Embedded Systems
INSTITUTE

Quantitative Testing Theory

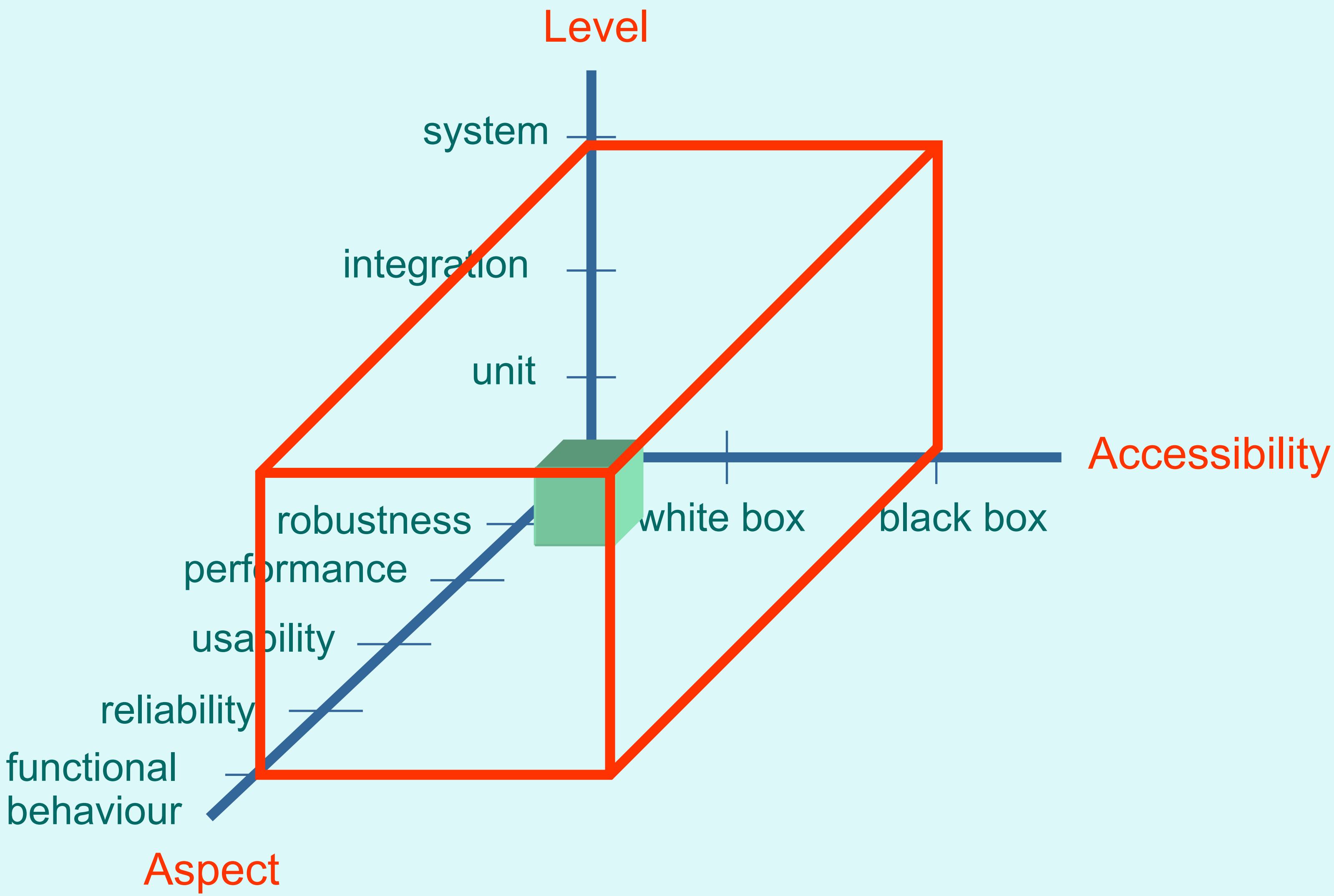
ARTIST2 Summer School
Autrans, September 8th, 2008

Ed Brinksma
Embedded Systems Institute
Eindhoven, NL

Contents

Contents

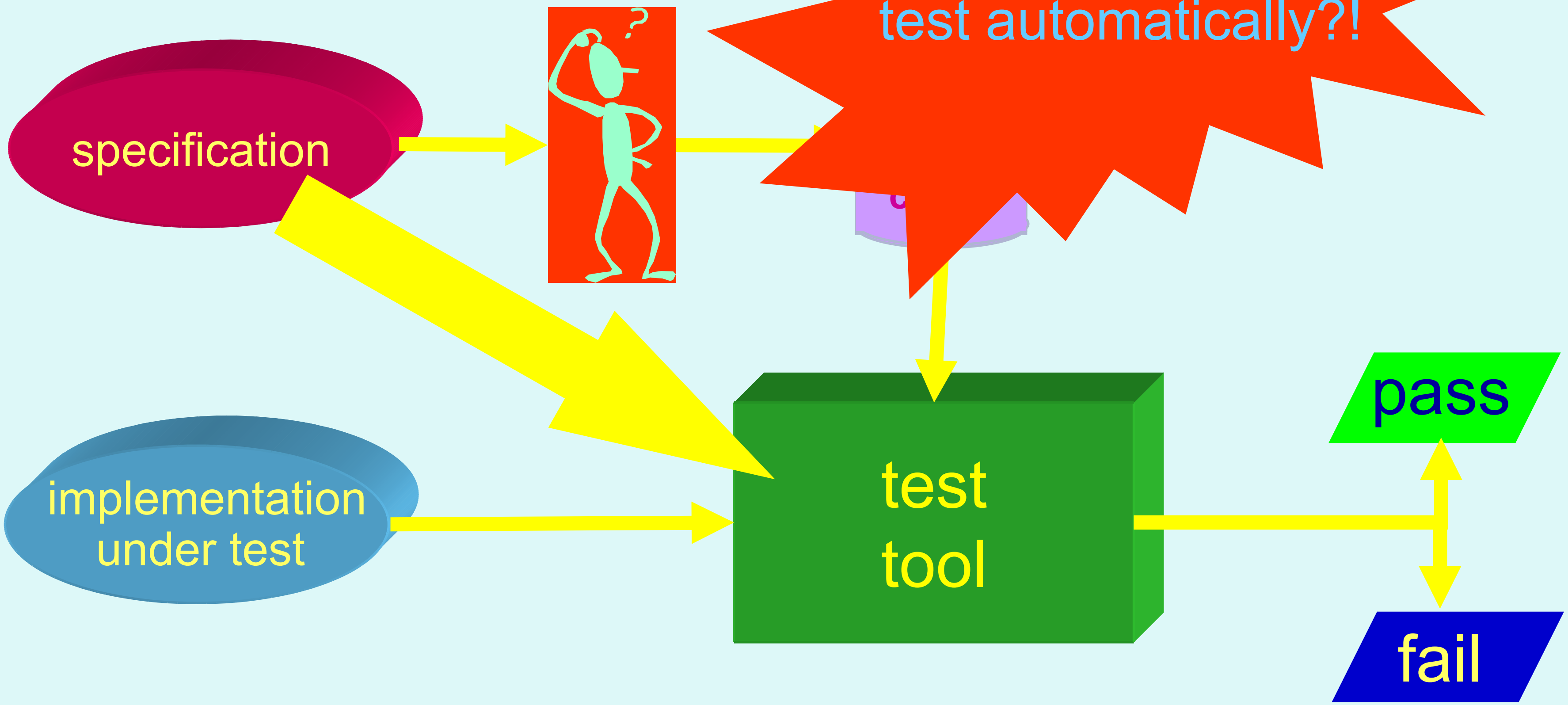
Types of Testing



Test automation

Traditional test automation
= tools to execute an

Why not generate
test automatically?!



Our Context

Formal methods:

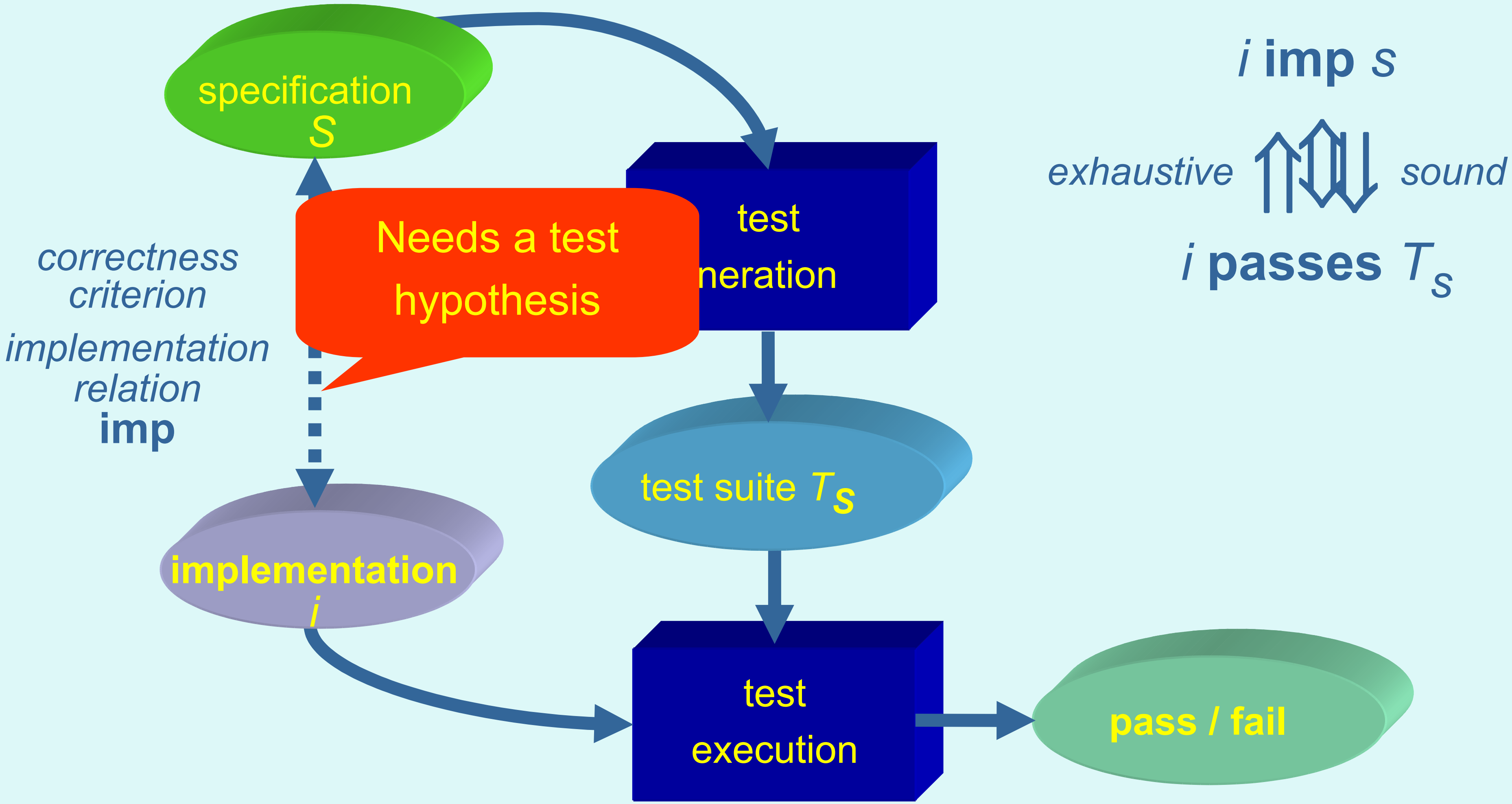
- unambiguous specification ("model-driven")
- precise notion of correctness
- formal validation of tests
- algorithmic generation of tests

