Model-Based Testing of Real-time Systems

using UPPAAL COVER and UPPAAL TRON
UPPAAL FOR TESTING REALTIME SYSTEMS ONLINE

Uppaal's Branches for Testing

- Uppaal's branches for testing:
  - Uppaal-TRON
  - Uppaal-Cover

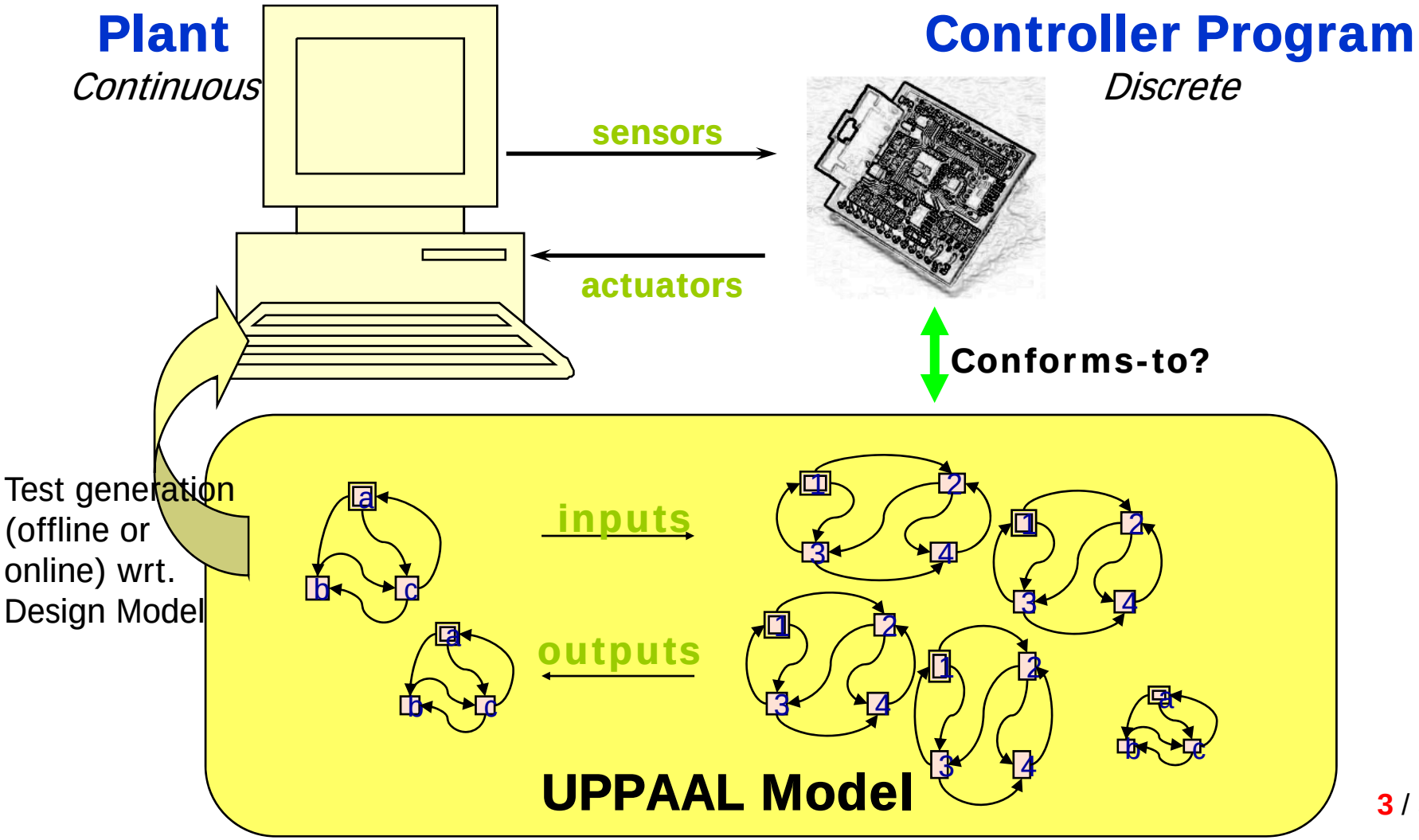
UPPAAL TRON
UPPAAL FOR TESTING REALTIME SYSTEMS ONLINE

(Off-line testing)

UPPAAL COVER

(On-line testing)

Real-time Model-Based Testing

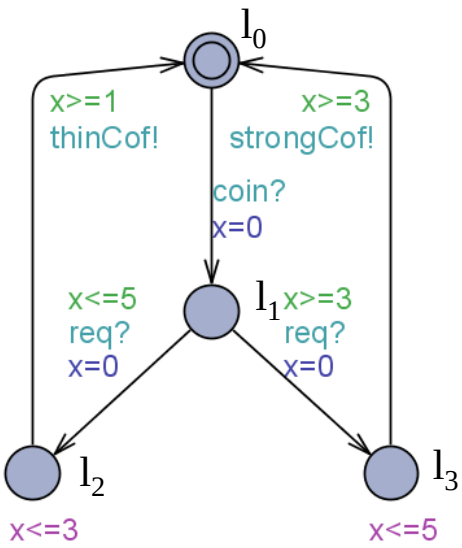


Timed System Testing

- Model:
  - Timed Input-Output Labelled Transition System (**Timed IOLTS**)
- Conformance relation:
  - Timed Input-Output Conformance (**Timed ioco**)

Timed IOLTS by Example

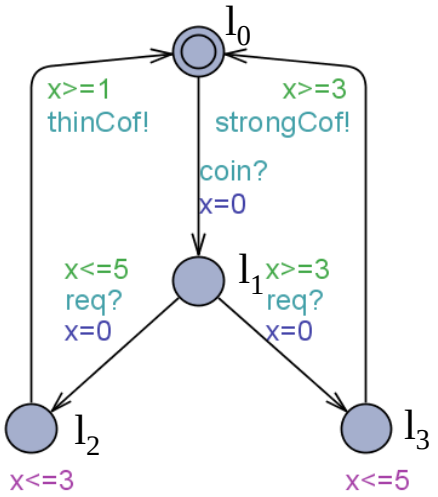
- Given a timed automaton:
  - location: $\{l_0, l_1, l_2, l_3\}$
  - actions:
    - {coin?, req?} --- input actions
    - {thinCof!, strongCof!} --- output actions
  - clock: {x}



- Semantic state:
  - e.g.: $(l_0, x=0)$, $(l_0, x=2)$, $(l_1, x=4)$
- Semantic transition:
  - e.g.: $(l_0, x=0)$ --delay(2)--> $(l_0, x=2)$,
    $(l_0, x=2)$ --coin?--> $(l_1, x=0)$,

Such a transition system is a timed IOLTS
- as semantic interpretation of TA
- typically infinite transition systems (because clocks are real variables)

Timed Conformance: tioco



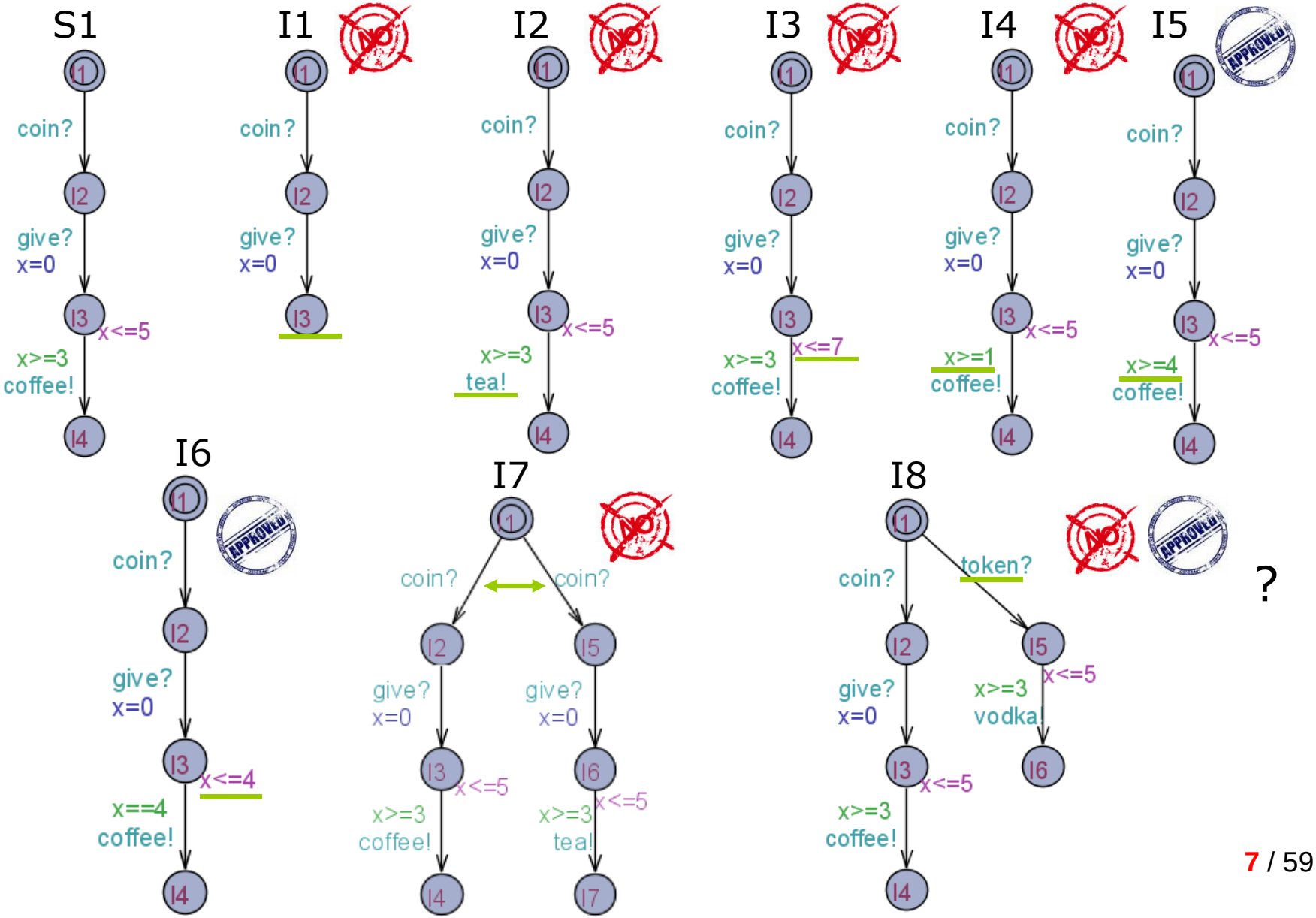
- Derived from Tretman's ioco
- Let I, S be two timed IOLTS's, P a set of states
  - TTr(P): the set of timed traces from a state in P
    - eg.: σ = coin?.5.req?.2.thinCoffee!.9.coin?
  - Out(P after σ) = possible outputs and delays after σ
    - eg. out ($\{l_2, x=1\}$): {thinCoffee, 0...2}

- I tioco S $=_{def}$
  - $\forall \sigma \in$ TTr(S): Out(I after σ) $\subseteq$ Out(S after σ), or
  - TTr(i_0) $\subseteq$ TTr(s_0), where i_0 and s_0 are the initial states of I and S respectively

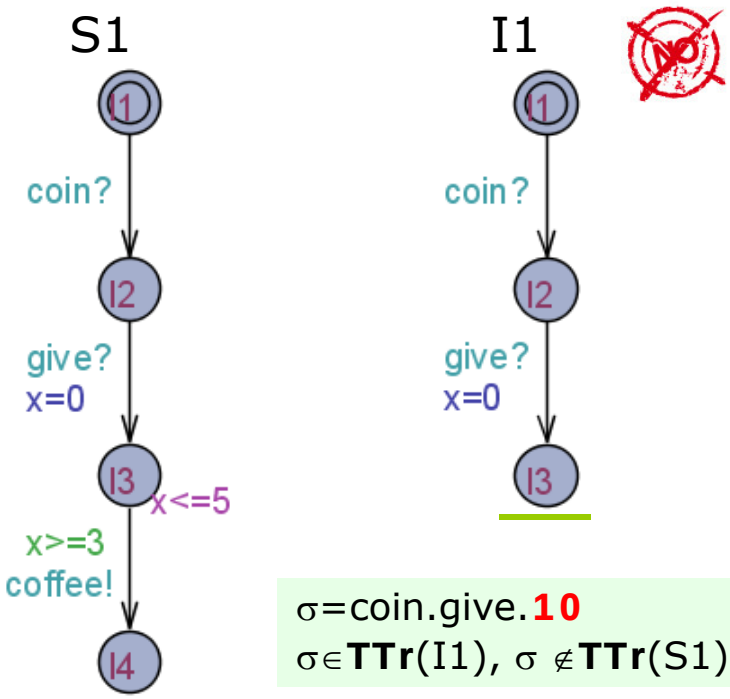
- Intuition
  - IUT can accept all inputs for SPEC (and perhaps some other inputs)
  - if IUT ever produces an output as required by SPEC, it should be produced in time
  - but IUT is not allowed to produce any illegal output (w.r.t. SPEC)

See also [Krichen&Tripakis, Khoumsi]

Does I_n Conform-to S_1?



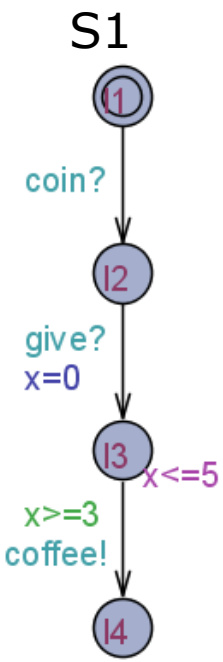
Does I_n Conform-to S_1?

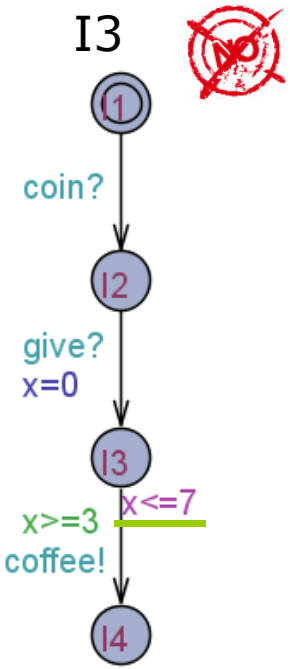


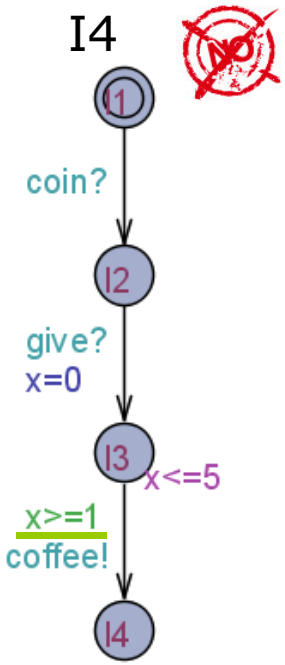
σ=coin.give.**10**
$\sigma \in$ **TTr**(I1), $\sigma \notin$ **TTr**(S1)

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?







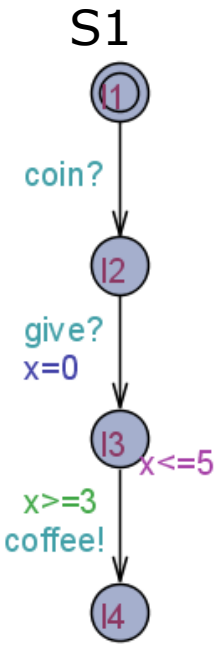
σ=coin.give.**7**.coffee
σ∈**TTr**(I3), σ ∉TTr(S1)

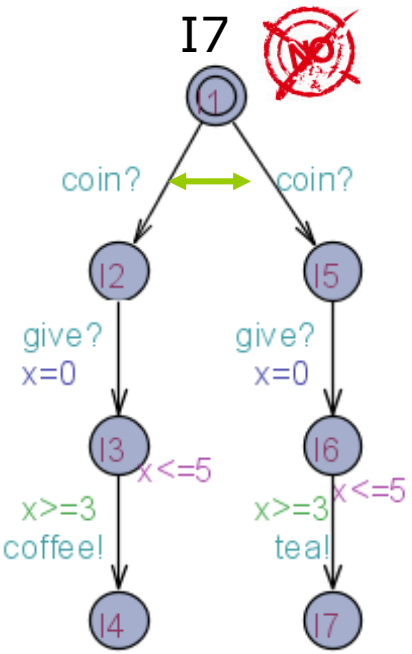
σ=coin.give.**1**.coffee
σ∈**TTr**(I4), σ ∉TTr(S1)

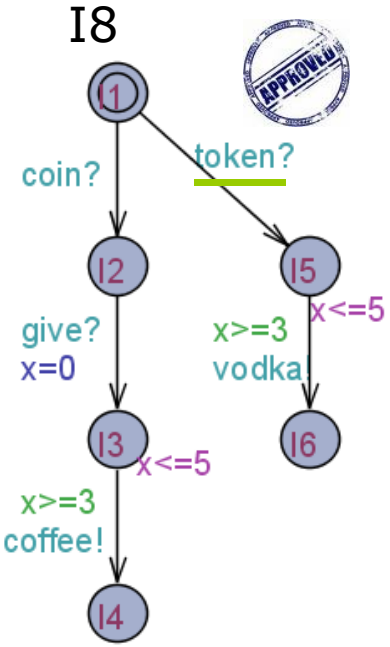
out(I3 after coin.give.7)={coffee,0}
⊄
out(S1 after coin.give.7)={}

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

Does I_n Conform-to S_1?







