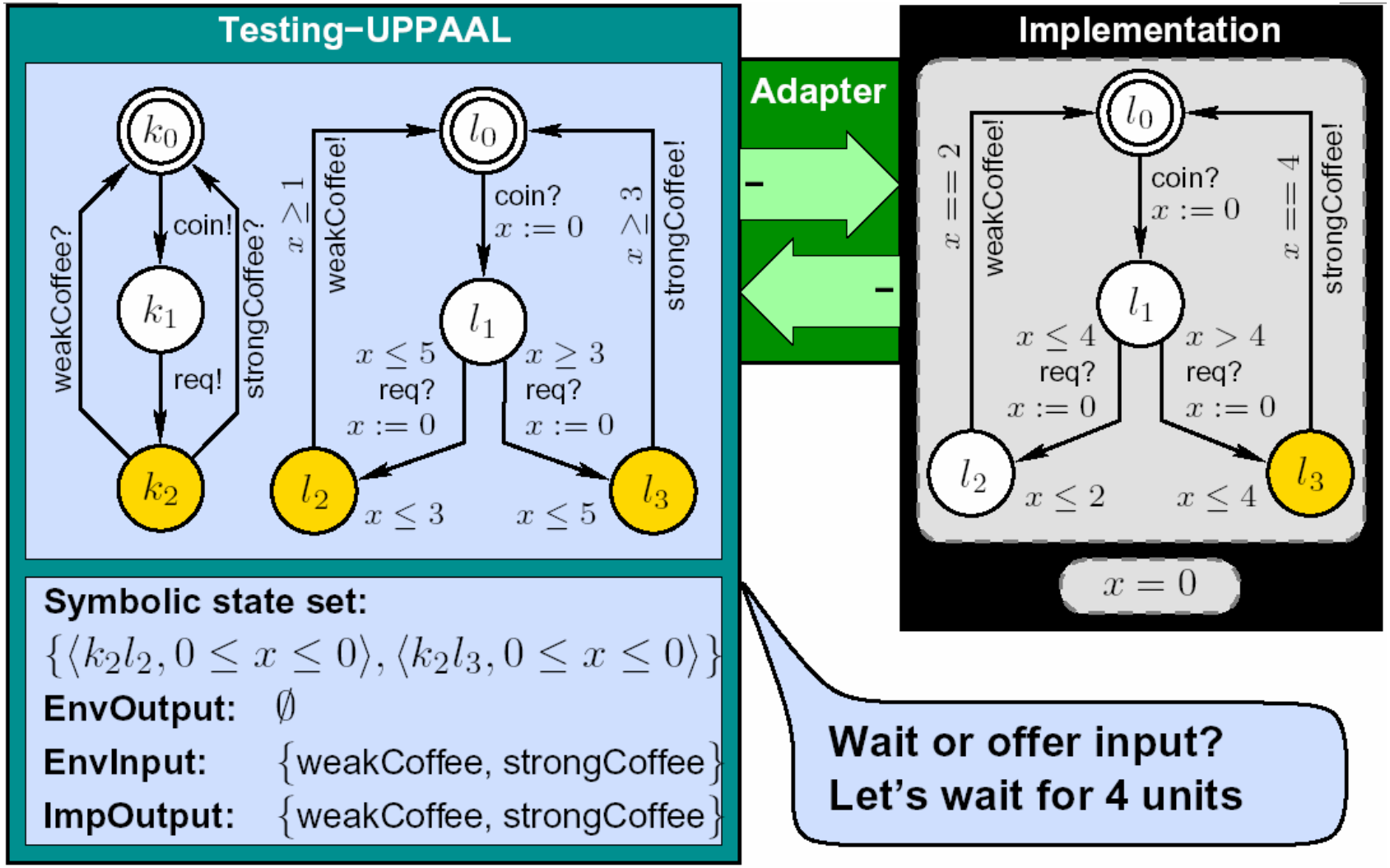
CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Online Testing