Dynamic behaviour:

- concentrate on control behaviour
- concurrency and non-determinism

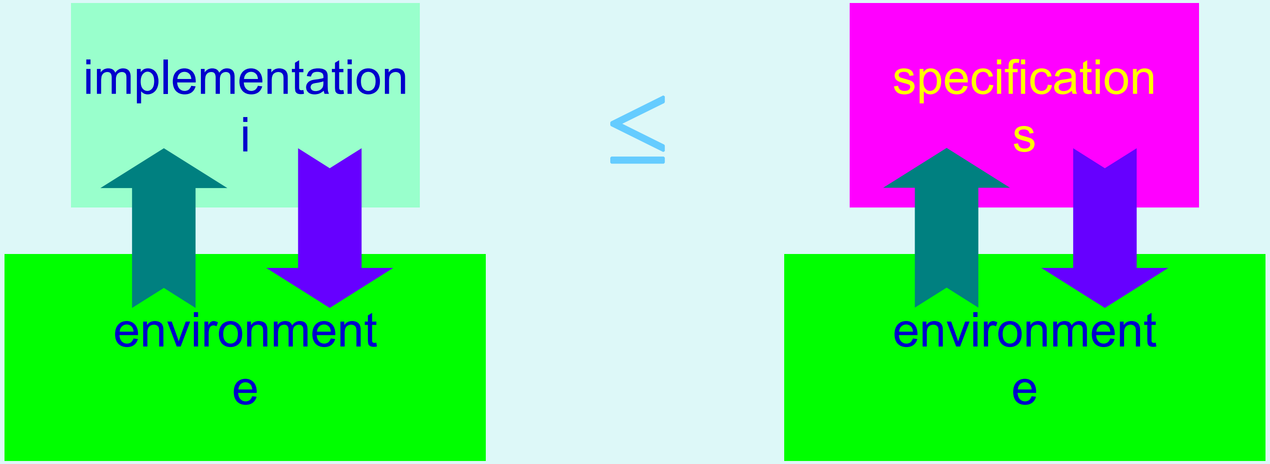
Embedded Systems INSTITUTE

Formal Testing



Testing preorders

Embedded Systems INSTITUTE

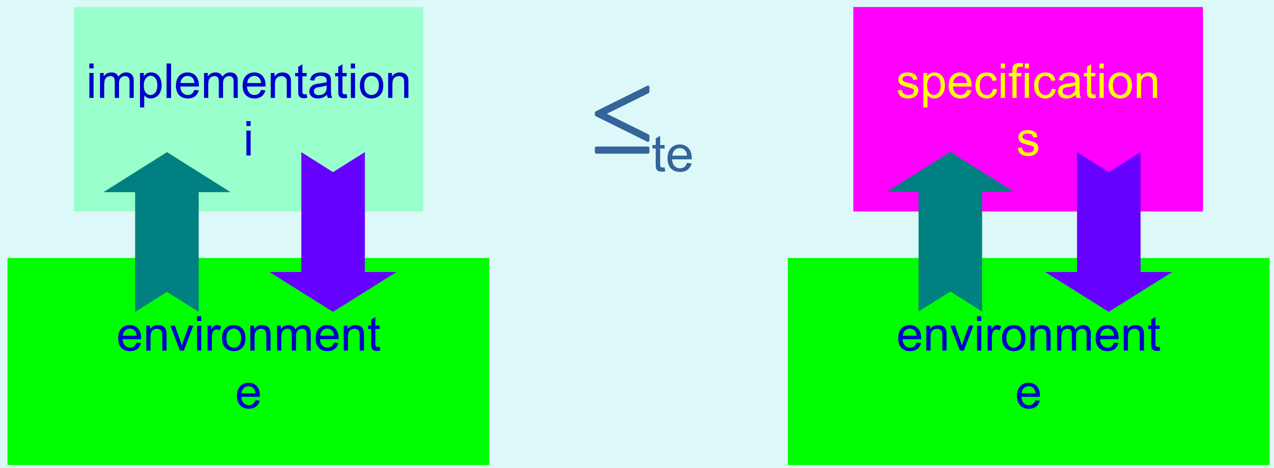


For all environments e
all observations of an implementation i in e
should be explained by
observations of the specification s in e.

$i \leq s \iff \forall\, e \in Env\,.\ obs(e, i) \subseteq obs(e, s)$

? ? ?

Classical Testing Preorder



Philosophical question:

can we observe deadlocks?

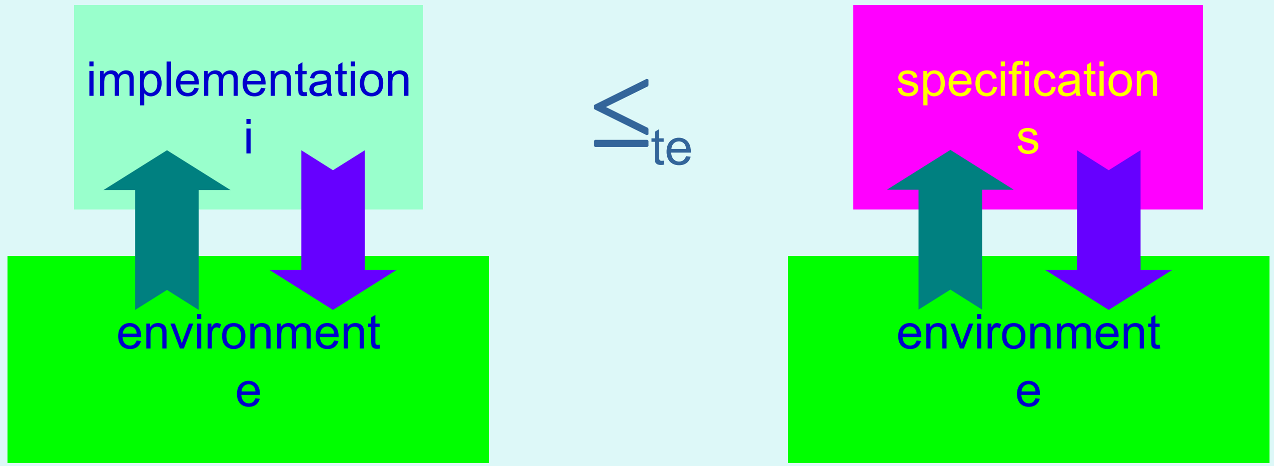
bs (e, i) $\subseteq$ obs (e, s)

↓ ↓

LTS(L) Deadlocks(e||s)

Embedded Systems INSTITUTE

Classical Testing Preorder



$i \leq_{te} s \iff \forall e \in LTS(L) \,.\, \forall \sigma \in L^* \,.$

$e \| i$ **deadlocks after** $\sigma \Rightarrow e \| s$ **deadlocks after** σ

$\{ \sigma \mid i \| e$ **deadlocks after** $\sigma \} \subseteq \{ \sigma \mid s \| e$ **deadlocks after** $\sigma \}$

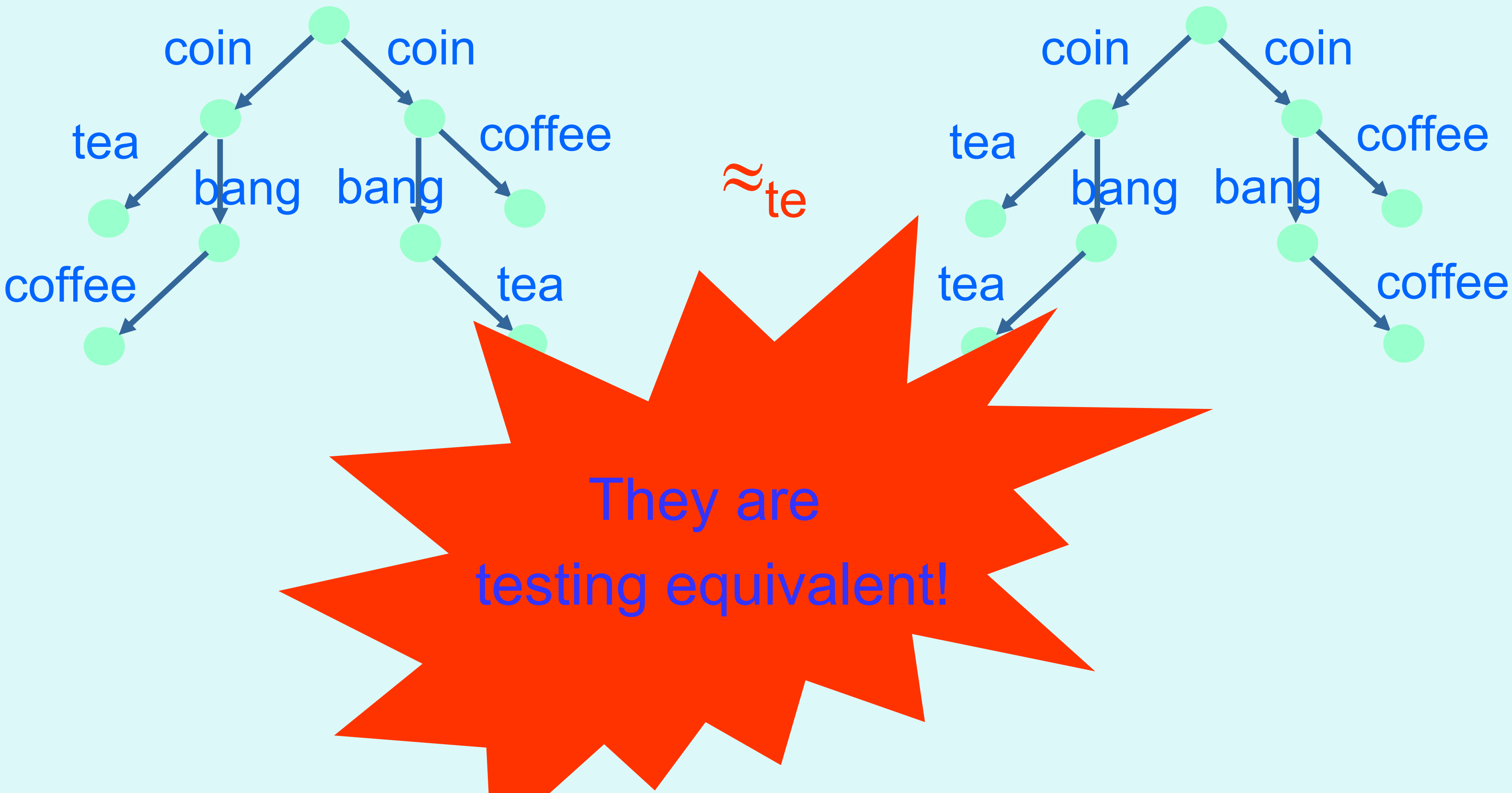
$\iff$ **FP**$(i) \subseteq$ **FP**(s)

FP(p) **=** $\{ \langle \sigma, A \rangle \mid p$ **after** σ **refuses** $A \}$

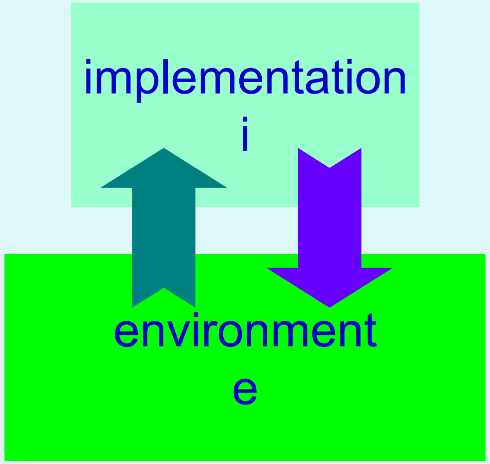
Embedded Systems INSTITUTE

Quirky Coffee Machine [Langerak]

Can we distinguish between these machines?



Refusal Preorder



$\leq_{rf}$

specification

Deadlocks$_{\delta}$(e||i) =
{σ∈(L∪{δ})* |
e||i **deadlocks after** σ}

$i \leq_{rf} s \Leftrightarrow \forall e \in E\ .\ obs(e, i) \subseteq$ … s (e, s)

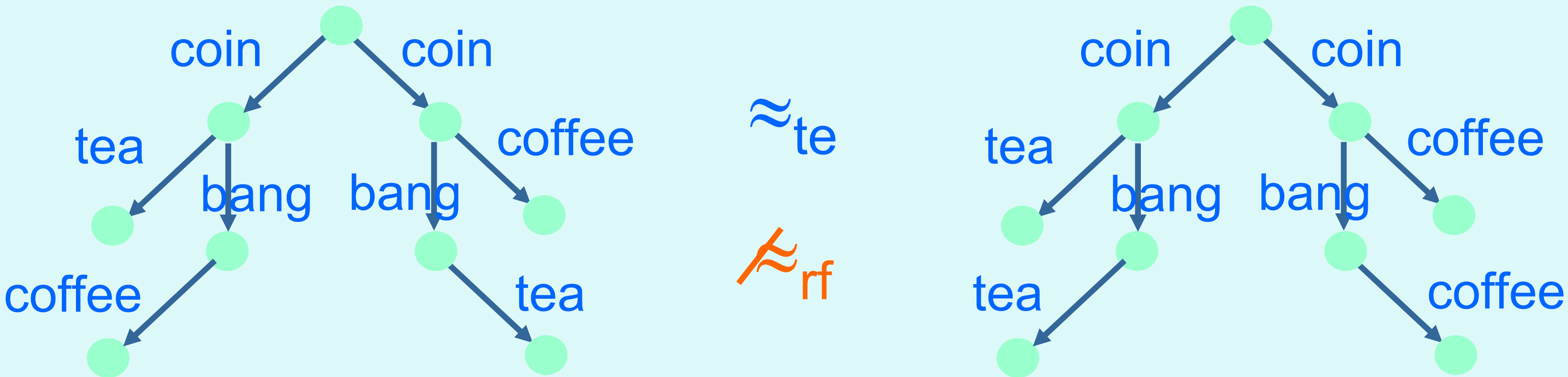
↓ ↓

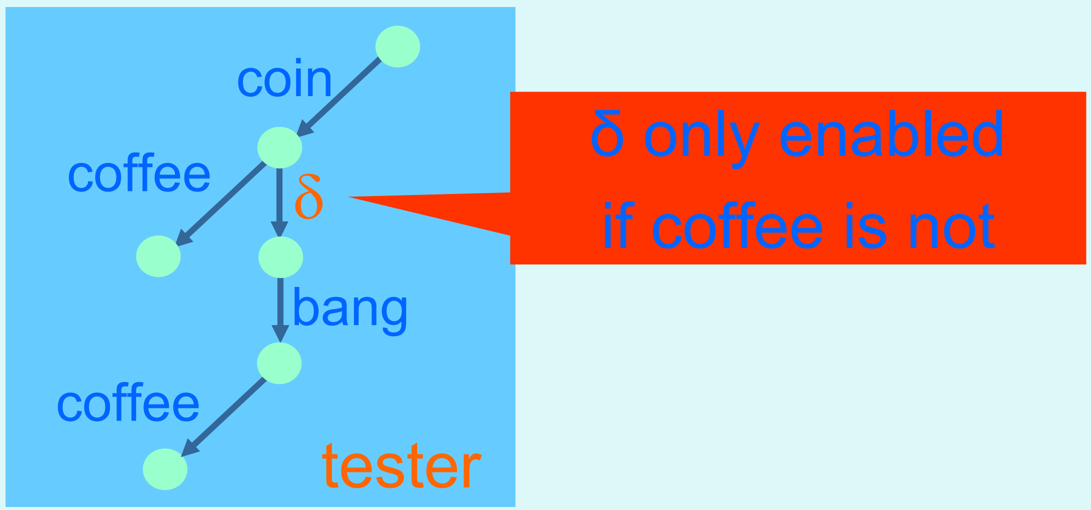
TS(L∪{δ}) Deadlocks $_{\delta}$(e||i)

e observes with δ
deadlock on all
alternative actions

Embedded Systems INSTITUTE

Quirky Coffee Machine Revisited





Embedded Systems INSTITUTE

I/O Transition Systems

- testing actions are usually directed, i.e. there are inputs and outputs

  $L = L_{in} \cup L_{out}$ with $L_{in} \cap L_{out} = \emptyset$

- systems can always accept all inputs (input enabledness)

  for all states s, for all $a \in L_{in}$ $s \overset{a}{\Rightarrow}$

- testers are I/O systems
  - output (stimulus) is input for the SUT
  - input (response) is output of the SUT

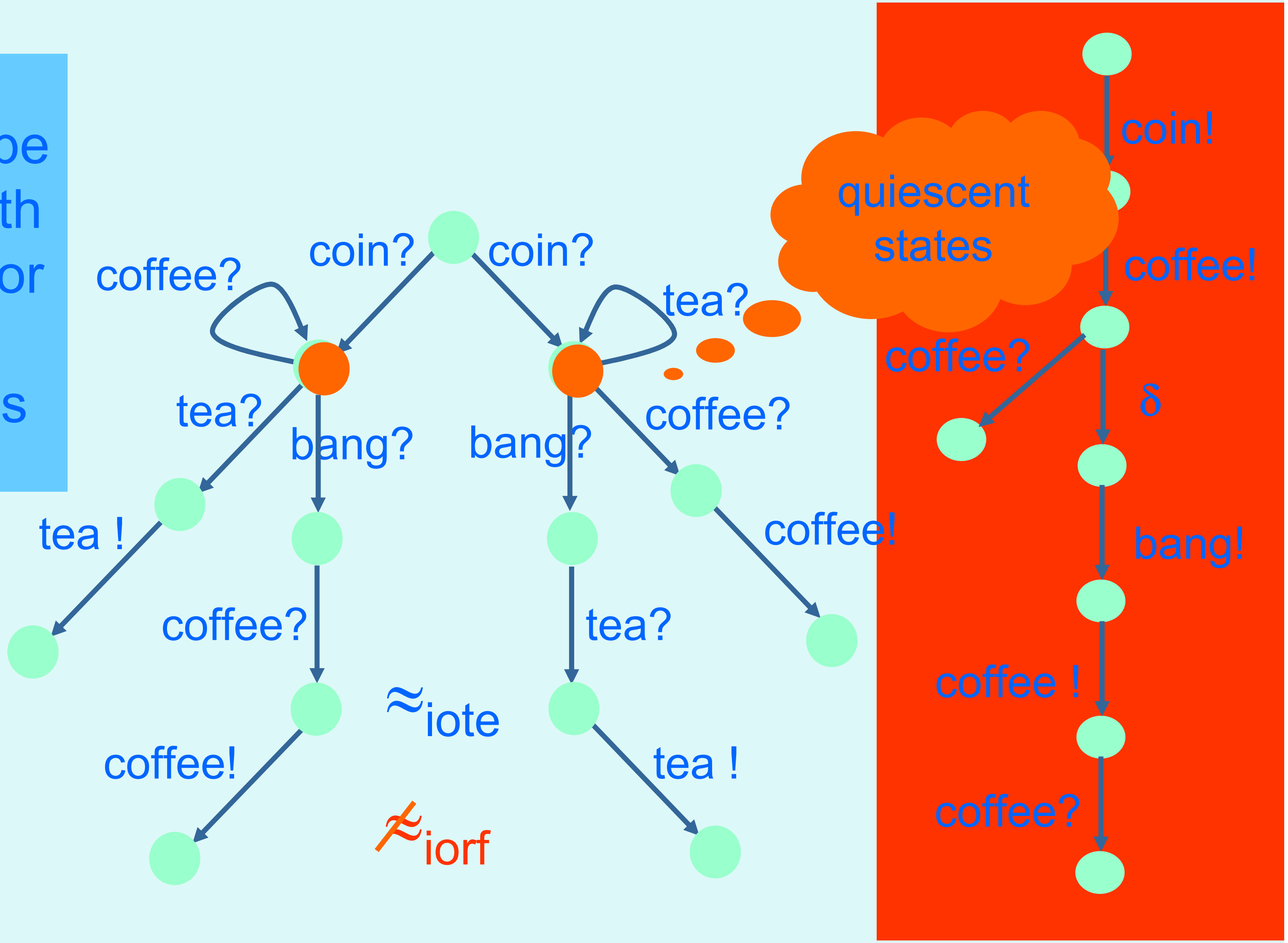
Embedded Systems INSTITUTE

Quiescence

- Because of input enabledness S||T deadlocks iff T produces no stimuli and S no responses. This is known as quiescence
- Observing quiescence leads to two implementation relations for I/O systems I and S:
  1. I $\leq_{iote}$ S iff for all I/O testers T:
     - Deadlocks(I||T) $\subseteq$ Deadlocks(S||T) (quiescence)
  2. I $\leq_{iorf}$ S iff for all I/O testers T:
     - $\text{Deadlocks}_{\delta}$(I||T) $\subseteq$ $\text{Deadlocks}_{\delta}$(S||T) (repetitive quiescence)

Input-Output QCM

states must be saturated with input loops for input enabledness



Implementation Relation ioco

Correctness expressed by implementation relation **ioco**:

$$i \textbf{ ioco } s \quad =_{def} \quad \forall \sigma \in Traces_{\delta}(s): \quad out\,(i \textbf{ after } \sigma) \subseteq out\,(s \textbf{ after } \sigma)$$

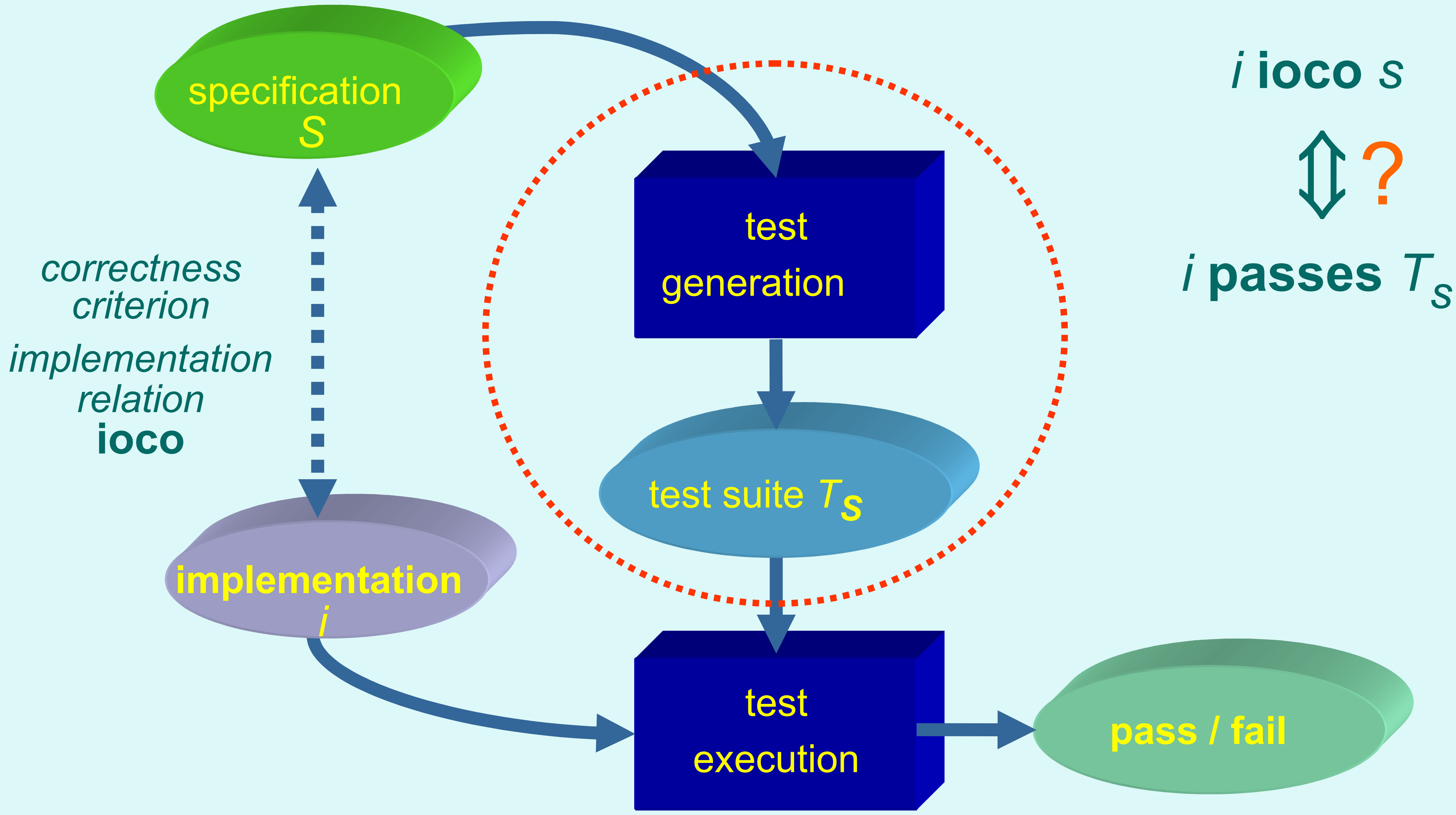
Intuition:

i **ioco**-conforms to s, iff

- if i produces output x after trace σ, then s can produce x after σ
- if i cannot produce any output after trace σ, then s cannot produce any output after σ (*quiescence* δ)

Embedded Systems INSTITUTE

Formal Testing



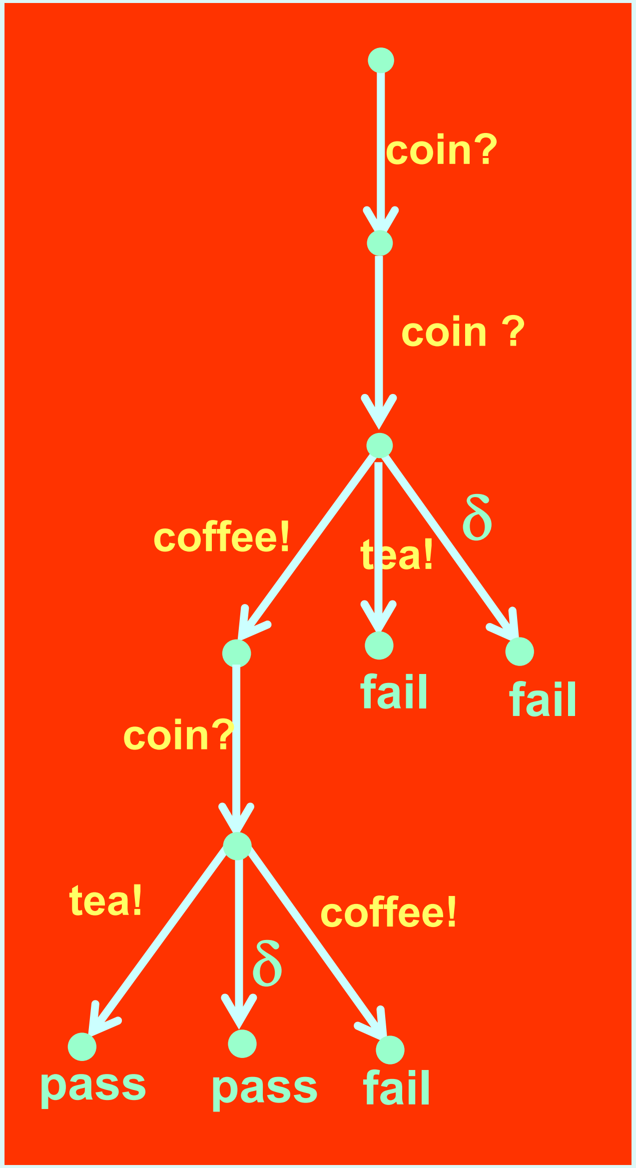
Embedded Systems INSTITUTE

Test Cases

Test case $t \in TTS$

TTS - Test Transition System :

- labels in $L \cup \{\delta\}$
- tree-structured
- finite, deterministic
- final states pass and fail
- from each state $\neq$ pass, fail
  - either one input i?
  - or all outputs o! and δ



Embedded Systems INSTITUTE

Test Generation Algorithm

Algorithm

To genera... test case t(S) ...
specificatio... et of s...

Apply the f...

1

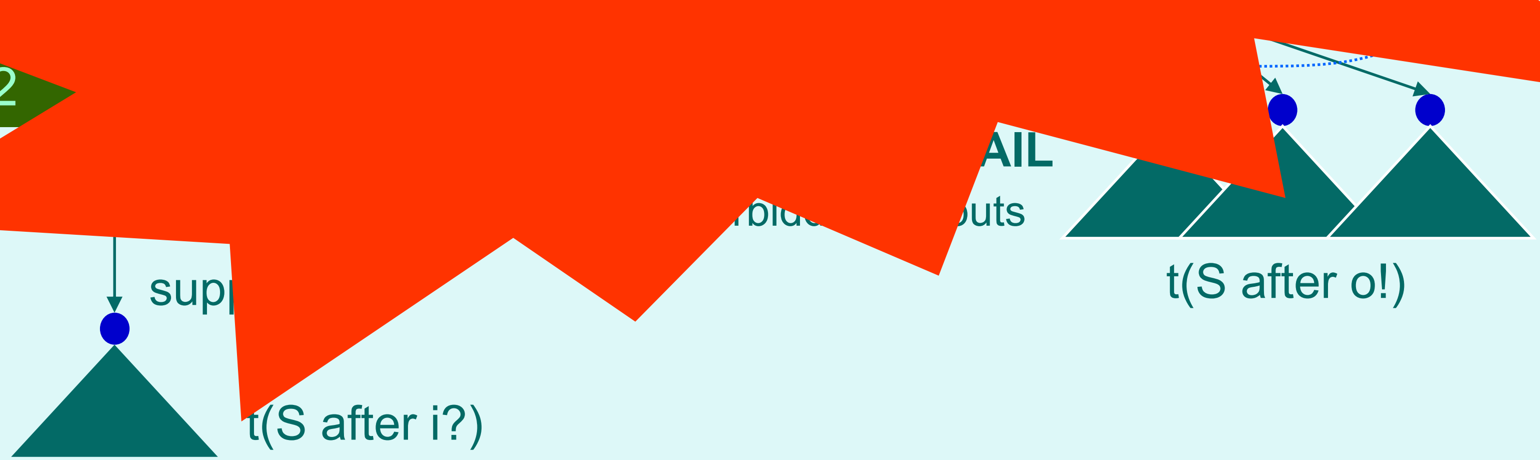
2

ioco-sound, i.e

no conforming implementation rejected

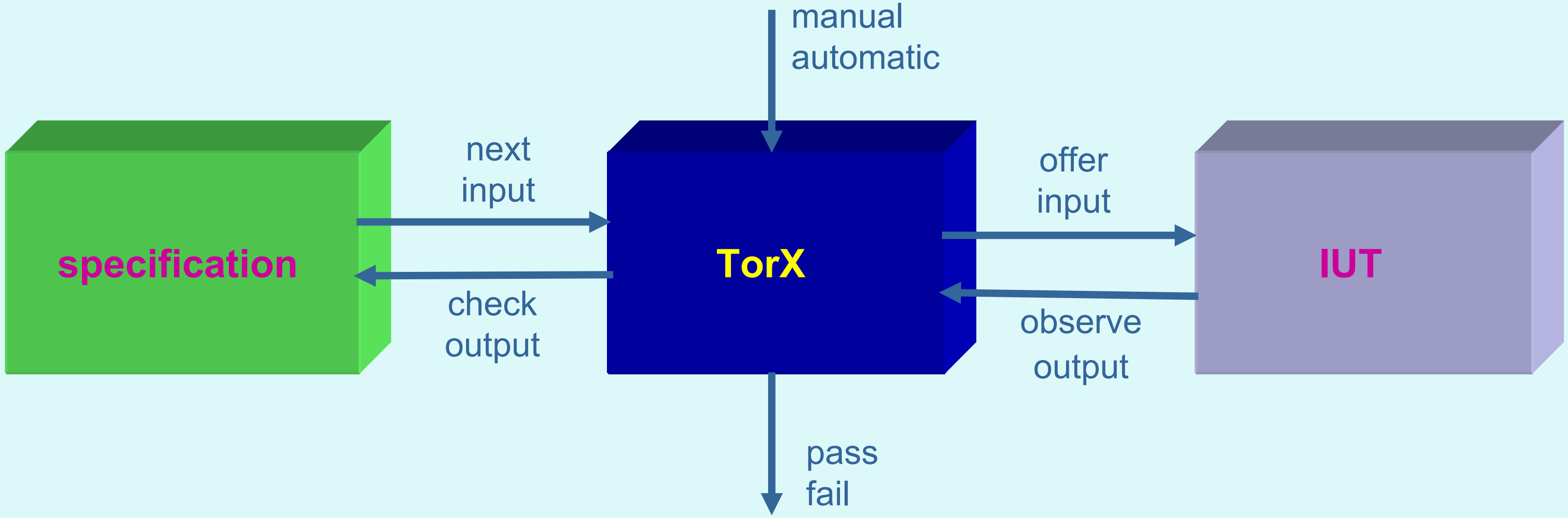
& (in the limit) ioco-complete, i.e

all non-conforming implementations rejected



Embedded Systems INSTITUTE

TorX



- On-the-fly test generation and test execution
- Implementation relation: **ioco**
- Specification languages LOTOS, Promela, FSP, etc.

Contents

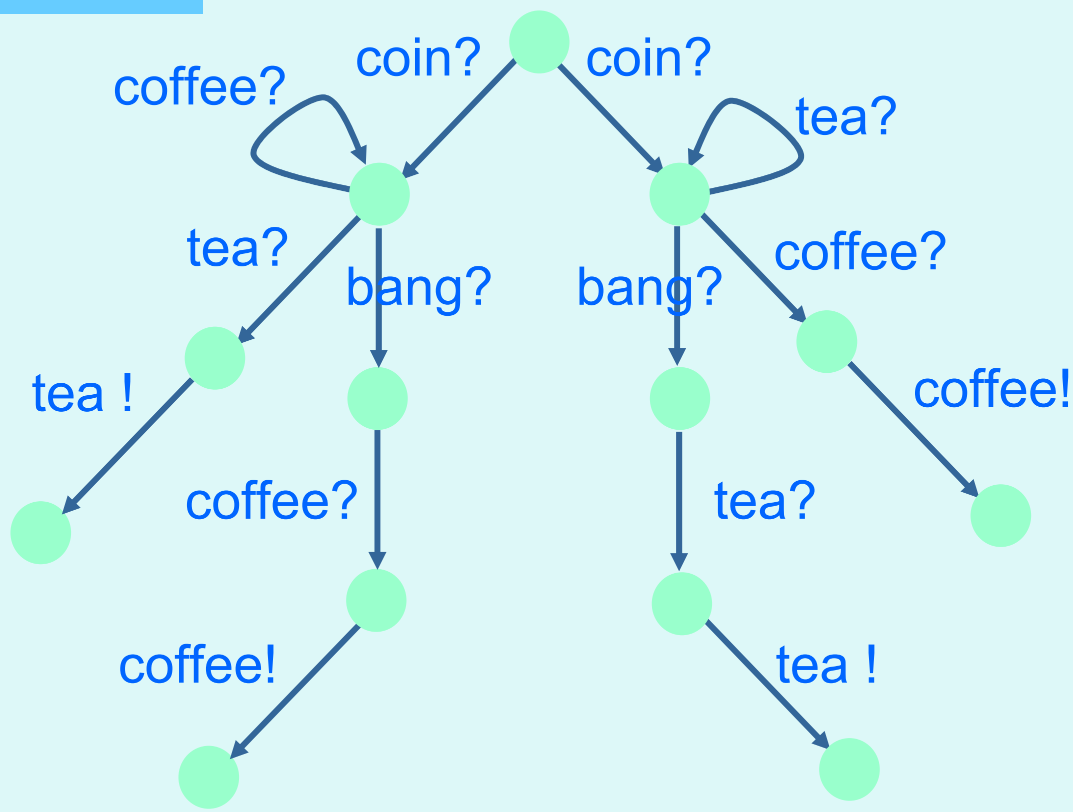
Embedded Systems
INSTITUTE

With real-time do we still need quiescence?

Embedded Systems INSTITUTE

Can't we make all useful distinctions using timed trace inclusion?

No!
Our standard example processes would become identical again in a real-time context



Embedded Systems INSTITUTE

Real-time and quiescence

- s is quiescent iff:

  for no output action a and delay d: $s \overset{a(d)}{\Rightarrow}$

- special transitions:

  $s \overset{\delta}{\rightarrow} s$ for every quiescent system state s

- testers observing quiescence take time:

  $Test_M$: set of test processes having only $\delta(M)$-actions to observe quiescence

- assume that implementations are M-quiescent:

  for all reachable states s and s':

  if $s \overset{\varepsilon(M)}{\Rightarrow} s'$ then s' is quiescent

Real-time and quiescence

$i \leq^{M}_{tiorf} s \iff \forall \mathbf{T} \in Test_M:$

$Deadlocks_{\boldsymbol{\delta}}(i||\mathbf{T}) \subseteq Deadlocks_{\boldsymbol{\delta}}(s||\mathbf{T})$

$\iff \forall \boldsymbol{\sigma} \in (L \cup \{\delta(M)\})^*:$

$out_M\,(\,i\ \textbf{after}\ \boldsymbol{\sigma}) \subseteq out_M\,(\,s\ \textbf{after}\ \boldsymbol{\sigma})$

$i\ \textbf{tioco}_M\ s \iff \forall \boldsymbol{\sigma} \in Traces_{\boldsymbol{\delta}(M)}(\,s\,):$

$out_M\,(\,i\ \textbf{after}\ \boldsymbol{\sigma}) \subseteq out_M\,(\,s\ \textbf{after}\ \boldsymbol{\sigma})$

Property

If implementation and specification are both M-quiescent then $\textbf{tioco}_M$ coincides with timed trace inclusion:

For all s,i M-quiescent

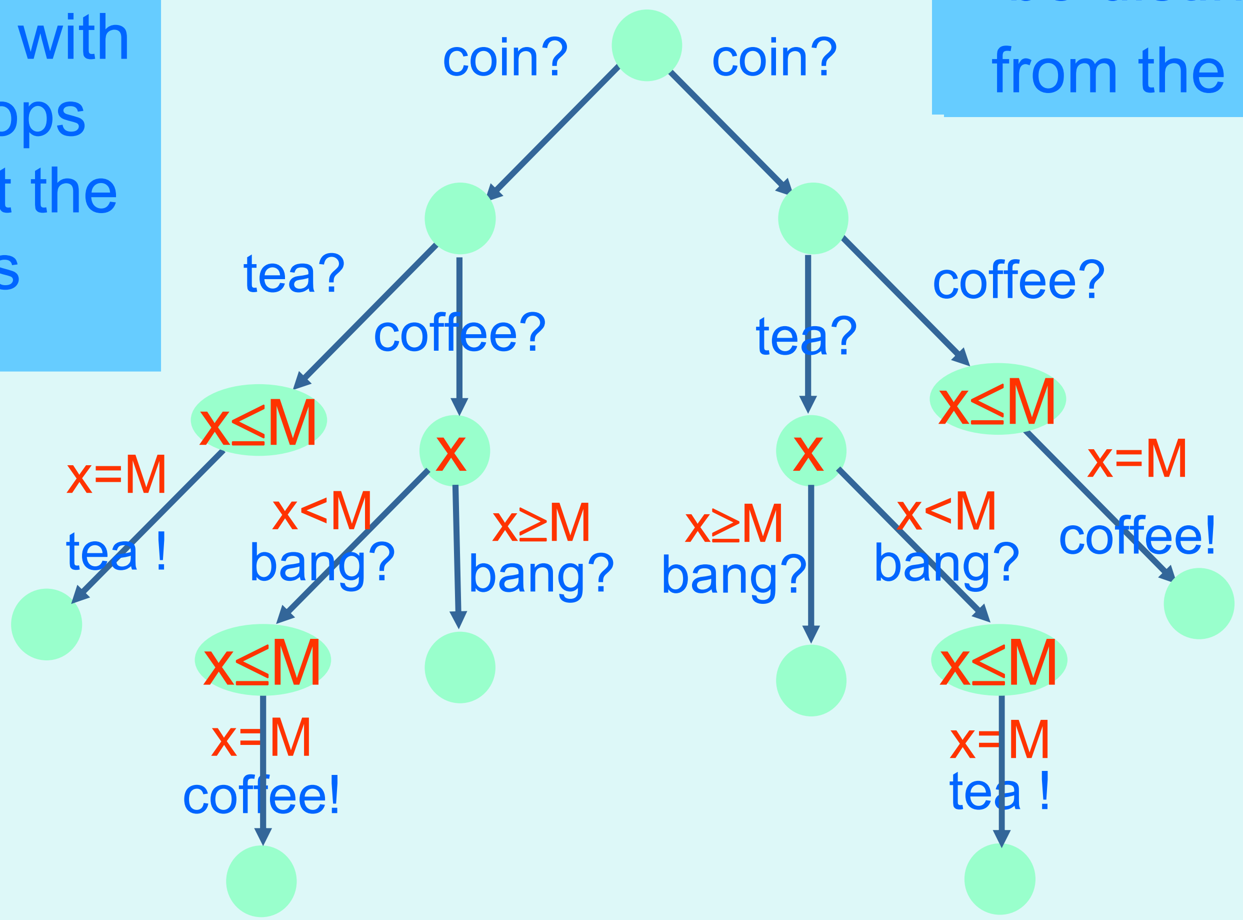
$$\text{Traces}(i) \cap \text{Traces}(s) \subseteq \text{Traces}(s) \quad \text{iff} \quad i\ \textbf{tioco}_M\ s$$

Embedded Systems INSTITUTE

A limitation

states are saturated with input loops that reset the clocks

this process cannot be distinguished from the previous



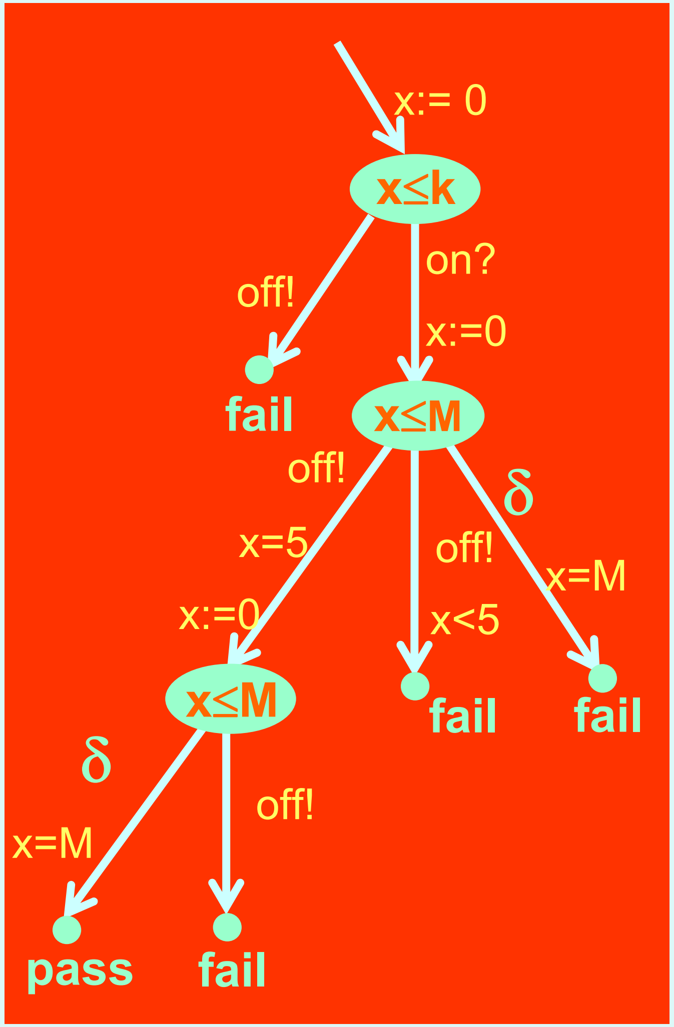
Real-time test cases

Embedded Systems INSTITUTE

Test case t ∈ TTA

TTA – Test Timed Automata :

- labels in $L \cup \{\delta\}$, G(d)
- tree-structured
- finite, deterministic
- final states pass and fail
- from each state ≠ pass, fail
  - choose an input **i?** and a time **k** and wait for the time **k** accepting all outputs **o!** and after k time units provide input **i?**
  - or wait for time **M** accepting all outputs **o!** and δ



Timed test generation proto-algorithm

To generate a test case $t(S)$ from a timed transition system specification with S set of states (initially $S = \{s_0\}$)

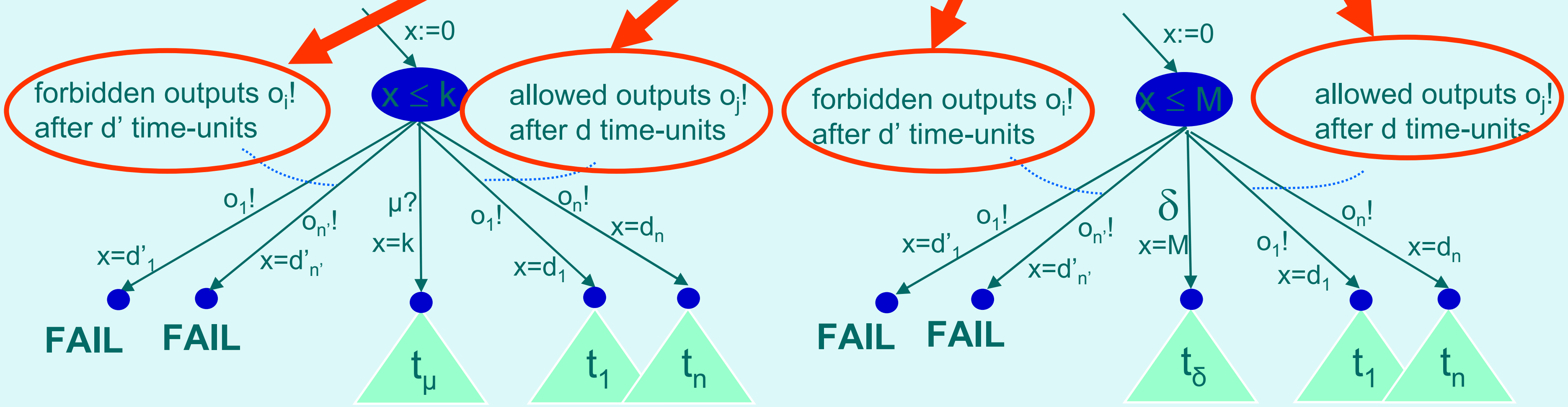
Apply the following steps recur

These sets must be computable!

1. end test case ● **PASS**

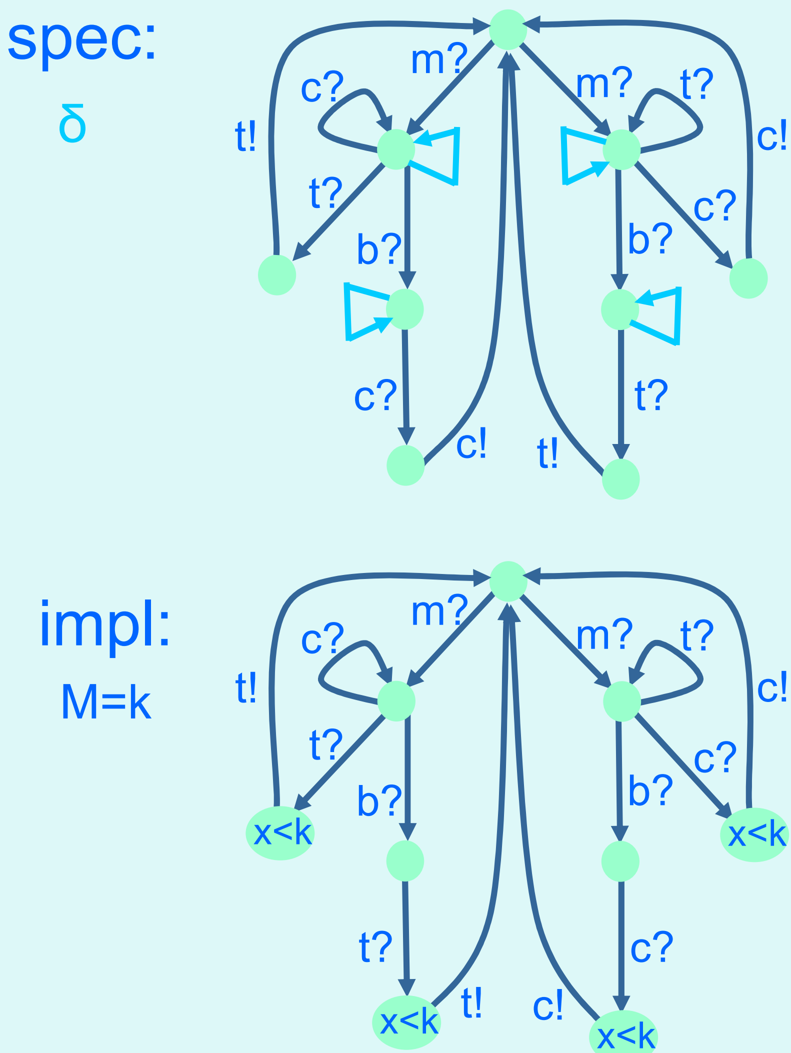
2. choose $k \in (0, M)$ and input μ

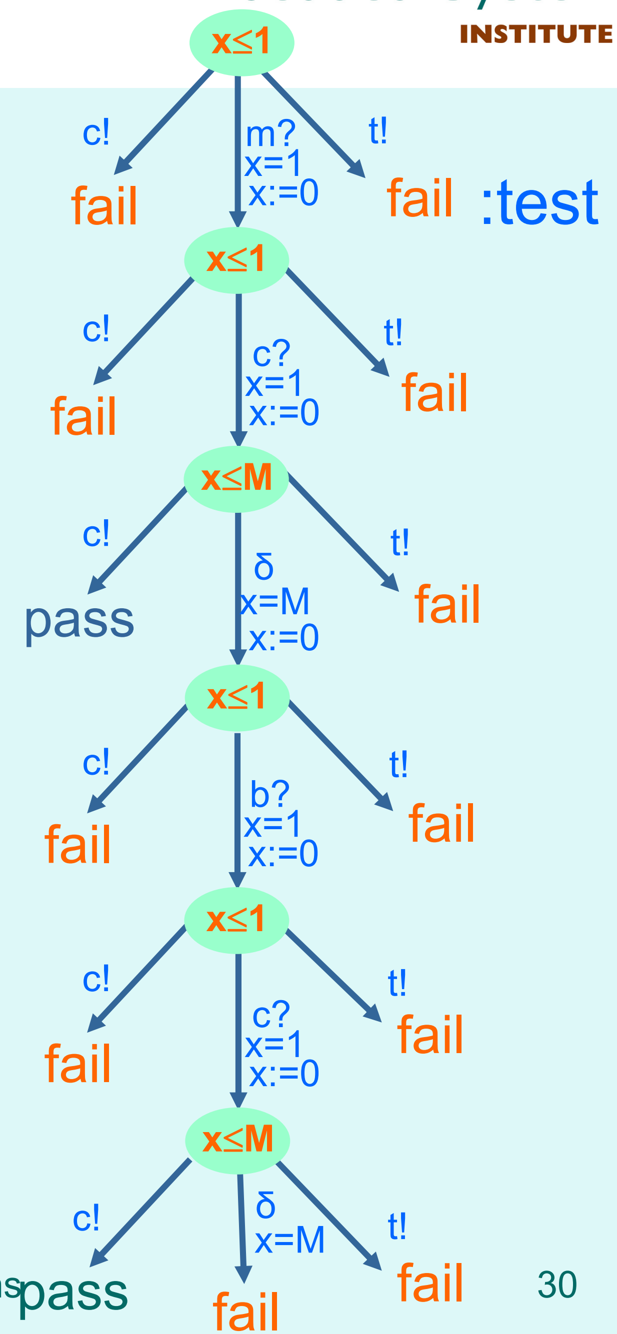
3. wait for observing possible output



Embedded Systems INSTITUTE

Example





Soundness & completeness

- the non-timed generation algorithm can be shown to generate only sound real-time test cases
- test generation is complete

  for every erroneous trace it can generate a test that exposes it
- test generation is not limit complete

  because of continuous time there are uncountably many timed error traces and only countably many test are generated by repeated runs
- test generation is almost limit complete

  repeated test geration runs will eventually generate a test case that will expose one of the non-spurious errors of a non-conforming implementation

non-spurious errors
=
errors with a positive probability of occurring

Contents

Embedded Systems INSTITUTE

Coverage: motivation

- **Testing is inherently incomplete**
  - Test selection is crucial
- **Coverage metrics**
  - Quantitative evaluation of test suite
  - Count how much of specification/implementation has been examined
- **Examples:**
  - **White box (implementation coverage):**
    - **Statement, path, condition coverage**
  - **Black box (specification coverage)**
    - **State, transition coverage**

Traditional coverage measures

Traditional measures are:

- based on syntactic model features
  - states, transitions, statements, tests
- uniform
  - all system parts treated as bequally important

Disadvantages:

- replacing the spec by an equivalent one yields different coverage
  - we need a *semantic* approach
- some bugs are more important than others;
  - test crucial behaviour first and better

Our Approach

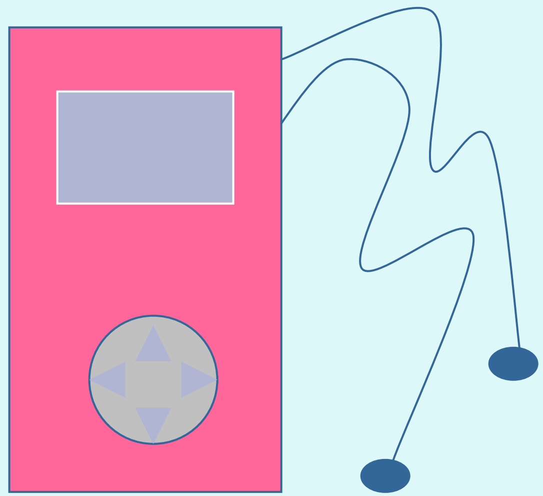
Embedded Systems INSTITUTE

- **Considers black box coverage**
  - similar ideas could apply to white box coverage
- **Is semantic**
  - Semantically equivalent specs yield same coverage
- **Is risk-based**
  - more important bugs/system parts
  - → higher contribution to coverage
- **Allows for optimization**
  - Cheapest test suite with 90% coverage
  - Maximal coverage within cost budget

Fault models

- f: *Observation* → $R^{\geq 0}$
  - f(σ) = 0 : correct behaviour
  - f(σ) > 0 : incorrect behaviour

    : f(σ) severity
  - $0 < \Sigma_\sigma f(\sigma) < \infty$
- Observations are traces
  - *Observations* = L^*
  - $L = (L_I, L_U)$
- How to obtain f?
  - E.g. via fault automaton

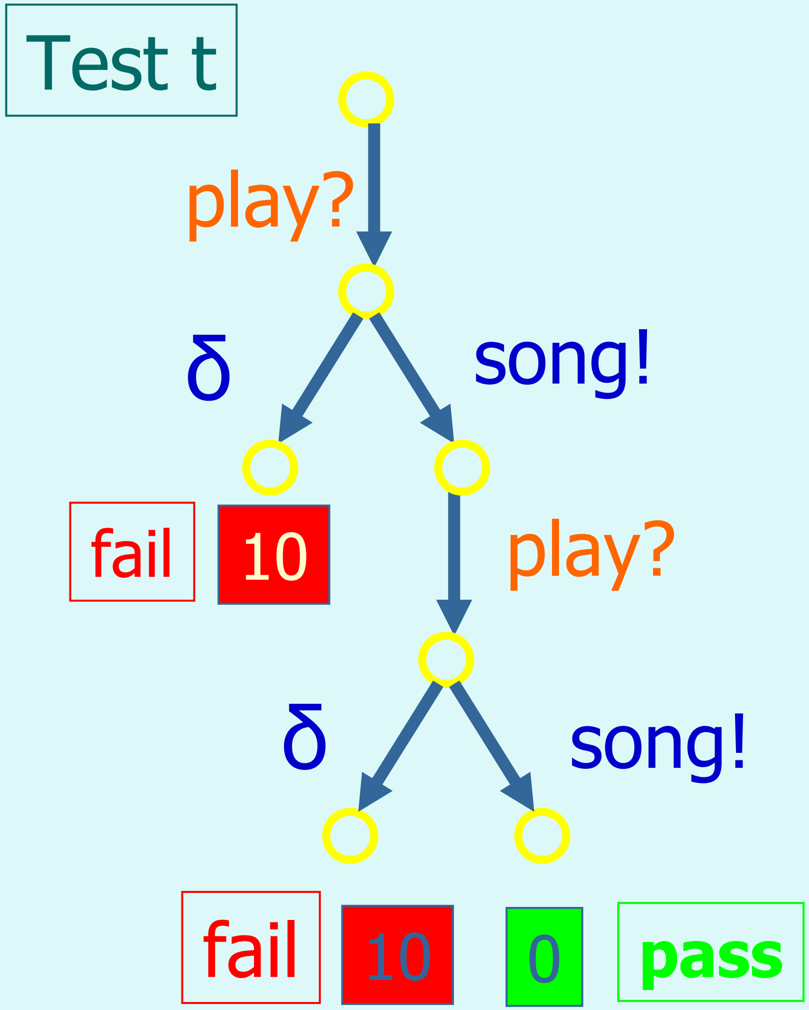


f: L* → $R^{\geq 0}$

f(play? song!) = 0 correct
f(play? silence!) = 10 incorrect
f(song!) = 3 incorrect

Example test case

Embedded Systems INSTITUTE



- f: L* → R
  - f(play? song!) = 0
  - f(play? δ) = 10
  - f(play? song! play? δ) = 10
  - f(song!) = 3
- $\Sigma_\sigma f(\sigma) = 100$ (assumption)

- **Absolute Coverage abscov(f,t)**
  - sum the error weights
  - 10 + 10 + 0 = 20

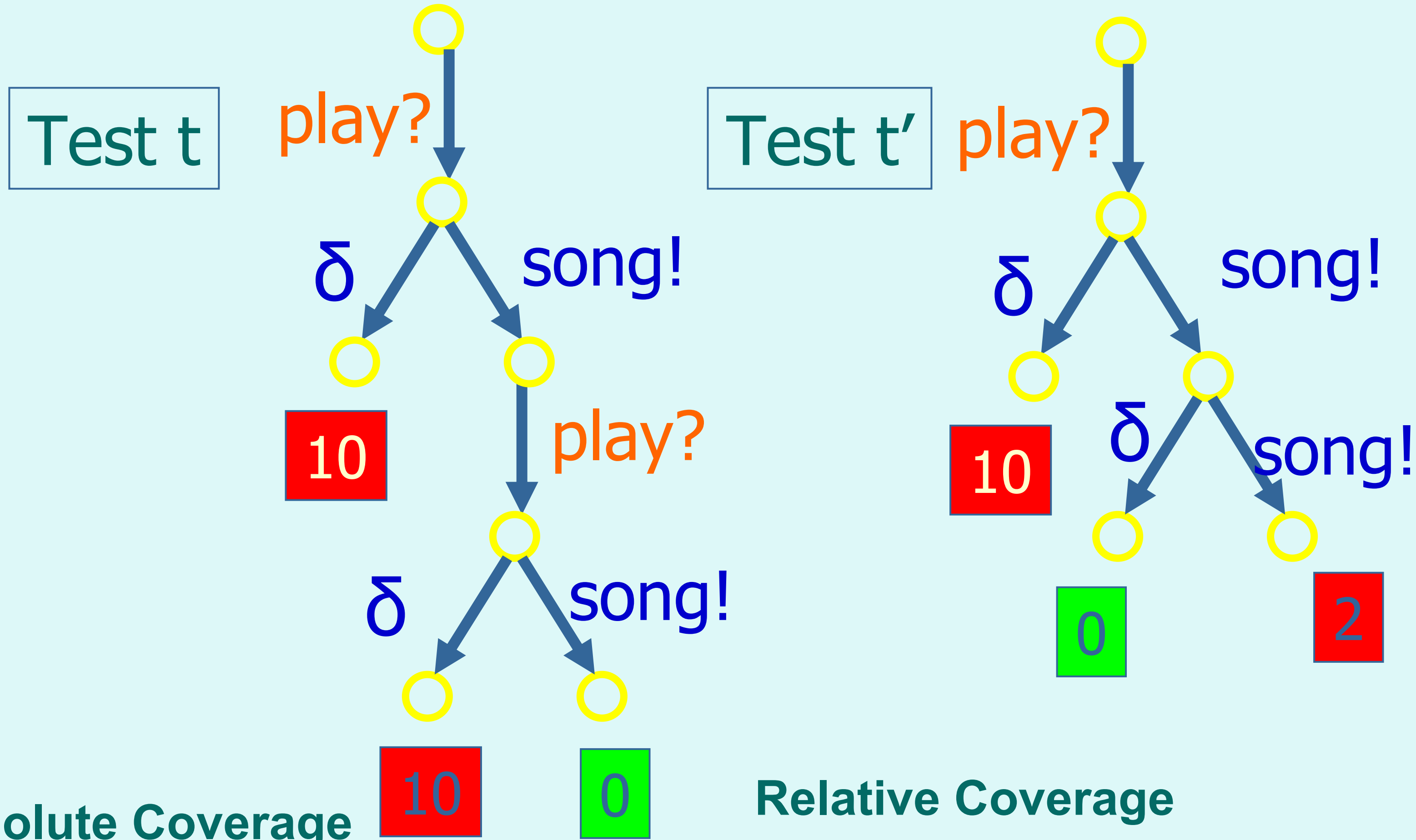
- **Relative Coverage**

$$\frac{abscov(f,t)}{totcov(f)} = \frac{20}{100}$$

should be ≠ 0, ≠∞

Example test suite

Embedded Systems
INSTITUTE



Absolute Coverage

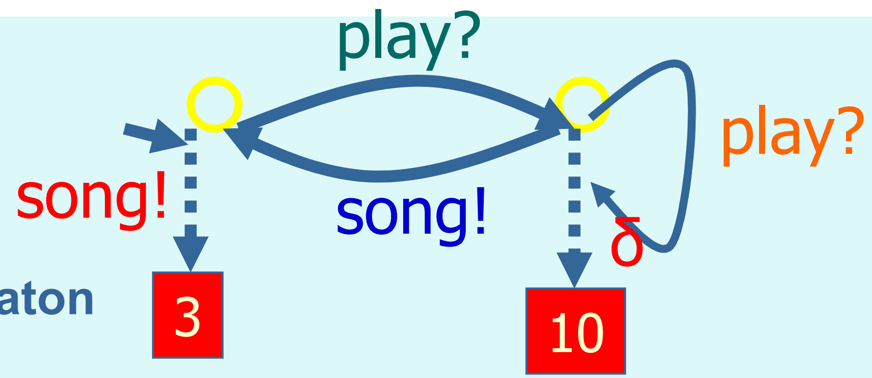
- count each trace once !
- 10 + 10 + 0 + 0 + 2 = 22

Relative Coverage

$$\frac{abscov(f,t)}{totcov(f)} = \frac{22}{100} = 22\%$$

Fault specifications

Embedded Systems INSTITUTE



fault model

- $f(\sigma) = 0$ if σ trace of automaton
- $f(\sigma) = 3 \cdot \alpha^{|\sigma|-1}$ if σ ends in 3-state
- $f(\sigma) = 10 \cdot \alpha^{|\sigma|-1}$ if σ ends in 10-state

Use your favorite Formalism, e.g. UML state charts, LOTOS, etc

infinite total coverage !!

- $\Sigma_\sigma f(\sigma) = 3 + 10 + 3 + 10 + \ldots = \infty$

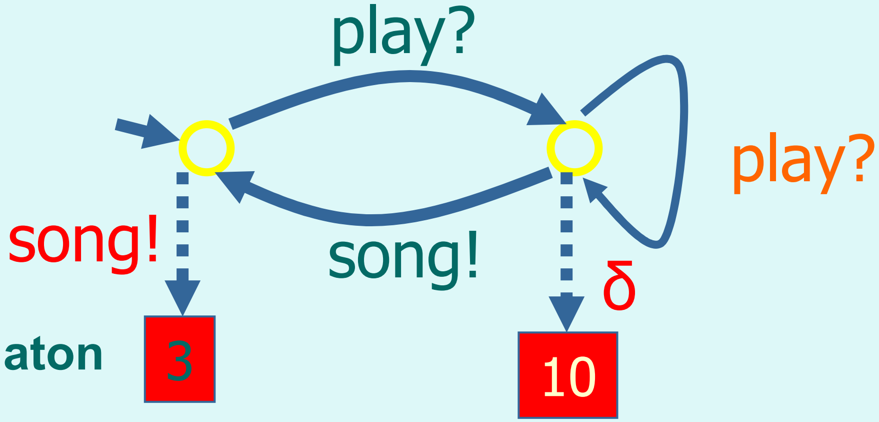
Solution 1: restrict to traces of lenght k

- Omit here, works as solution 2, less efficient, more boring

Solution 2: discounting

- errors in short traces are worse
- Lower the weight proportional to length

Fault specifications



fault model

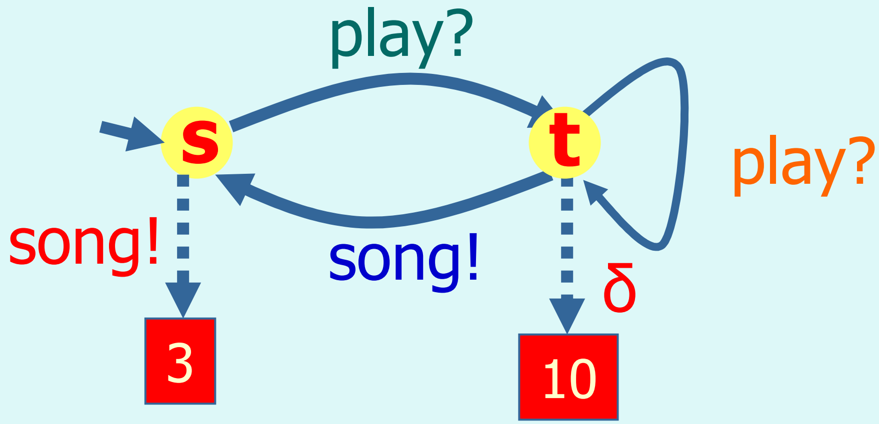
- $f(\sigma) = 0$ if σ trace of automaton
- $f(\sigma) = 3 \cdot \alpha^{|\sigma|-1}$ if σ end in 3-state
- $f(\sigma) = 10 \cdot \alpha^{|\sigma|-1}$ if σ ends in 10-state

Example

- f(play?) = 0
- f(play? δ) = 10 · α
- f(play? song! song!) = $3 \cdot \alpha^2$
- ....