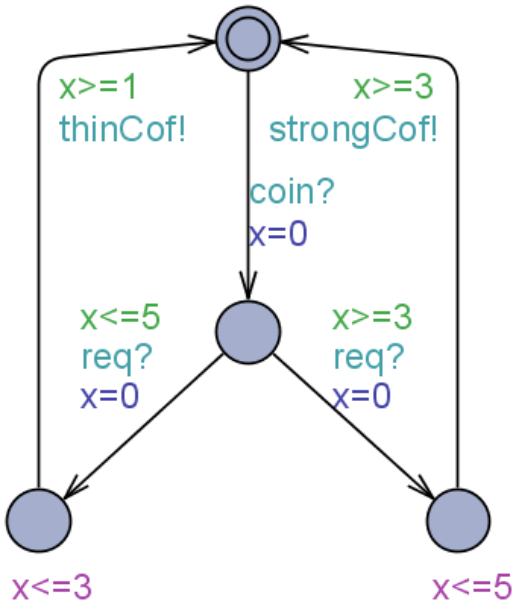
σ=coin.give.5.**tea**
σ∈**TTr**(I7), σ ∉**TTr**(S1)

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

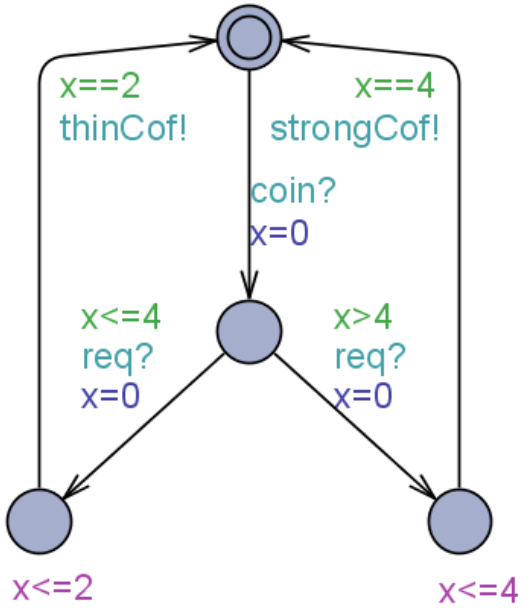
out(I7 **after** coin.give.5)={tea, coffee,0}
⊄
out(S1 **after** coin.give.5)={coffee,0}

Now, Back to Timed Coffee Machine

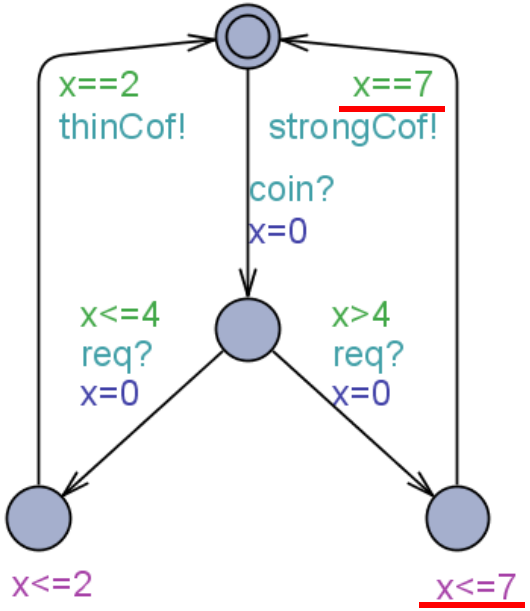
Specification



Implementation 1



Implementation 2



Example Traces

- c?.2.r?.2.weakC
- c?.5.r?.4.strongC

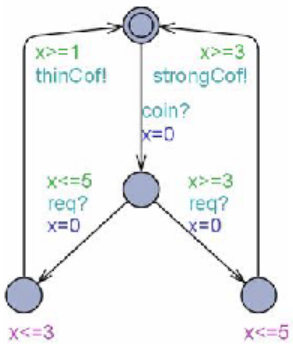
I1 **rt-ioco** S

- c?.2.r?.2.weakC
- c?.5.r?.7

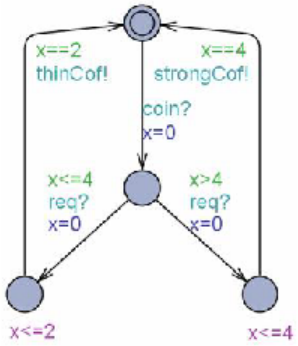
I2 **rt-ioco** S

Essence of "Timed ioco"?

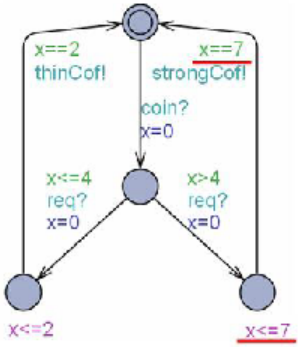
Specification



Implementation 1



Implementation 2



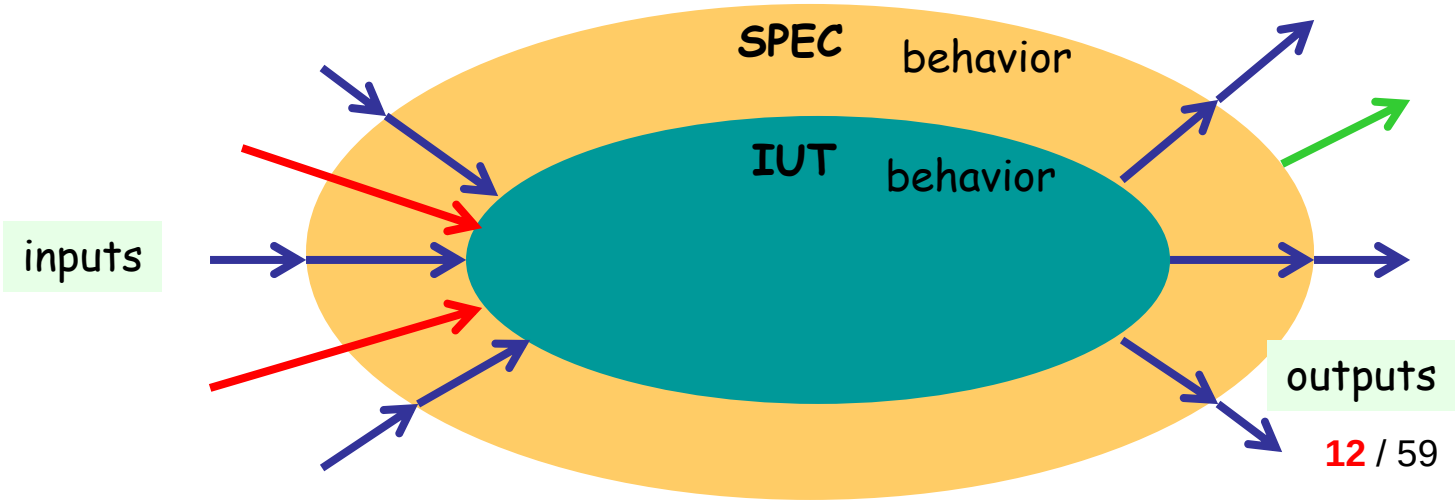
Example Traces

- c?.2.r?.2.weakC
- c?.5.r?.4.strongC

I1 **rt-ioco** S

- c?.2.r?.2.weakC
- c?.5.r?.7

I2 **rt-ioco** S



Explicit Environment Modelling

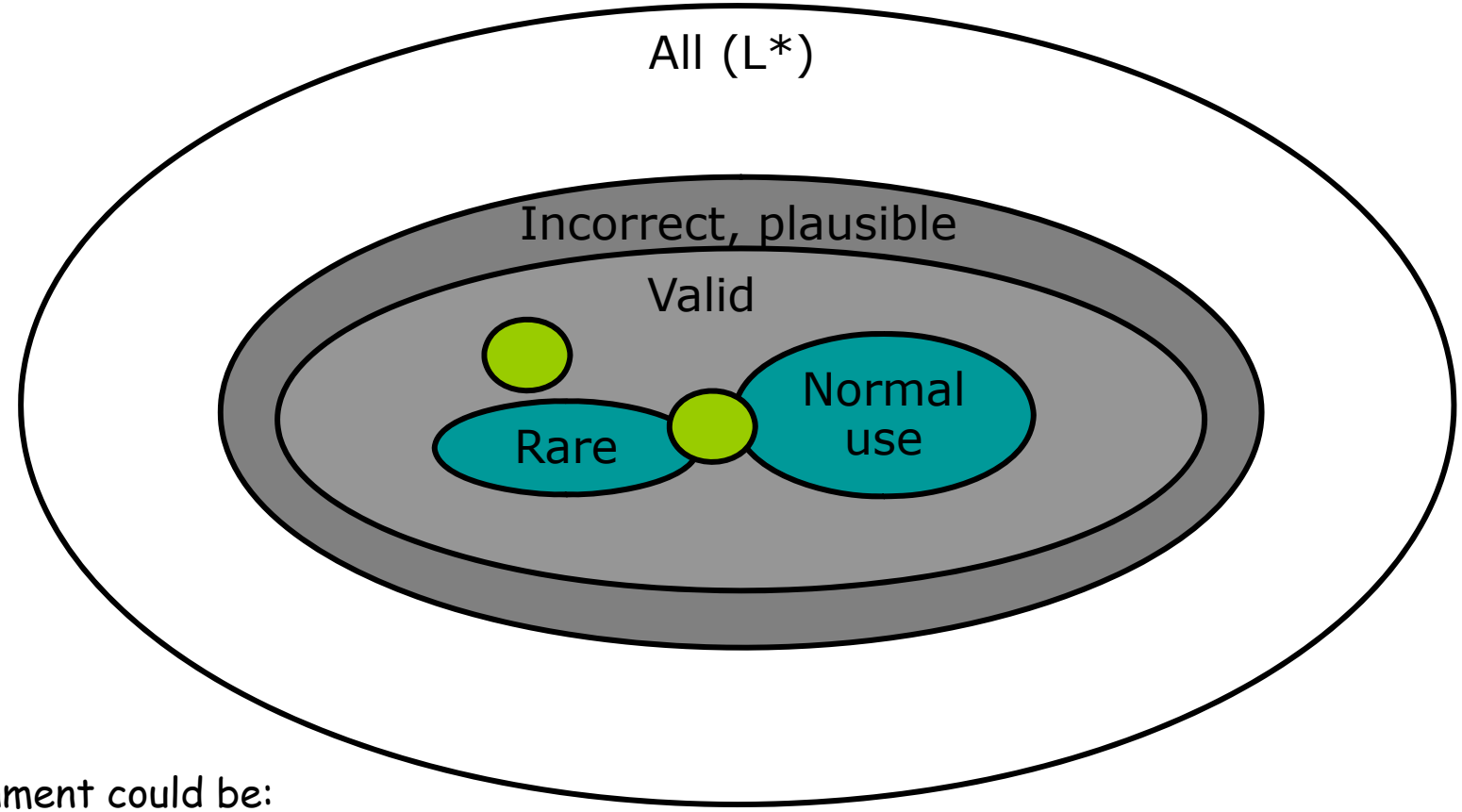
Recall that in "ioco" conformance…

- I tioco S $=_{def}$
  - $\forall \sigma \in TTr(S)$: Out(I after σ) $\subseteq$ Out(S after σ), or
  - $TTr(i_0) \subseteq TTr(s_0)$, where i_0 and s_0 are the initial states of I and S respectively

- Note that:
  - TTr(S) is a very big (infinite) set
  - We are usually interested in only a small portion of the behavior

- A solution:
  - To explicitly model the environment that the IUT will be operated in

The Environment "Universe"



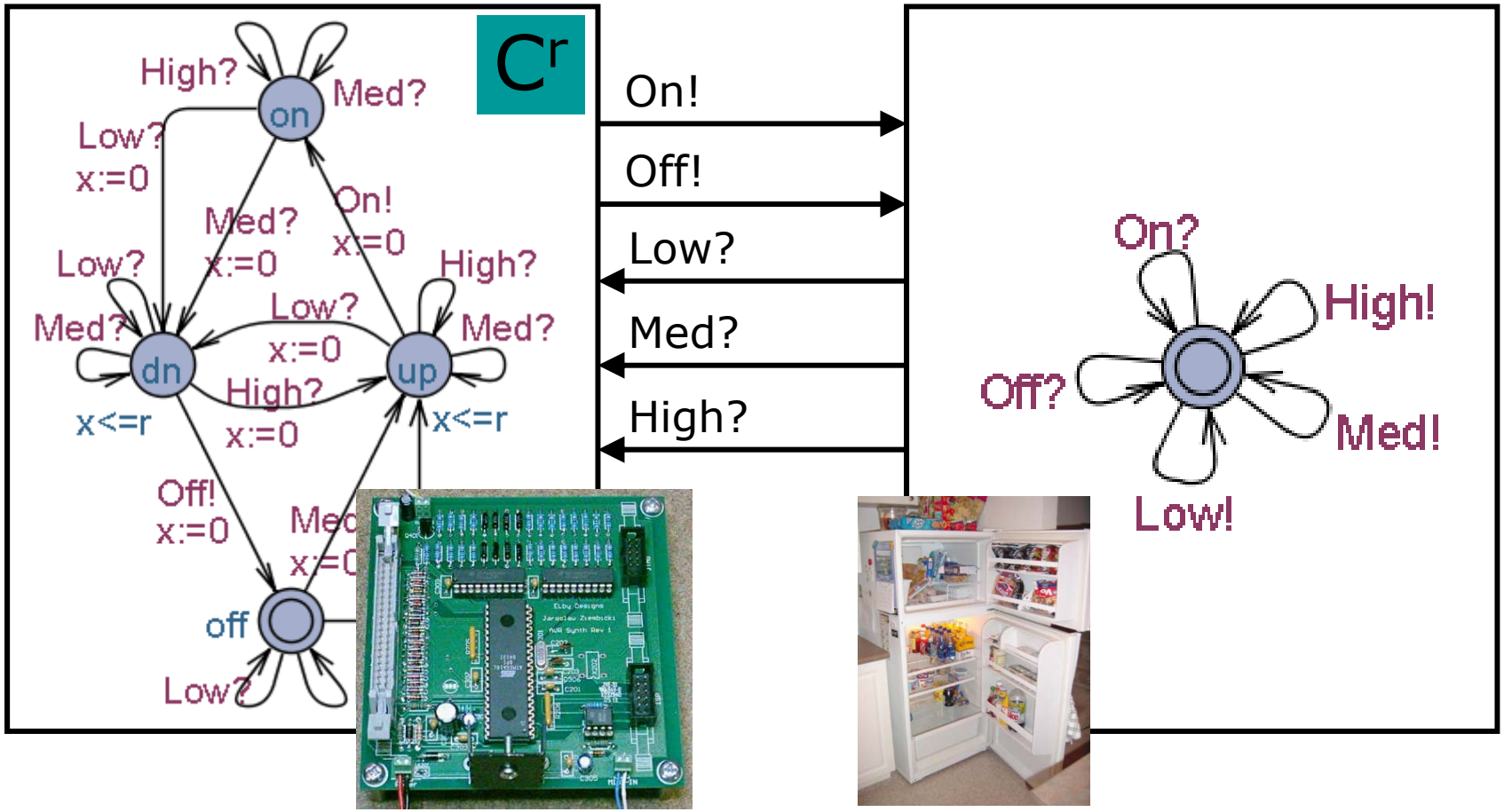
environment could be:

- Other external systems (Dedicate / open protocols)
- Other internal systems (eg powersupply, radio)
- Human Users
- Physical Plant via sensors / actuators

Sample Cooling Controller

IUT-model

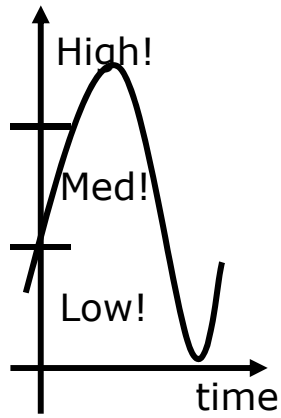
Env-model

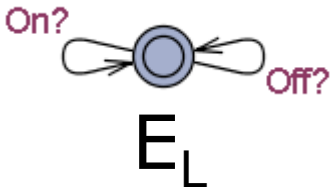


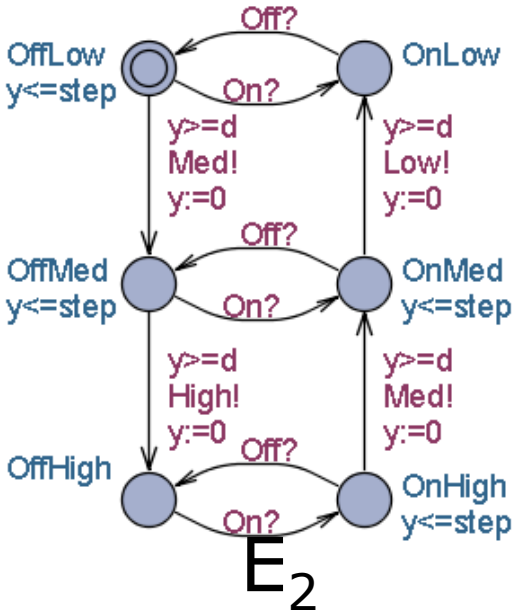
- When T is high (low) switch on (off) cooling within r secs.
- When T is medium cooling may be either on or off (impl. freedom)

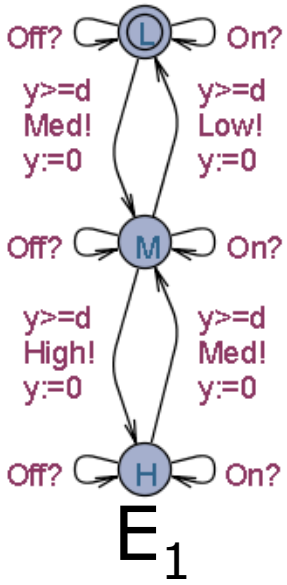
Environment Modelling

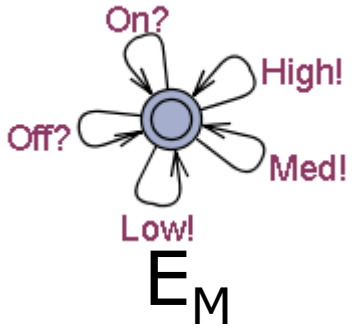
- E_M Any action possible at any time
- E_1 Only realistic temperature variations
- E_2 Temperature never increases when cooling
- E_L No inputs (completely passive)