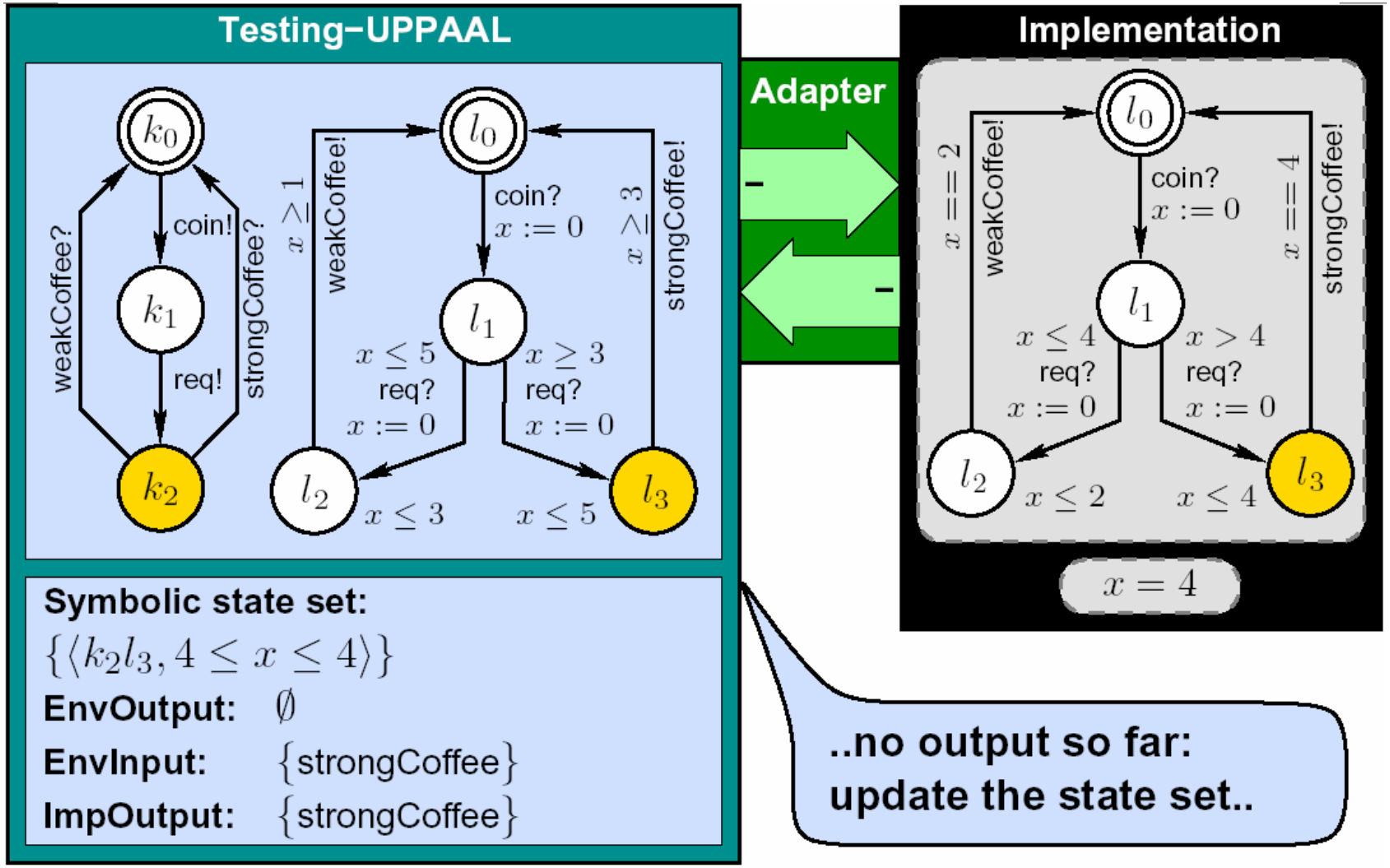
BRICS
Basic Research
in Computer Science

