Model-Based Testing with Labelled Transition Systems

There is Nothing More Practical than a Good Theory

Jan Tretmans

TNO – ESI
Eindhoven, NL

Radboud University
Nijmegen, NL

jan.tretmans@tno.nl

Model-Based Testing
with Labelled Transition Systems (LTS)

Overview of a Theory

Embedded Systems Innovation BY TNO

- Models LTS
- Comparing LTS
  - equivalences
- Correctness
  - implementation relation
  - **ioco**
- Testing LTS
  - test generation
  - test execution
- Correctness & Testing
  - soundness
  - exhaustiveness
- SUT: Black-Box & Formal
  - test assumption
- Consequences
  - (non) compositionality
  - variations of **ioco**

Labelled Transition Systems

Labelled Transition Systems

Labelled Transition System $\langle$ **S**, **L**, **T**, s_0 $\rangle$

states

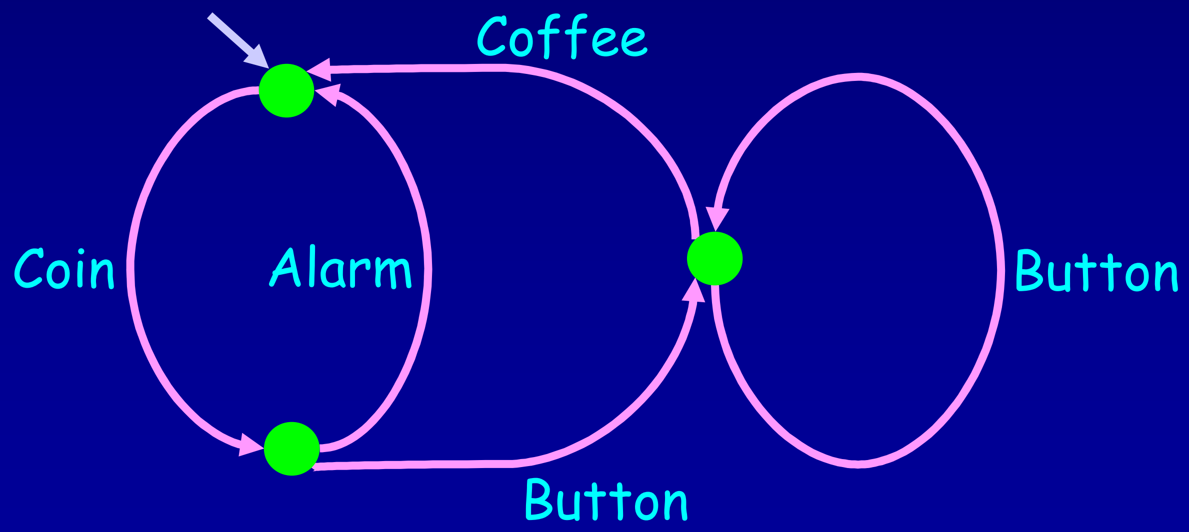
actions

transitions

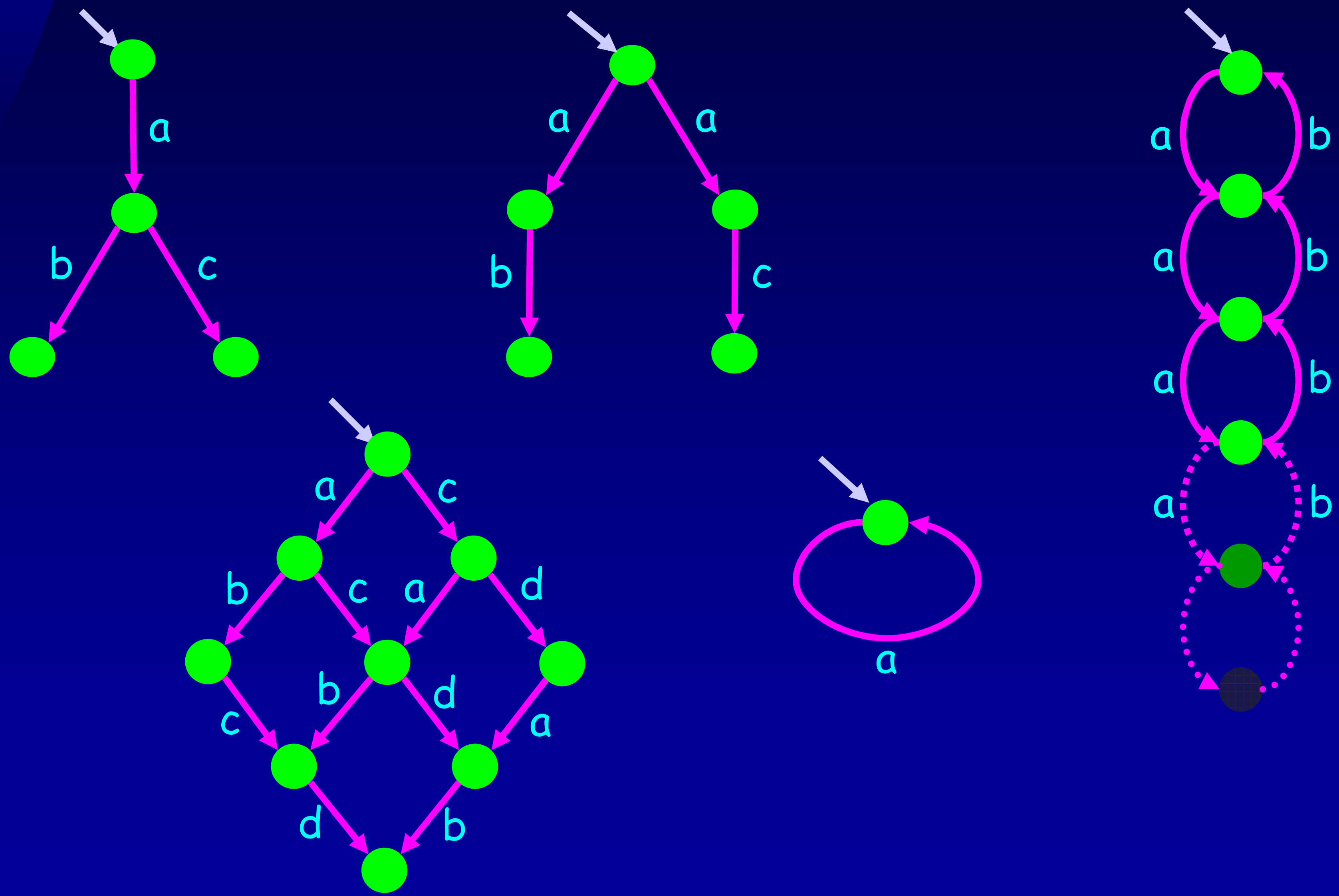
$T \subseteq S \times (L \cup \{\tau\}) \times S$

initial state

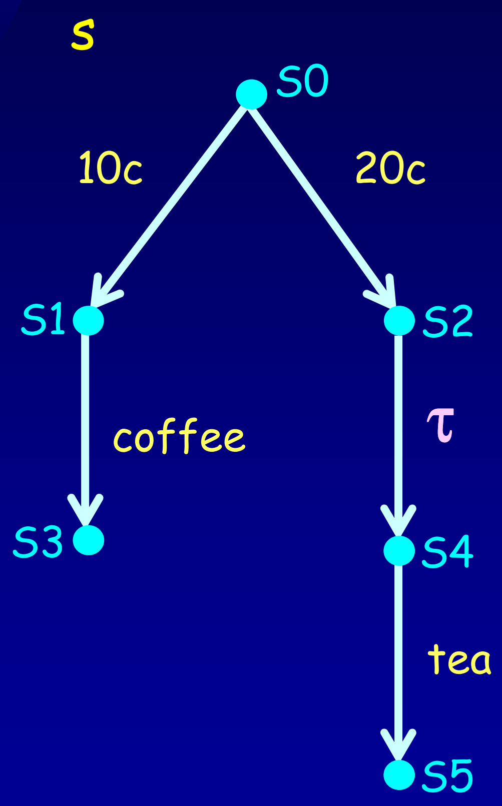
$s_0 \in S$



Labelled Transition Systems



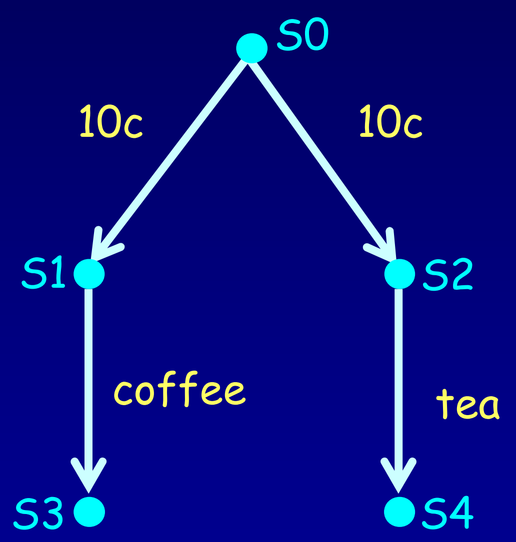
Labelled Transition Systems



$L = \{ 10c, 20c, coffee, tea, soup \}$

$S0 \xrightarrow{10c} S1$	transition
$S0 \xRightarrow{10c\ \ coffee} S3$	transition composition
$S0 \xRightarrow{20c\ \ tea}$	executable sequence
$S0 \not\xRightarrow{20c\ \ soup}$	non-executable sequence
LTS(L)	all transition systems over L

Labelled Transition Systems



Sequences of observable actions:

traces (s) = $\{ \sigma \in L^* \mid s \overset{\sigma}{\Longrightarrow} \}$

traces(s) = $\{ \varepsilon$, 10c, 10c coffee, 10c tea $\}$

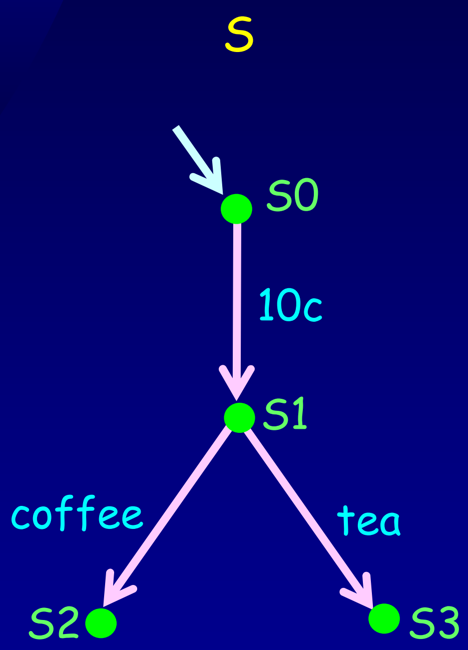
Reachable states:

s after σ = $\{ s' \mid s \overset{\sigma}{\Longrightarrow} s' \}$

s after 10c = { S1, S2 }

s after 10c tea = { S4 }

Representation of LTS



☞ Explicit :

⟨ {S0,S1,S2,S3},

{10c,coffee,tea},

{ (S0,10c,S1), (S1,coffee,S2), (S1,tea,S3) },

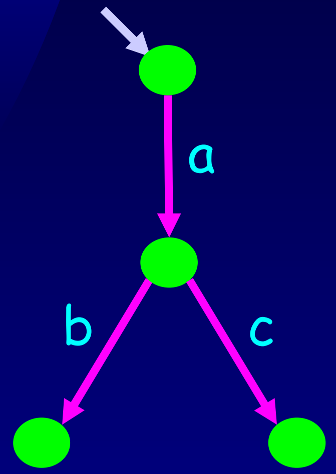
S0 ⟩

☞ Transition tree / graph

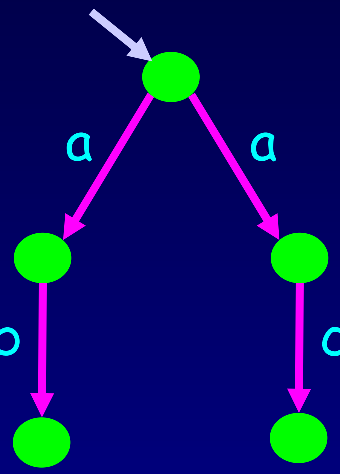
☞ Language / behaviour expression :

S := 10c ; (coffee ; **stop** [] tea ; **stop**)

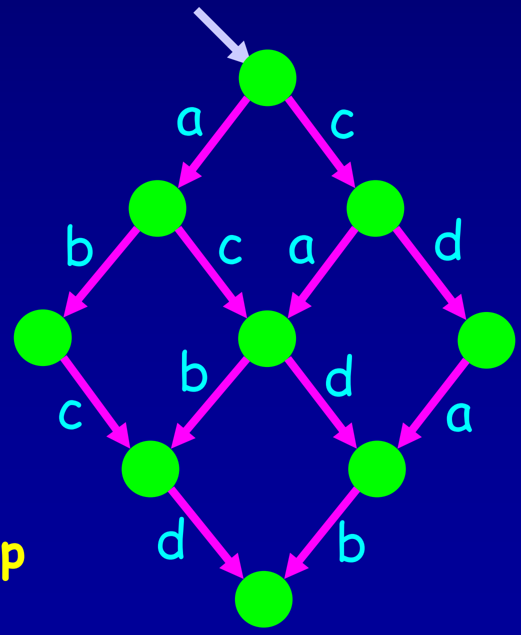
Representation of LTS



a ; (b ; **stop** [] c ; **stop**)

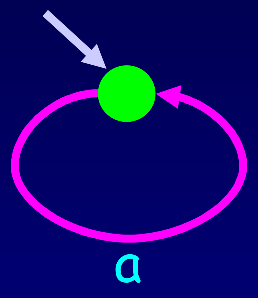


a ; b ; **stop** [] a ; c ; **stop**



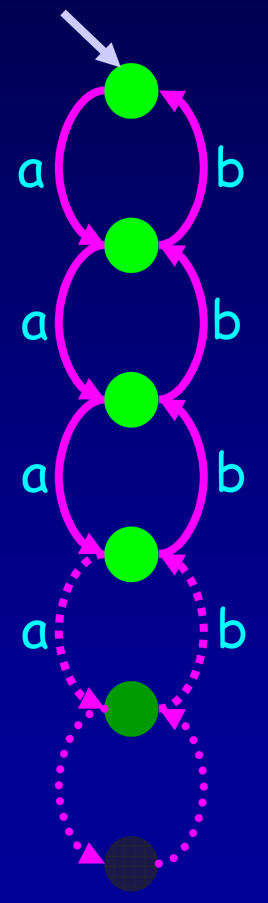
a ; b ; **stop** ||| c ; d ; **stop**

Representation of LTS



Q, where

Q := a ; (b ; **stop** ||| Q)

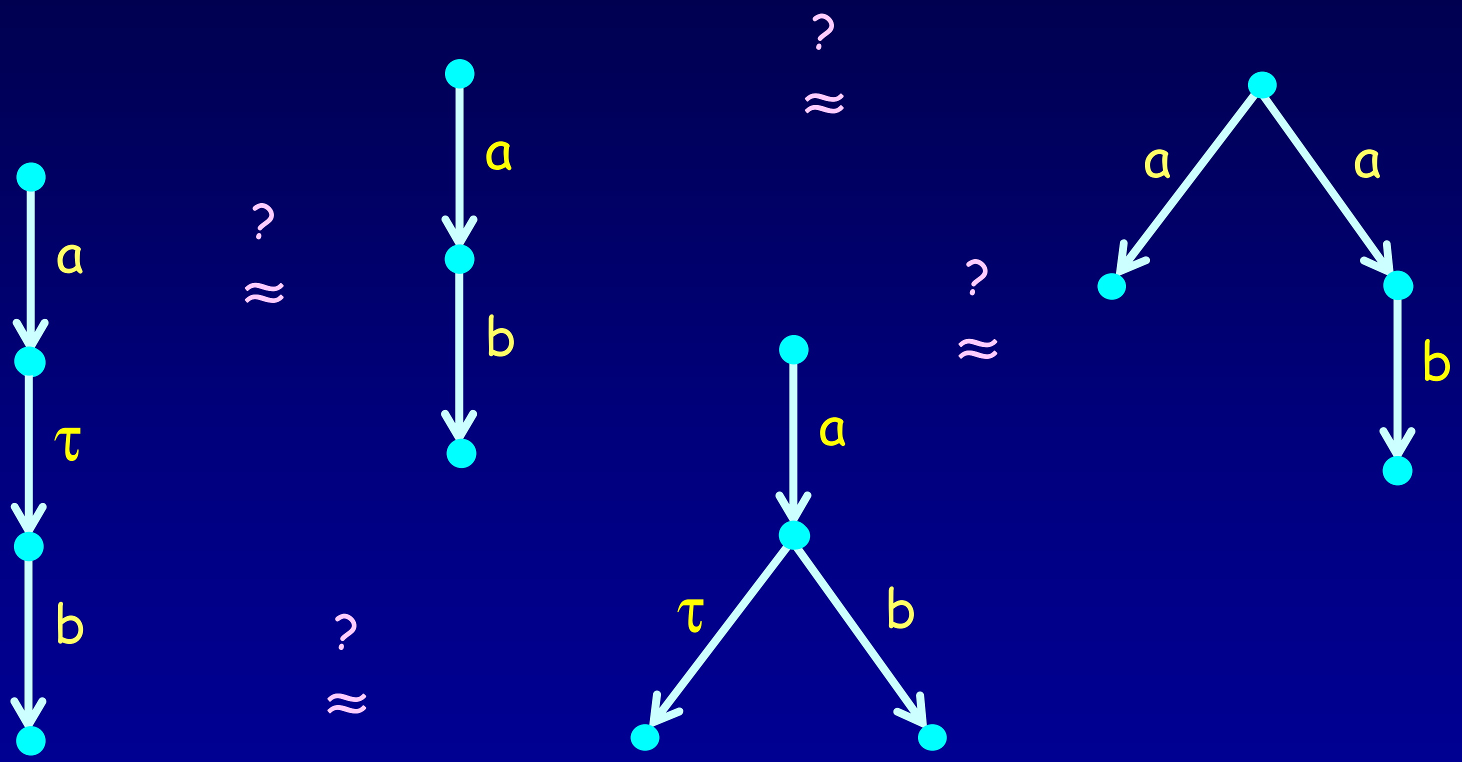


P, where

P := a ; P

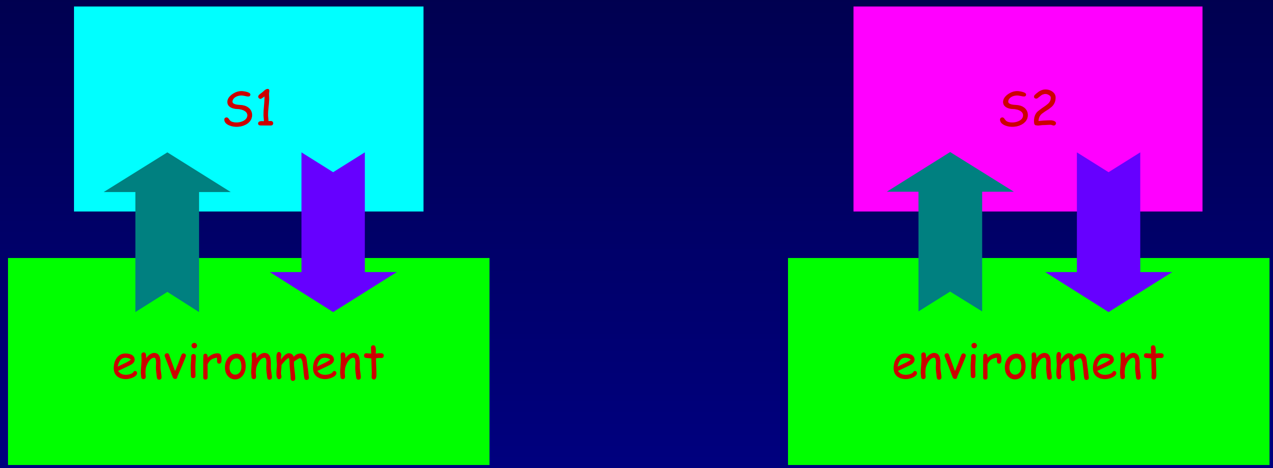
Equivalences on Labelled Transition Systems

Observable Behaviour



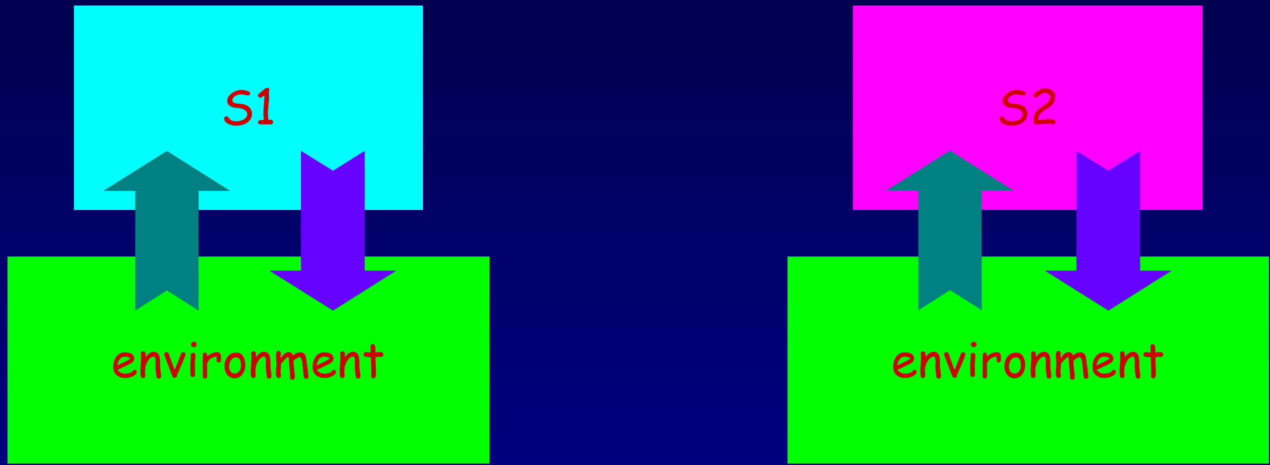
" Some transition systems are more equal than others "

Comparing Transition Systems



- ☞ Suppose an environment interacts with the systems:
  - the environment tests the system as black box by observing and actively controlling it;
  - the environment acts as a tester;
- ☞ Two systems are equivalent if they pass the same tests.