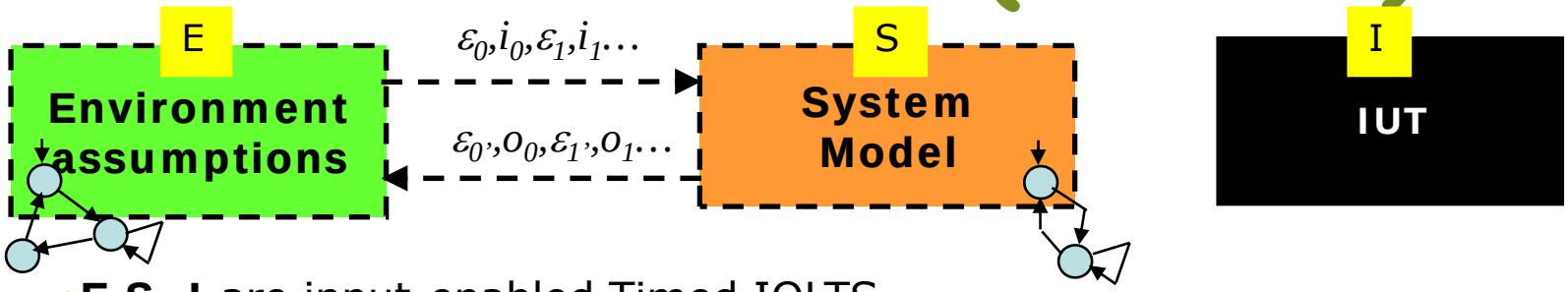




(strict) ⟷ (loose)

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

Relativized Timed Input-Output Conformance (rt-ioco)



- **E,S, I** are input-enabled Timed IOLTS
- **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)

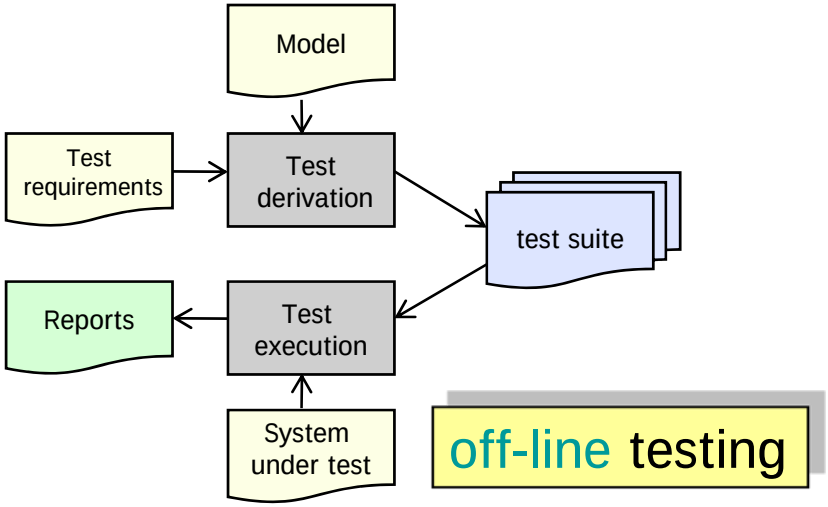
or

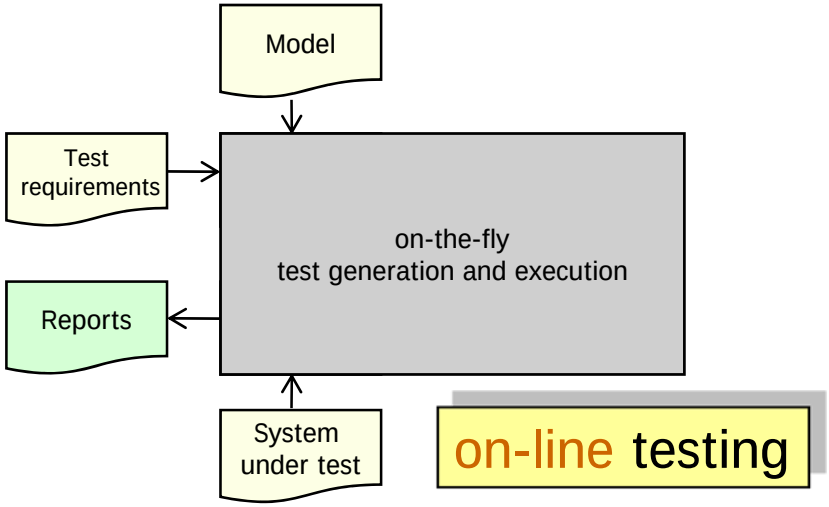
- I rt-ioco$_E$ S iff TTr(I) $\cap$ TTr(E) $\subseteq$ TTr(S) $\cap$ TTr(E) // *input enabled*

- **Intuition: for all assumed environment behaviors, the IUT**
  - **never produces illegal output, and**
  - **if ever produces required output, then produces it in time**

See also [Larsen 04 FATES]

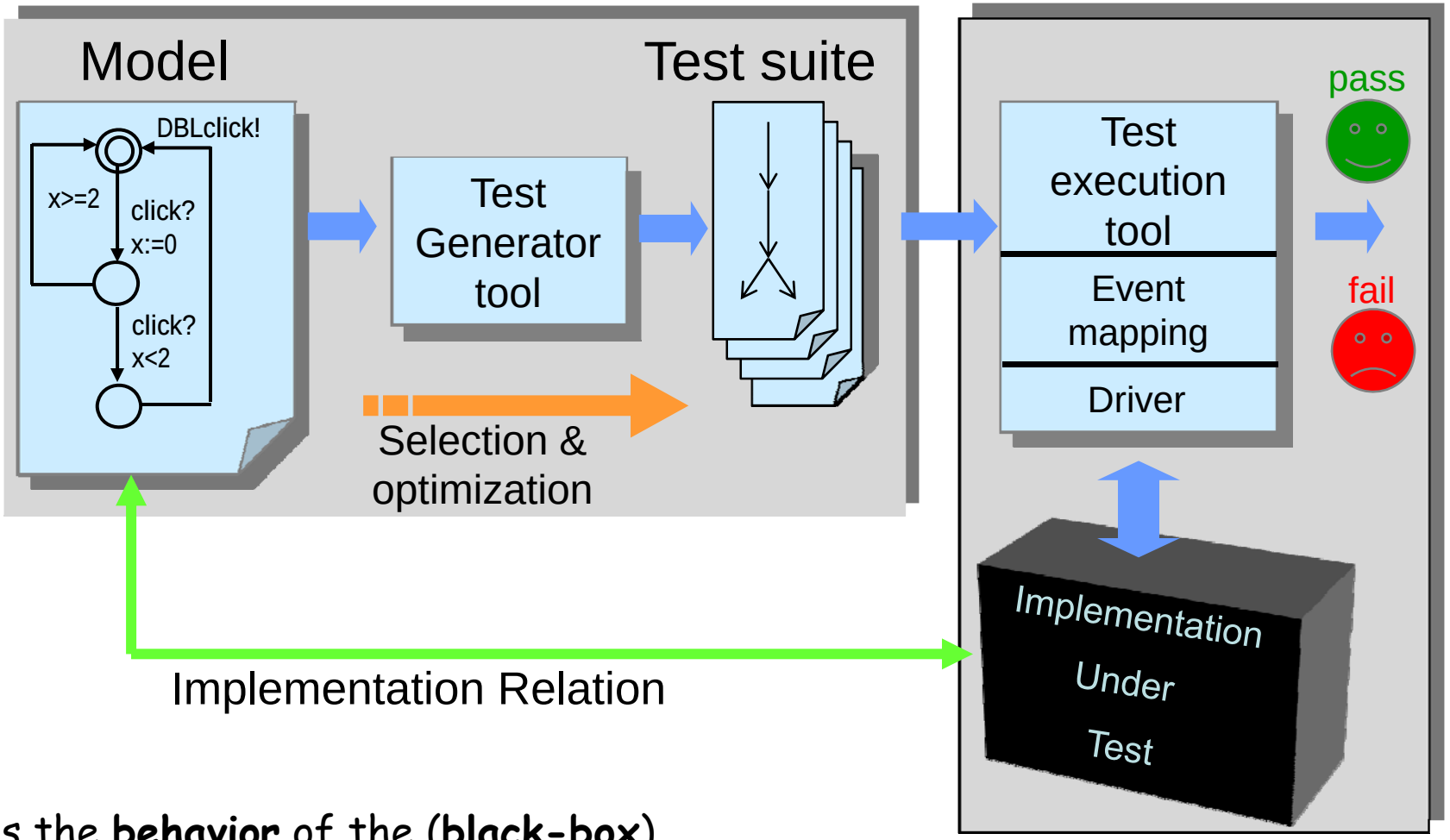
Off-line and On-line Testing





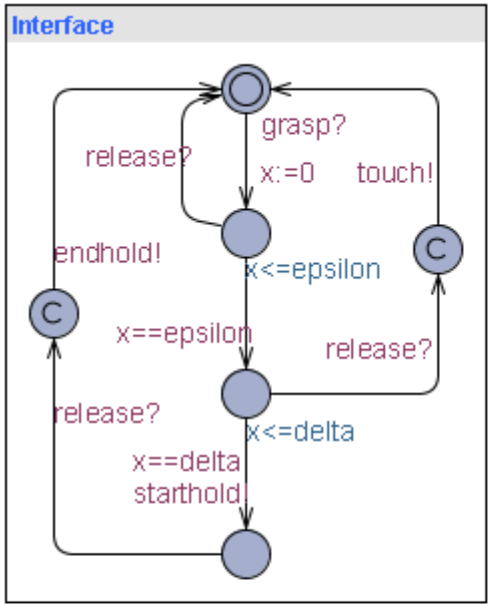
Model-Based Off-line Testing of Timed Systems

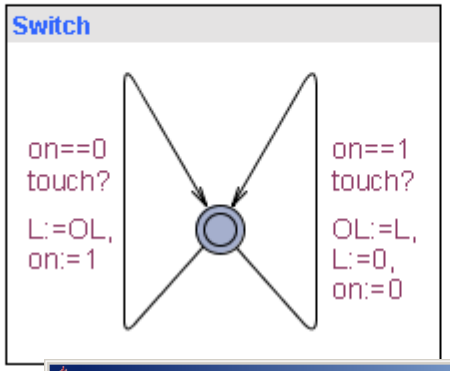
Automated Model-Based Off-line Conformance testing

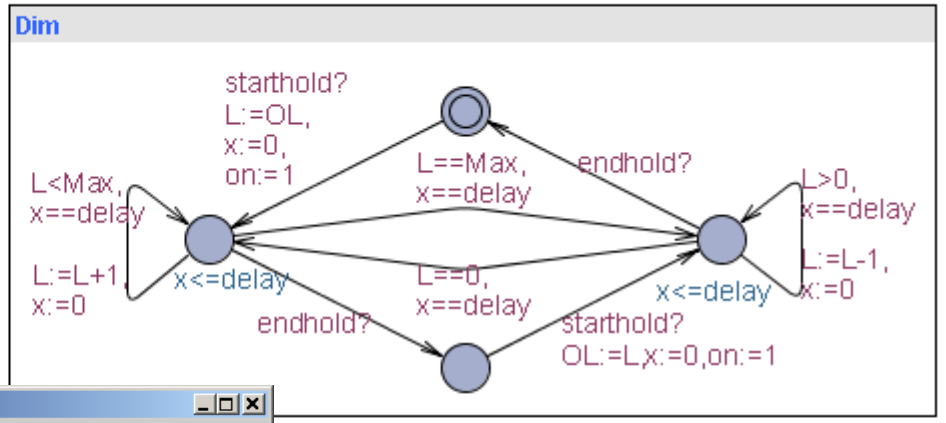


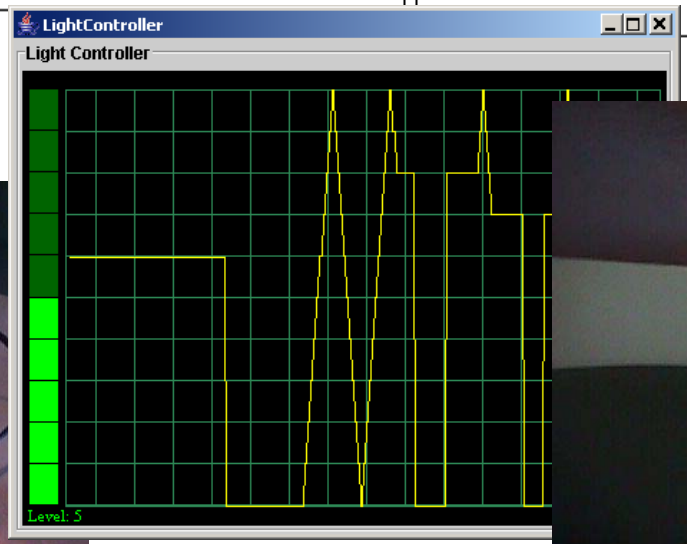
Does the **behavior** of the (**black-box**) implementation ***comply*** to that of the specification?

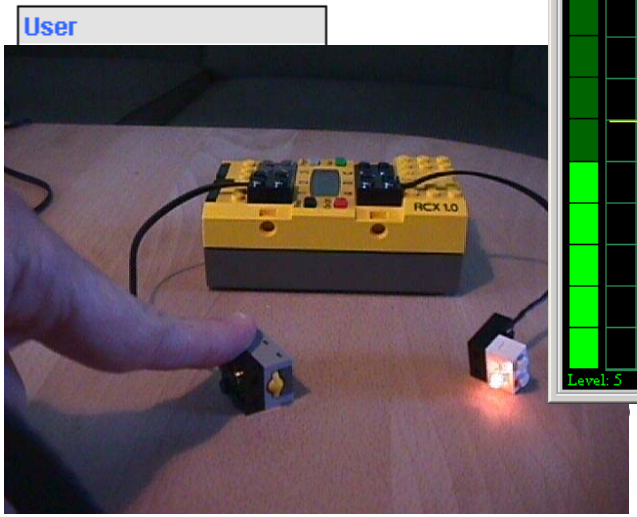
Touch-sensitive Light Controller



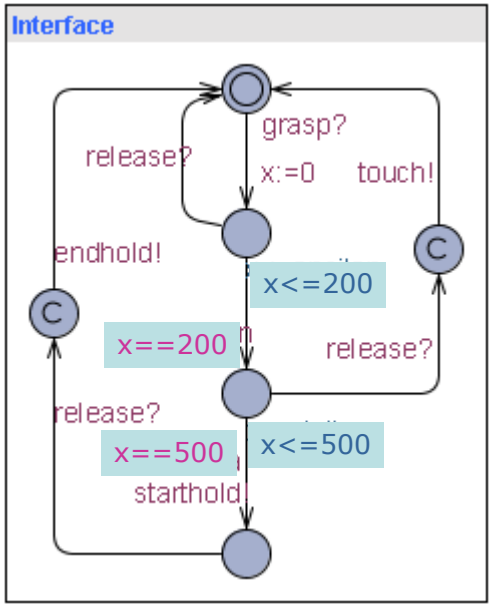


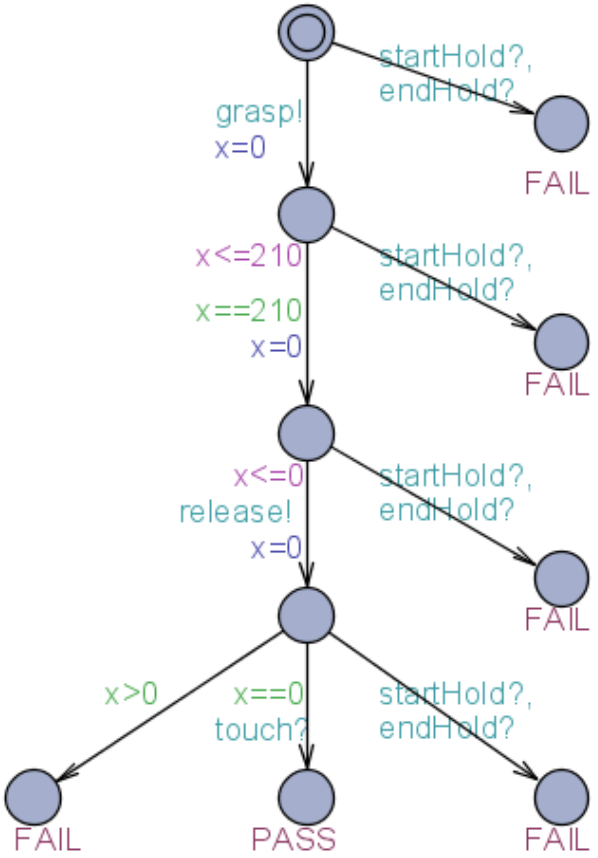






Timed Tests





EXAMPLE test cases for **Interface**

0·grasp!·210·release!·touch?.PASS

0·grasp!·317·release!·touch?·2½·grasp!·220·release!·touch?·PASS

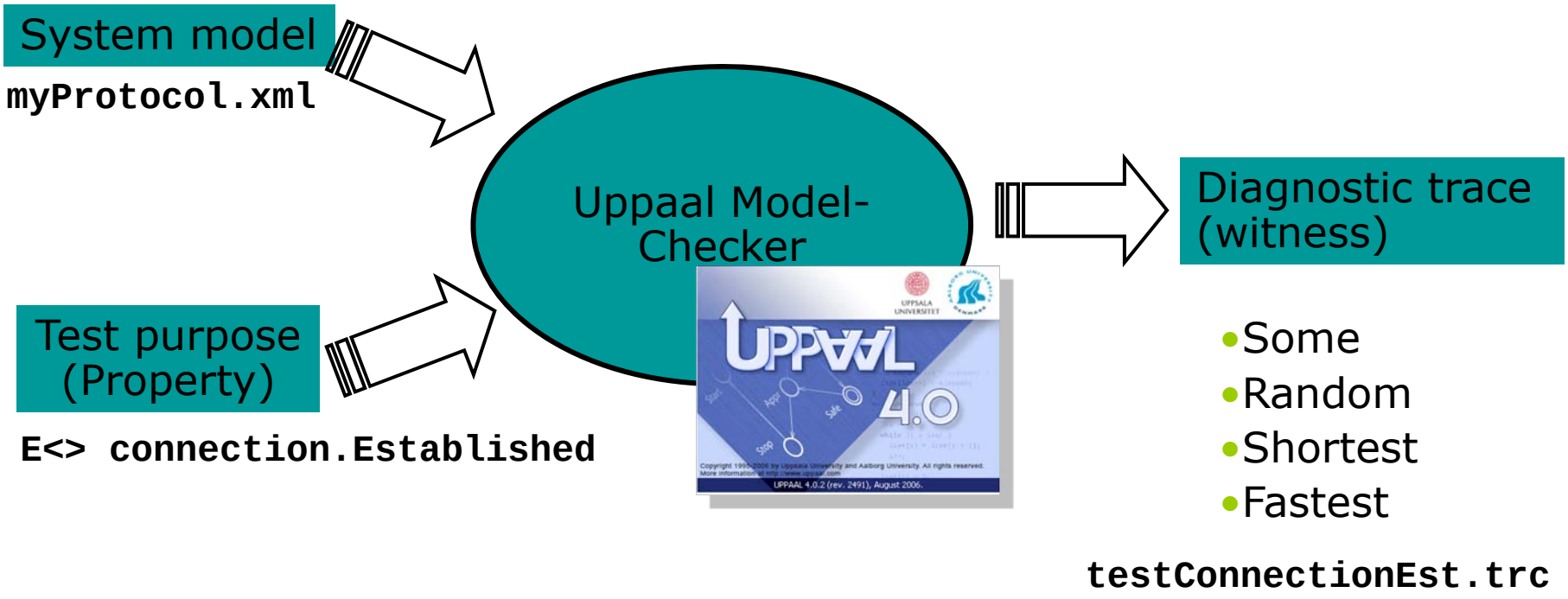
1000·grasp!·517·starthold?·100·release!·endhold?·PASS

Infinitely many sequences!!!!!!

Test Selection?

- Infinitely many sequences…
- But testing practice should definitely be finite
- To select finitely many out from an infinitely large pool
  - Test coverage criteria
  - Test purposes

Test Generation by Model-Checking



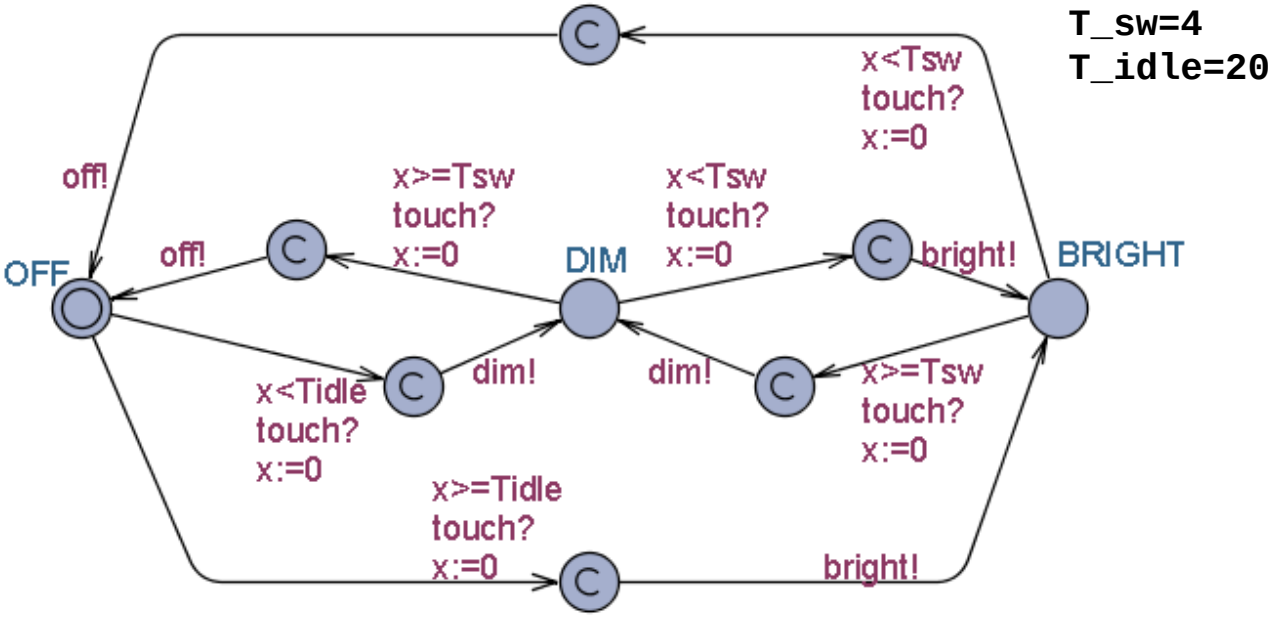
- Use diagnostic trace as test case??!!

Controllable Timed Automata

- "**DOUTA**"-**Model**
  - **D**eterministic: two transitions with same input/output leads to the same state
  - **O**utput-**U**rgent: enabled outputs will occur immediately
  - **Isolated Outputs**: if an output is enabled, no other output is enabled
  - **Input-Enabled**: all inputs can always be accepted

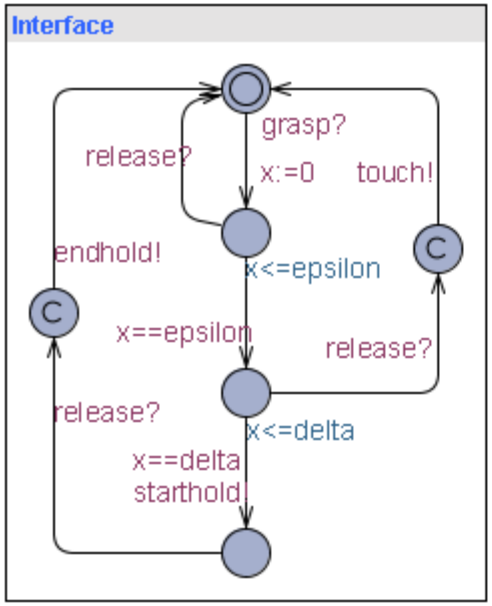
A DOUTA Timed Automaton

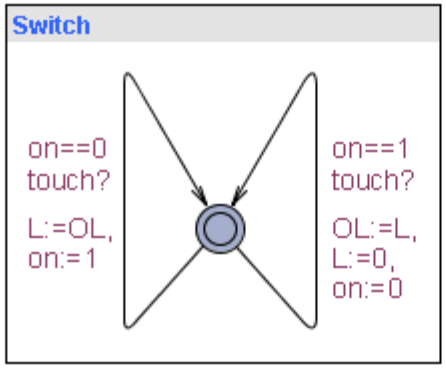
Deterministic,
Output-Urgent,
Isolated Outputs,
Input-Enabled

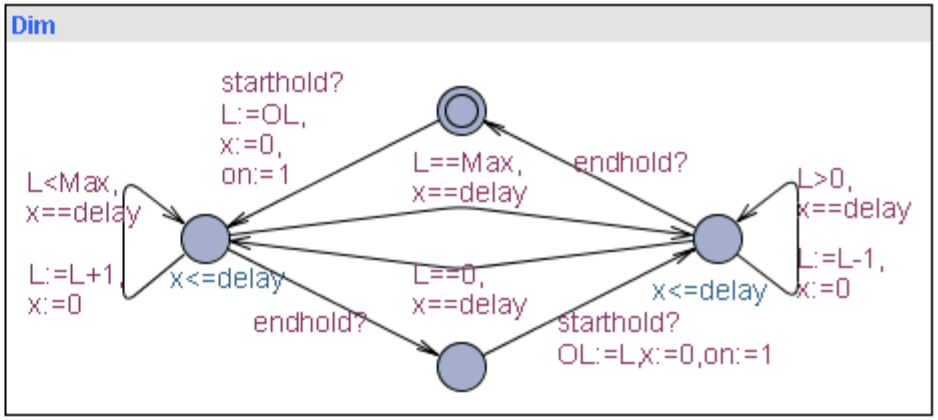


WANT: if touch is issued twice quickly
then the light will get brighter; otherwise the light is turned off.

Without Test Purpose







EXAMPLE test cases for **Interface**

- Epsilon=200ms
- Delta=500ms

0·grasp!·210·release!·touch?.PASS

0·grasp!·317·release!·touch?·2½·grasp!·220·release!·touch?·PASS

1000·grasp!·517·starthold?·100·release!·endhold?·PASS

Infinitely many sequences!!!!!!

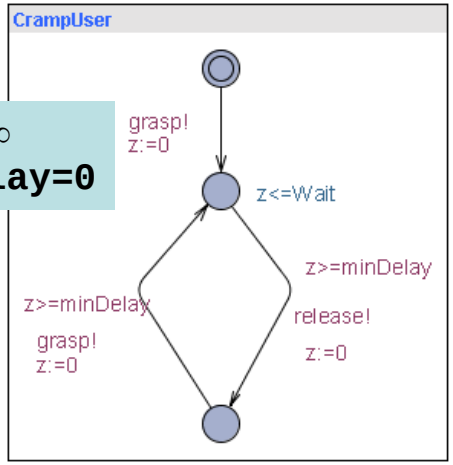
Test Purpose #1

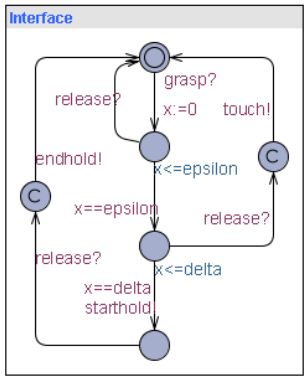
Test Purpose: *A specific test objective (or observation) the tester wants to make on SUT*

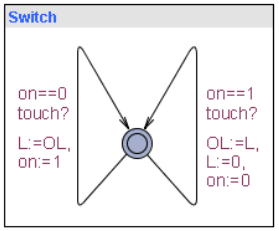
Environment model

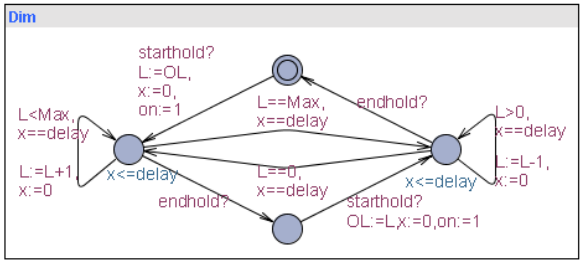
System model

Wait=∞
minDelay=0









TP1: Check that the light can become bright:

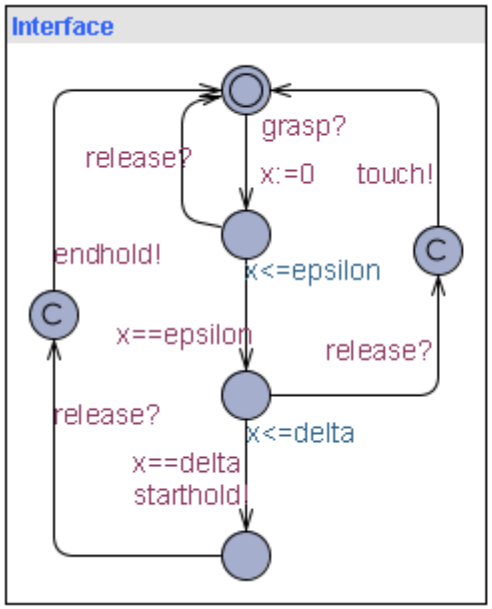
E<> L==10

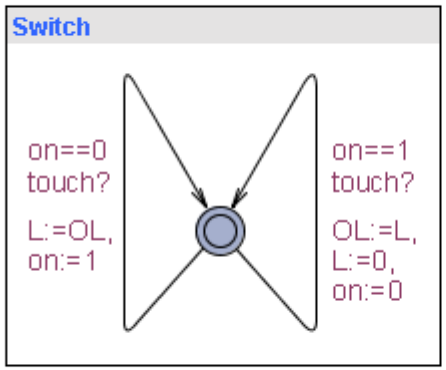
- *Shortest* (and *fastest*) Test:

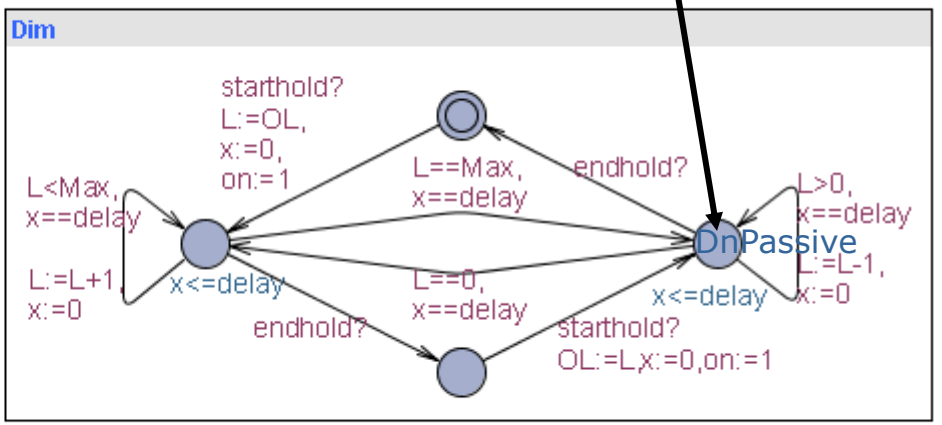
```
out(IGrasp);silence(500);in(OSetLevel,0);silence(1000);
in(OSetLevel,1);silence(1000);in(OSetLevel,2); silence(1000);
in(OSetLevel,3);silence(1000);in(OSetLevel,4);silence(1000);
in(OSetLevel,5);silence(1000);in(OSetLevel,6);silence(1000);
in(OSetLevel,7);silence(1000);in(OSetLevel,8);silence(1000);
in(OSetLevel,9);silence(1000);in(OSetLevel,10);
out(IRelease);
```

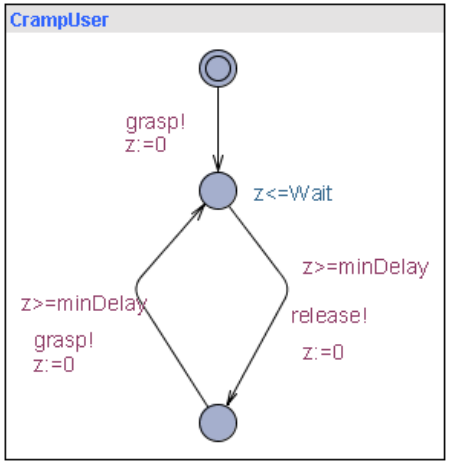
Test Purpose #2

TP2: Check that controller can enter location 'DnPassive':
E<> Dim.DnPassive







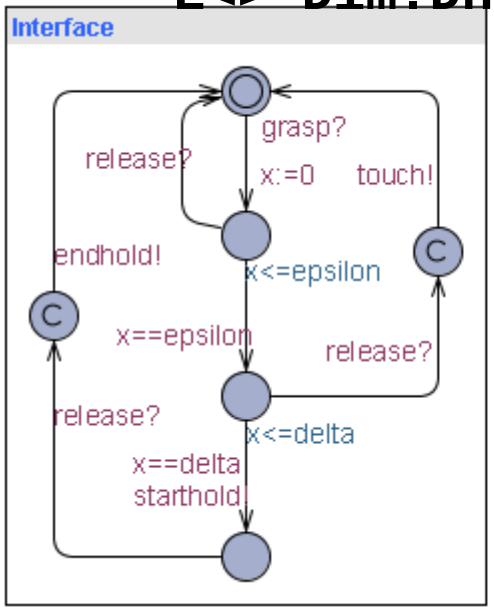


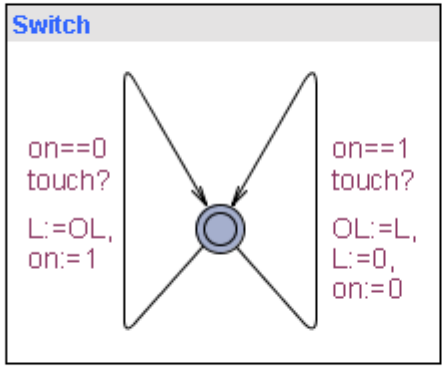
- If delay=1000
- *Shortest* (and *fastest*) Test:

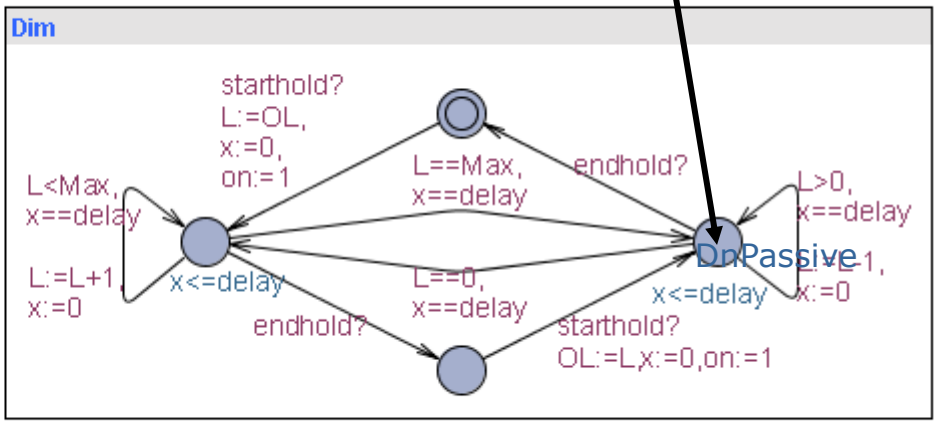
```
out(IGrasp);
silence(500);
in(OSetLevel,0);
out(IRelease);
out(IGrasp);
silence(500);
```

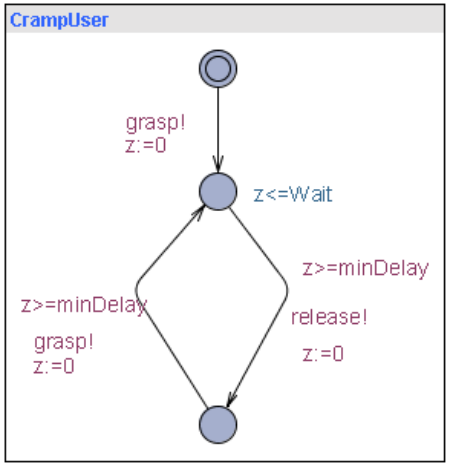
Test Purpose #2

TP2: Check that controller can enter location 'DnPassive':
E<> Dim.DnPassive









- If delay=40?
- *Shortest* Test:

```
out(IGrasp);
silence(500);
in(OSetLevel,0);
out(IRelease);
out(IGrasp);
silence(500);
```

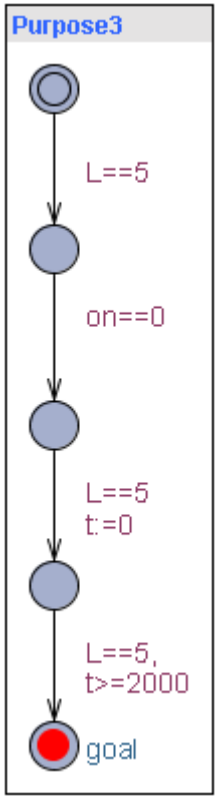
- *Fastest* Test:

```
out(IGrasp);silence(500);in(OSetLevel,0);silence(40);
in(OSetLevel,1);silence(40);in(OSetLevel,2); silence(40);
in(OSetLevel,3);silence(40);in(OSetLevel,4); silence(40);
in(OSetLevel,5);silence(40);in(OSetLevel,6); silence(40);
in(OSetLevel,7);silence(40);in(OSetLevel,8); silence(40);
in(OSetLevel,9);silence(40);in(OSetLevel,10);silence(40);
```

Test Purpose #3

TP3: Check that controller resets light level to previous value after switch-on.

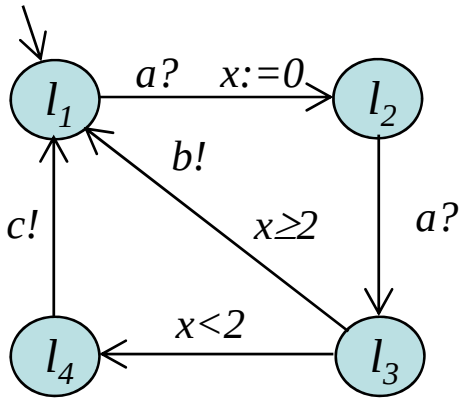
E<> Purpose3.goal




```
out(IGrasp);   //set level to 5
silence(500);
in(OSetLevel,0);
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);
out(IRelease);

out(IGrasp);   //touch To Off
silence(200);
out(IRelease);
in(OSetLevel,0);

out(IGrasp);   //touch To On
silence(200);
out(IRelease);
in(OSetLevel,5);

silence(2000);
```

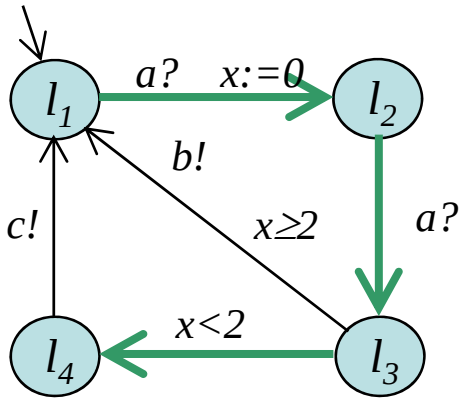
Coverage-Based Test Generation

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



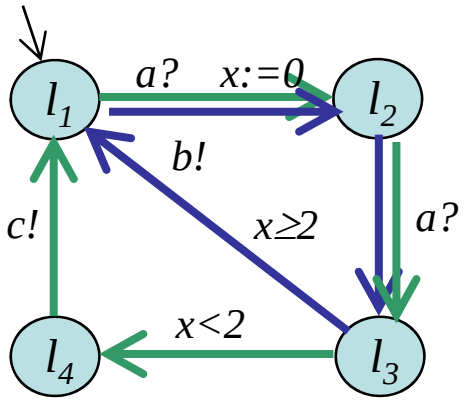
Location Coverage

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



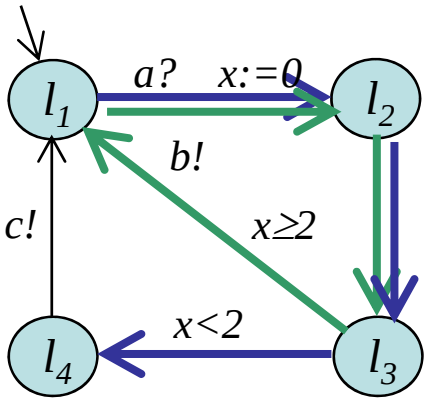
Edge Coverage

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



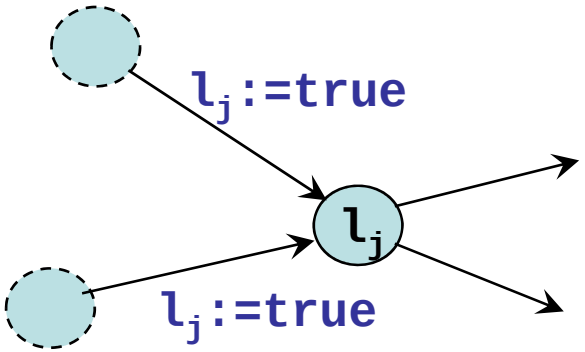
Definition/Use Pair Coverage

- Multi purpose testing
- Cover measurement
- Examples:
  - Location Coverage,
  - Edge Coverage,
  - **Definition/Use Pair Coverage**



Implementing Location Coverage

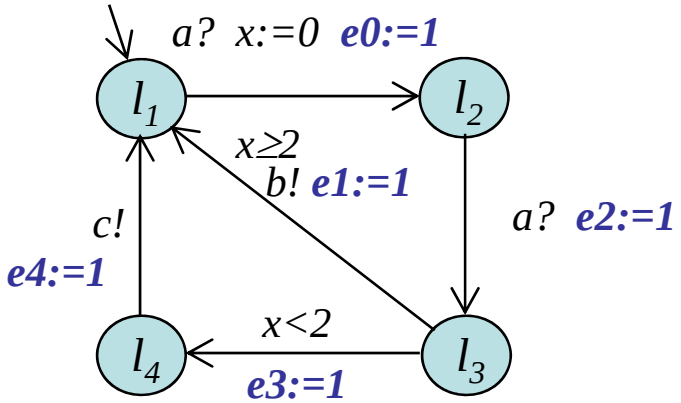
- Test sequence traversing all locations
- Encoding:
  - Enumerate locations $l_0, \ldots, l_n$
  - Add an auxiliary variable l_i for each location
  - Label each ingoing edge to location i with l_i:=true
  - Mark initial visited l_0:=true
- Check: E<>(l_0=true $\wedge$ … $\wedge$ l_n=true)



UPPAAL COVER

Implementing Edge Coverage

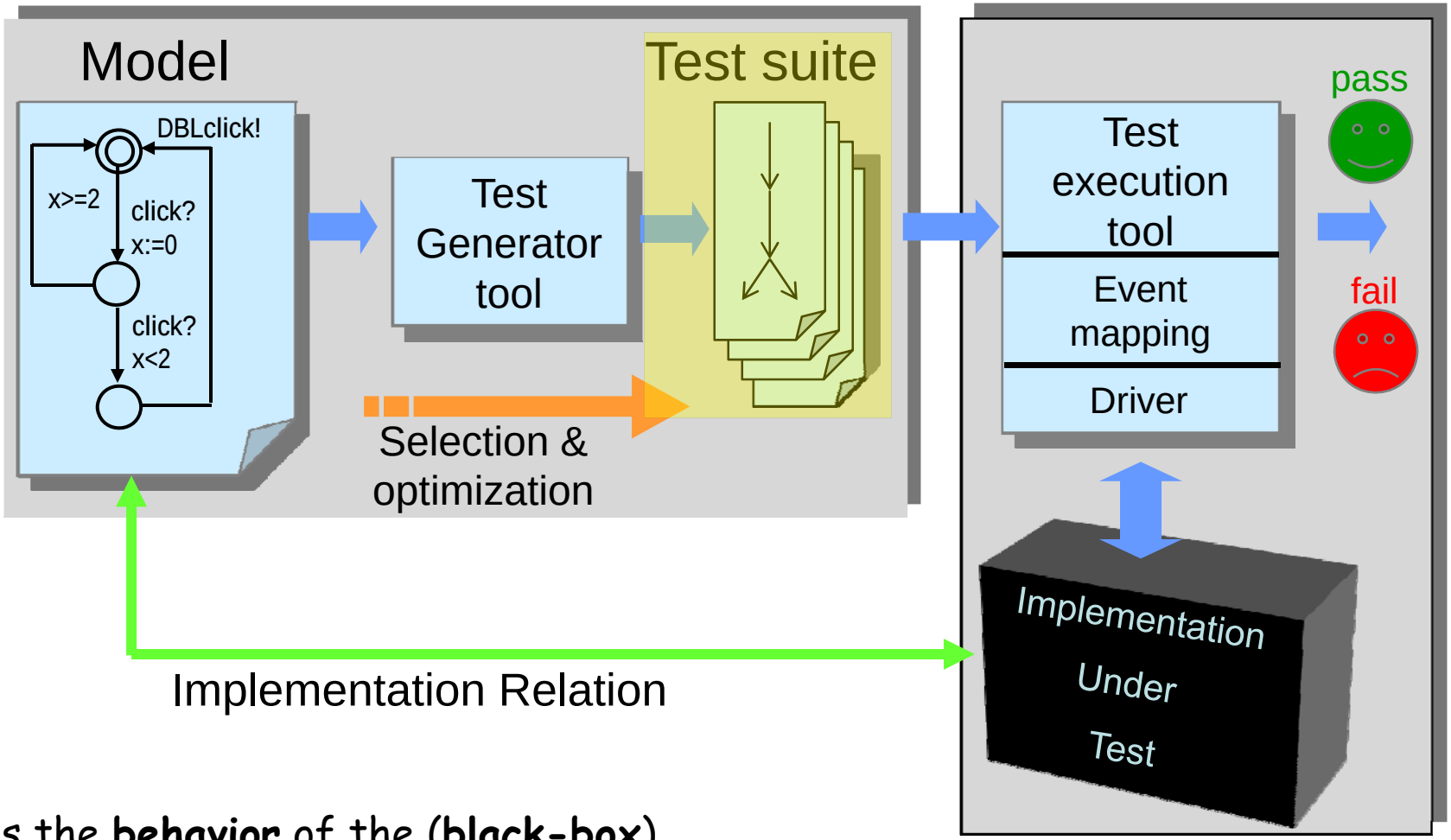
- Test sequence traversing all edges
- Encoding:
  - Enumerate edges $e_0, \ldots, e_n$
  - Add auxiliary variable e_i for each edge
  - Label each edge e_i:=true
- Check: E<>(e_0=true $\wedge$ … $\wedge$ e_n=true)



Model-Based On-line Testing of Timed Systems

Automated Model-Based Off-line Conformance testing

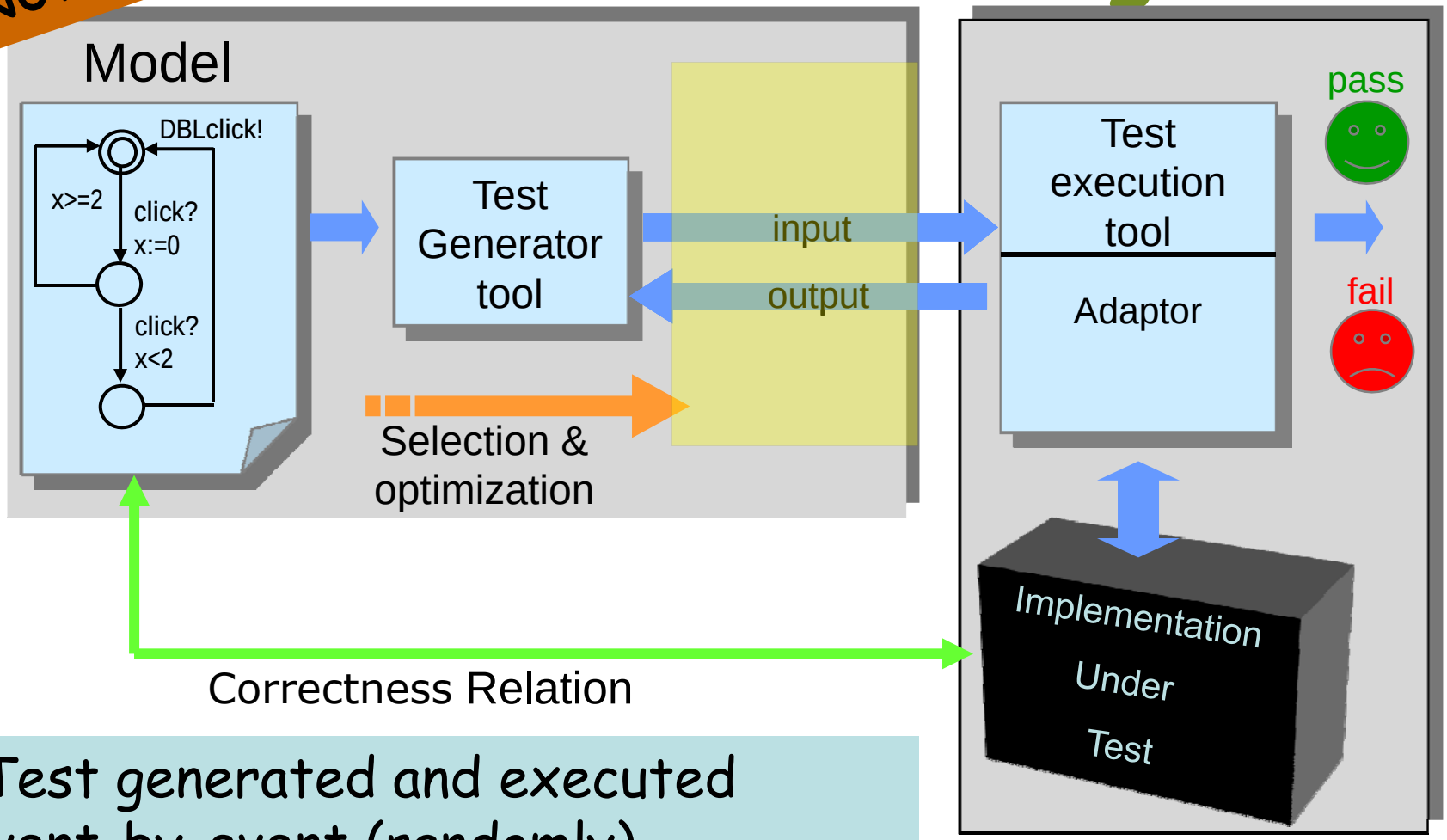
Recall...



Does the **behavior** of the (**black-box**) implementation ***comply*** to that of the specification?

Automated Model-Based On-line Conformance testing

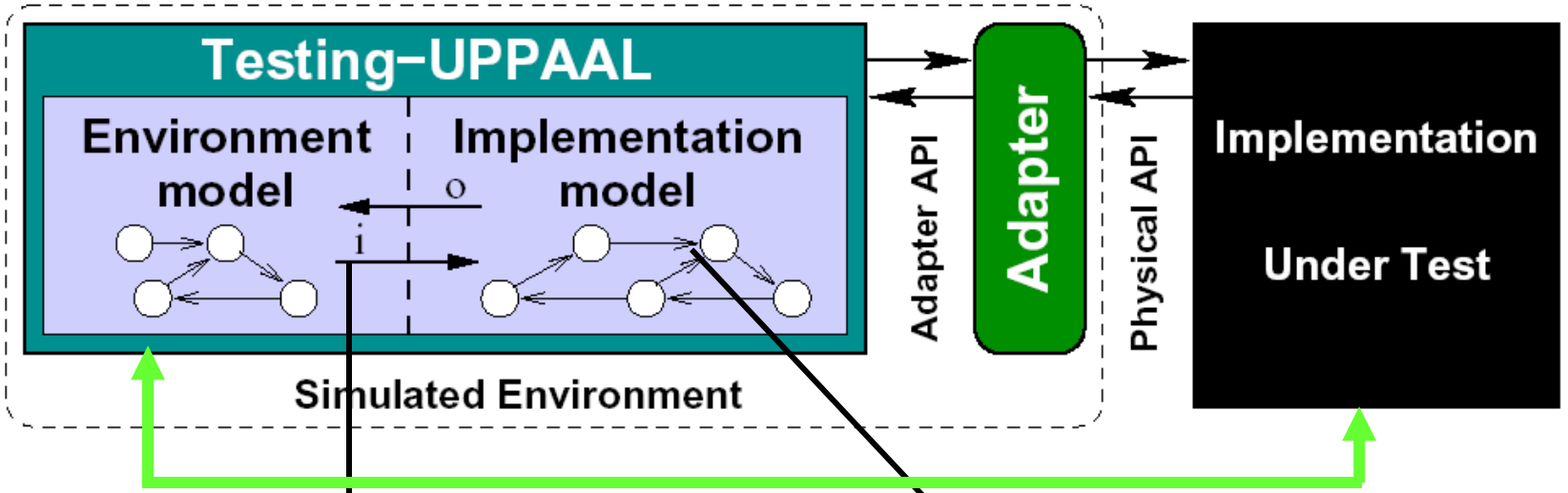
Now…



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

The Framework of Uppaal-TRON

- ***UppAal Timed Automata*** *Network: Env || IUT*



"Relativized Timed i/o Conformance" Relation (rt-ioco)

- Relevant input event sequences
- Load model

Correct system behavior

- Test Oracle
- Monitor

- *Complete and sound algorithm*
- Efficient symbolic reachability algorithms
- **Uppaal-TRON:** Testing Real-time Systems ONline
- Release 1.4 http://www.cs.aau.dk/~marius/tron/

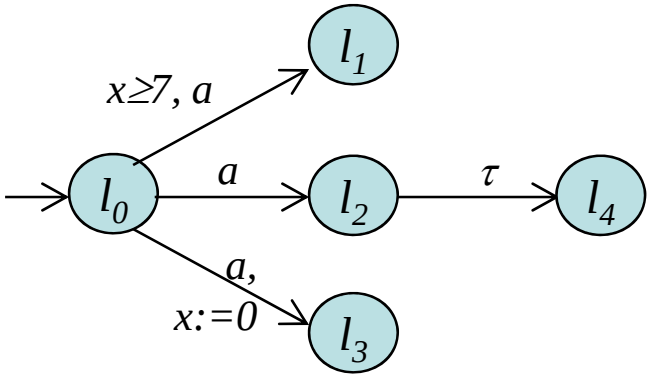
On-line Testing

- Characteristica
  - very imaginative, "ingenious" tests sequences
  - long test sequences
  - stressful load
  - effective fault detection
- Tools exists but mostly NON-real-time
  - **So-far systematic and explicit handling of real-time constraints missing**

State-set Computation

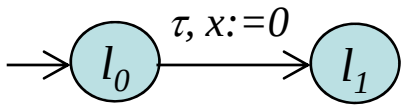
- Compute all potential states the model can occupy after the timed trace $\varepsilon_0, i_0, \varepsilon_1, o_1, \varepsilon_2, i_2, o_2, \ldots$
- *Let Z be a set of states*

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



$\{ \langle l_0, x=3 \rangle \}$ **after** a =
$\{ \langle l_2, x=3 \rangle, \langle l_4, x=3 \rangle, \langle l_3, x=0 \rangle \}$

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



$\{ \langle l_0, x=0 \rangle \}$ **after** 4 =
$\{ \langle l_0, x=4 \rangle, \langle l_1, 0 \leq x \leq 4 \rangle \}$

$\langle l_0, x=0 \rangle \xrightarrow{1} \langle l_0, x=1 \rangle \xrightarrow{\tau} \langle l_1, x=0 \rangle \xrightarrow{3} \langle l_1, x=3 \rangle$

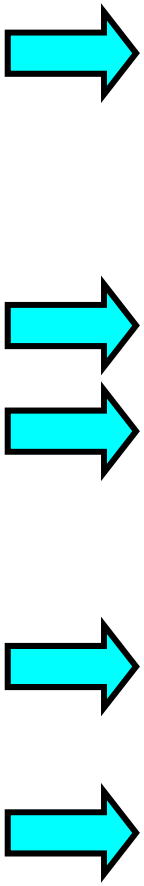
Algorithm Idea:
State-set tracking

- Dynamically compute all potential states that the model M can reach after the timed trace
  [Tripakis] Failure Diagnosis
  $\varepsilon_0, i_0, \varepsilon_1, o_1, \varepsilon_2, i_2, o_2, \ldots$

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

- *If* $Z = \emptyset$ *then IUT has made a computation not in model:* ***FAIL***
- *i is a relevant input in Env iff* $i \in$ *EnvOutput(Z)*

Uppaal-TRON On-line Testing Algorithm (skeleton)

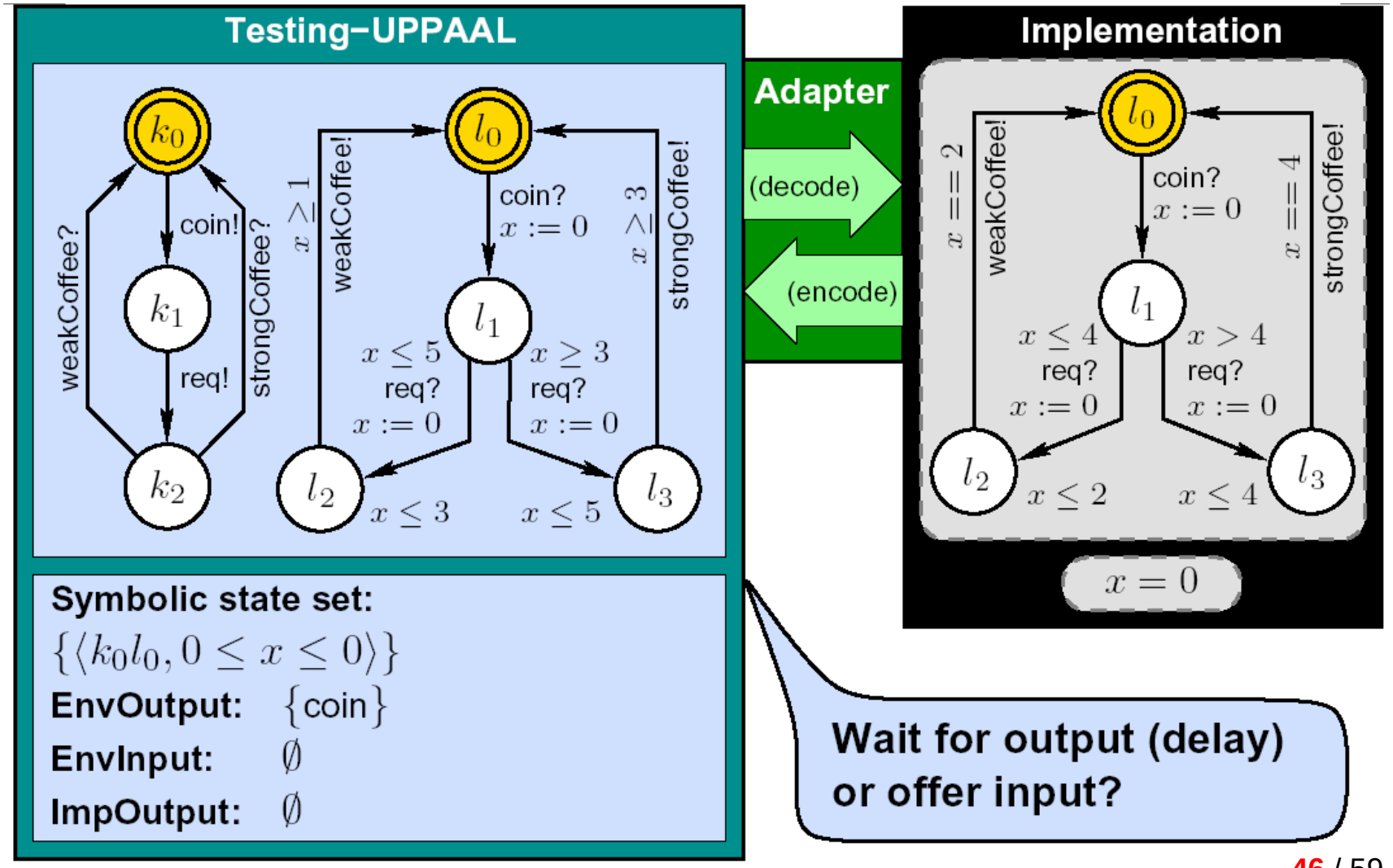


Algorithm *TestGenExe* (*S*, *E*, *IUT*, *T*) **returns** {**pass**, **fail**}
Z := {(s0, e0)}.
while Z ≠ ∅ ∧ ♯iterations ≤ T **do either** randomly:

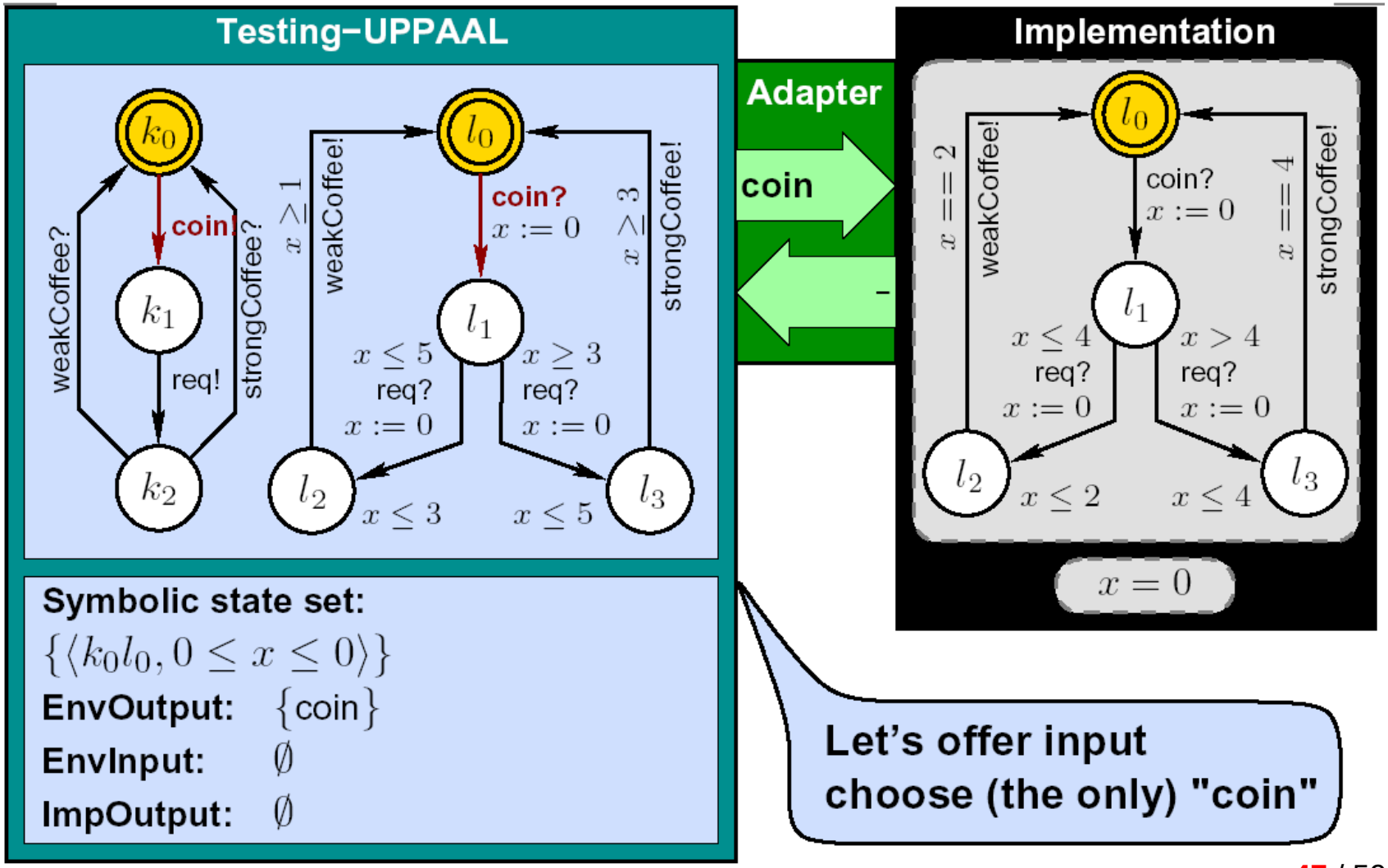
1. // offer an input
   if *EnvOutput(Z)* ≠ ∅
   randomly choose *i* ∈ *EnvOutput(Z)*
   send i to IUT
   Z := Z *After* i
2. // wait d for an output
   randomly choose d ∈*Delays(Z)*
   wait (for d time units or output o at d′ ≤ d)
   if o occurred **then**
   Z := Z *After* d′
   Z := Z *After* o // 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**

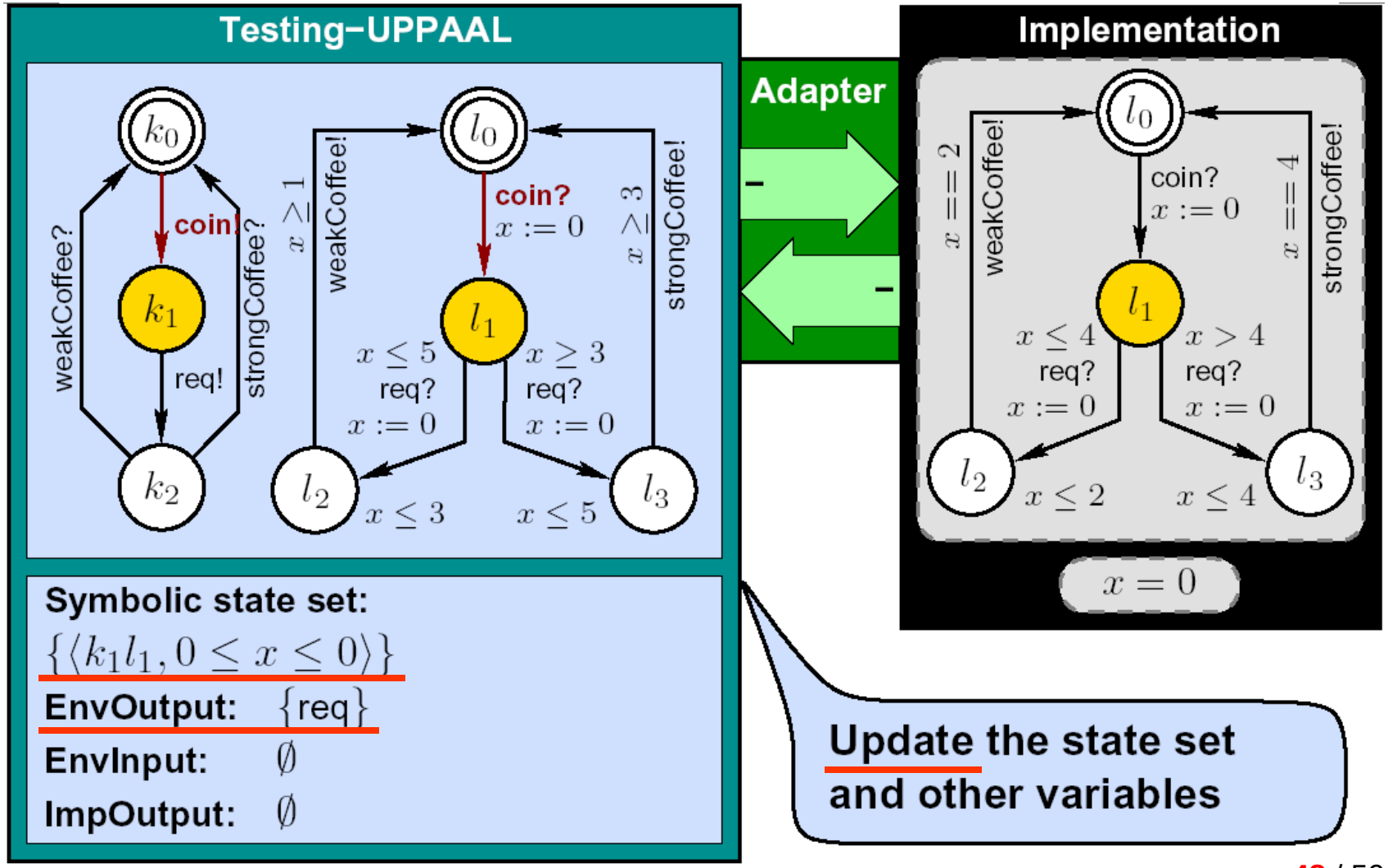
On-line Testing Example (1)



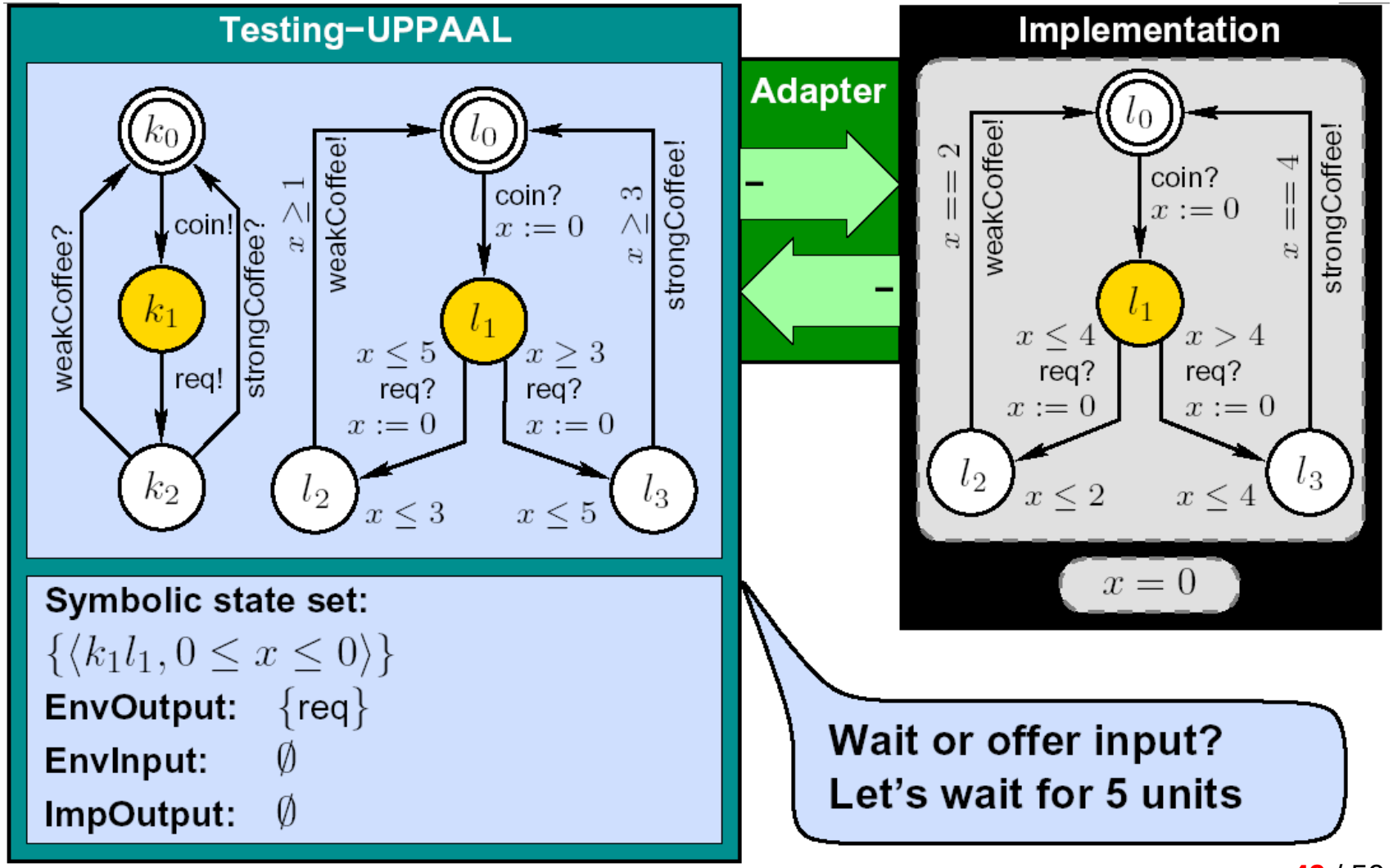
On-line Testing Example (2)



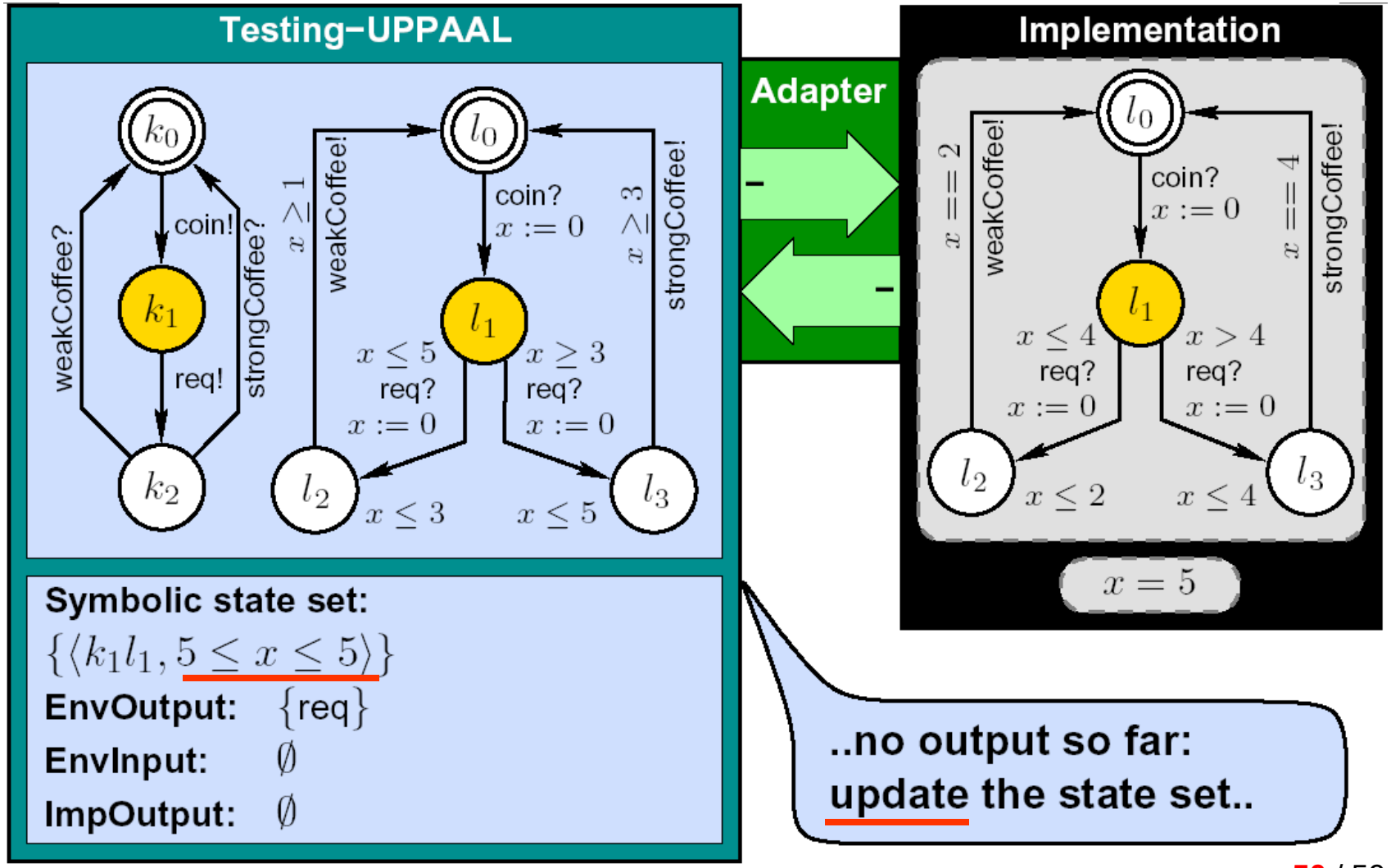
On-line Testing Example (3)



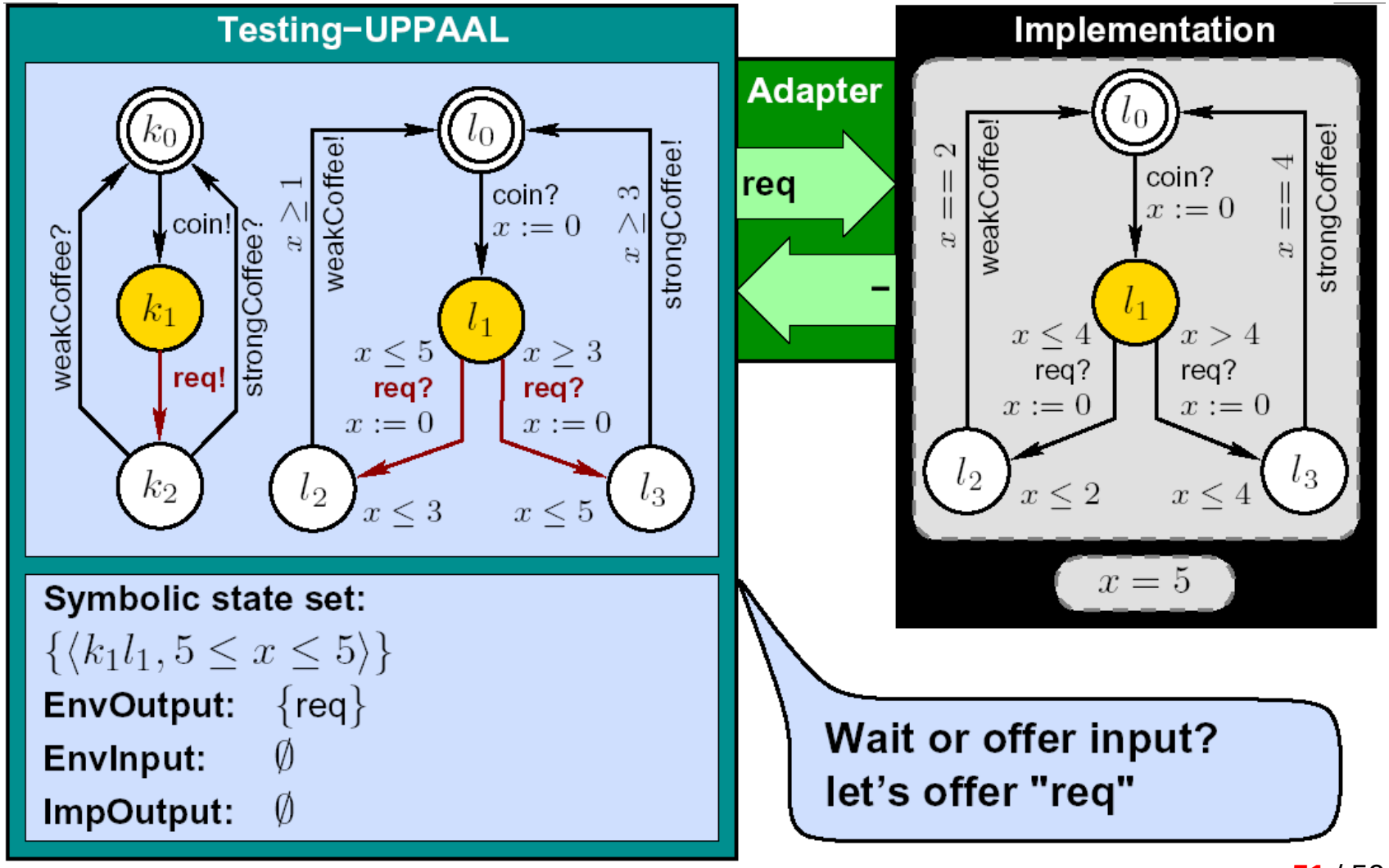
On-line Testing Example (4)



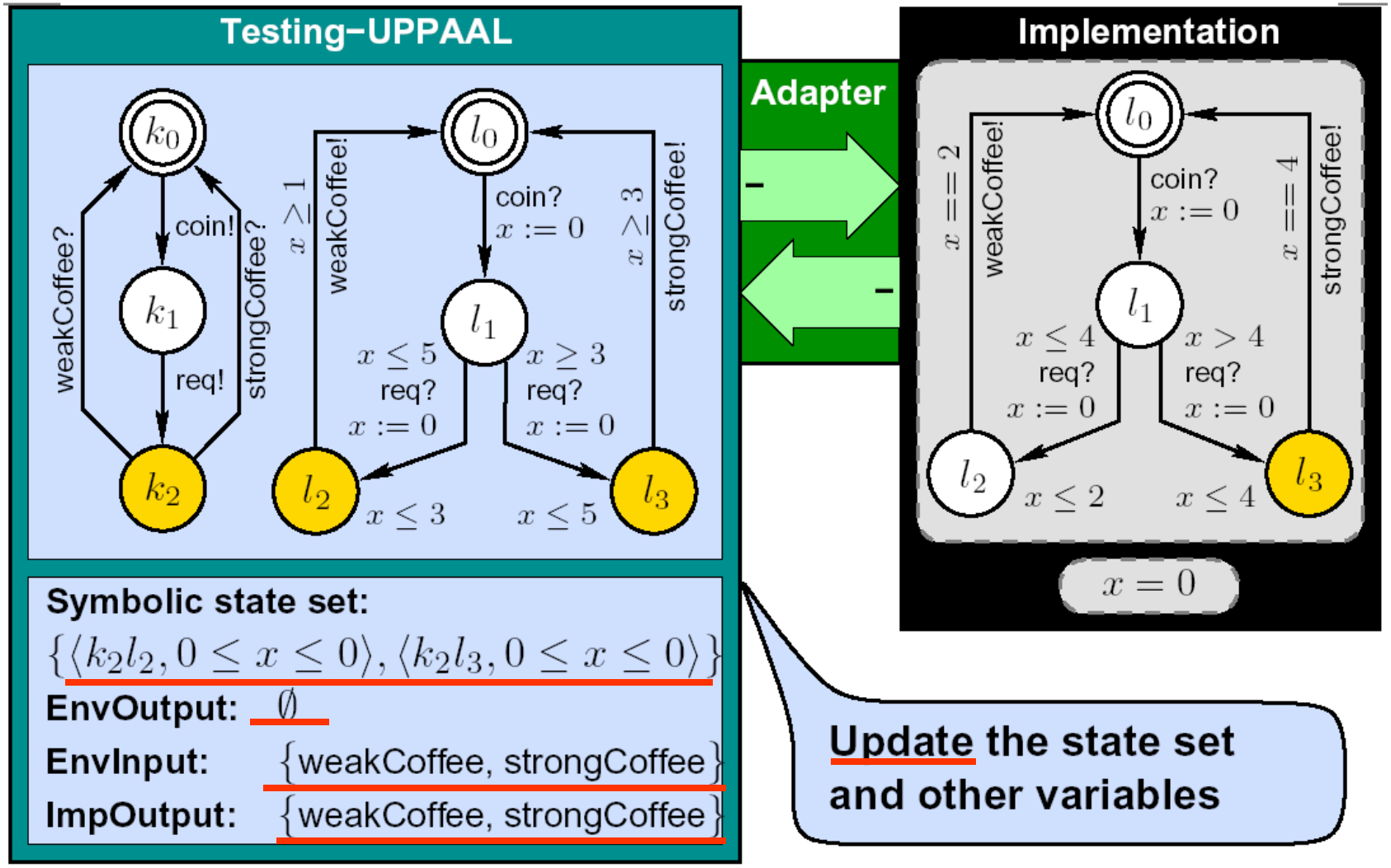
On-line Testing Example (5)



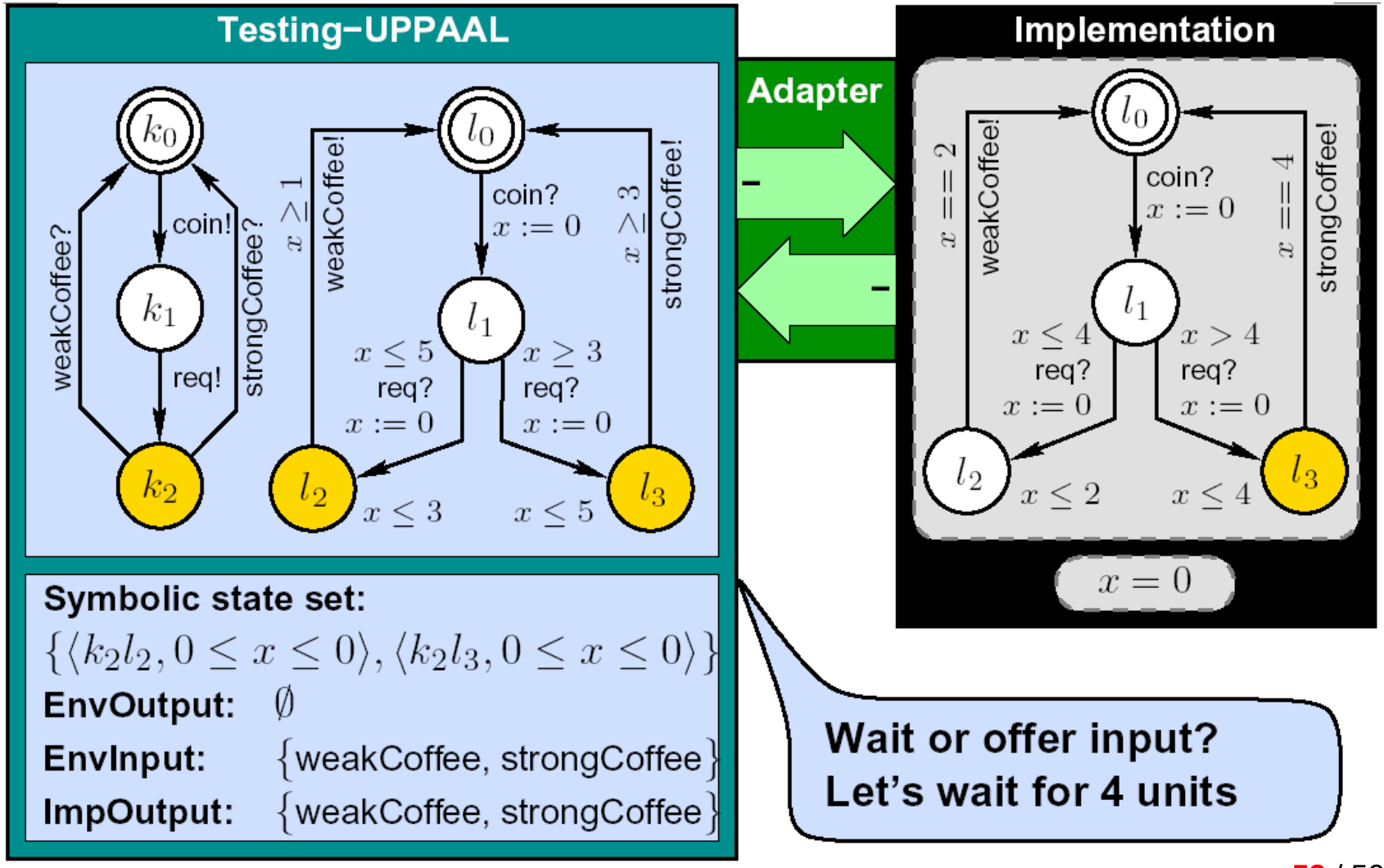
On-line Testing Example (6)



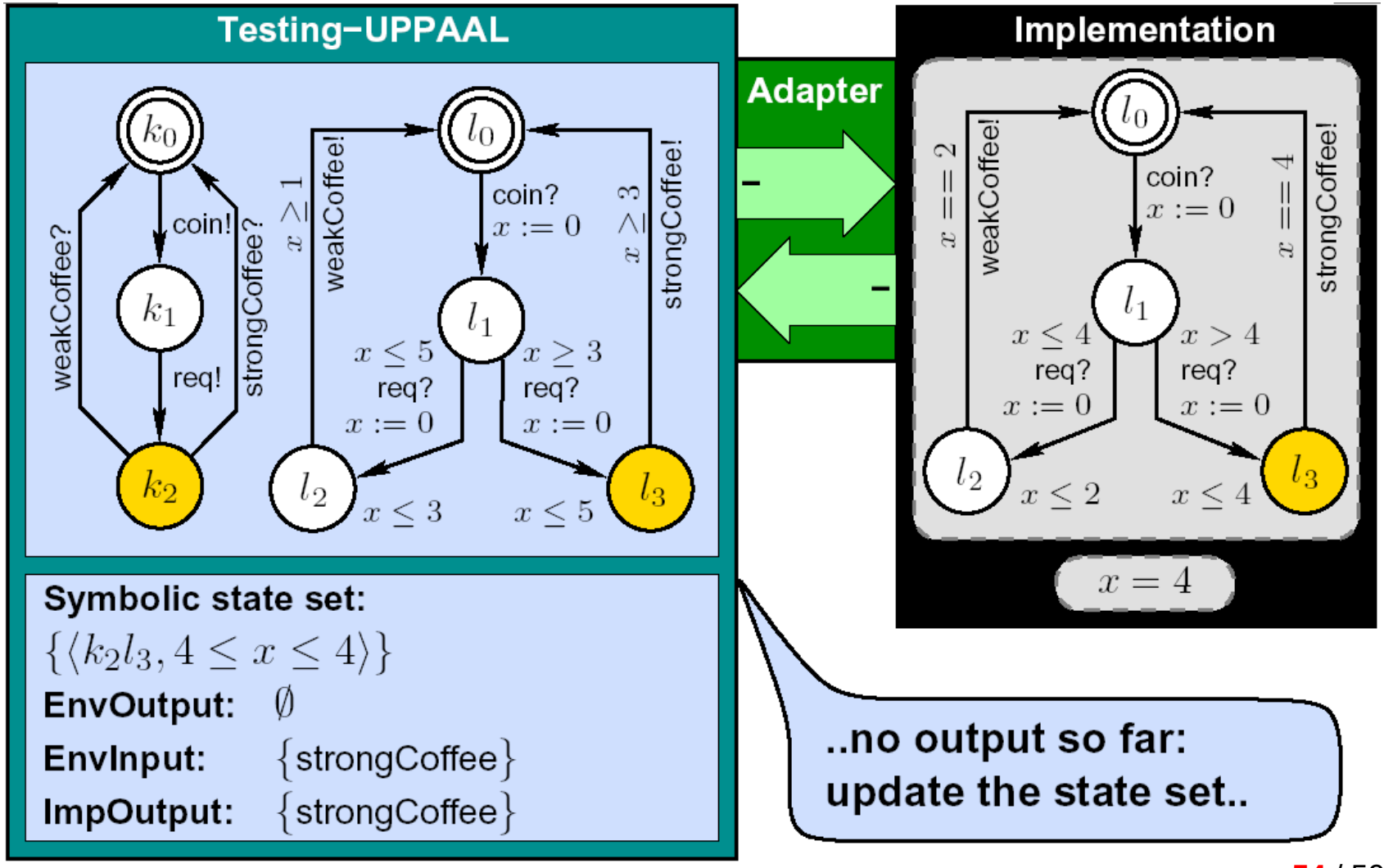
On-line Testing Example (7)