Online Testing

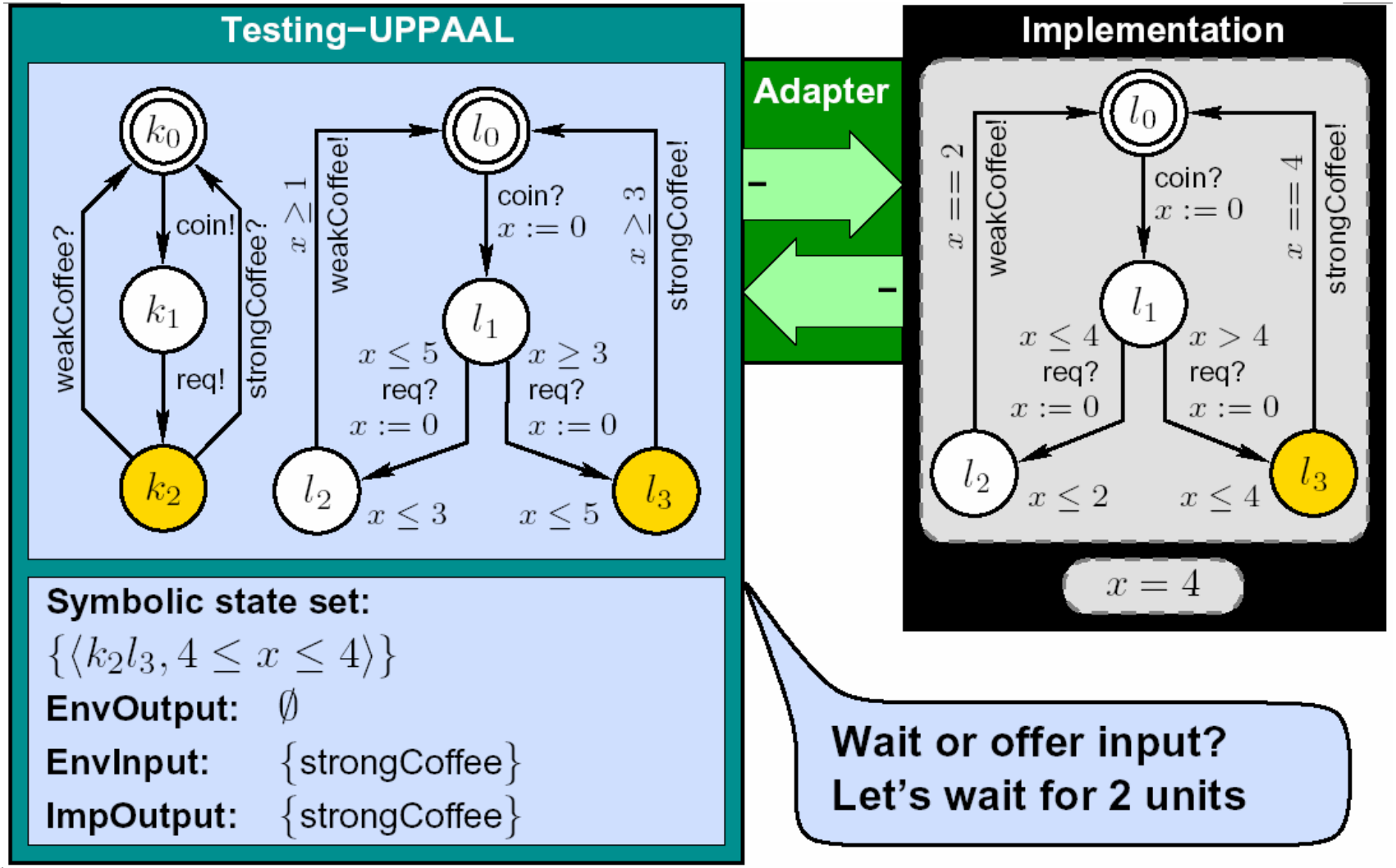


CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

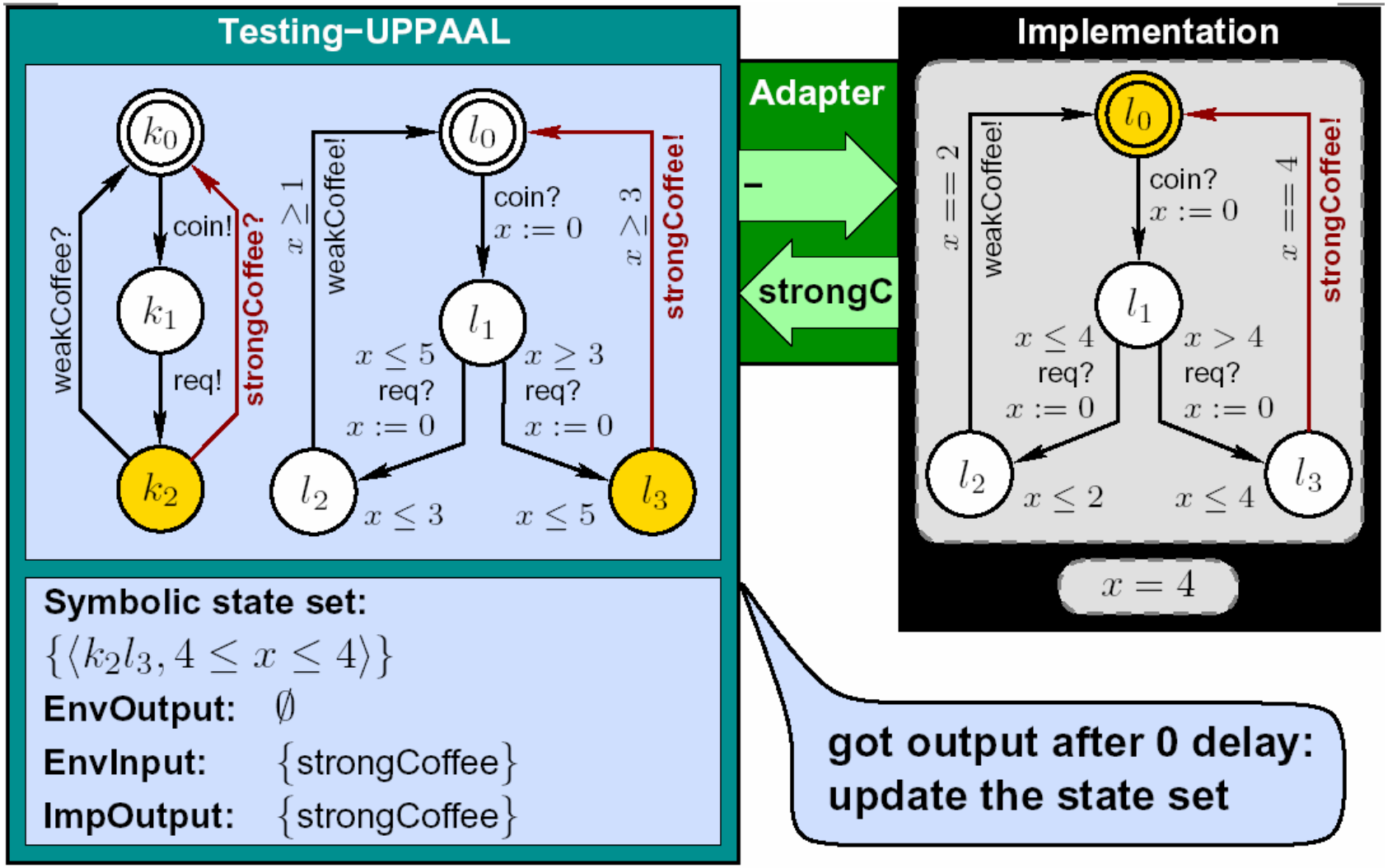
Online Testing