> - α < 1/out(spec) = 1/2
> - α can vary per transition
> - tune α

Fault specifications



Total coverage becomes finite & computable:

$tc(s) = 3 + \alpha\, tc(t)$

$tc(t) = 10 + \alpha\, tc(t) + \alpha\, tc(s)$

$tc(x) = wgt(x) + \alpha \sum_{y:\ succ(x)} tc(y)$

Solve linear equations

$tc = wgt\, (I - \alpha A)^{-1}$ A adjacency matrix

tc(s) =

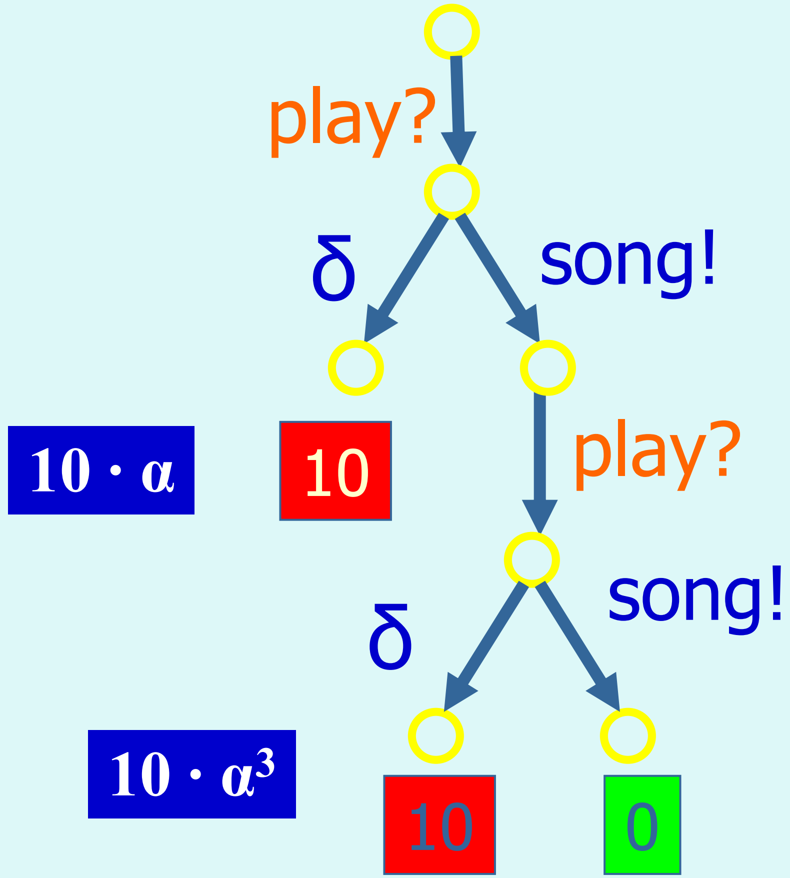
$$\frac{10 + 7\alpha}{1 - \alpha - \alpha^2}$$

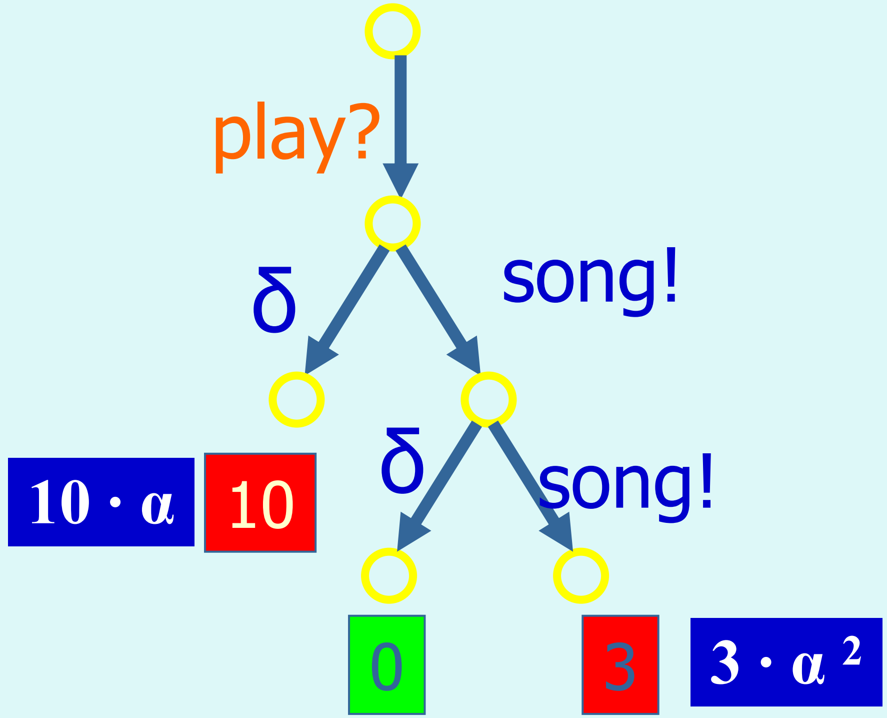
Relative Coverage

$$\frac{abscov(f,t)}{totcov(f)}$$

Test suite coverage

Embedded Systems INSTITUTE





Absolute test suite coverage

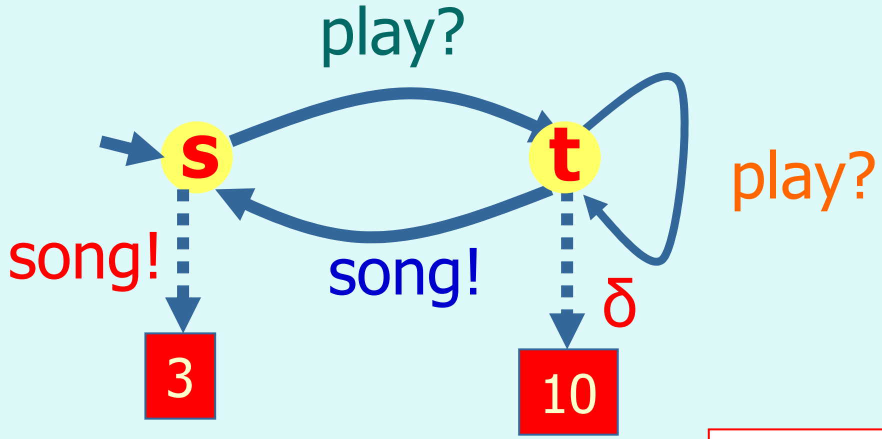
- count each trace once!
- merge test cases first

Relative test suite coverage

$$\frac{\text{abscov}(f,t)}{\text{totcov}(f)} = \frac{10\alpha + 3\alpha^2 + 10\alpha^3}{\dfrac{10 + 7\alpha}{1 - \alpha - \alpha^2}}$$

Optimization

Embedded Systems INSTITUTE



Find best test case of lenght *n*

$v_1(s) = 3$

$v_1(t) = 10$

$v_{k+1}(s) = \max(3, \alpha\, v_k(t))$

$v_{k+1}(t) = \max(10 + \alpha\, v_k(s), \alpha\, v_k(t))$

Complexity: O(*n* #transitions in spec)

More optimizations:

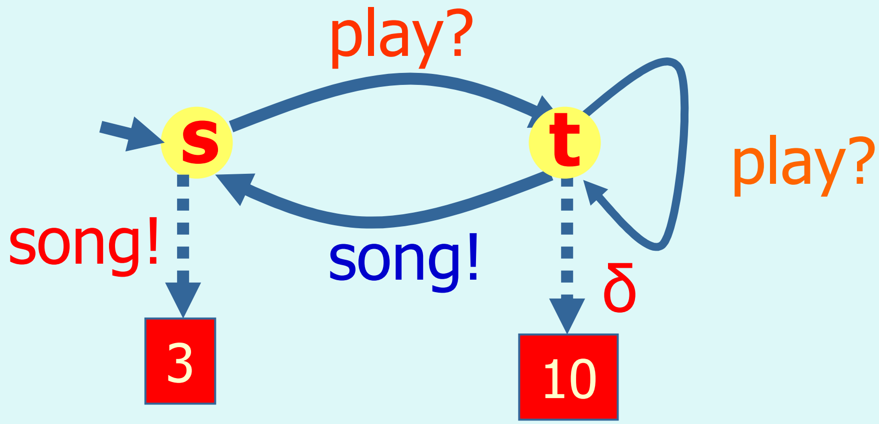
- Test suite of k tests & lenght n;
- Best test case in budget;
- Add costs
- ....

Properties

Framework for black box coverage

- **robustness**
  - small changes in weight yield small changes in coverage
  - relcov(s) continuous
- **tunable (calibration)**
  - change α : get as much total coverage as desired

Calibration



α **small** → present is important, future unimportant

→ few small test cases with high (>0.999) coverage

tune α → make tests with lenght >*k* important, i.e make $cov(T_k, f)$ as small as desired.

→ $\alpha(s) = 1/n(s) - \varepsilon$ **n(s) =outinf(s)**

→ $\lim_{\varepsilon \to 0} cov(T_k, f_\alpha) = 0$ **for all k**

CONCLUSIONS

Embedded Systems
INSTITUTE

Conclusions

- model-based testing offers theory and tools for (real-time) conformance testing, in particular:
  - test generation
  - test execution
  - test evaluation
  - coverage analysis
- ioco-theory, TorX and related tools have been evaluated against many industrial cases
  - on-the-fly application very productive
  - good coverage with random test execution
- current theory is control-oriented
  - OK for classical embedded applications
  - must be extended to cope with data-intensive systems

Future work

Embedded Systems
INSTITUTE

- **integration with data-oriented testing**
  - *classical, symbolic*
- **stochastic systems**
  - *continuous & discrete time Markov chains*
- **quality of service**
  - *performance testing*
- **hybrid systems**
  - *testing discrete vs polling continuous behaviour*
- **actual coverage measures**
  - *actual coverage during test execution*
- **integration white/black box spectrum**
  - *grey-box testing*
- **...**

Resources

- http://fmt.cs.utwente.nl/tools/torx/introduction.html
- http://www.testingworld.org/
- http://www.laquso.com/knowledge/toolstable.php
- http://www.irisa.fr/vertecs/
- http://www.cs.aau.dk/~marius/tuppaal/