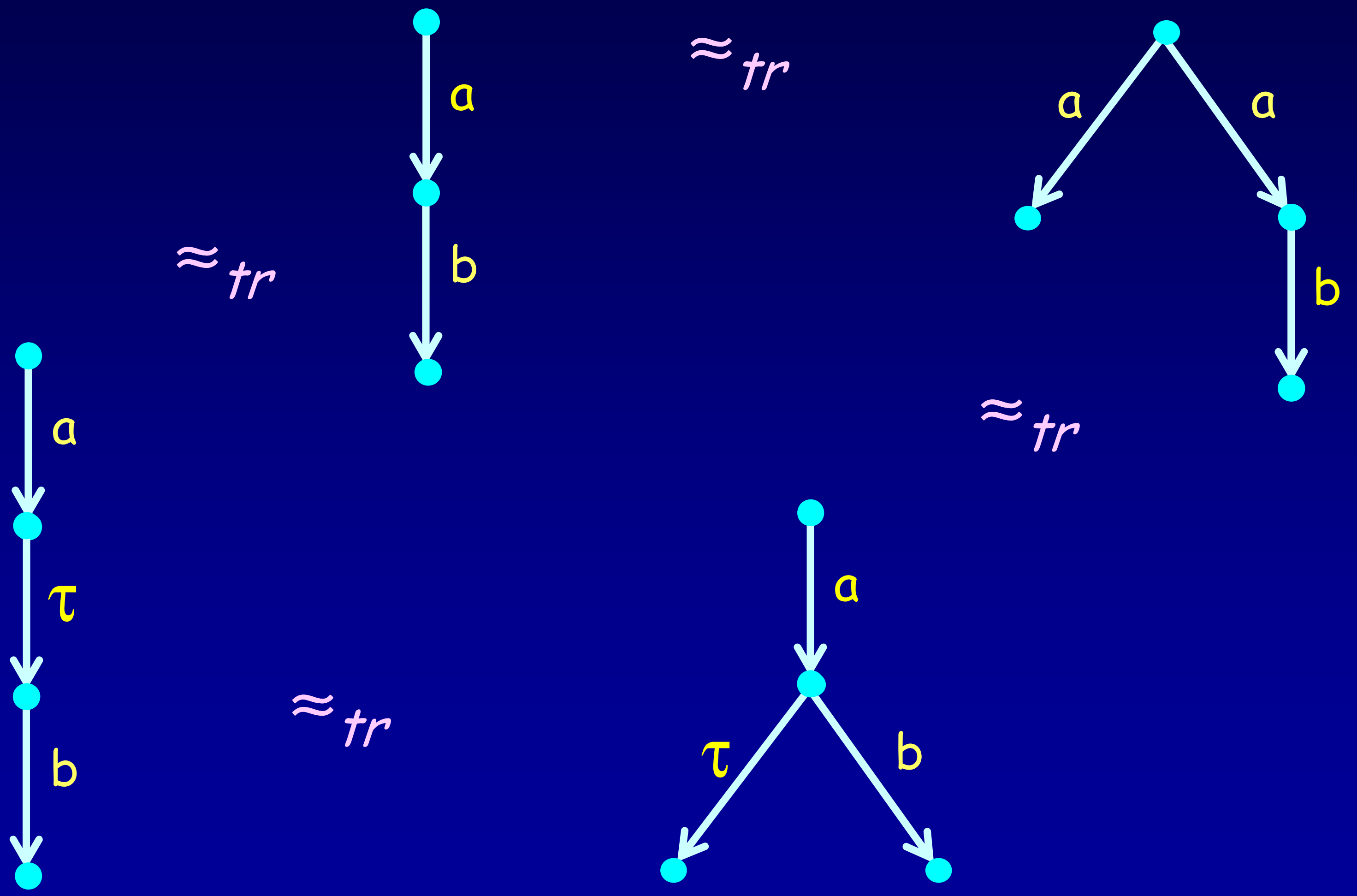
Trace Equivalence



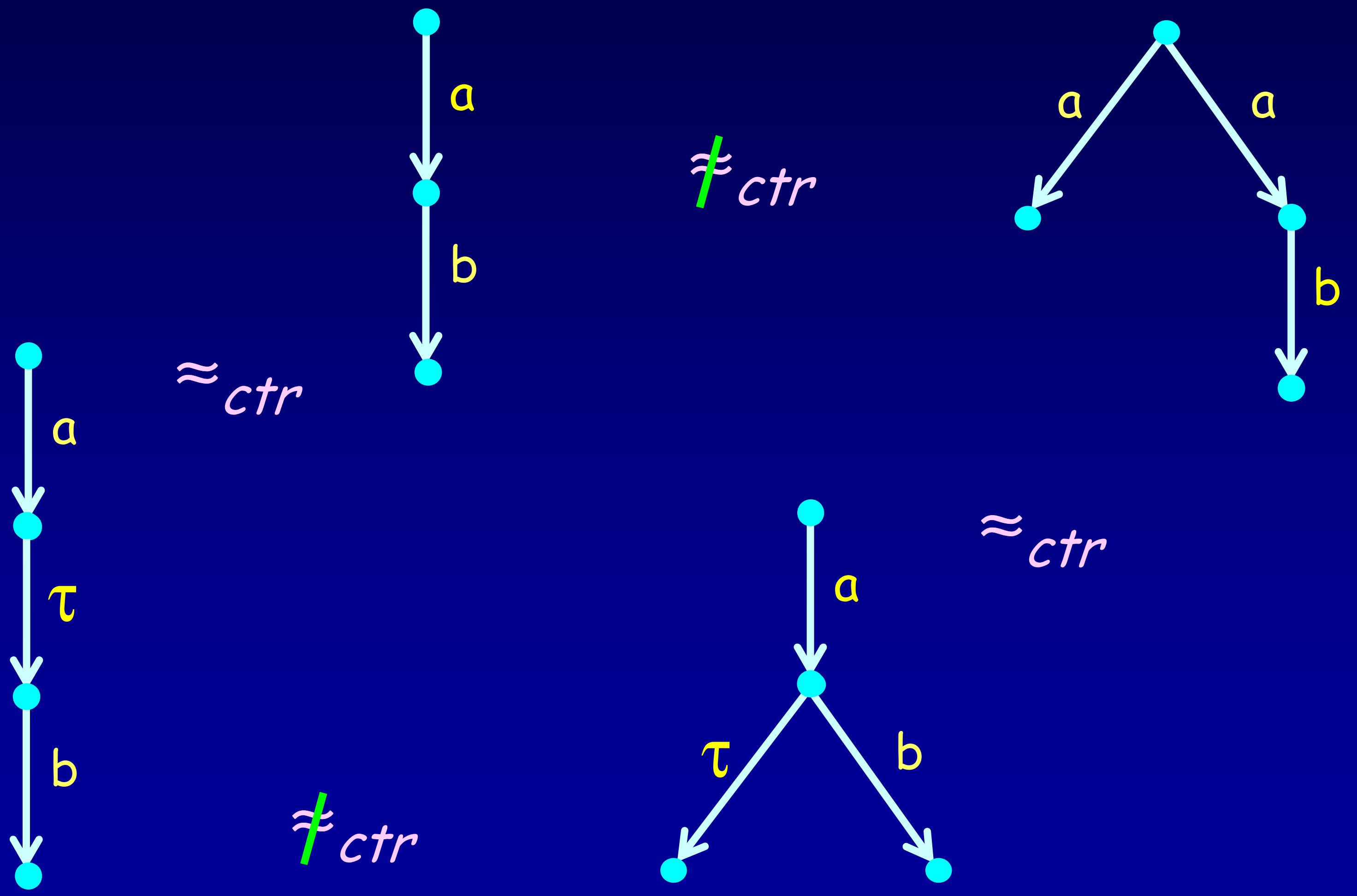
$$S1 \approx_{tr} S2 \quad \Leftrightarrow \quad traces(S1) = traces(S2)$$

Traces: $$traces(s) = \{ \sigma \in L^* \mid s \stackrel{\sigma}{\Longrightarrow} \}$$

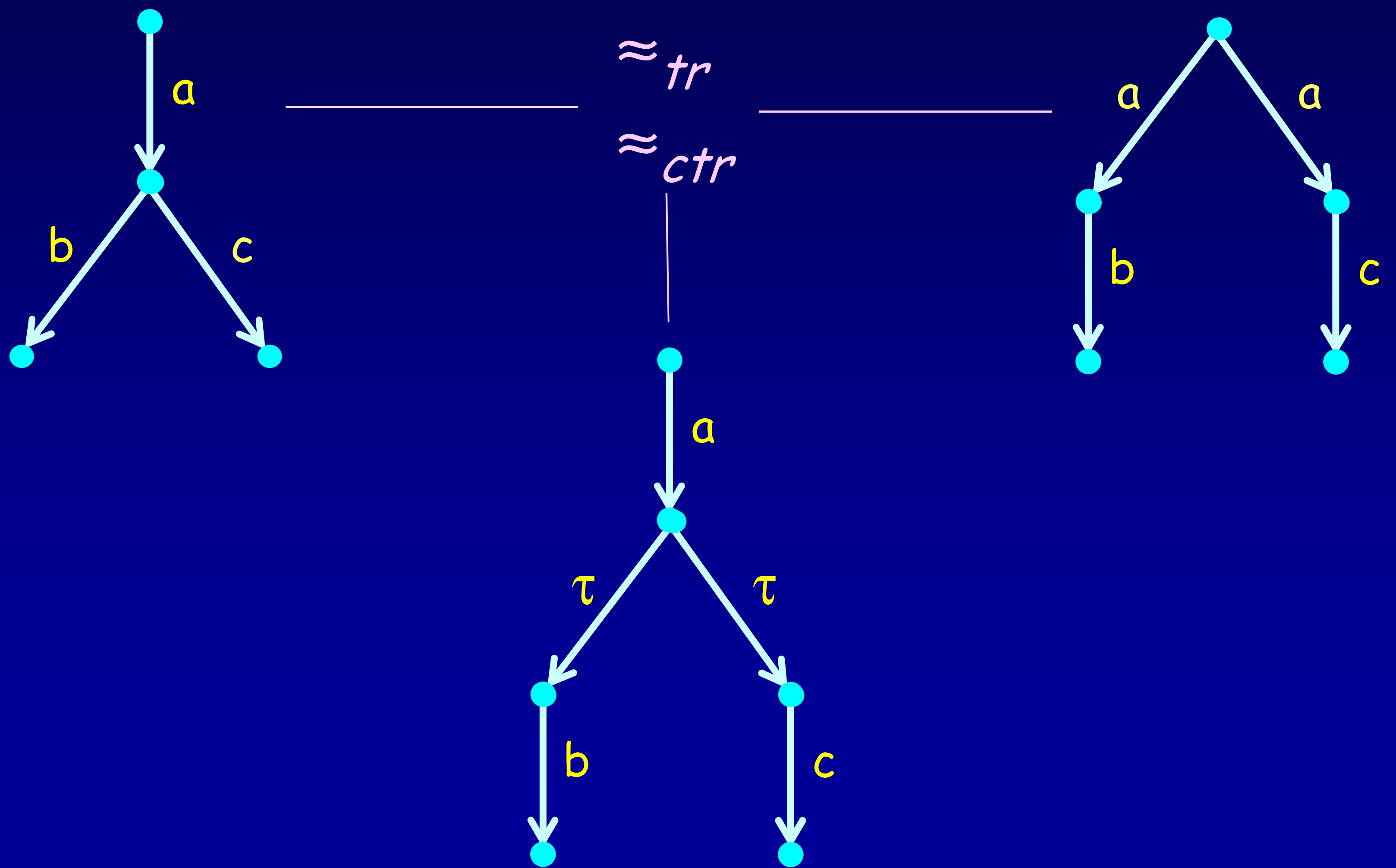
Trace Equivalence



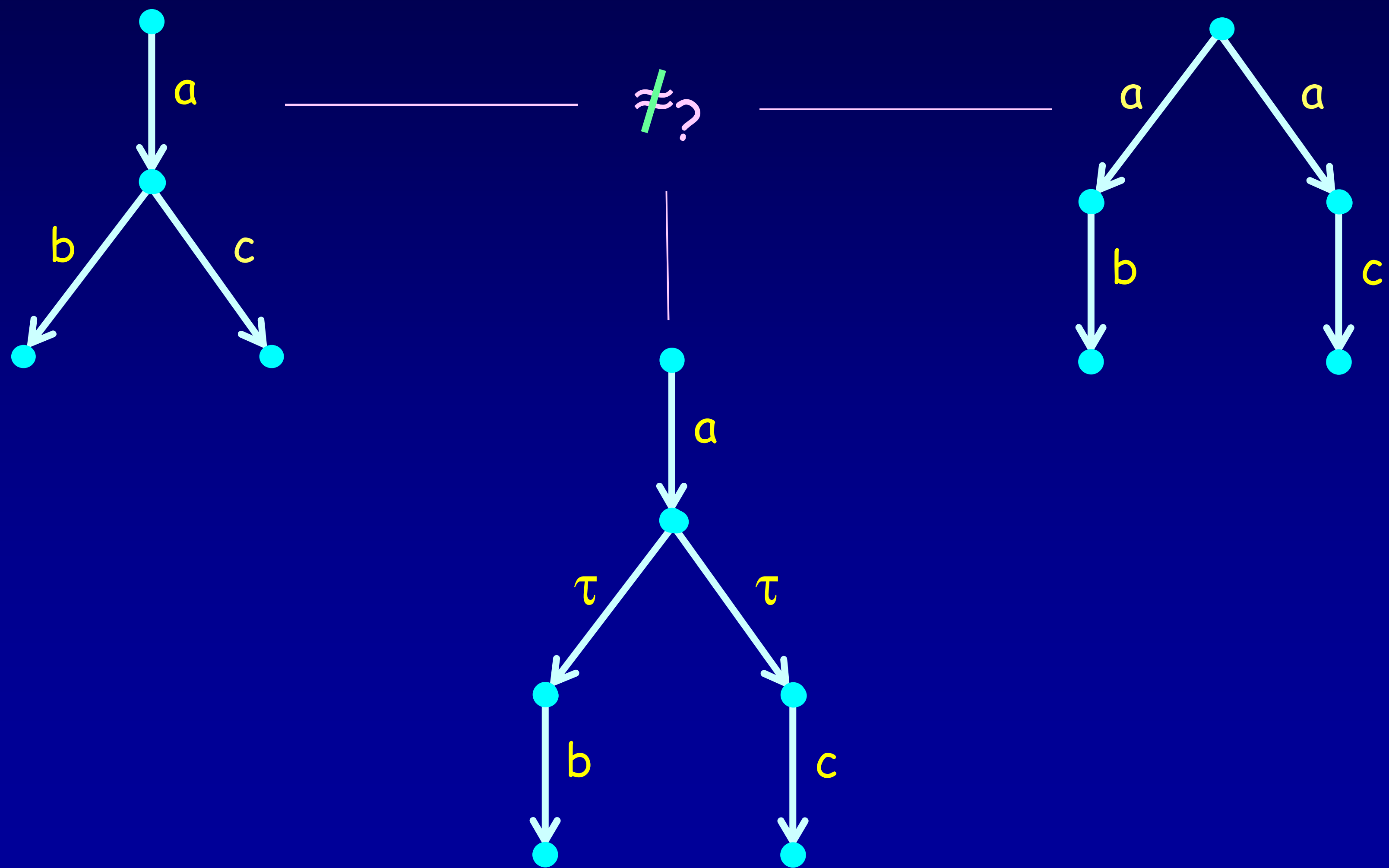
Completed Trace Equivalence



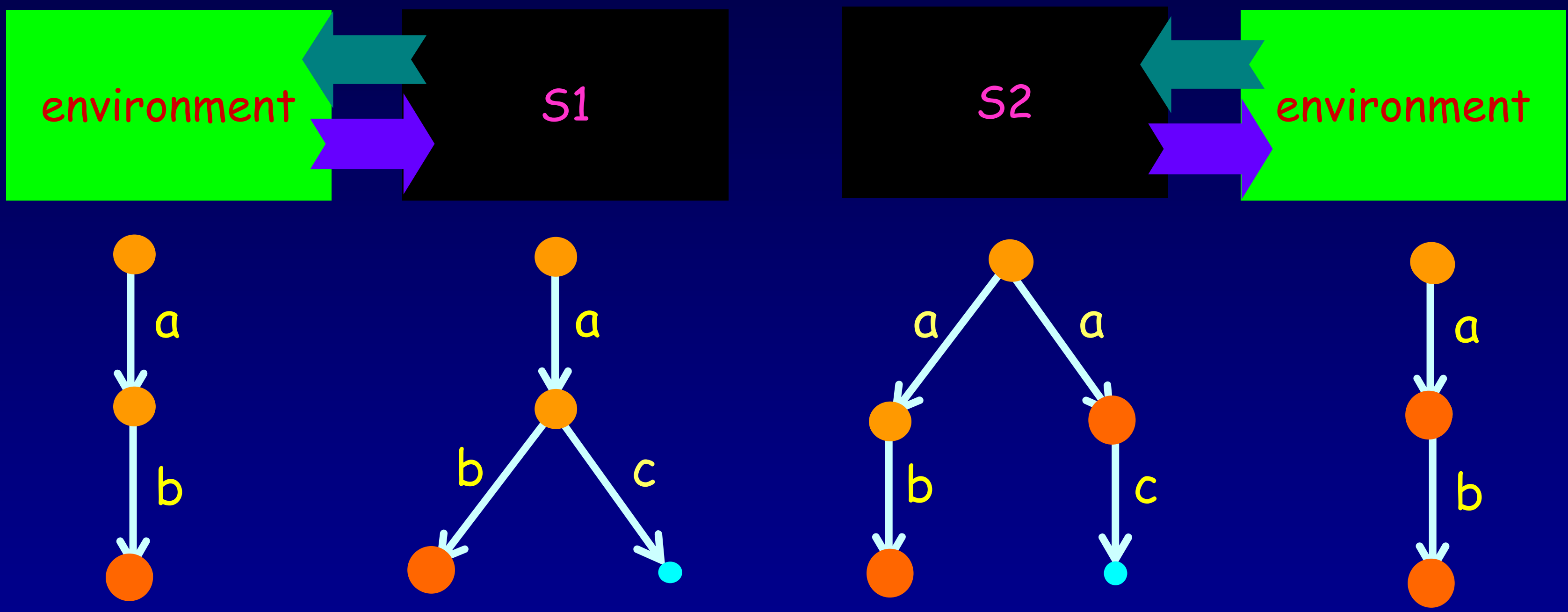
Completed Trace Equivalence



(Completed) Trace Equivalence : Others ?