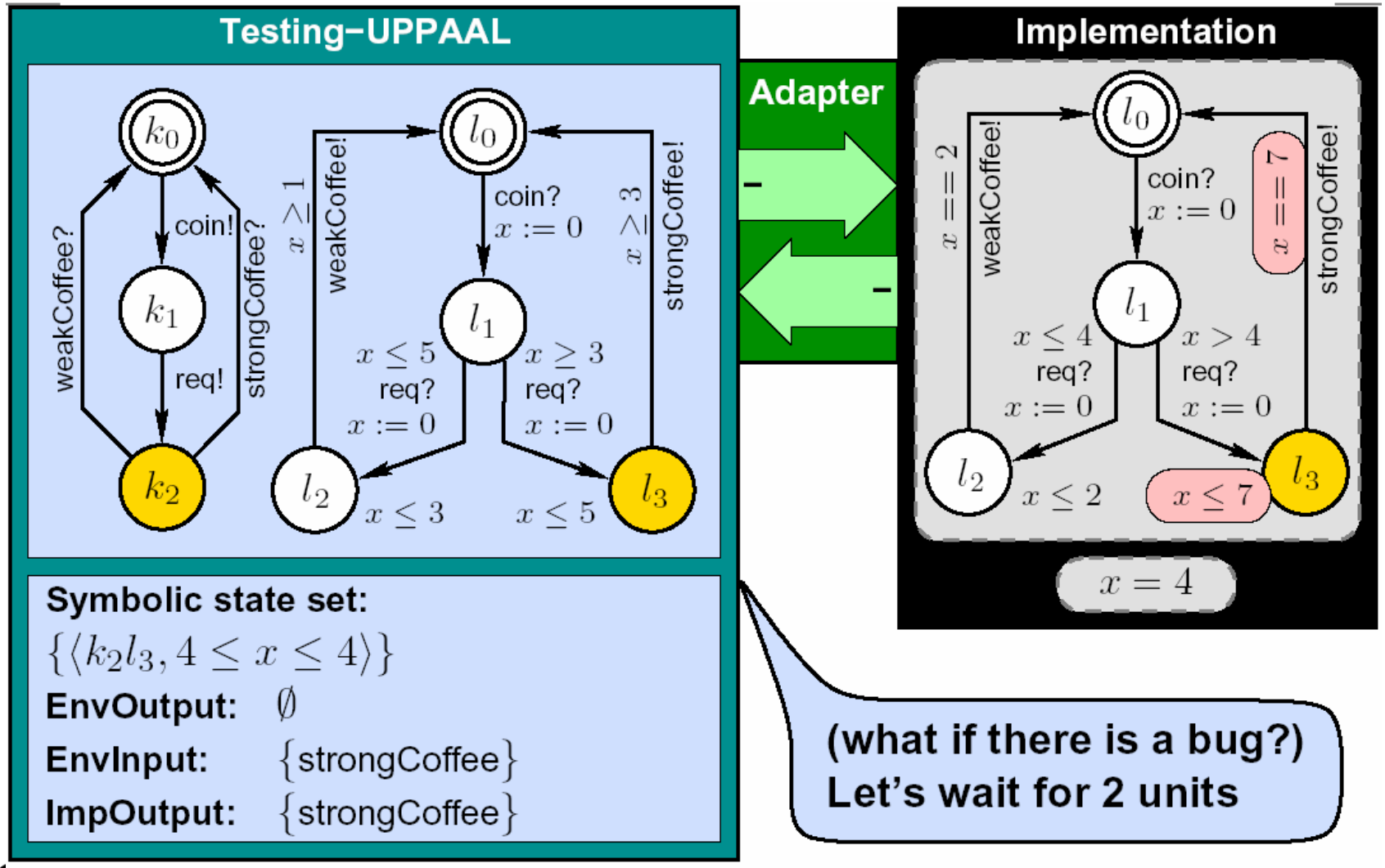
Online Testing



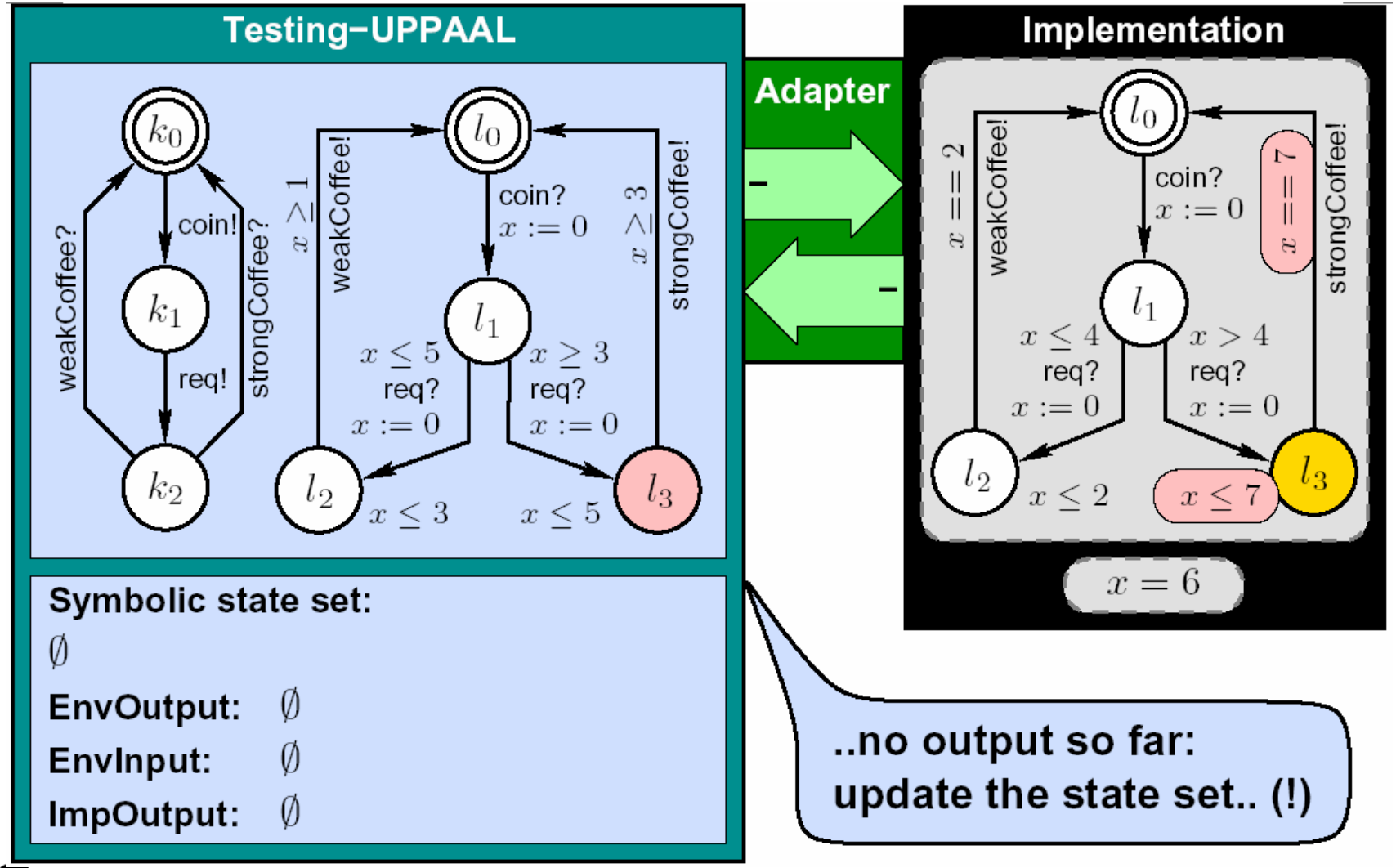
Online Testing



Online Testing



Online Testing



Industrial Application:

Danfoss Electronic Cooling Controller

Sensor Input
- air temperature sensor
- defrost temperature sensor
- (door open sensor)

Keypad Input
- 2 buttons (~40 user settable parameters)

Output Relays
- compressor relay
- defrost relay
- alarm relay
- (fan relay)

Display Output
- alarm / error indication
- mode indication
- current calculated temperature