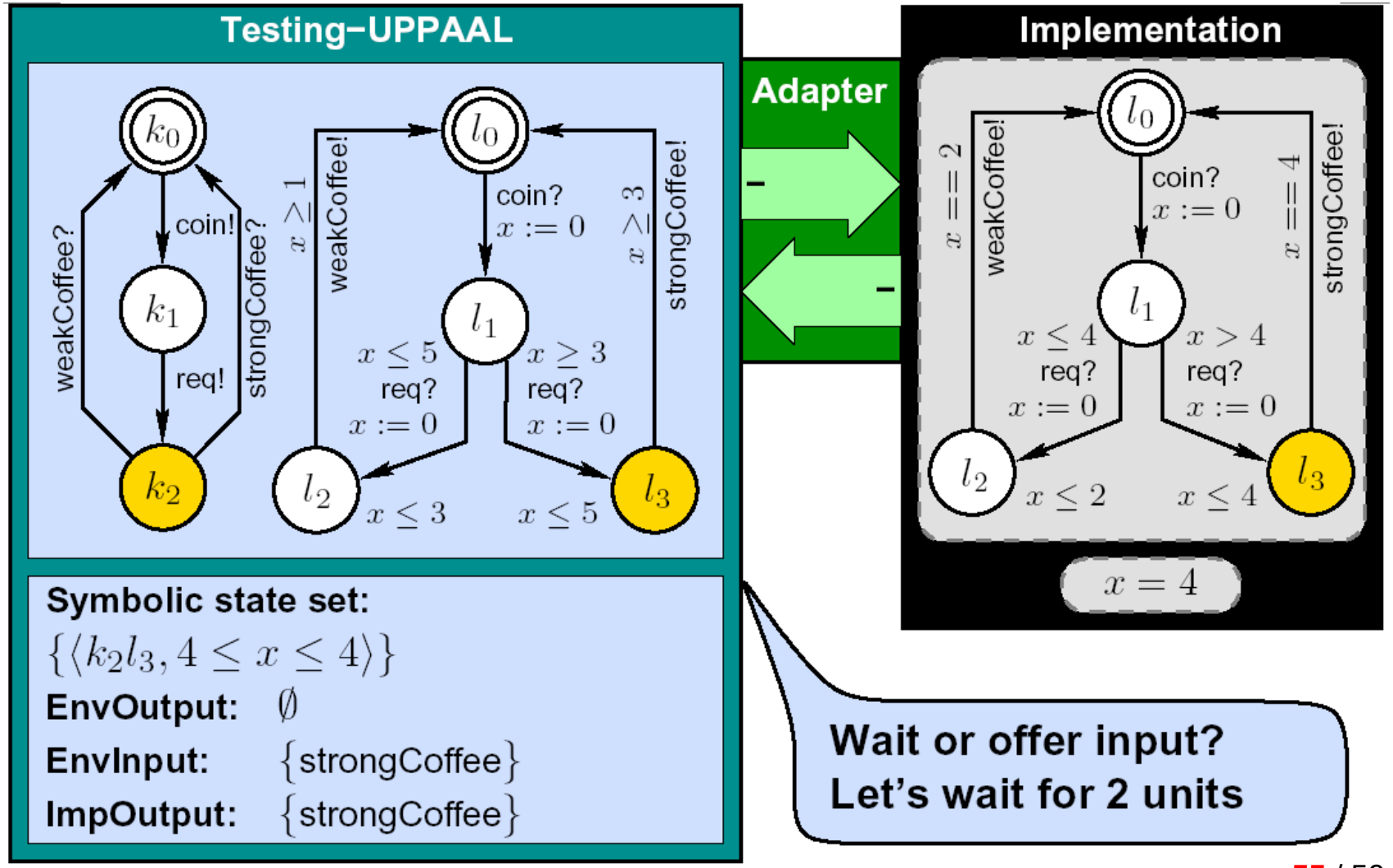
On-line Testing Example (8)



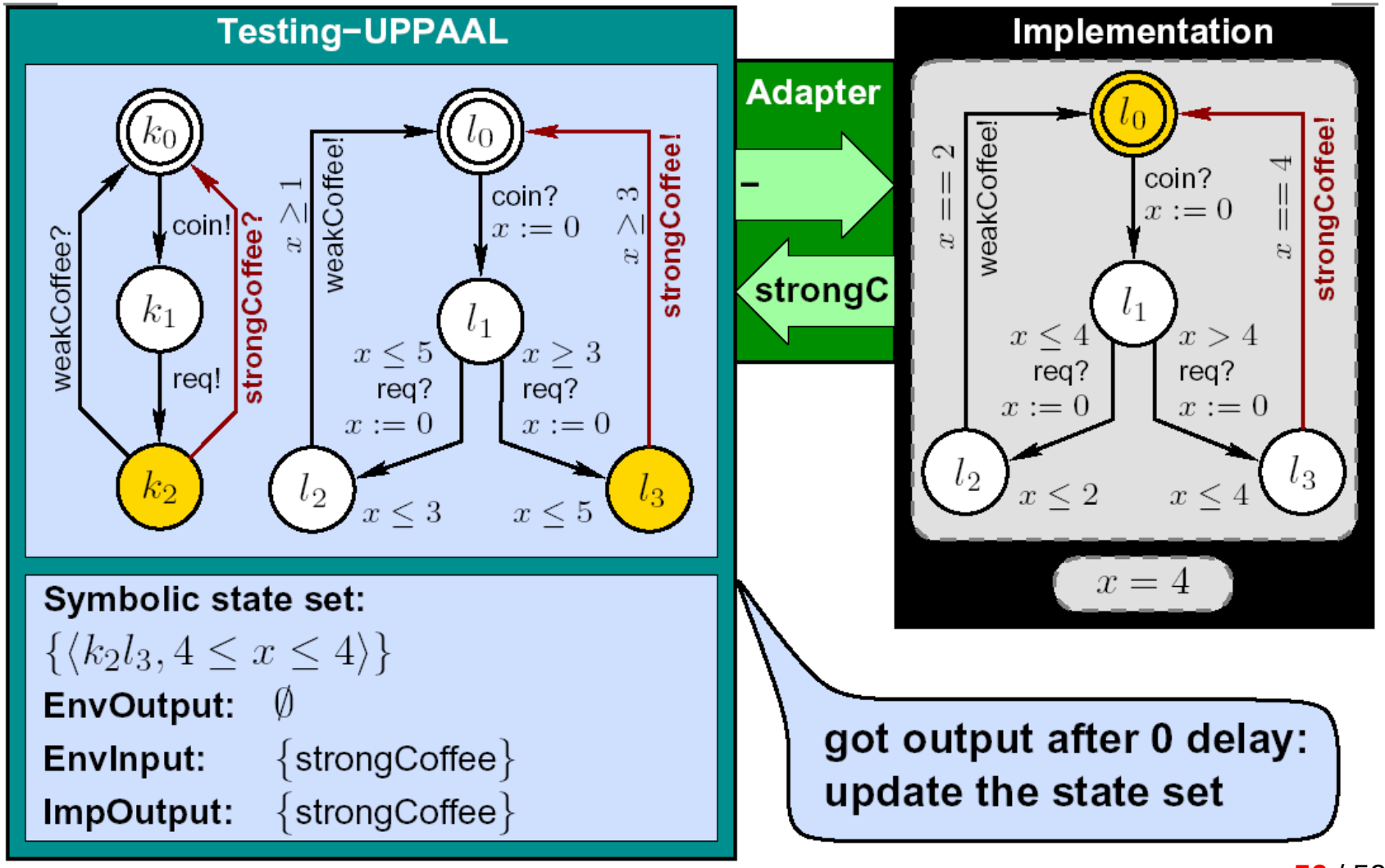
On-line Testing Example (9)



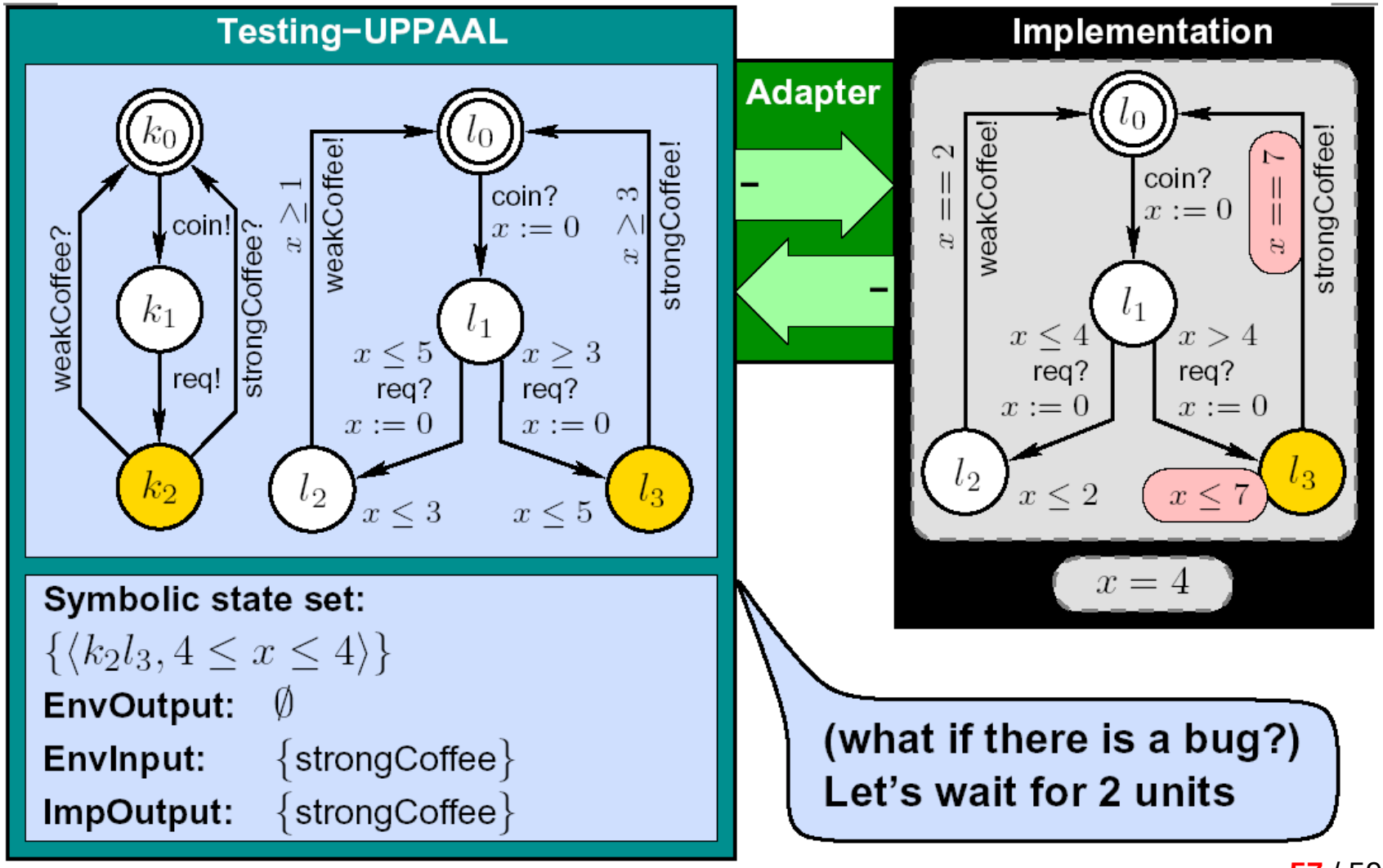
On-line Testing Example (10)



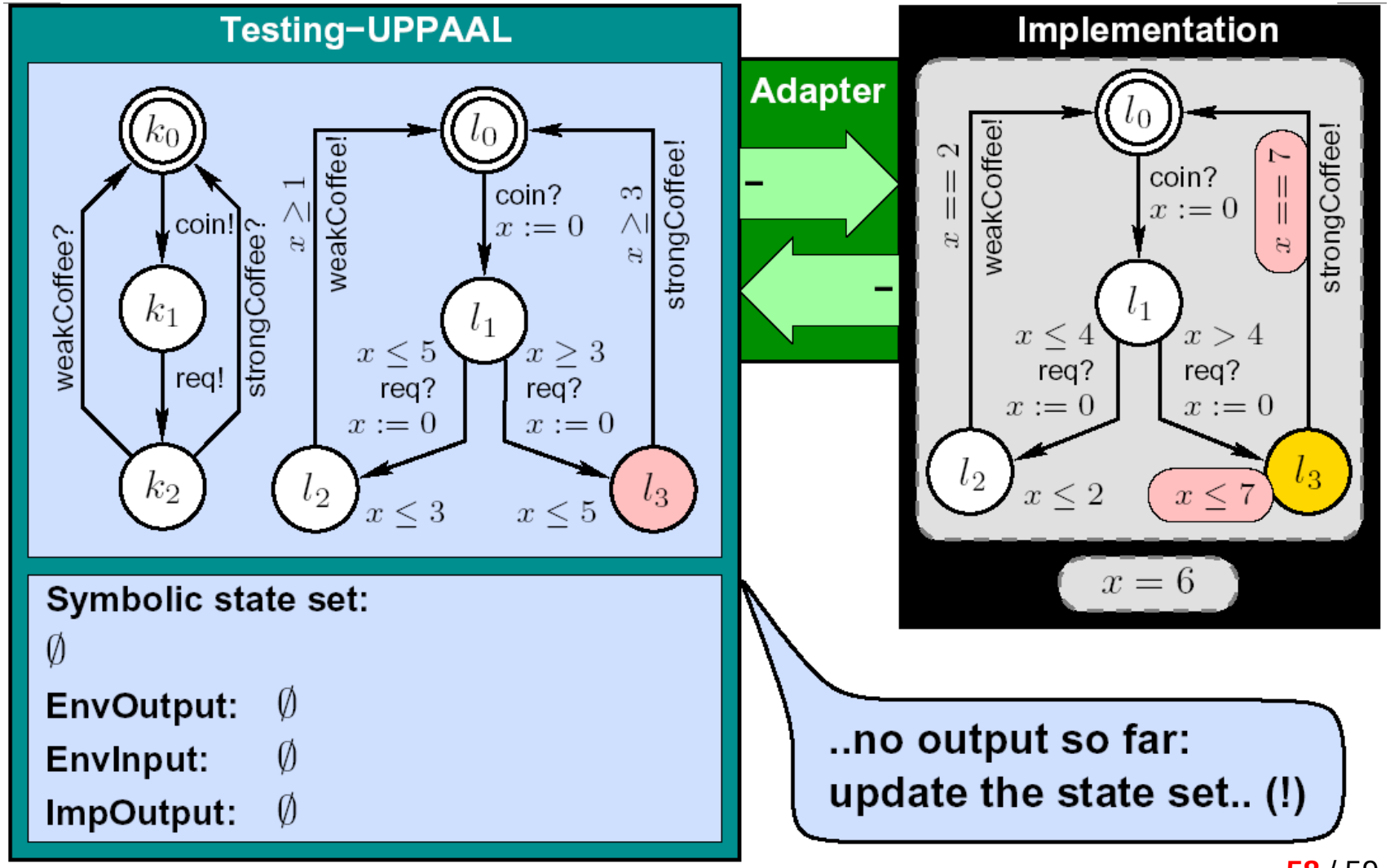
On-line Testing Example (11)



On-line Testing Example (12)



On-line Testing Example (13)



Further information

- Main Readings
  - Anders Hessel, Kim Guldstrand Larsen, Marius Mikucionis, Brian Nielsen, Paul Pettersson, and Arne Skou. "**Automated Model-Based Conformance Testing of Real-Time Systems**". In: Formal Methods and Testing, An Outcome of the FORTEST Network, Revised Selected Papers. Lecture Notes in Computer Science 4949 Springer 2008, pp.1-38.

- Industrial case studies
  - Anders Hessel, Paul Pettersson. "**Model-Based Testing of a WAP Gateway: An Industrial Case-Study**". FMICS/PDMC 2006: 116-131 (Uppaal-Cover)
  - Kim Guldstrand Larsen, Marius Mikucionis, Brian Nielsen, Arne Skou. "**Testing real-time embedded software using UPPAAL-TRON: an industrial case study**". EMSOFT 2005: 299-306 (Uppaal-TRON)