Comparing Systems : Testing Equivalence



a b √

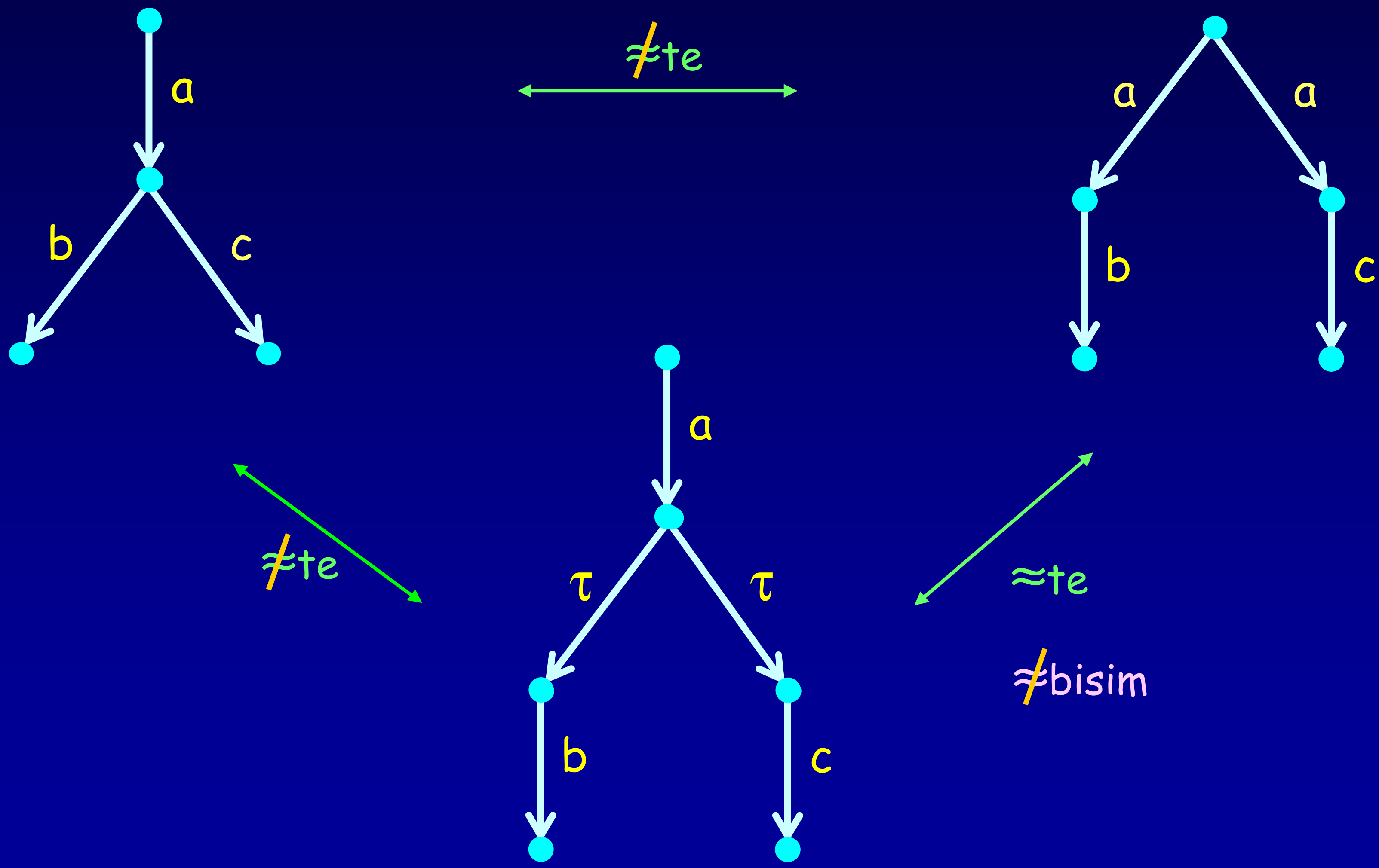
≉te

a b √
a √

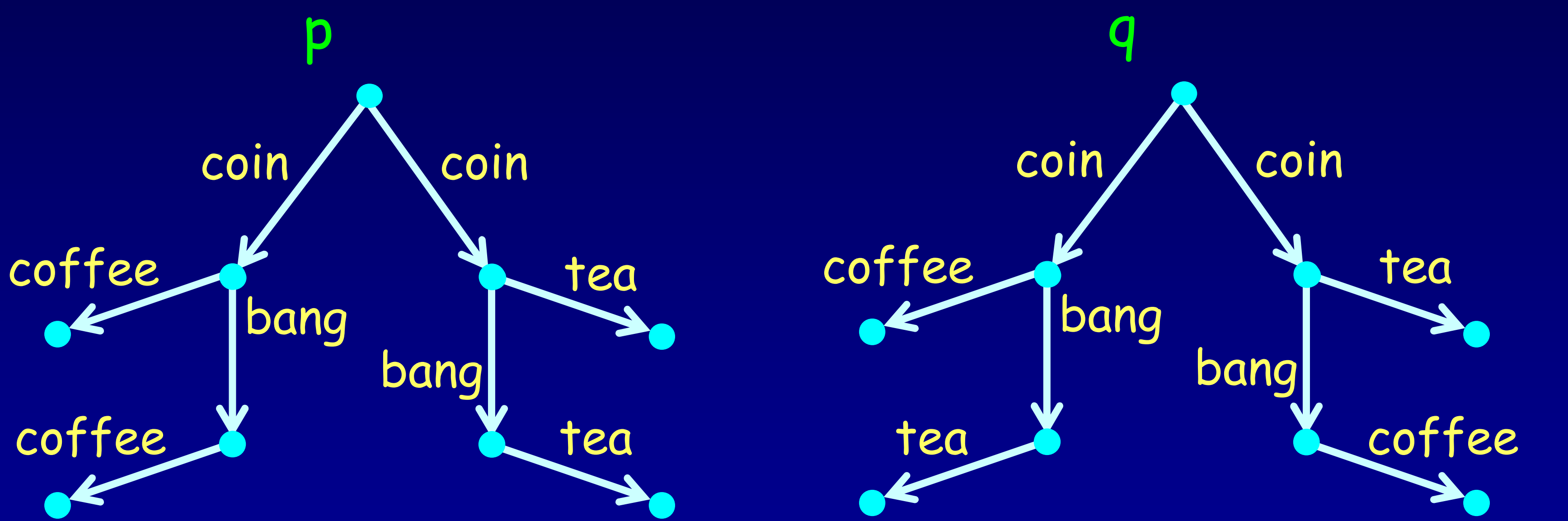
S1 after ~~a refuses~~ {b}

S2 after a refuses {b}

Testing Equivalence



Testing Equivalence

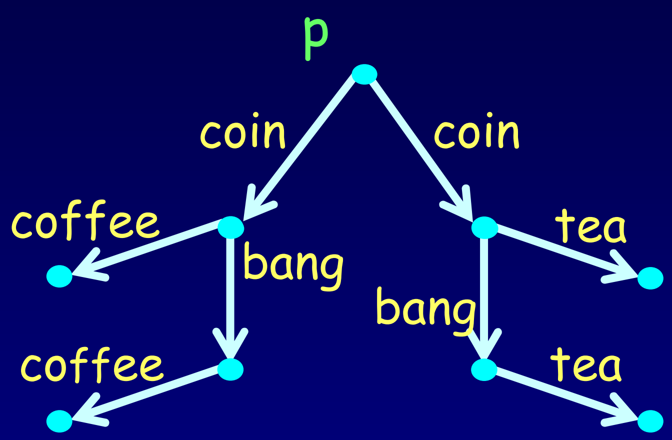


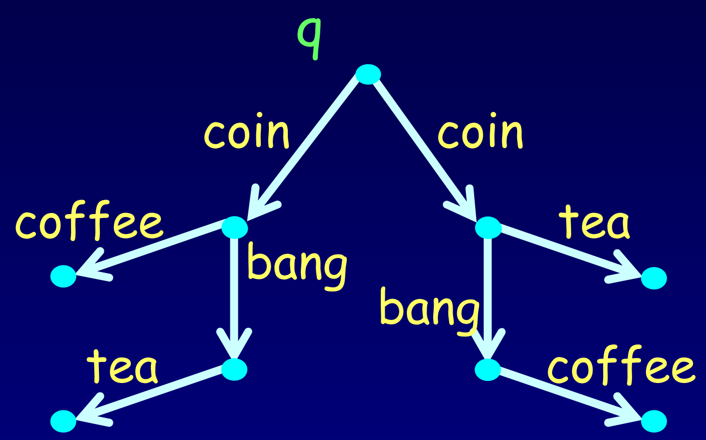
p $\approx_{te}$ q

But:

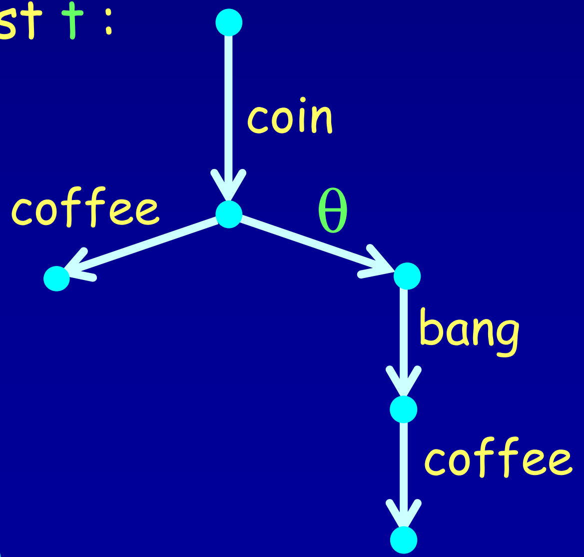
if you want coffee you will eventually always succeed in q but not p !?

Refusal Equivalence





Test t :



θ only possible
if nothing else is possible

coin θ bang coffee √ $\notin$ obs (p || t)

coin θ bang coffee √ $\in$ obs (q || t)

$p \not\approx_{rf} q$

Equivalences on Transition Systems

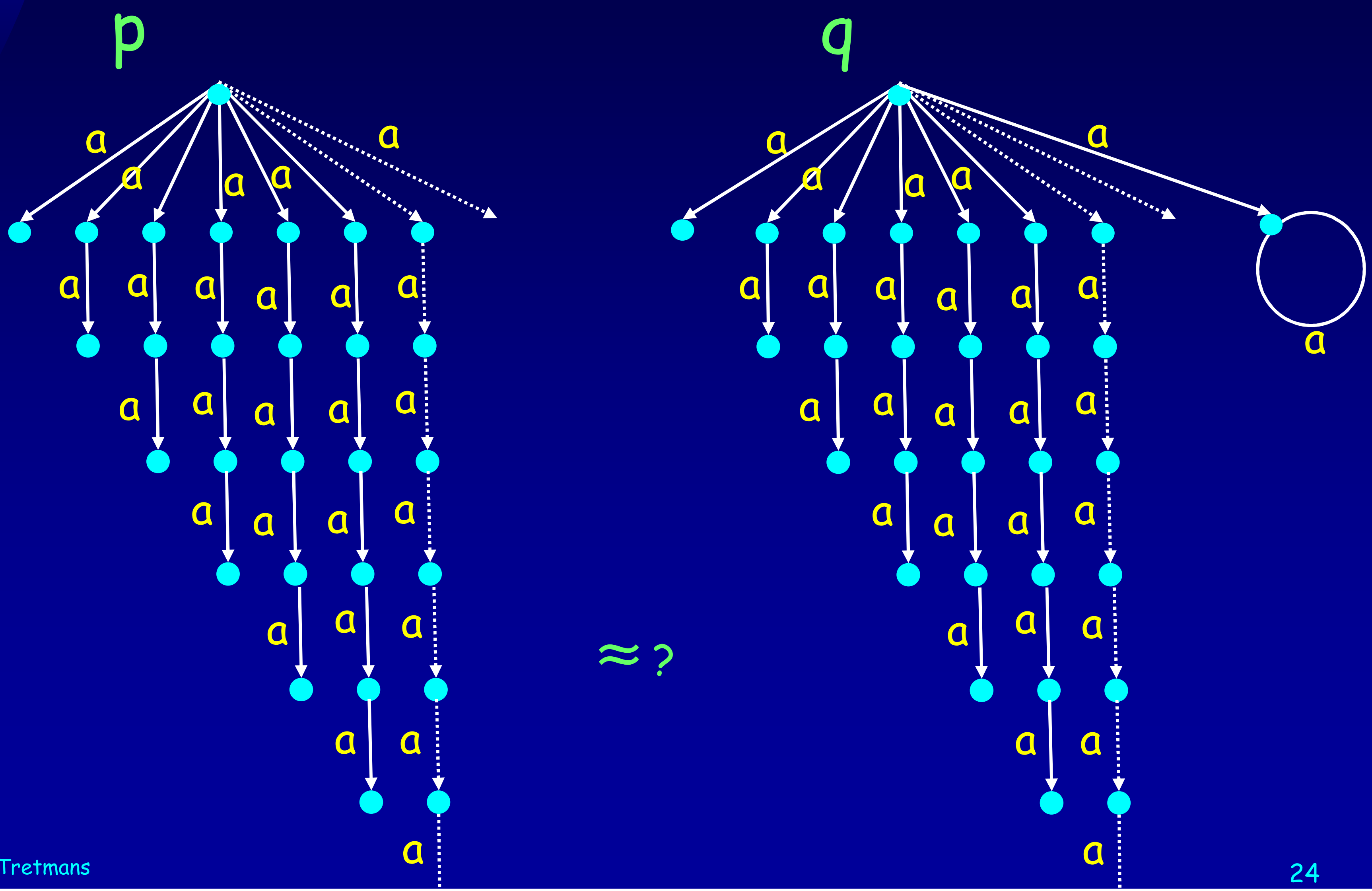
strong

↕

weak

isomorphism	now you need to observe τ's ……
bisimulation (weak)	test an LTS with another LTS, and undo, copy, repeat as often as you like
failure trace = refusal	test an LTS with another LTS, and try again (continue) after failure
failures = testing	test an LTS with another LTS
completed trace	observing sequences of actions and their end
trace	observing sequences of actions

Equivalences : Examples



Non-Equivalence Relations on Labelled Transition Systems

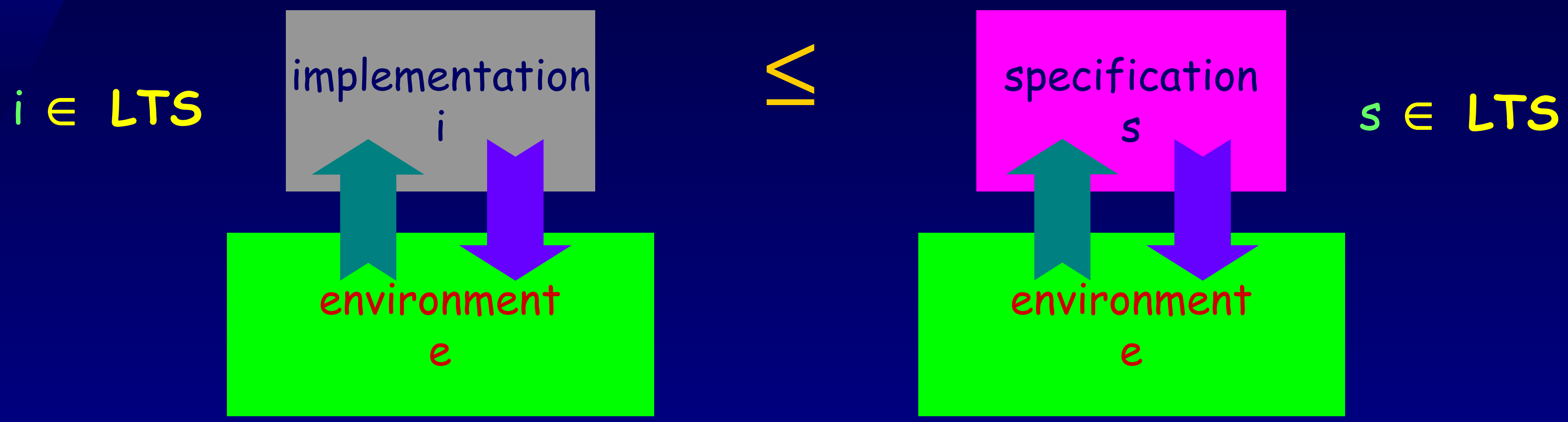
Implementation Relations

Conformance Relations

Refinement Relations

Pre-Orders

Preorders on Transition Systems

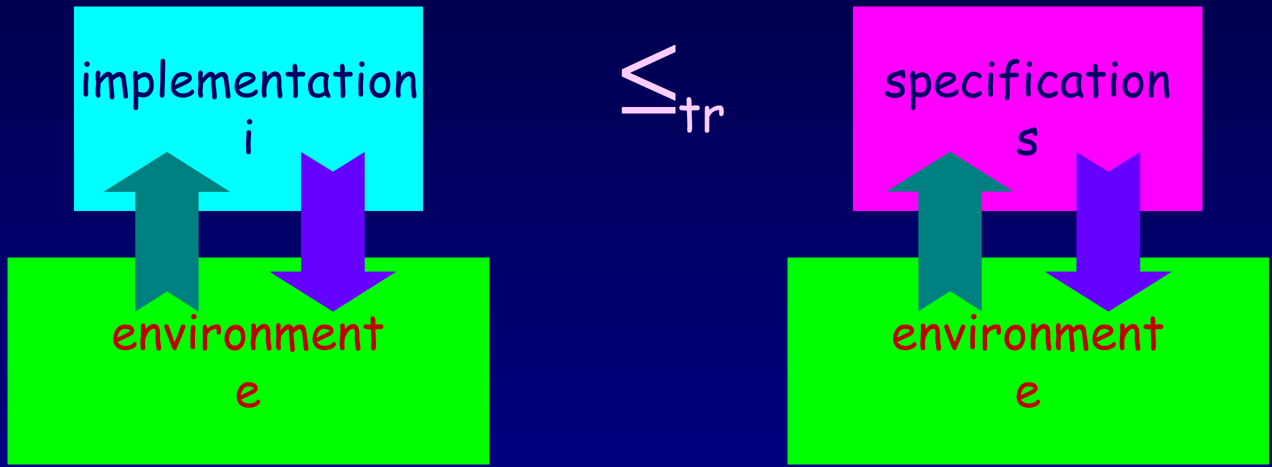


☞ Suppose an environment interacts with the black box implementation i and with the specification s :

- i correctly implements s

  if all observation of i can be related to observations of s

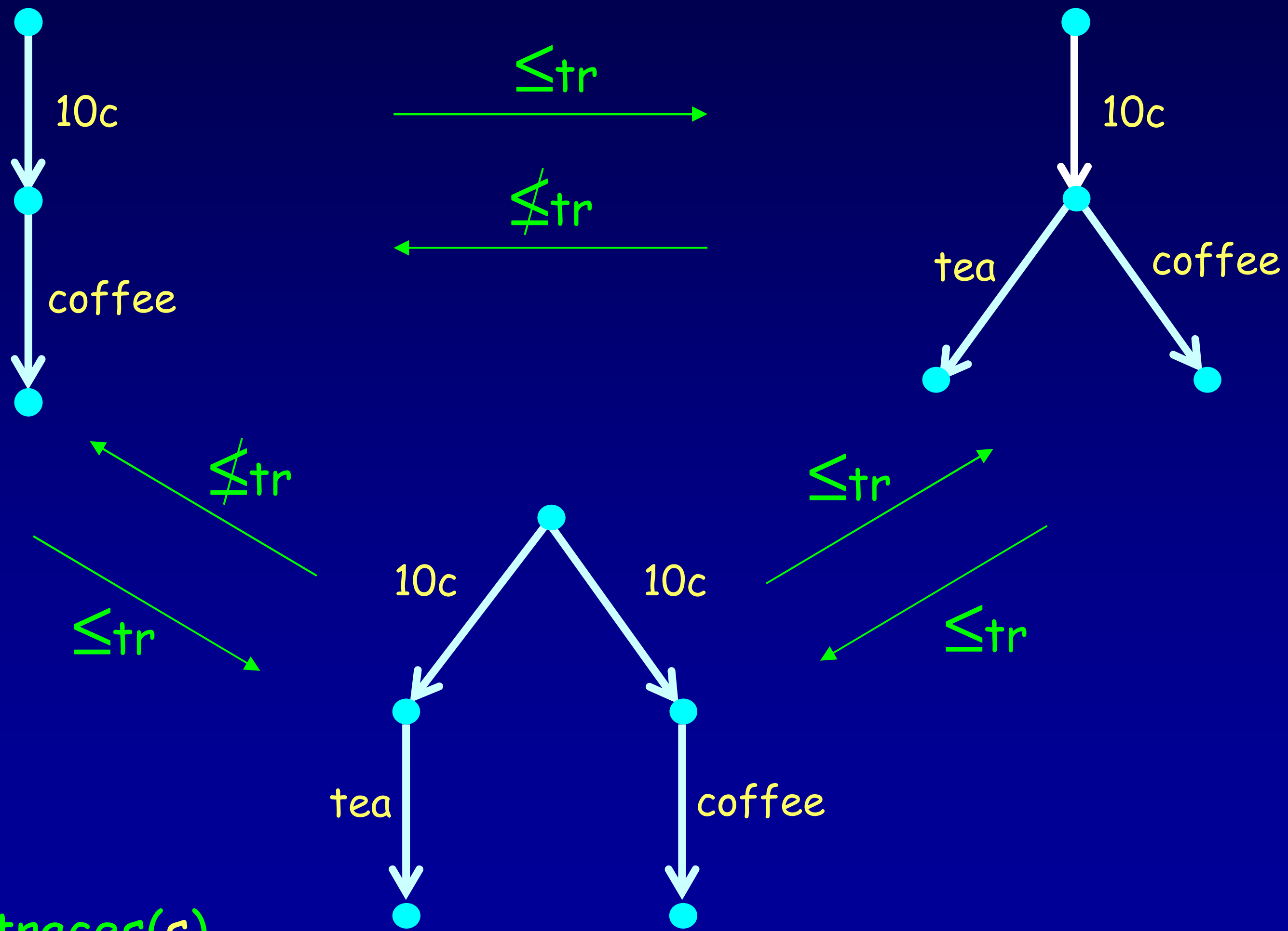
Trace Preorder



$$i \leq_{tr} s \quad \Leftrightarrow \quad traces(i) \subseteq traces(s)$$

Traces: $traces(s) = \{ \sigma \in L^* \mid s \stackrel{\sigma}{\Longrightarrow} \}$

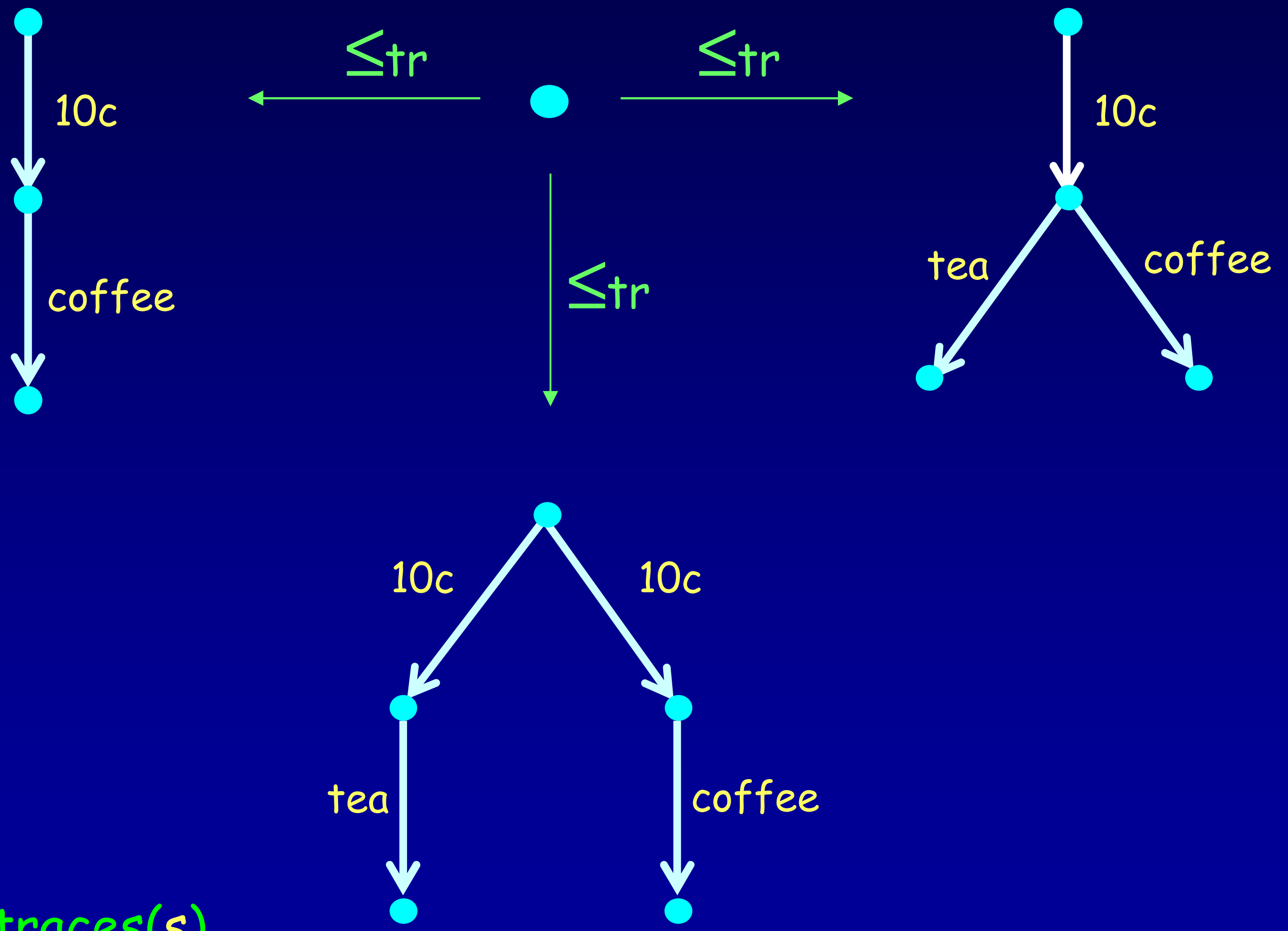
Trace Preorder



i ≤tr s =

traces(i) ⊆ traces(s)

Trace Preorder

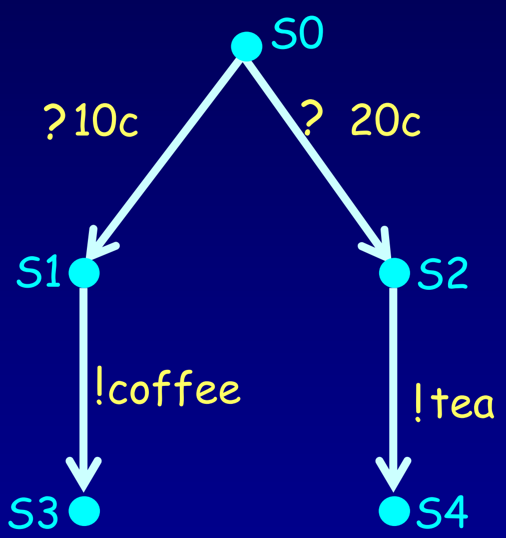


$i \leq_{tr} s$ =

traces(i) $\subseteq$ traces(s)

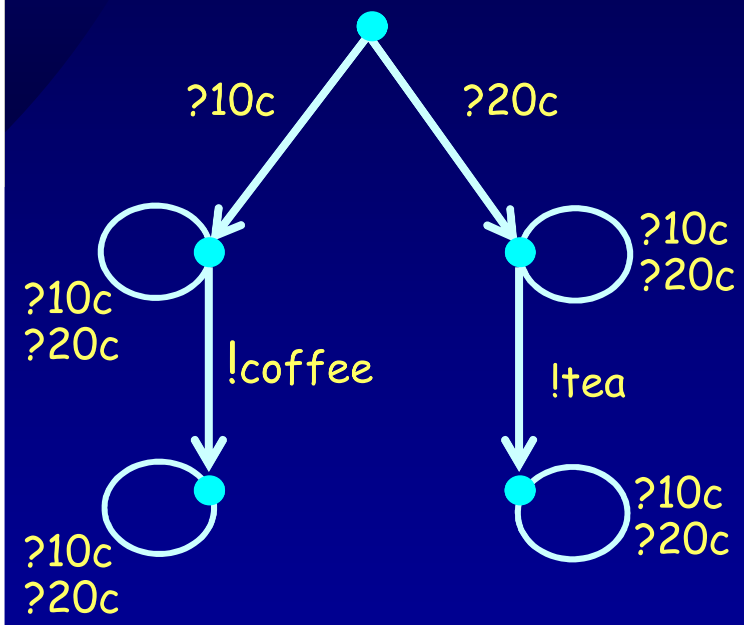
Implementation Relation ioco for Labelled Transition Systems with Inputs and Outputs

Input-Output Transition Systems



	10c, 20c	coffee, tea
	from user to machine initiative with user machine cannot refuse	from machine to user initiative with machine user cannot refuse
	input L_I	output L_U
L_I = { ?10c, ?20c } L_U = { !coffee, !tea }	$L_I \cap L_U = \emptyset$	$L_I \cup L_U = L$

Input-Output Transition Systems



L_I = { ?10c, ?20c }

L_U = { !coffee, !tea }

Input-Output Transition Systems

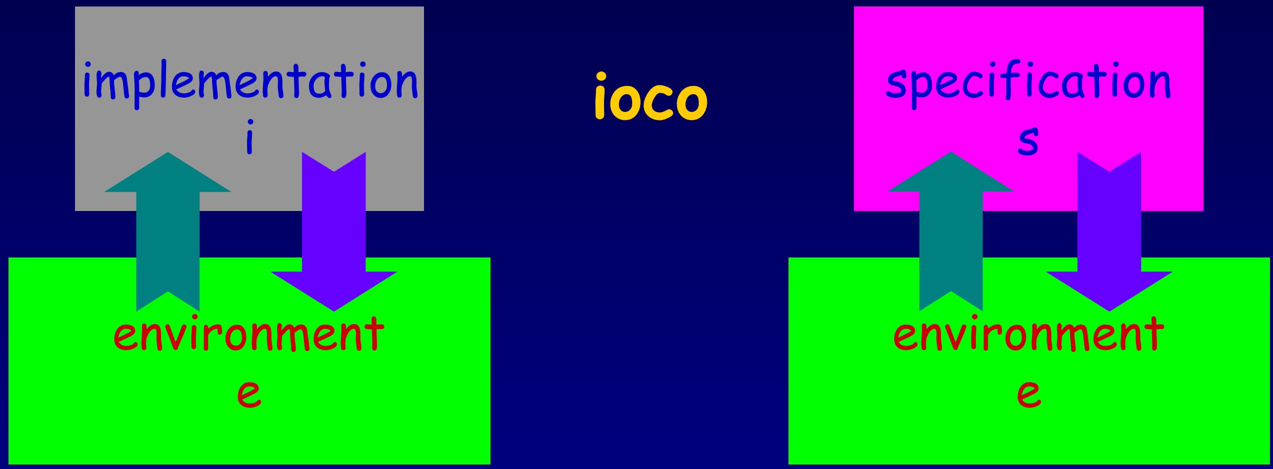
IOTS (L_I, L_U) $\subseteq$ LTS ($L_I \cup L_U$)

IOTS is LTS with Input-Output and always enabled inputs:

for all states s,

for all inputs $?a \in L_I$: $s \xRightarrow{?a}$

Input-Output Transition Systems with ioco



$i \in$ IOTS(L_I,L_U)

$s \in$ LTS(L_I,L_U)

ioco $\subseteq$ IOTS (L_I,L_U) $\times$ LTS (L_I,L_U)

Observing IOTS where system inputs
interact with environment outputs, and v.v.

Correctness

Implementation Relation **ioco**

$$i \ \mathbf{ioco} \ s \ =_{def} \ \forall \sigma \in \mathit{Straces}(s) : \ \mathit{out}(i \ \mathbf{after} \ \sigma) \subseteq \mathit{out}(s \ \mathbf{after} \ \sigma)$$

$$p \xrightarrow{\delta} p \quad = \quad \forall \, !x \in L_U \cup \{\tau\} . \ p \overset{!x}{\not\longrightarrow}$$

$$\mathit{Straces}(s) \quad = \quad \{ \ \sigma \in (L \cup \{\delta\})^* \ | \ s \overset{\sigma}{\Longrightarrow} \ \}$$

$$p \ \mathbf{after} \ \sigma \quad = \quad \{ \ p' \ | \ p \overset{\sigma}{\Longrightarrow} p' \ \}$$

$$\mathit{out}(P) \quad = \quad \{ \, !x \in L_U \ | \ p \xrightarrow{!x} , \ p \in P \} \cup \{ \ \delta \ | \ p \xrightarrow{\delta} p, \ p \in P \ \}$$

Correctness

Implementation Relation **ioco**

i **ioco** s $=_{def}$ $\forall \sigma \in \mathit{Straces}(s) : \mathit{out}(i \textbf{ after } \sigma) \subseteq \mathit{out}(s \textbf{ after } \sigma)$

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)

Correctness

Implementation Relation **ioco**

$$i \;\mathbf{ioco}\; s \;=_{def}\; \forall \sigma \in \mathit{Straces}(s) : \mathit{out}(i \;\mathbf{after}\; \sigma) \subseteq \mathit{out}(s \;\mathbf{after}\; \sigma)$$

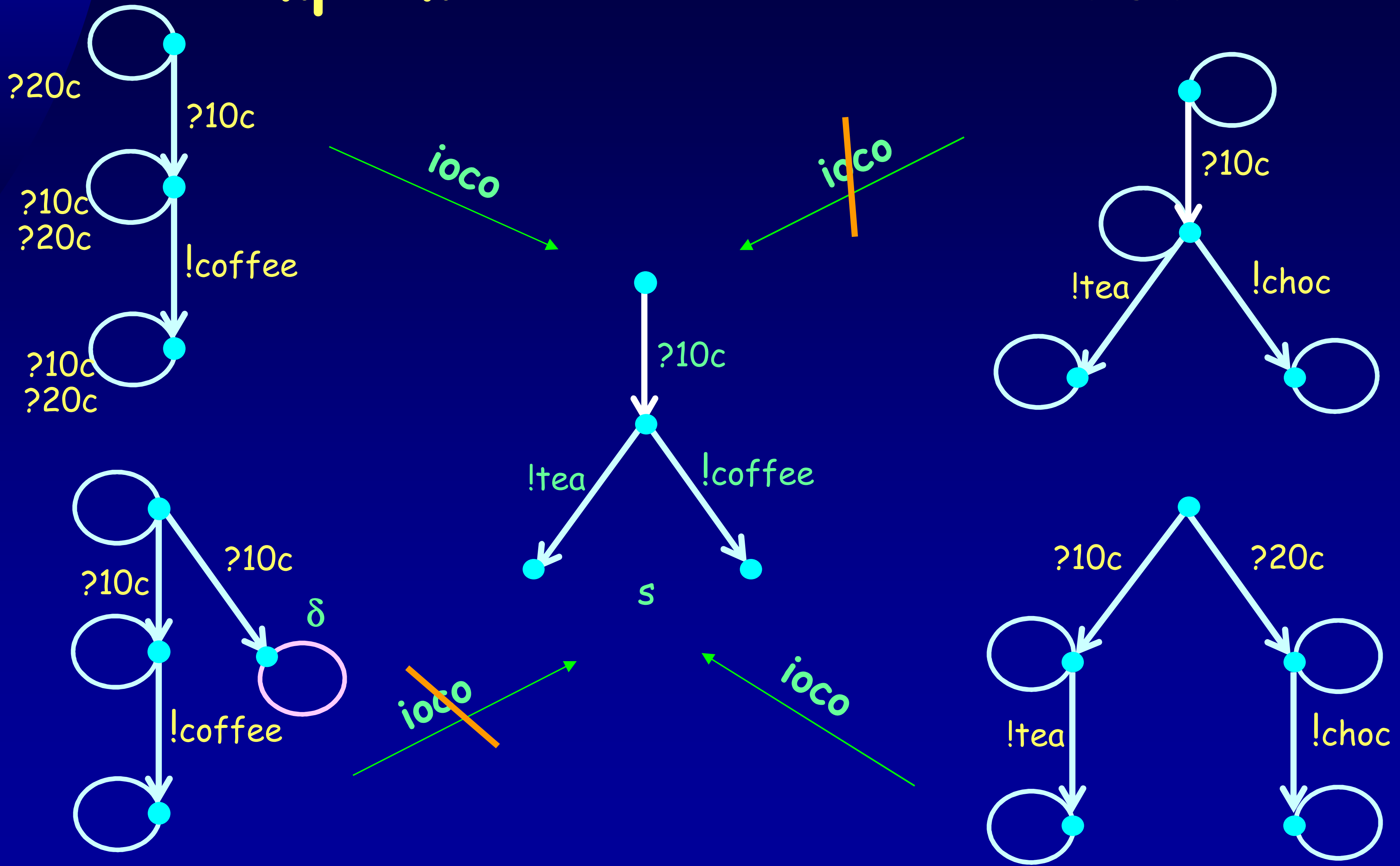
$$p \xrightarrow{\delta} p \quad = \quad \forall\, !x \in L_U \cup \{\tau\} .\; p \stackrel{!x}{\not\longrightarrow}$$

$$\mathit{Straces}(s) \quad = \quad \{\; \sigma \in (L \cup \{\delta\})^* \mid s \stackrel{\sigma}{\Longrightarrow} \;\}$$

$$p \;\mathbf{after}\; \sigma \quad = \quad \{\; p' \mid p \stackrel{\sigma}{\Longrightarrow} p' \;\}$$

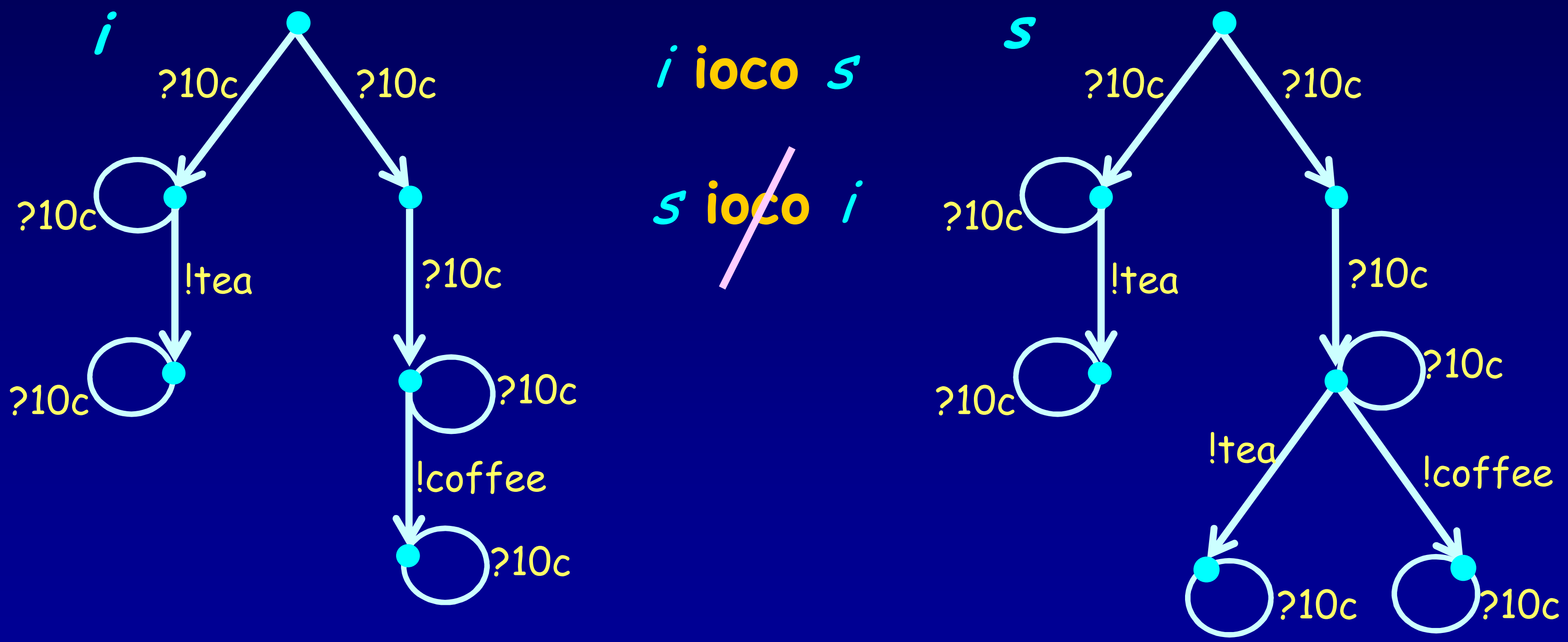
$$\mathit{out}(P) \quad = \quad \{\, !x \in L_U \mid p \xrightarrow{!x},\; p \in P \,\} \cup \{\, \delta \mid p \xrightarrow{\delta} p,\; p \in P \,\}$$

Implementation Relation ioco



Implementation Relation ioco

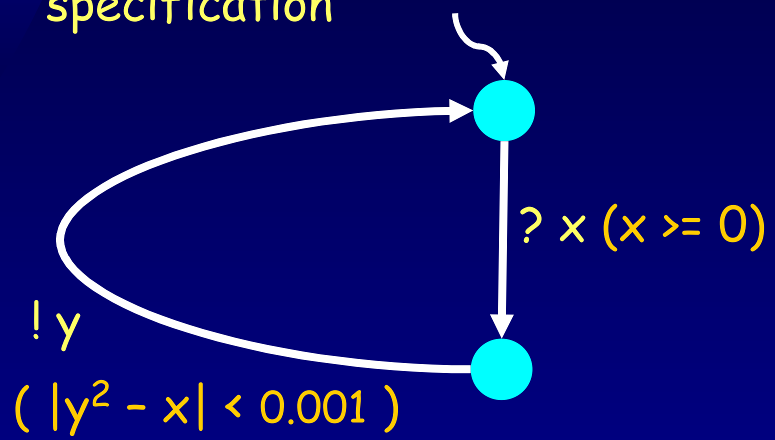
i **ioco** s $=_{def}$ $\forall \sigma \in$ *Straces* (s) : *out* (i **after** σ) $\subseteq$ *out* (s **after** σ)

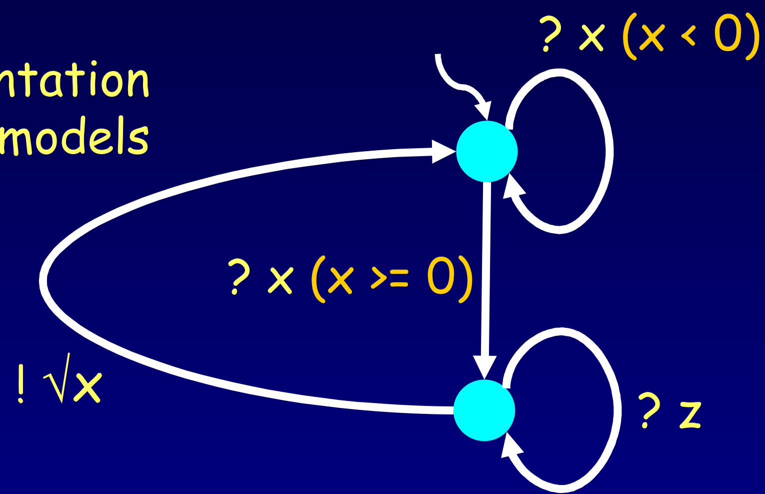


out (i **after** ?10c.?10c) = *out* (s **after** ?10c.?10c) = { !tea, !coffee }

out (i **after** ?10c.δ.?10c) = { !coffee } $\neq$ *out* (s **after** ?10c.δ.?10c) = { !tea, !coffee }

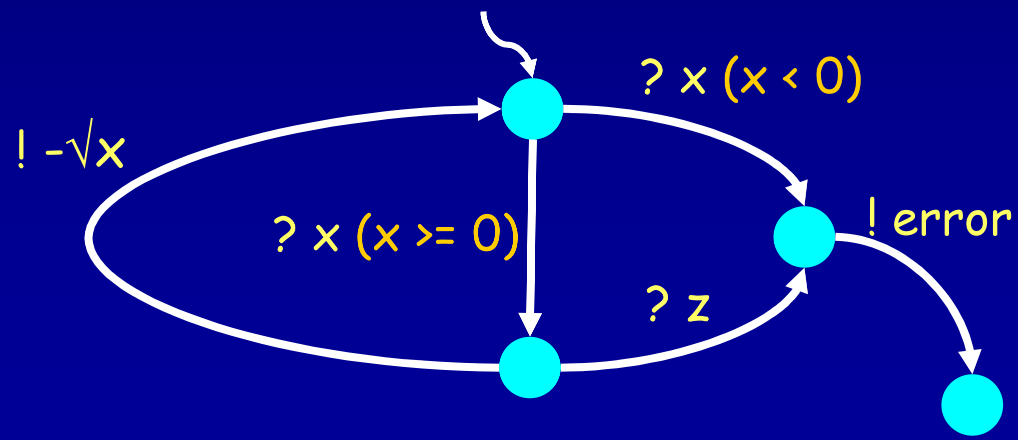
Implementation Relation ioco



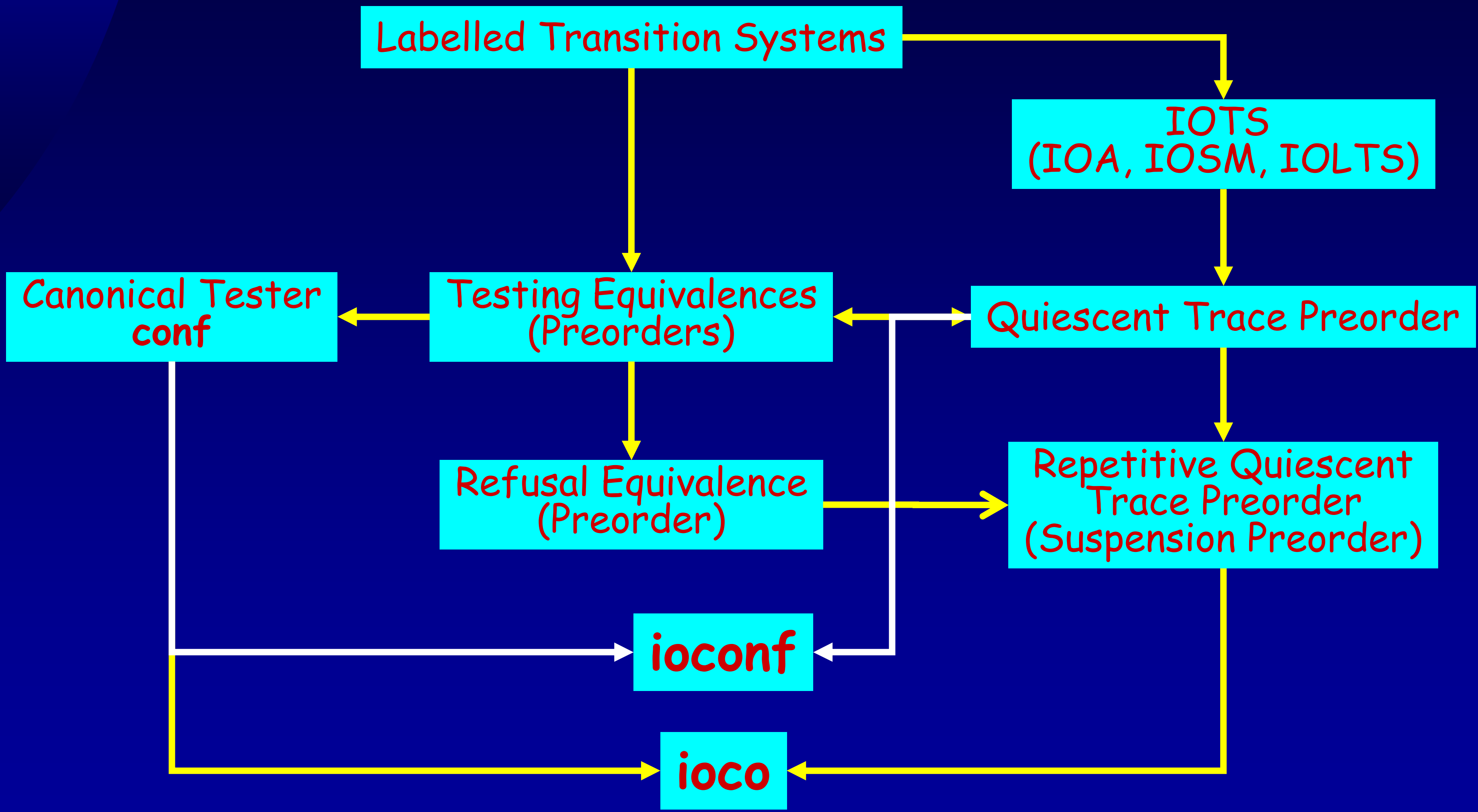


LTS and ioco allow:

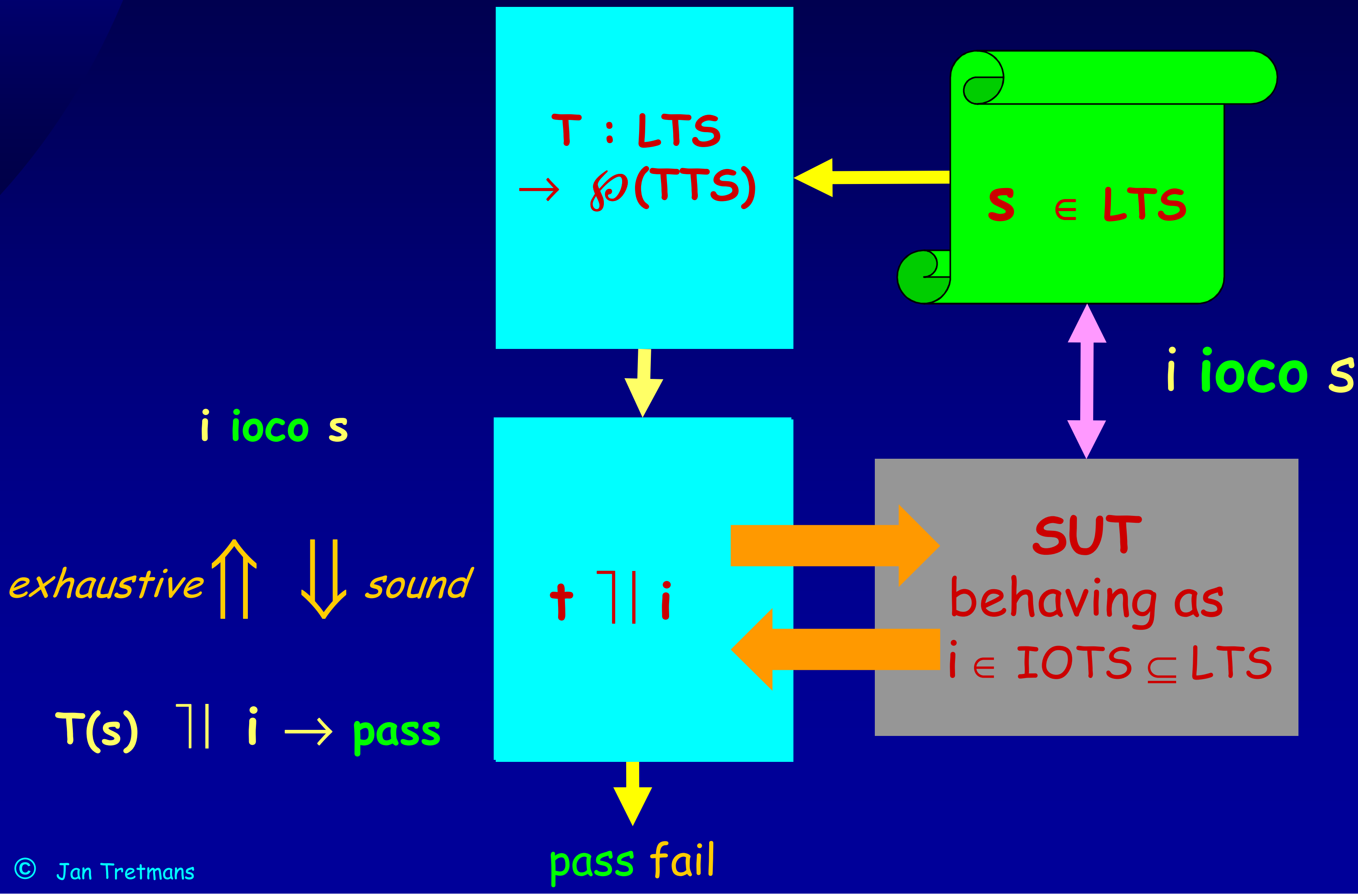
- non-determinism
- under-specification
- the specification of properties rather than construction



Genealogy of ioco



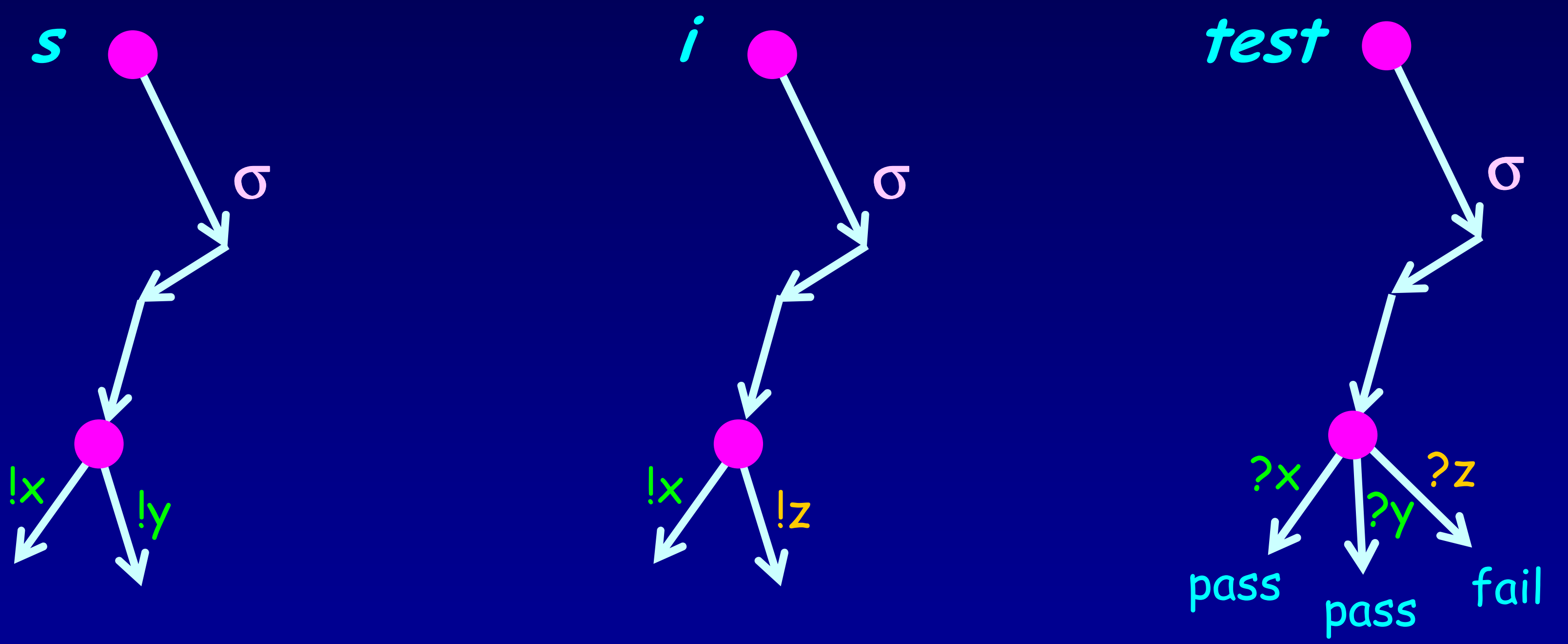
Model Based Testing with Transition Systems



Test Cases, Test Generation, and Test Execution for Labelled Transition Systems

Test Generation

i **ioco** s $=_{def}$ $\forall \sigma \in$ *Straces* (s) : *out* (i **after** σ) $\subseteq$ *out* (s **after** σ)



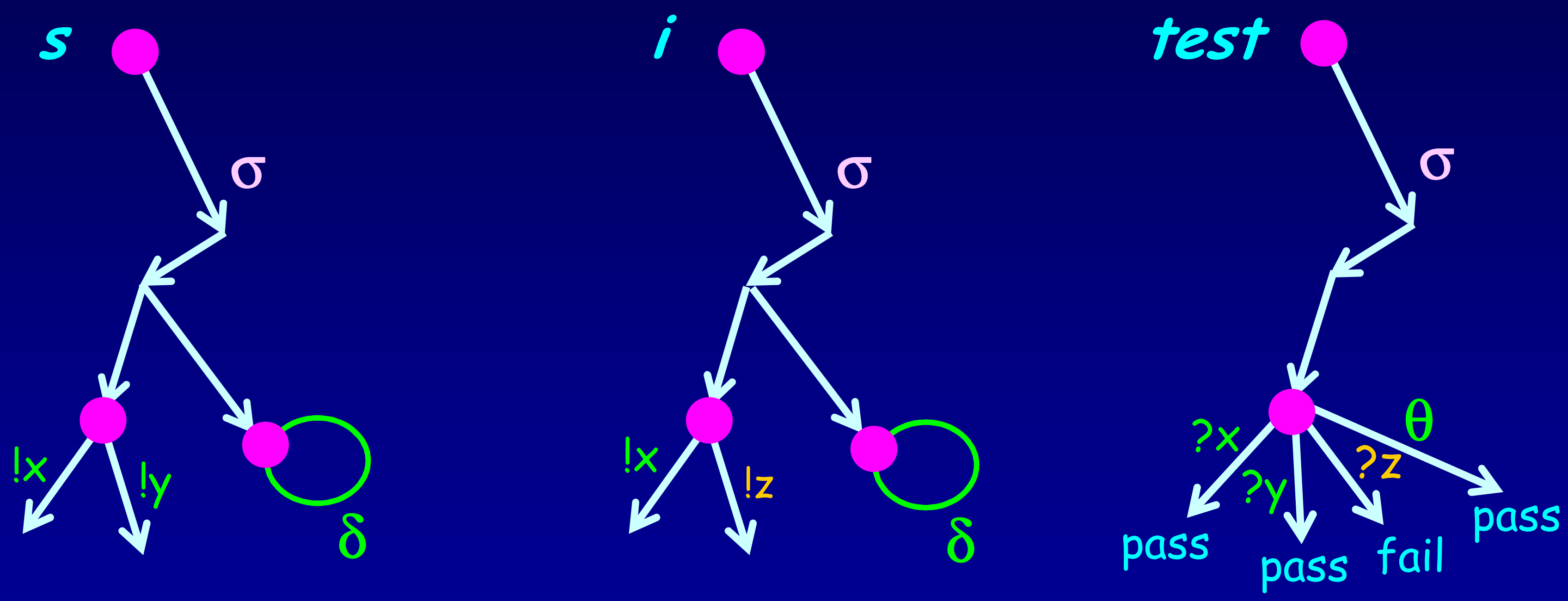
out (s **after** σ)
= { !x, !y }

out (i **after** σ)
= { !x, !z }

out (*test* **after** σ) = L_U

Test Generation

$i \ \textbf{ioco} \ s \ =_{def} \ \forall \sigma \in \mathit{Straces}(s) : \ \mathit{out}(i \ \textbf{after} \ \sigma) \subseteq \mathit{out}(s \ \textbf{after} \ \sigma)$



$\mathit{out}(s \ \textbf{after} \ \sigma)$
$= \{ !x, !y, \delta \}$

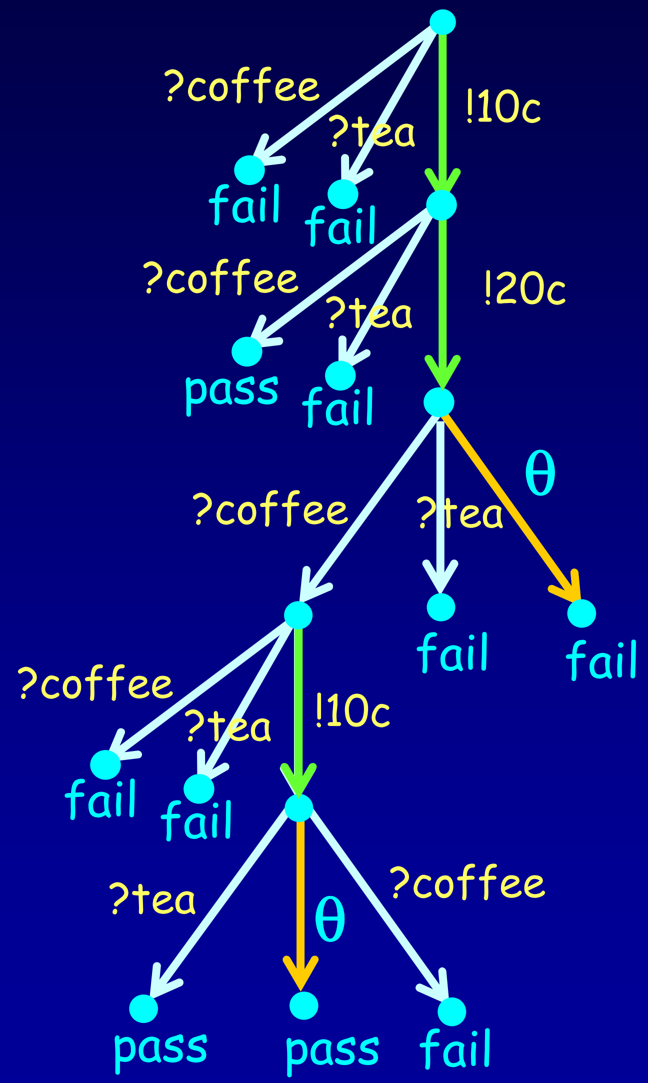
$\mathit{out}(i \ \textbf{after} \ \sigma)$
$= \{ !x, !z, \delta \}$

$\mathit{out}(\mathit{test} \ \textbf{after} \ \sigma)$
$= L_U \cup \{ \theta \}$

Test Cases

Model of a test case = transition system :

- labels in $L \cup \{\theta\}$
  - 'quiescence' label θ
- tree-structured
- 'finite', deterministic
- sink states **pass** and **fail**
- from each state:
  - either one input !a and all outputs ?x
  - or all outputs ?x and θ



Test Generation Algorithm

Algorithm

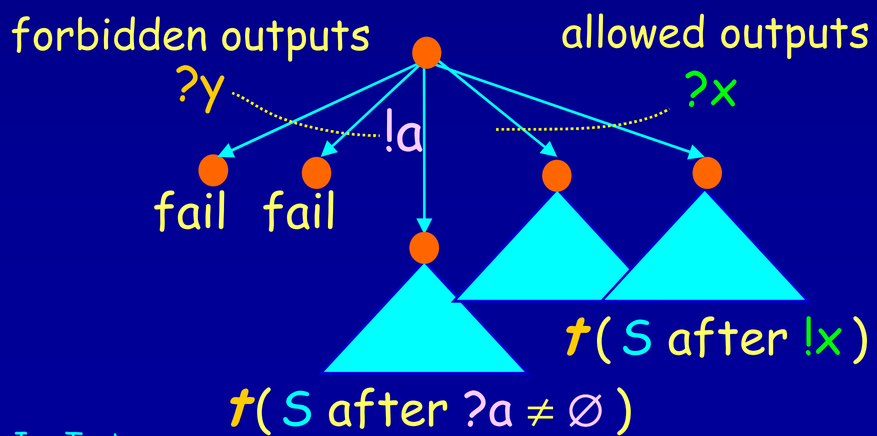
To generate a test case $t(S)$ from a transition system specification S, with $S \neq \emptyset$: set of states (initially $S = s_0$ after ε)

Apply the following steps recursively, non-deterministically:

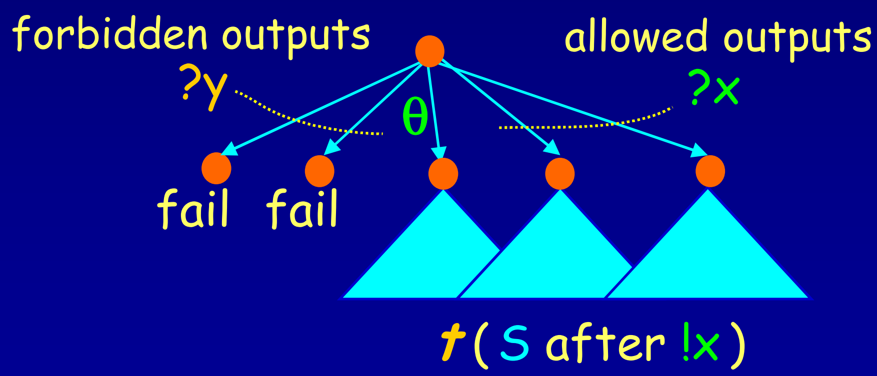
1 end test case

● pass

2 supply input !a



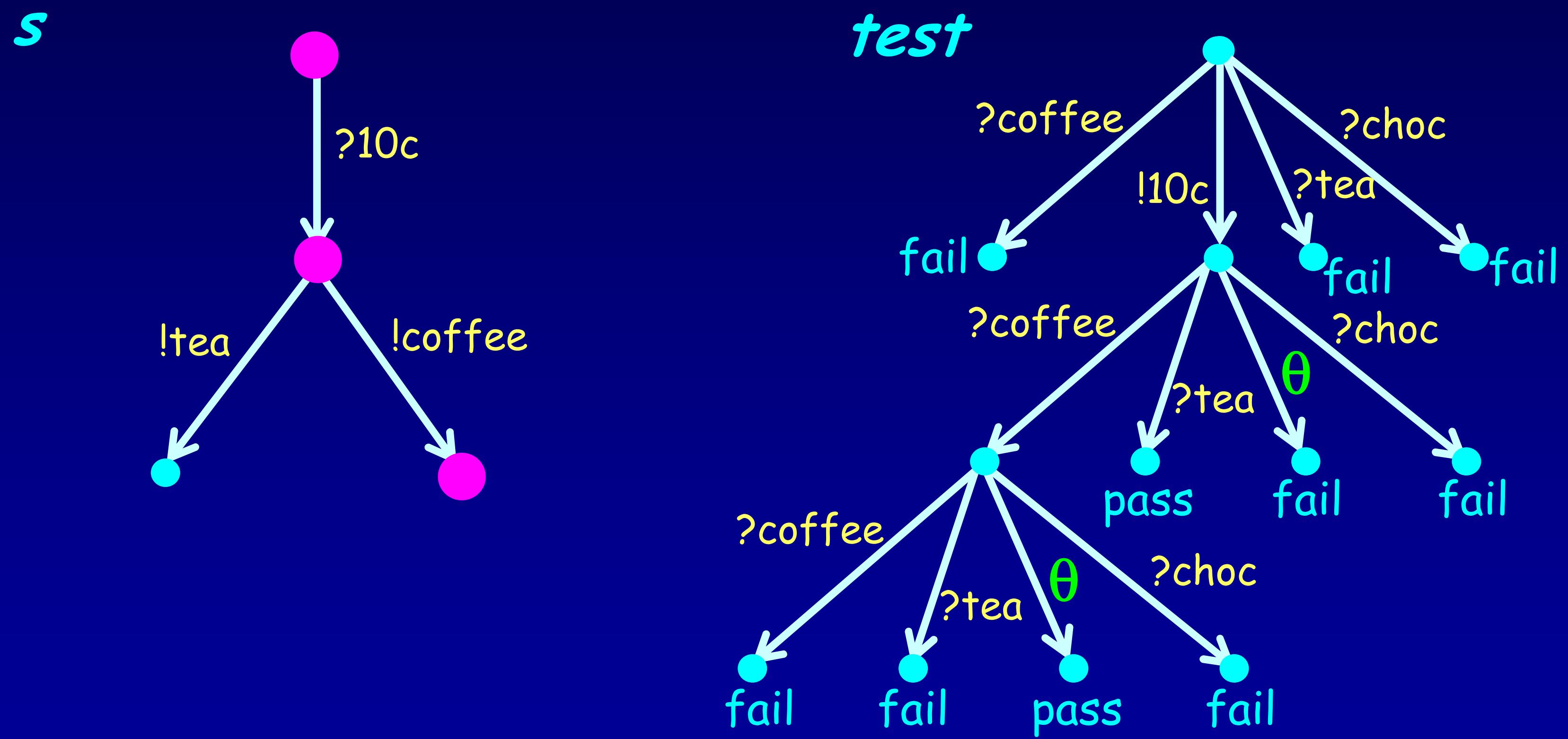
3 observe all outputs



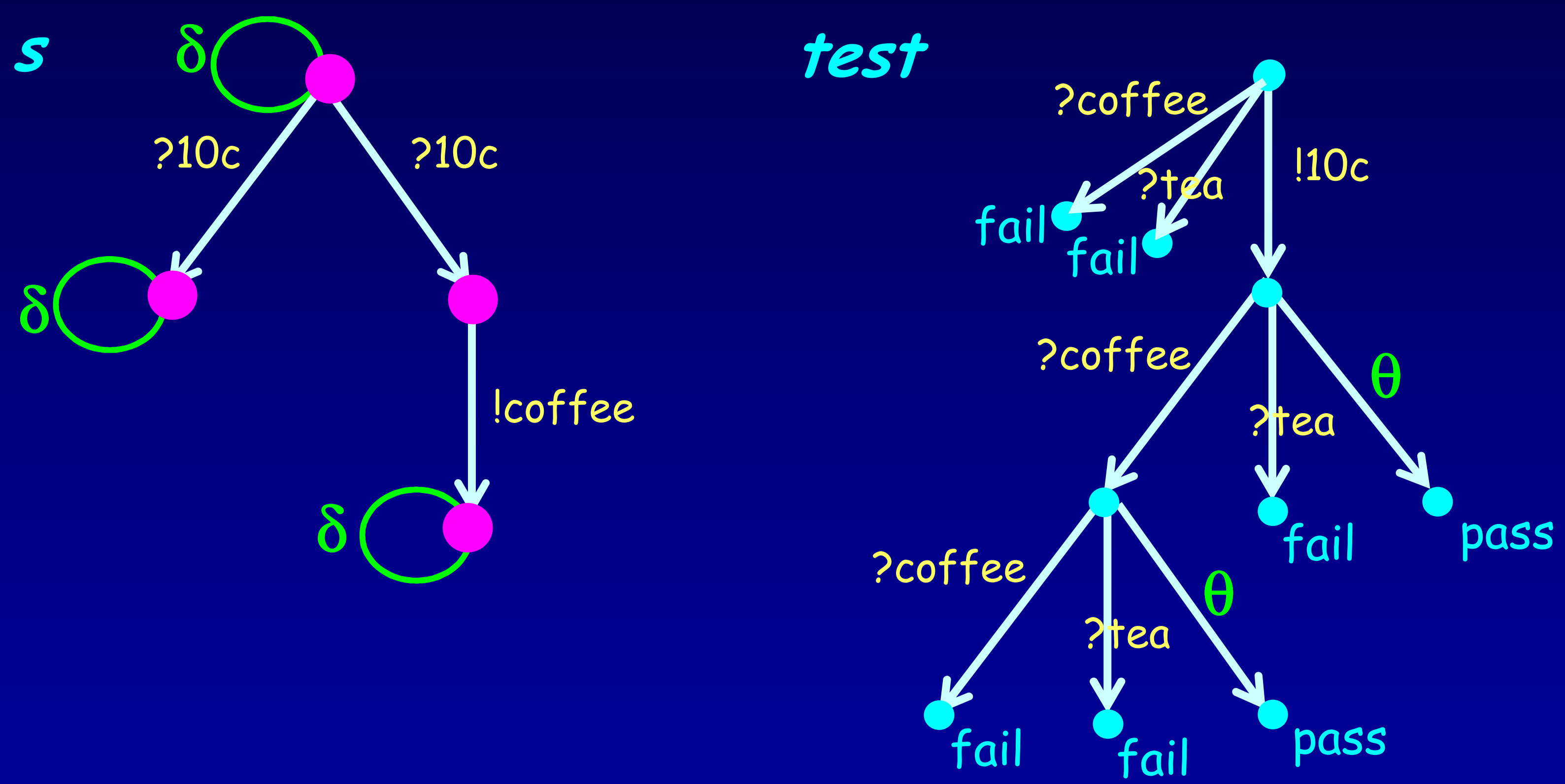
allowed outputs (or δ): $!x \in out(S)$

forbidden outputs (or δ): $!y \notin out(S)$

Test Generation Example

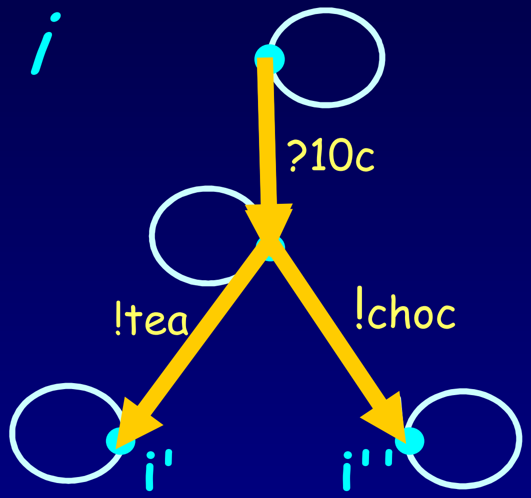


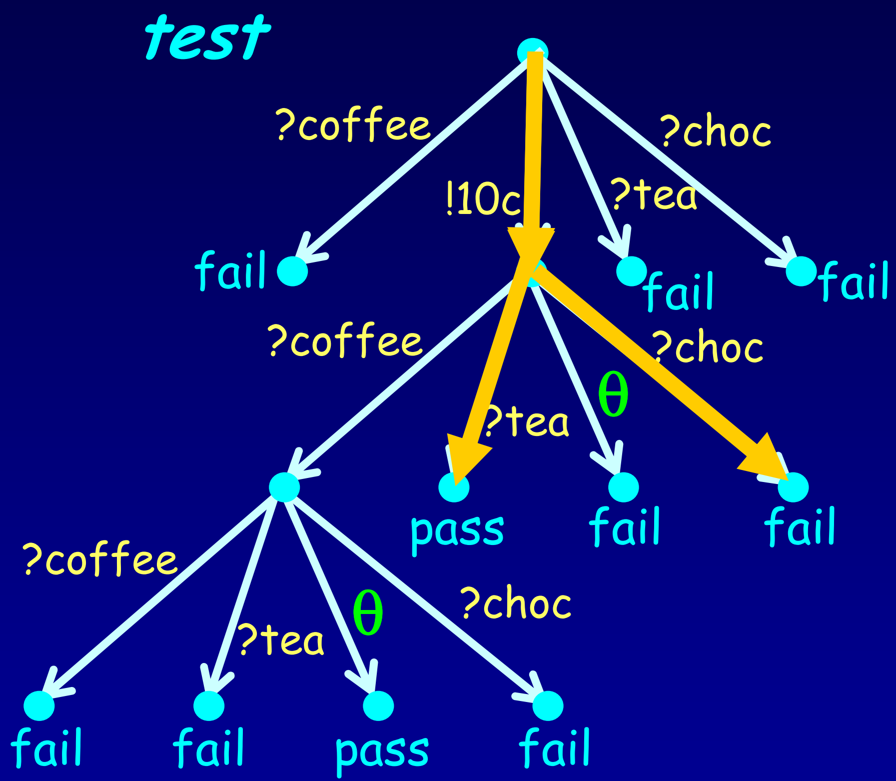
Test Generation Example



To cope with non-deterministic behaviour, tests are not linear traces, but trees

Test Execution Example





Two test runs :

$t \rceil\!| \, i \xRightarrow{10c \; tea} pass \rceil\!| \, i'$

$t \rceil\!| \, i \xRightarrow{10c \;\; choc} fail \rceil\!| \, i''$

i fails t

Test Execution

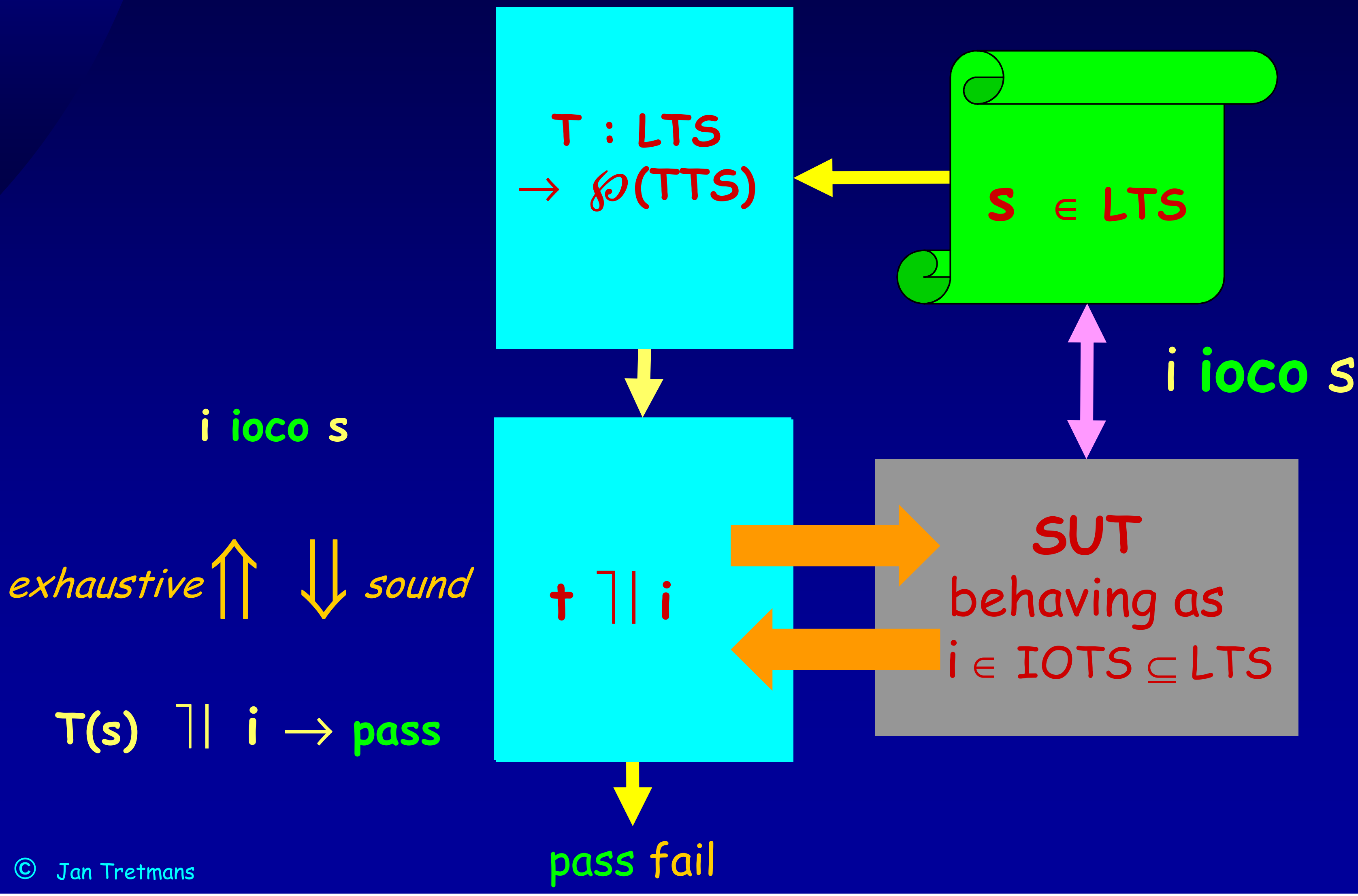
Test execution = all possible parallel executions (test runs) of test t with implementation i going to state pass or fail

Test run : $t \rceil| \, i \overset{\sigma}{\Longrightarrow} \text{pass} \rceil| \, i'$ or $t \rceil| \, i \overset{\sigma}{\Longrightarrow} \text{fail} \rceil| \, i'$

$$\frac{t \xrightarrow{a} t', \quad i \xrightarrow{a} i'}{t \rceil| \, i \xrightarrow{a} t' \rceil| \, i'} \qquad \frac{i \xrightarrow{\tau} i'}{t \rceil| \, i \xrightarrow{\tau} t \rceil| \, i'}$$

$$\frac{t \xrightarrow{\theta} t', \quad i \xrightarrow{\delta} i'}{t \rceil| \, i \xrightarrow{\theta} t' \rceil| \, i'}$$

Model Based Testing with Transition Systems



Soundness and Exhaustiveness

Validity of Test Generation

For every test t generated with algorithm we have:

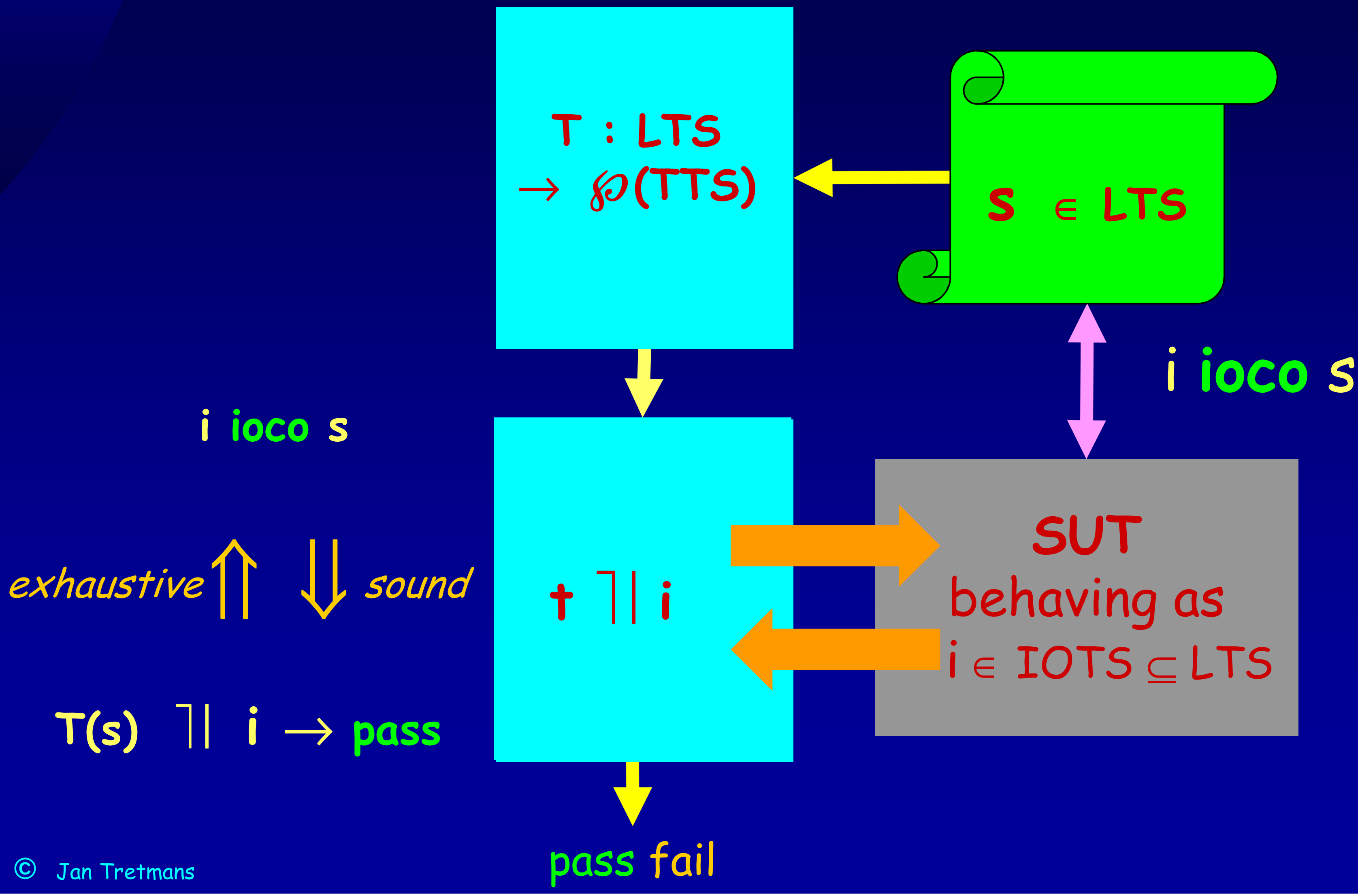
☞ Soundness :
t will never fail with correct implementation

i **ioco** s   implies   i passes t

☞ Exhaustiveness :
each incorrect implementation can be detected with a generated test t

i ~~ioco~~ s   implies   ∃ t : i fails t

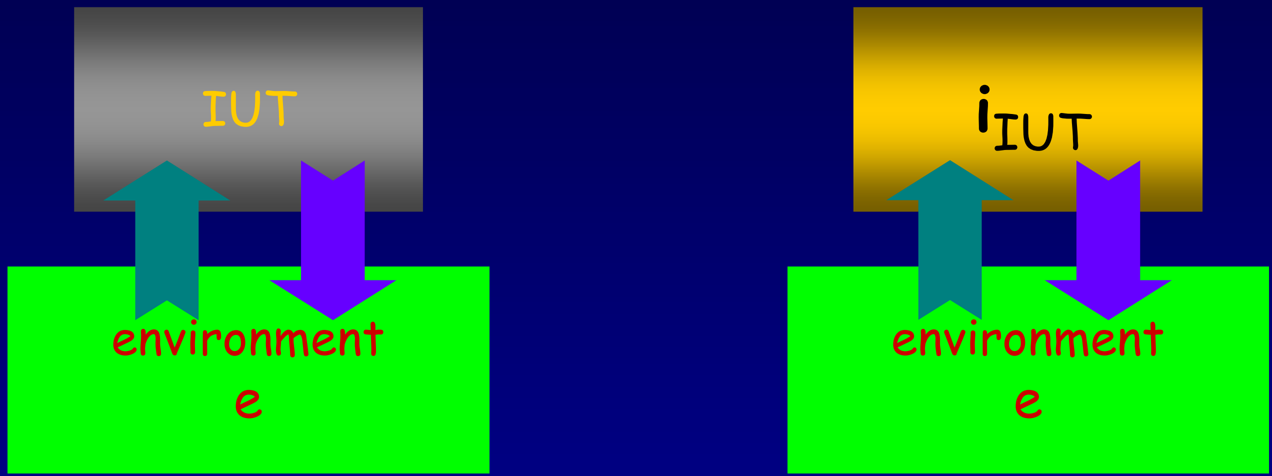
Model Based Testing with Transition Systems



Test Assumption

(Test Hypothesis)

Comparing Transition Systems: An Implementation and a Model



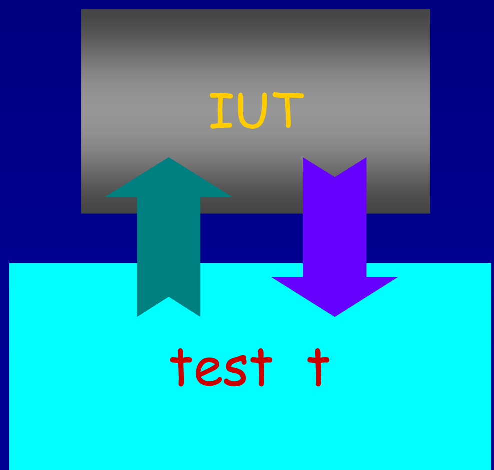
$$IUT \approx i_{IUT} \iff \forall\, e \in E \,.\; obs(e, IUT) = obs(e, i_{IUT})$$

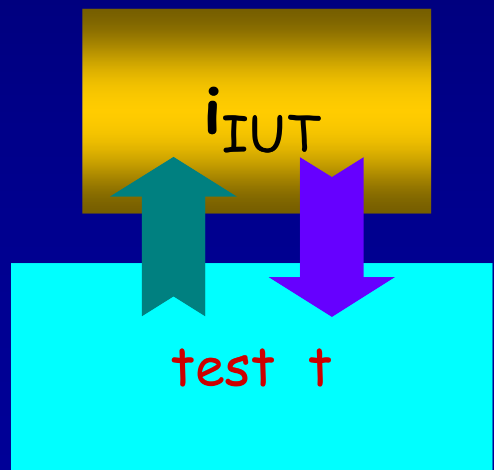
Formal Testing : Test Assumption

Test assumption :

$\forall$ IUT . $\exists$ $i_{IUT} \in$ MOD.

$\forall$ t $\in$ TEST . IUT passes t $\Leftrightarrow$ i_{IUT} passes t





Completeness of Formal Testing

$$\text{IUT } \textbf{passes } T_s \overset{?}{\Leftrightarrow} \text{IUT } \textbf{confto } s$$

IUT **passes** T_s

IUT **passes** $T_s \Leftrightarrow_{def} \forall t \in T_s$. IUT **passes** t

$\Leftrightarrow \quad \forall t \in T_s$. IUT **passes** t

Test assumption : $\forall t \in$ TEST . IUT passes $t \Leftrightarrow i_{IUT}$ passes t

$\Leftrightarrow \quad \forall t \in T_s . i_{IUT}$ **passes** t

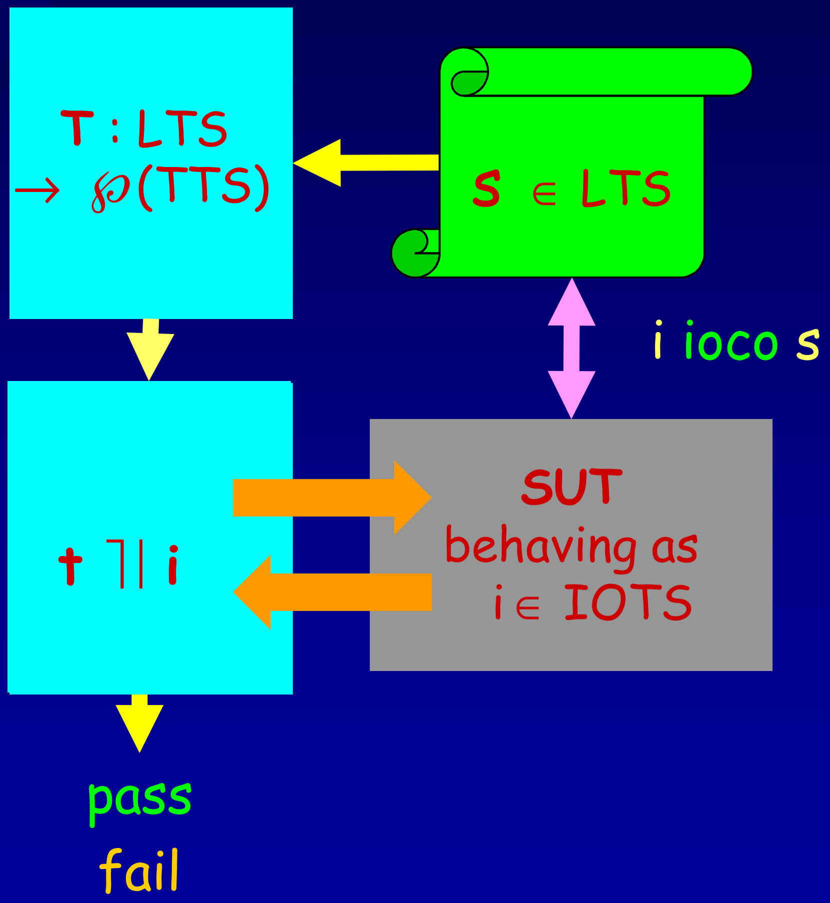
Proof obligation: $\forall i \in$ MOD ($\forall t \in T_s . i$ **passes** t) $\Leftrightarrow$ i **imp** s

$\Leftrightarrow \quad i_{IUT}$ **imp** s

Definition : IUT **confto** s

$\Leftrightarrow$ IUT **confto** s

Formal Testing with Transition Systems



Test assumption :

$\forall IUT \in IMP \,.\, \exists i_{IUT} \in IOTS \,.$

$\forall t \in TEST \,.\, IUT$ **passes** t

$\Leftrightarrow i_{IUT}$ passes t

Proof soundness and exhaustiveness:

$\forall i \in IOTS \,.$

$(\forall t \in$ **T**$(s) \,.\, i$ **passes** $t)$

$\Leftrightarrow$ i **ioco** s

SUT ioco s

exhaustive ⇑ ⇓ *sound*

SUT ⌉| T(s) → pass

Model-Based Testing :
There is Nothing More Practical than a Good Theory

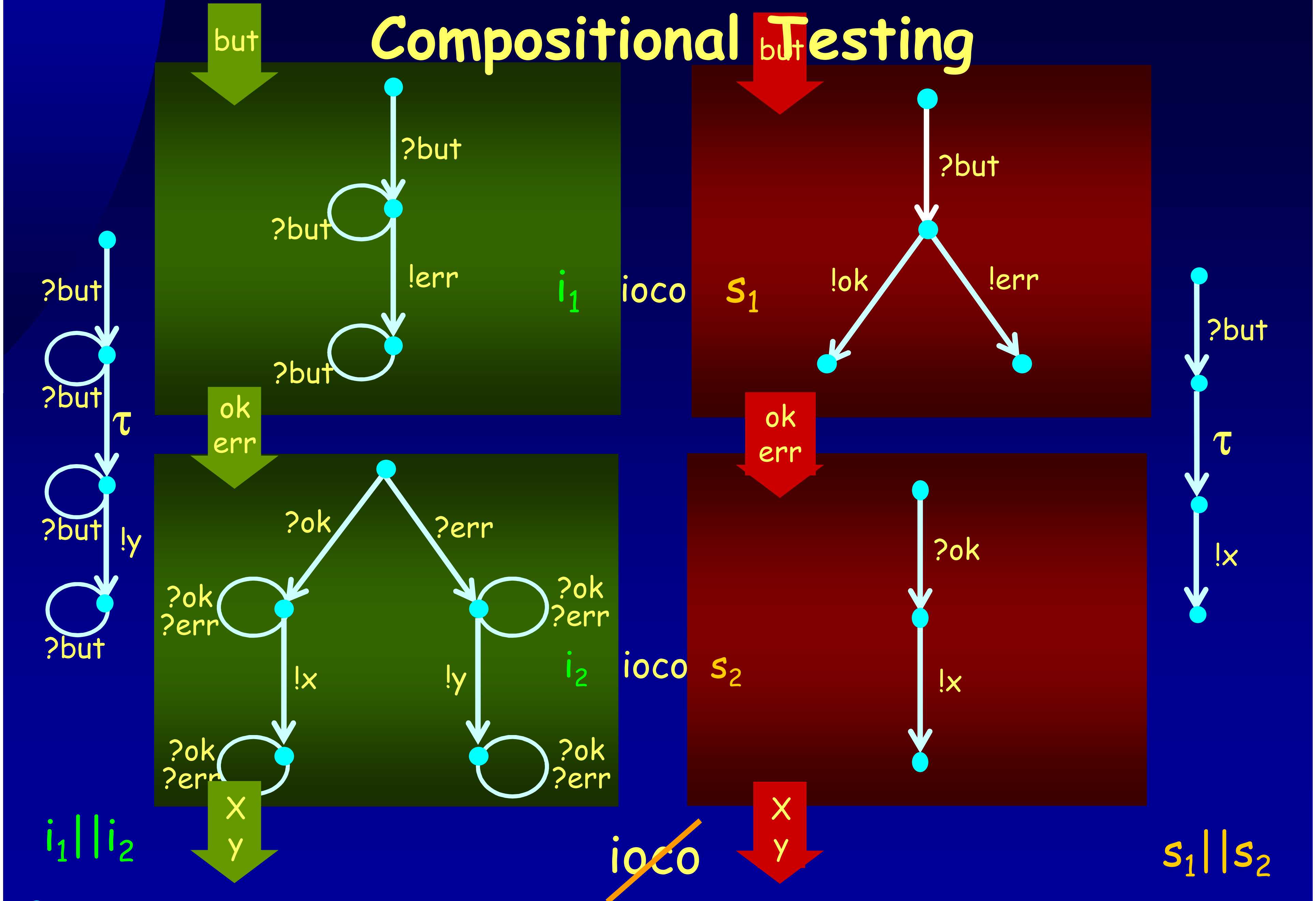
A well-defined and sound testing theory brings:

- ☞ Arguing about validity of test cases and correctness of test generation algorithms
- ☞ Explicit insight in what has been tested, and what not
- ☞ Use of complementary validation techniques: model checking, theorem proving, static analysis, runtime verification, ...
- ☞ Implementation relations for nondeterministic, concurrent, partially specified, loose specifications
- ☞ Comparison of MBT approaches and error detection capabilities

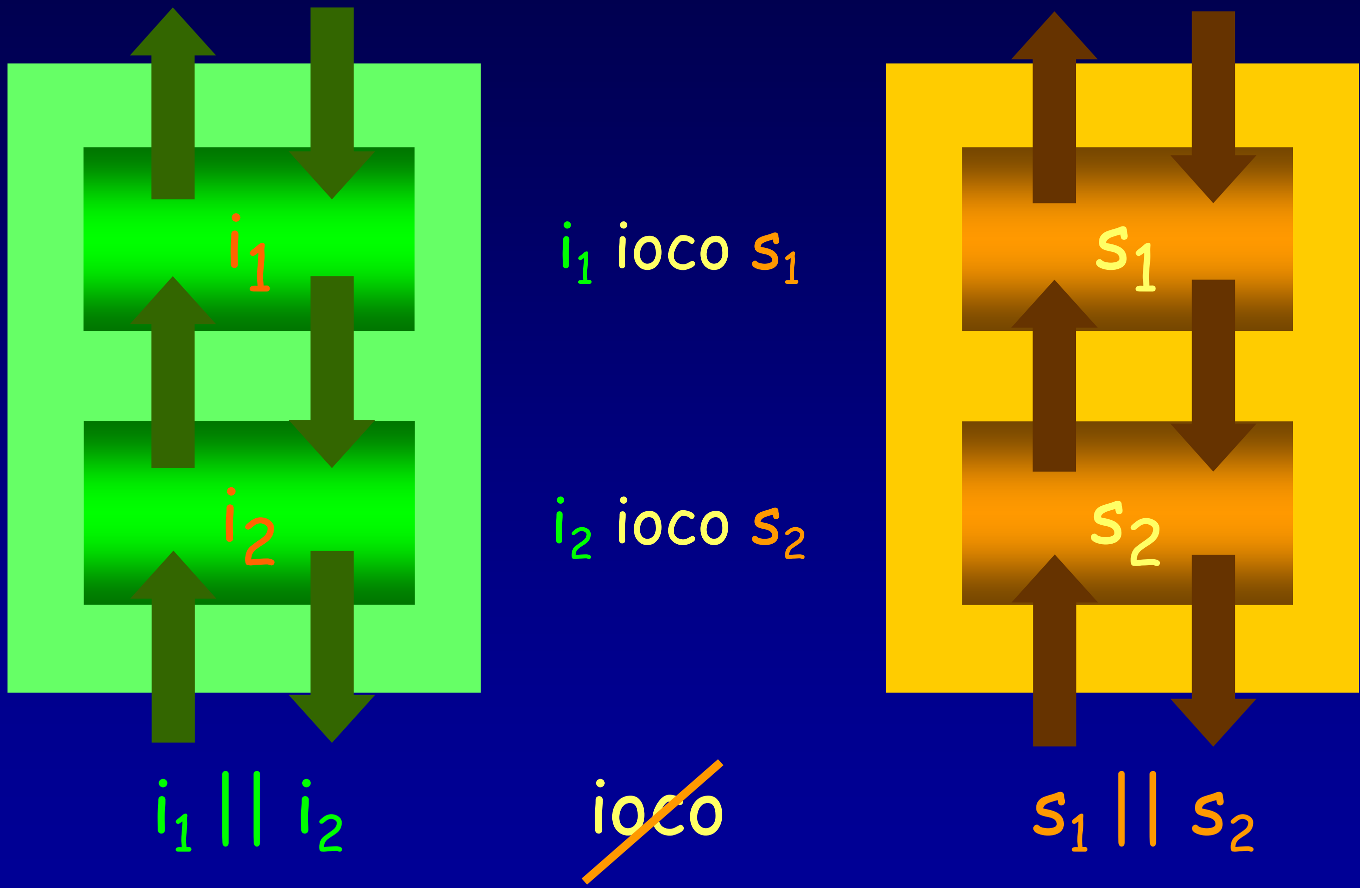
Embedded Systems Innovation BY TNO

A Consequence of ioco: (Non) Compositionality

Compositional Testing



Compositional Testing

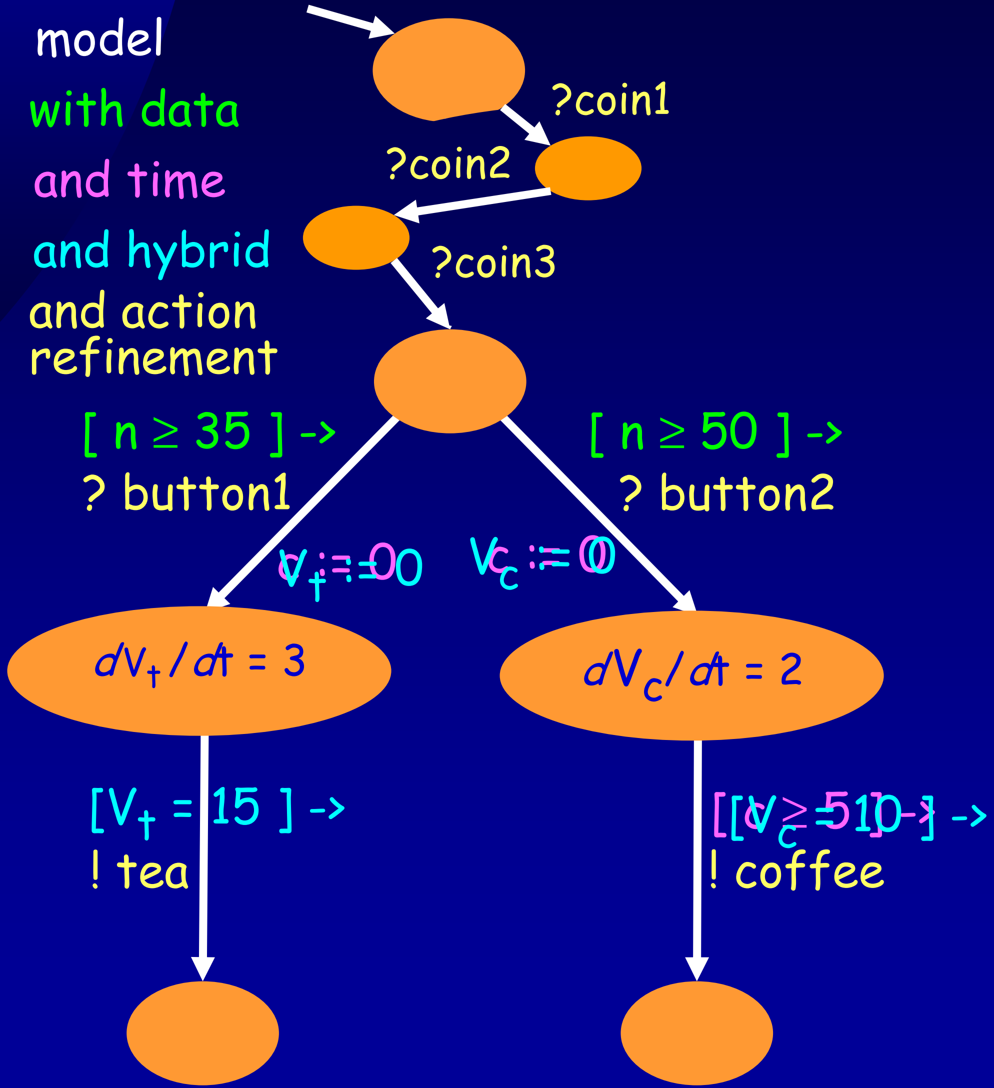


If s_1, s_2 input enabled - $s_1, s_2 \in$ IOTS - then ioco is preserved !

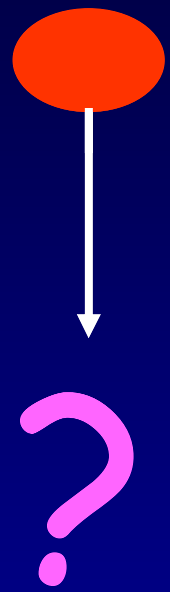
Variations of ioco

Testing Transition Systems: Variations

model
with data
and time
and hybrid
and action refinement



test case



Variations on a Theme

i **ioco** s $\Leftrightarrow$ $\forall \sigma \in Straces(s)$: *out* (i **after** σ) $\subseteq$ *out* (s **after** σ)

i $\leq_{ior}$ s $\Leftrightarrow$ $\forall \sigma \in (L \cup \{\delta\})^*$: *out* (i **after** σ) $\subseteq$ *out* (s **after** σ)

i **ioconf** s $\Leftrightarrow$ $\forall \sigma \in traces(s)$: *out* (i **after** σ) $\subseteq$ *out* (s **after** σ)

i $\mathbf{ioco}_F$ s $\Leftrightarrow$ $\forall \sigma \in F$: *out* (i **after** σ) $\subseteq$ *out* (s **after** σ)

i **uioco** s $\Leftrightarrow$ $\forall \sigma \in Utraces(s)$: *out* (i **after** σ) $\subseteq$ *out* (s **after** σ)

i **mioco** s multi-channel ioco

i **wioco** s non-input-enabled ioco

i **eco** e environmental conformance

i **sioco** s symbolic ioco

i **(r)tioco** s (real) timed tioco (Aalborg, Twente, Grenoble, Bordeaux,. . . .)

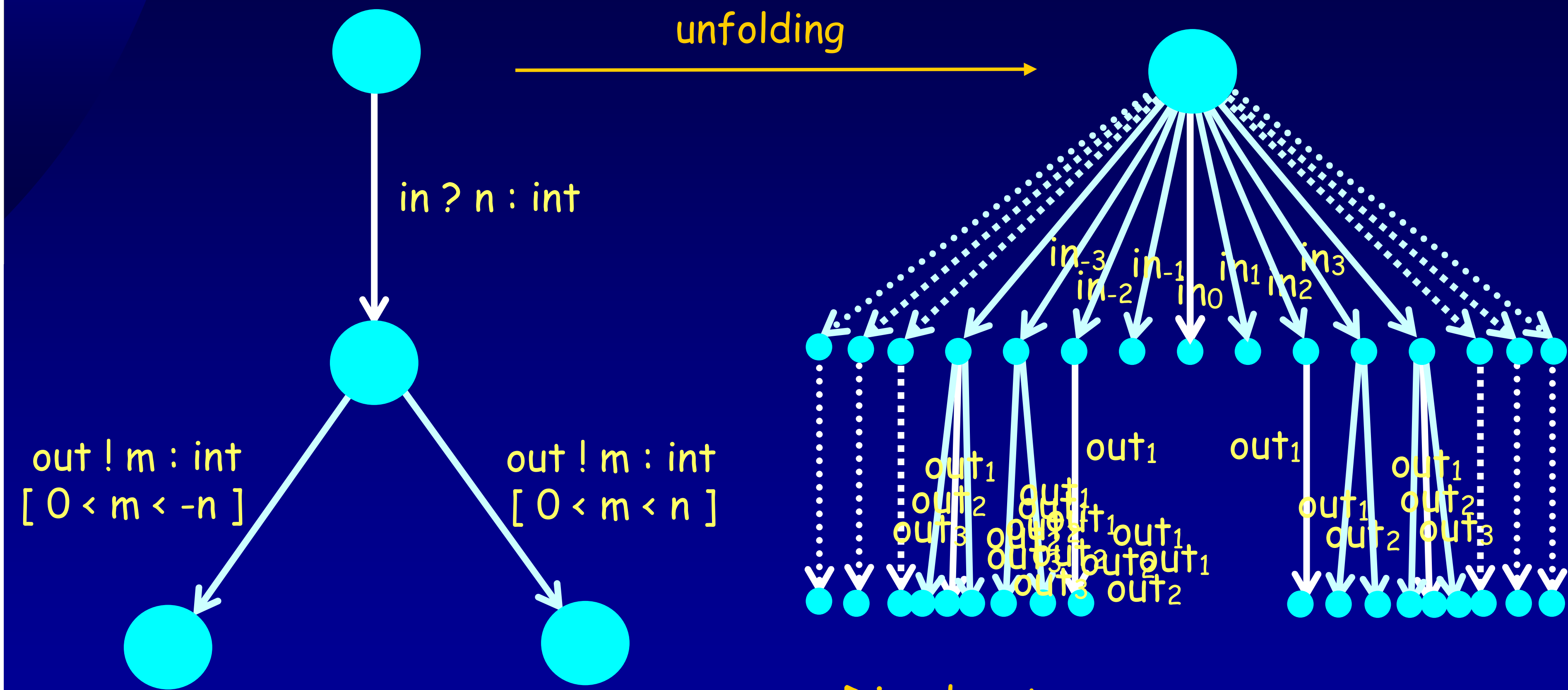
i $\mathbf{ioco}_r$ s refinement ioco

i **hioco** s hybrid ioco

i **qioco** s quantified ioco

Symbolic ioco

Transition System with Data



Disadvantages:

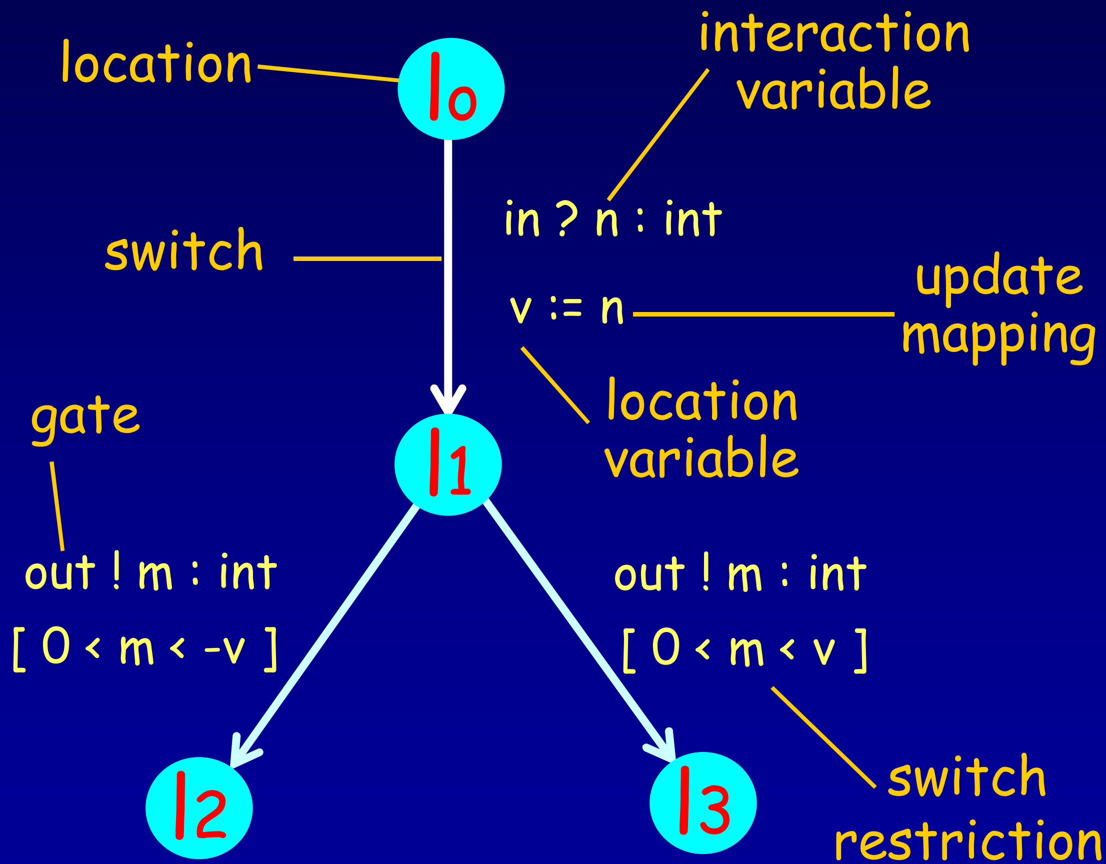
☞ infinity

☞ loss of information
(e.g. for test selection)

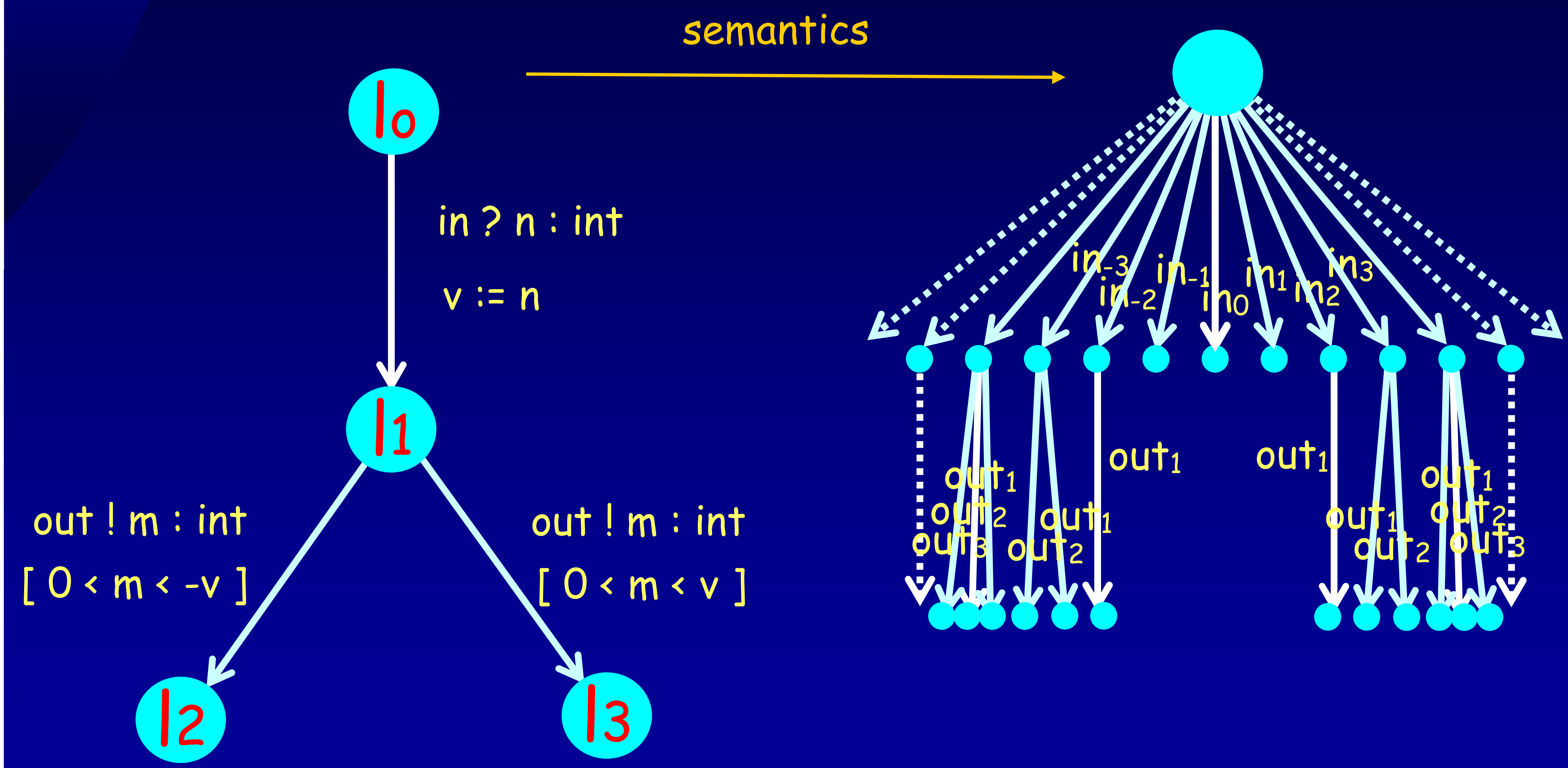
Symbolic Transition System

STS:

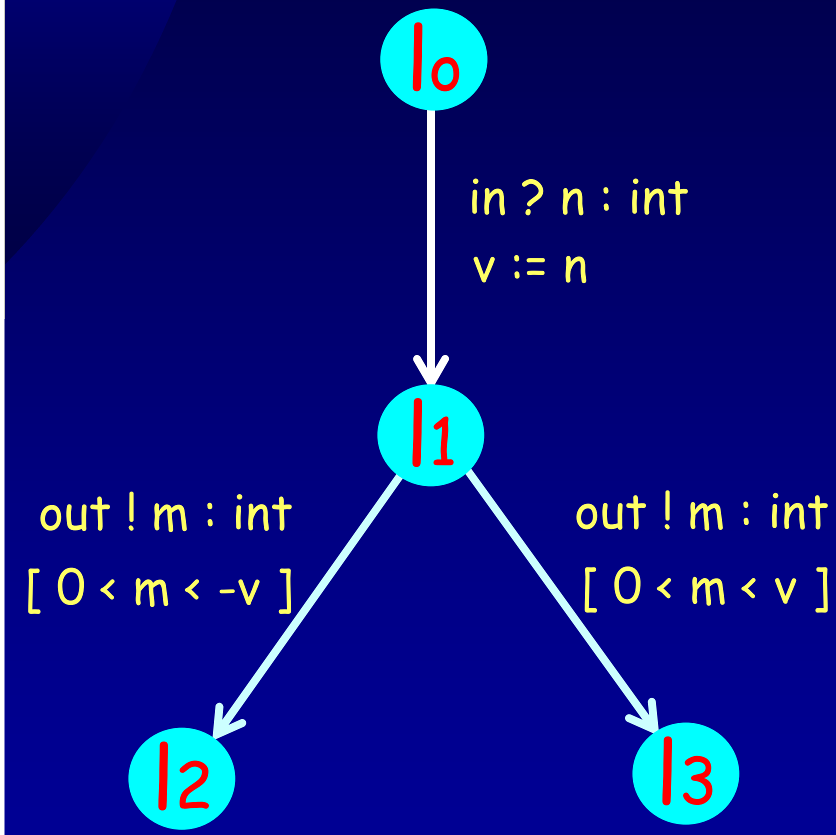
- LTS with explicit data, variables and constraints
- Data: first order logic
- Finite, symbolic representation



Symbolic Transition System



Symbolic Transitions



Generalised switch relation

$l_0 \xRightarrow{\text{in? } \ \text{true} \ \ v:=n_1} l_1$

$l_1 \xRightarrow{\text{out! } \ 0 < m_2 < v \ \ v:=v} l_3$

$l_0 \xRightarrow{\text{in? out! } \ 0 < m_2 < n_1 \ \ v:=n_1} l_3$

Symbolic states

$(l_1, [\text{true}], v:=n_1)$

$(l_2, [0 < m_2 < -n_1], v:=n_1)$

$(l_3, [0 < m_2 < n_1], v:=n_1)$

Symbolic Trace, After, . . .

Symbolic suspension trace

…… pair of …… (sequence of gates,
formula over indexed interaction
variables and location variables) ......

Symbolic $after_s$

…… <symbolic state> $after_s$ <symbolic suspension trace> ......

Lemma

[[<symbolic state> $after_s$ <symbolic suspension trace>]]

=

[[<symbolic state>]] after [[<symbolic suspension trace>]]

Symbolic ioco

Specification: IOSTS $\mathcal{S}(\iota_S) = \langle L_S, l_S, \mathcal{V}_S, \mathcal{I}, \Lambda, \rightarrow_S \rangle$
Implementation: IOSTS $\mathcal{P}(\iota_P) = \langle L_P, l_P, \mathcal{V}_P, \mathcal{I}, \Lambda, \rightarrow_P \rangle$
both initialised, implementation input-enabled, $\mathcal{V}_S \cap \mathcal{V}_P = \emptyset$
$\mathcal{F}_s$: a set of symbolic extended traces satisfying $[\![\mathcal{F}_s]\!]_{\iota_S} \subseteq \mathit{Straces}((l_0, \iota))$;

$\mathcal{P}(\iota_P)$ **sioco**$_{\mathcal{F}_s}$ $\mathcal{S}(\iota_S)$ iff

$$\forall(\sigma, \chi) \in \mathcal{F}_s \; \forall \lambda_\delta \in \Lambda_U \cup \{\delta\} : \iota_P \cup \iota_S \models \overline{\forall}_{\widehat{\mathcal{I}} \cup \mathcal{I}} \big(\Phi(l_P, \lambda_\delta, \sigma) \wedge \chi \rightarrow \Phi(l_S, \lambda_\delta, \sigma) \big)$$

$$\text{where } \Phi(\xi, \lambda_\delta, \sigma) = \bigvee \{ \varphi \wedge \psi \mid (\lambda_\delta, \varphi, \psi) \in \mathbf{out}_s((\xi, \top, \mathsf{id})_0 \, \mathbf{after}_s(\sigma, \top)) \}$$

Theorem 1.

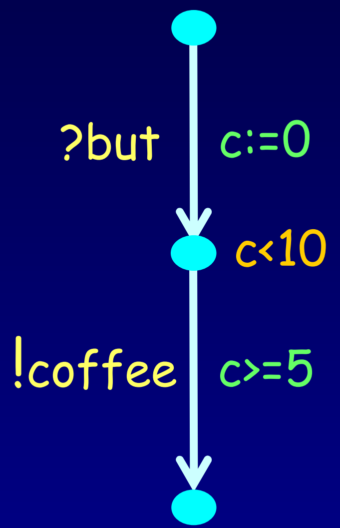
$$\mathcal{P}(\iota_P) \; \mathbf{sioco}_{\mathcal{F}_s} \; \mathcal{S}(\iota_S) \quad \mathit{iff} \quad [\![\mathcal{P}]\!]_{\iota_P} \; \mathbf{ioco}_{[\![\mathcal{F}_s]\!]_{\iota_S}} \; [\![\mathcal{S}]\!]_{\iota_S}$$

Real Time ioco

Real-Time Model-Based Testing

- In many systems real-time properties are crucial
- Approach:
  - Extension of IOTS/**ioco** theory
    - Timed Input Output Transition Systems (TIOTS)
    - Timed Implementation Relations: build on **ioco**
    - Concentrate on implementation relations: no test generation
- Challenges:
  - Is time input or output ?
  - Quiescence: How long is there never eventually no output?

Timed Input-Output Transition Systems



TIOTS : $\langle Q, L_I, L_U, \mathbf{R}_{\geq 0}, T, q_0 \rangle$

Observable actions: L_I, L_U

delay $d \in \mathbf{R}_{\geq 0}$

Unobservable action: τ

Specifications are TIOTS

Implementations are assumed

to behave as input-enabled TIOTS

Constraints:

- time additivity
- null delay
- time determinism
- no divergence
- progress: no forced inputs

The Untimed Implementation Relation ioco

$$i \ \textbf{ioco} \ s \ =_{def} \ \forall \sigma \in \mathit{traces}(s) : \ \mathit{out}(i \ \textbf{after} \ \sigma) \subseteq \mathit{out}(s \ \textbf{after} \ \sigma)$$

↓ *Straces* ↓ **after** ↓ *out*

$$\delta(p) \ = \ \forall \ !x \in L_U \cup \{\tau\} . \ p \overset{!x}{\not\longrightarrow}$$

$$\mathit{Straces}(s) \ = \ \{ \ \sigma \in (L \cup \{\delta\})^* \ | \ s \overset{\sigma}{\Longrightarrow} \ \}$$

$$\mathit{out}(p) \ = \ \{ \ !x \in L_U \ | \ p \overset{!x}{\Longrightarrow} \ \} \ \cup \ \{ \delta \ | \ \delta(p) \ \}$$

$$\mathit{out}(P) \ = \ \bigcup \{ \ \mathit{out}(p) \ | \ p \in P \ \}$$

$$p \ \textbf{after} \ \sigma \ = \ \{ \ p' \ | \ p \overset{\sigma}{\Longrightarrow} p' \ \}$$

A Timed Implementation Relation

i **ioco** s $=_{def}$ $\forall \sigma \in$ *traces* (s) : *out* (i **after** σ) $\subseteq$ *out* (s **after** σ)

↓ ↓ ↓ ↓

tioco$_X$? ? ?

A Timed Implementation Relation

i **ioco** s $=_{def}$ $\forall \sigma \in$ *traces* (s) : *out* (i **after** σ) $\subseteq$ *out* (s **after** σ)

- **ioco** ↓ **tioco**
- *traces* ↓ *ttraces*
- **after** ↓ **after**$_t$
- *out* ↓ *out*$_t$

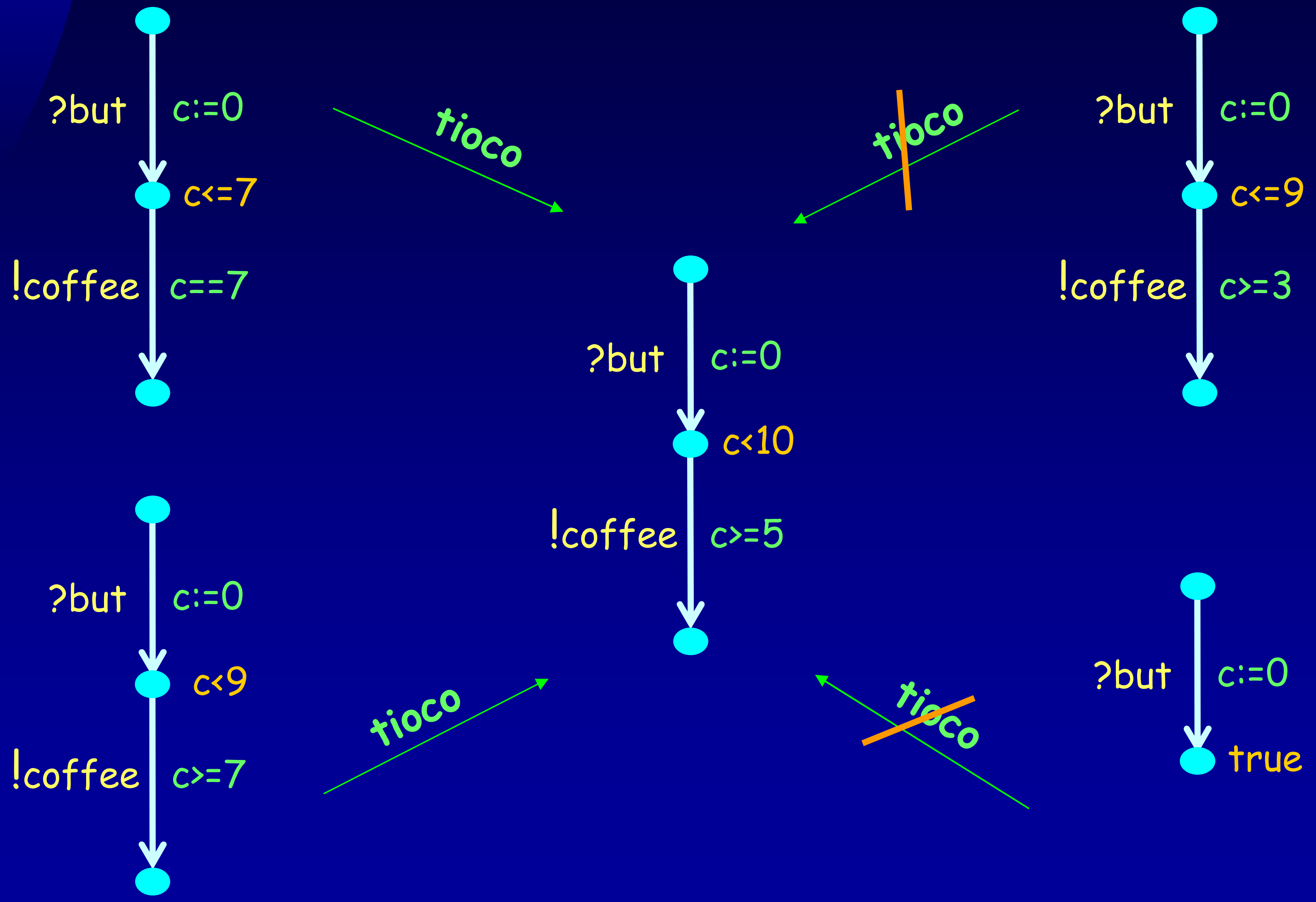
δ (p) = ✗

ttraces (s) = { $\sigma \in (L \cup \mathbf{R}_{\geq 0})^* \mid s \stackrel{\sigma}{\Longrightarrow}$ }

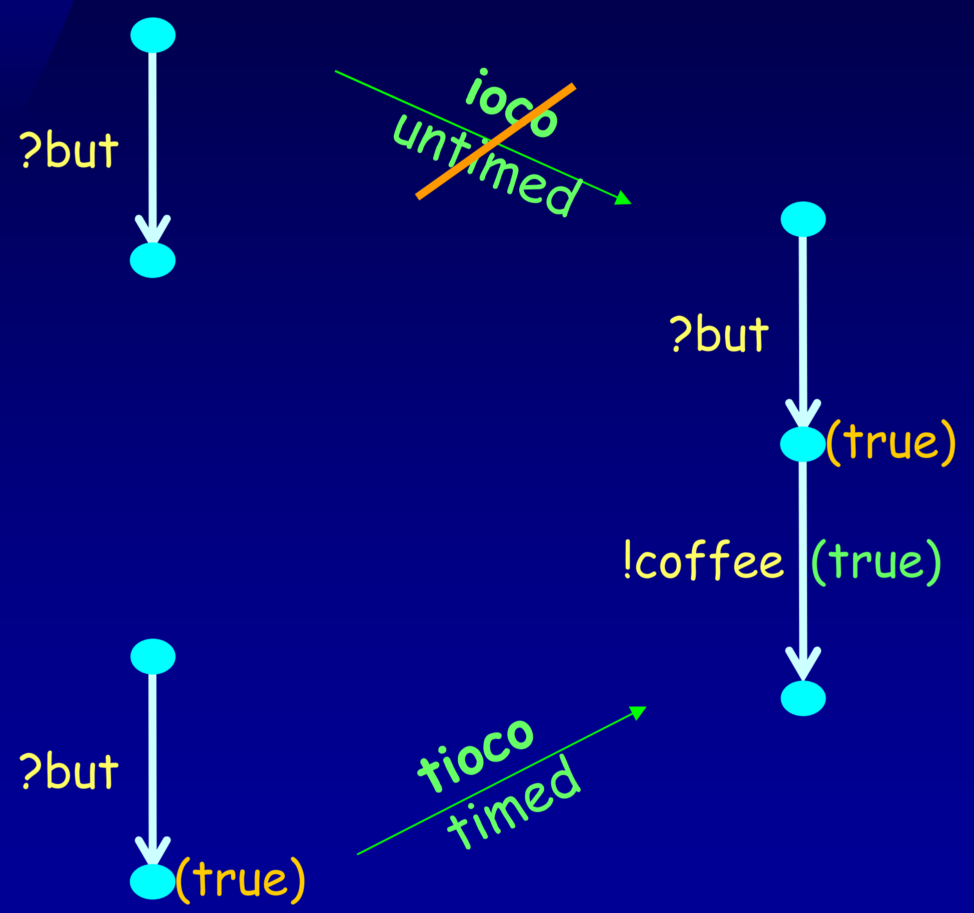
out$_t$ (p) = { $x \in L_U \cup \mathbf{R}_{\geq 0} \mid p \stackrel{x}{\Longrightarrow}$ }

p **after**$_t$ σ = { $p' \mid p \stackrel{\sigma}{\Longrightarrow} p', \ \sigma \in (L \cup \mathbf{R}_{\geq 0})^*$ }

A Timed Implementation Relation tioco



Not Just Adding Extra Constraints: Unbounded Delay



- And suppose you wish to reject this IUT: how long would you wait ?
- Untimed ioco: quiescence to express that there eventually is !coffee
- But when is eventually ?