- Optional real-time clock or LON network module

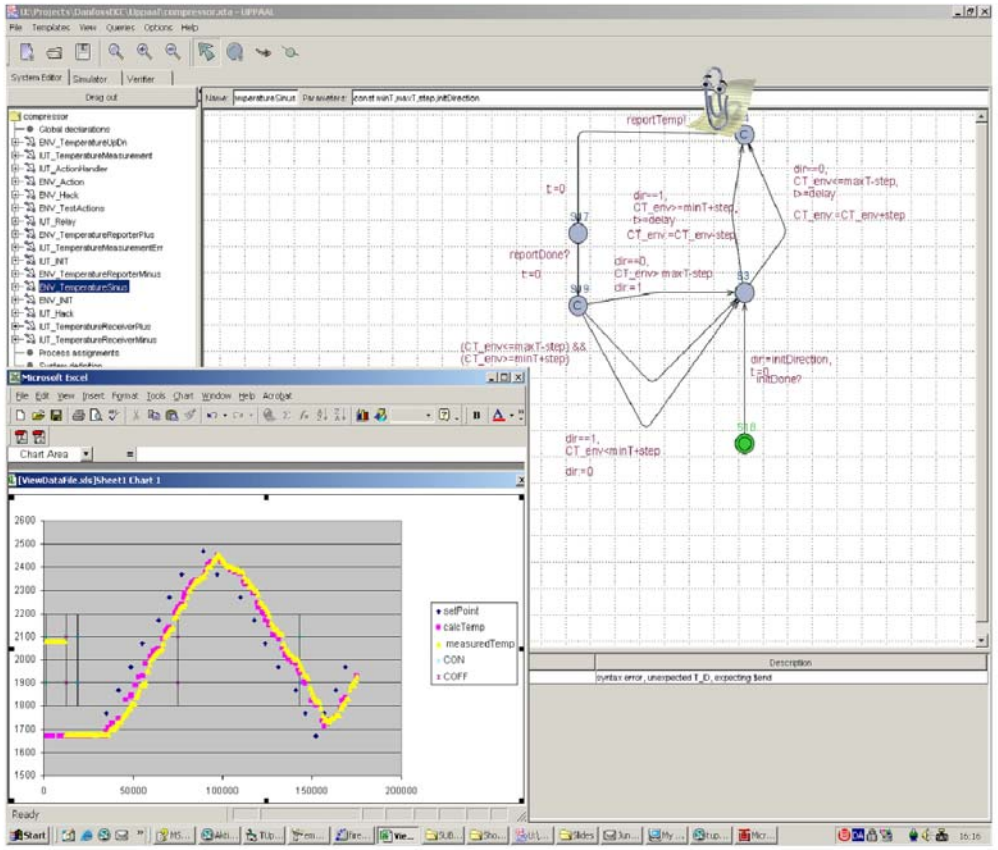
BRICS
Basic Research
in Computer Science

Industrial Application:

Danfoss Electronic Cooling Controller

Sensor Input

- air temperature sensor

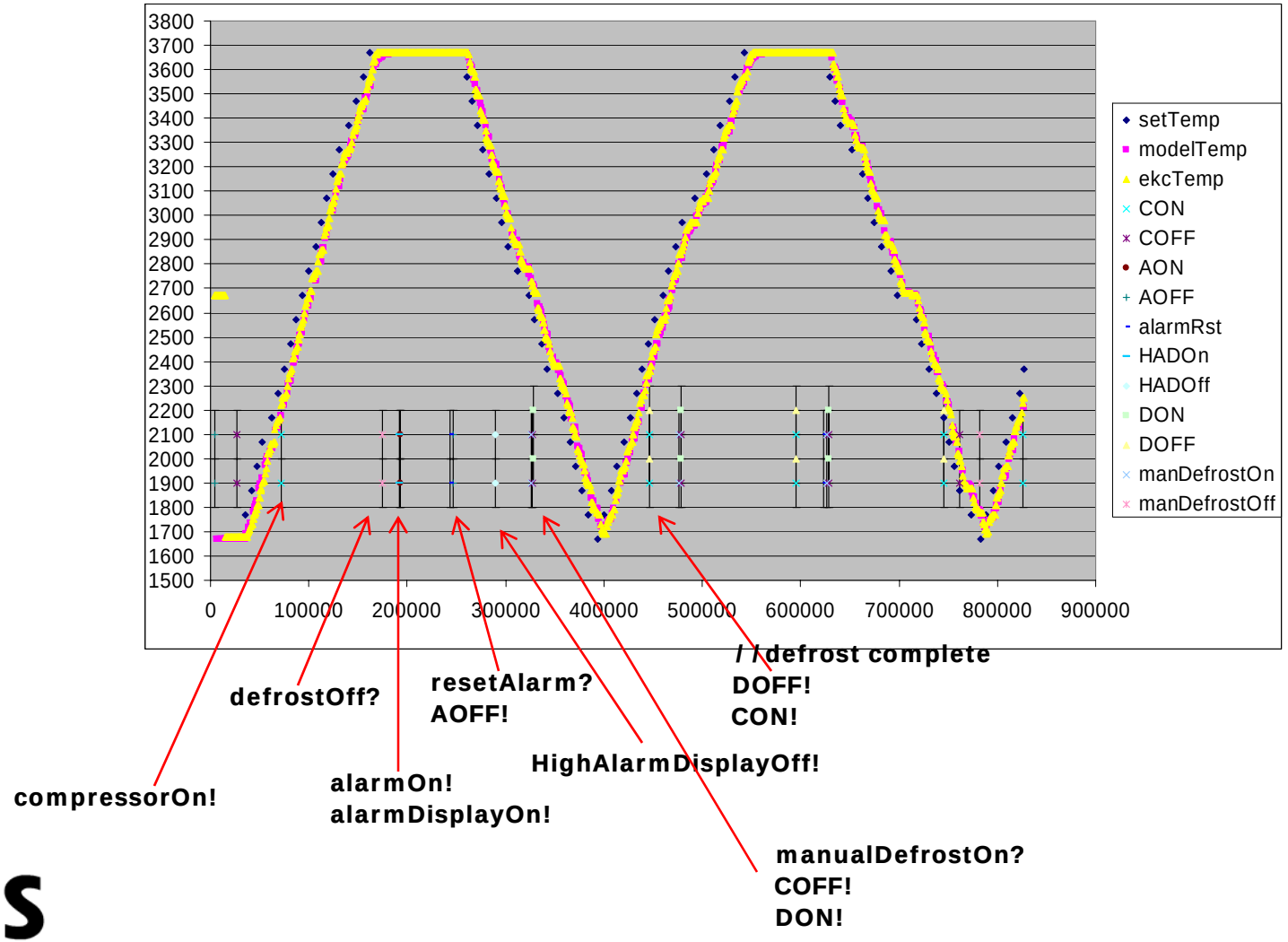


- Optional real-time clock or LON network module

CENTER FOR INDLEJEDE SOFTWARE SYSTEMER

Example Test Run

(log visualization)



Model-based Testing of Real Time Systems

Conclusions

BRICS
Basic Research
in Computer Science

BRICS
Basic Research
in Computer Science

Conclusions

- Testing real-time systems is theoretically and practically challenging
- Promising techniques and tools
- Explicit environment modeling
  - Realism and guiding
  - Separation of concerns
  - Modularity
  - Creative tool uses
  - Theoretical properties
- Real-time online testing from timed automata is feasible, but
  - Many open research issues

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

Research Problems

- Testing Theory
- Timed games with partial observability
- Hybrid extensions
- Other Quantitative Properties
- Probabilistic Extensions, Performance testing
- Efficient data structures and algorithms for state set computation
- Diagnosis & Debugging
- Guiding and Coverage Measurement
- Real-Time execution of TRON
- Adaptor Abstraction, IUT clock synchronization
- Further Industrial Cases

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

Related Work

- Formal Testing Frameworks
  - [Brinksma, Tretmans]
- Real-Time Implementation Relations
  - [Khoumsi'03, Briones'04, Krichen'04]
- Symbolic Reachability analysis of Timed Automata
  - [Dill'89, Larsen'97,…]
- Online state-set computation
  - [Tripakis'02]
- Online Testing
  - [Tretmans'99, Peleska'02, Krichen'04]

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER