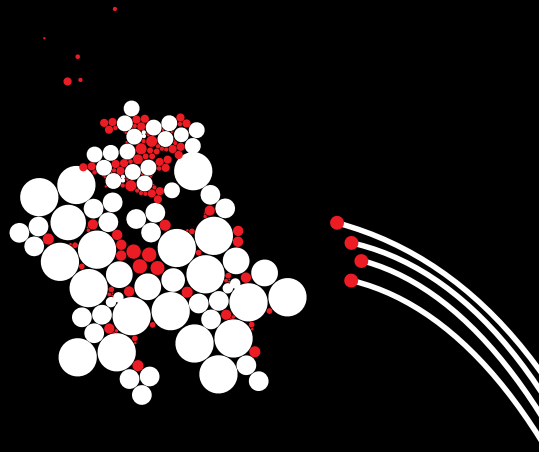
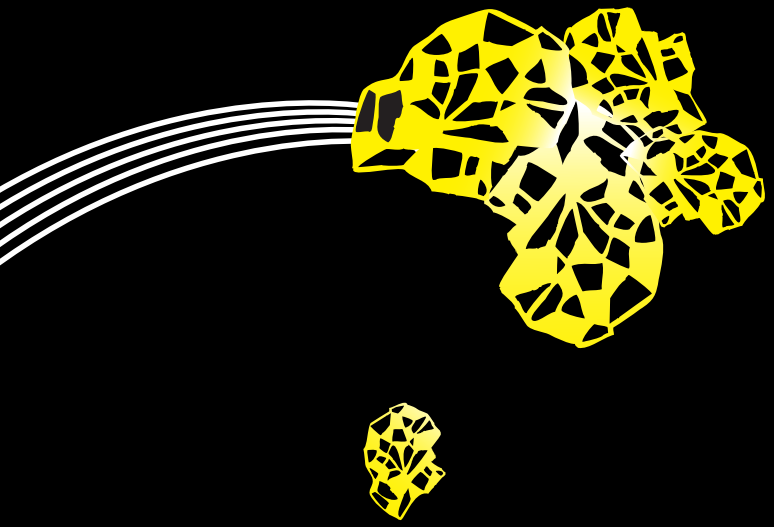
UNIVERSITY OF TWENTE.



Test generation & ioco

Software Testing and Risk Assessment

Lecture 5



Today

- after and test cases (revisited)
- Batch test generation
- On-the-fly Test generation
- Test assumptions
- ioco
- Sound and exhaustive test suites
- Underspecification
- uioco

after

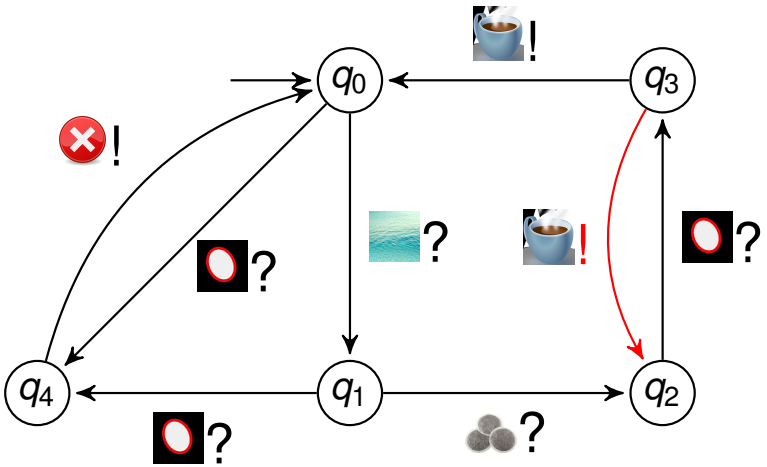
- Let $S = (Q, L_I, L_U, T, q_0)$ be an LTS.
- Let $L = L_I \cup L_U \cup \{\delta\}$.
- Let $q \in Q$ be a state, $\mu \in L$ a label, and $\rho \in L^*$ a sequence of labels.
- A state q is quiescent, denoted $\delta(q)$, if:
  $\forall x \in L_U, \forall q' \in Q : (q, x, q') \notin T$
- *after* : $Q \times L^* \to \mathbb{P}(Q)$
- q *after* $\epsilon = \{q\}$
- q *after* $\mu\rho = Q'$ *after* ρ where
  $Q' = \{q' \in Q \mid (q, \mu, q') \in T\} \cup \{q \mid \delta(q) \wedge \mu = \delta\}$
- Extend to *after* : $\mathbb{P}(Q) \times L^* \to \mathbb{P}(Q)$

after

- Let $S = (Q, L_I, L_U, T, q_0)$ be an LTS.
- Let $L = L_I \cup L_U \cup \{\delta\}$.
- Let $q \in Q$ be a state, $\mu \in L$ a label, and $\rho \in L^*$ a sequence of labels.
- A state q is quiescent, denoted $\delta(q)$, if:
  $\forall x \in L_U, \forall q' \in Q : (q, x, q') \notin T$
- *after* : $Q \times L^* \to \mathbb{P}(Q)$
- q *after* $\epsilon = \{q\}$
- q *after* $\mu\rho = Q'$ *after* ρ where
  $Q' = \{q' \in Q \mid (q, \mu, q') \in T\} \cup \{q \mid \delta(q) \wedge \mu = \delta\}$
- Extend to *after* : $\mathbb{P}(Q) \times L^* \to \mathbb{P}(Q)$
- Let $Q' \subseteq Q$.
- Q' *after* $\rho = \bigcup_{q \in Q'} q$ *after* ρ

Example for *after*

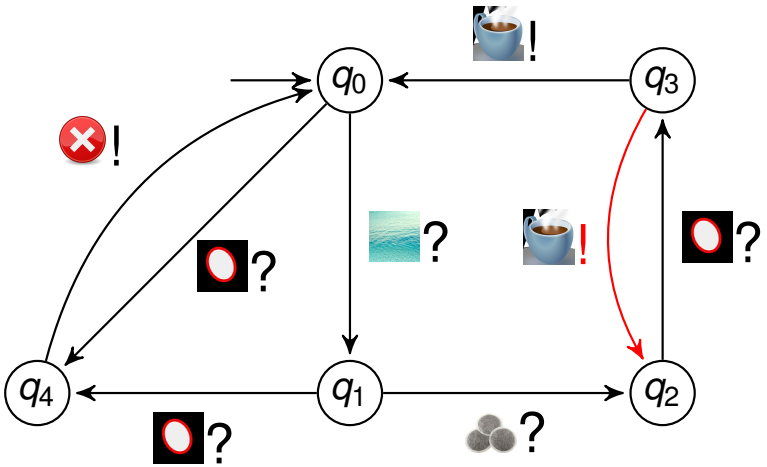
- q *after* $\mu\rho = Q'$ *after* ρ where
  $Q' = \{q' \in Q \mid (q, \mu, q') \in T\} \cup \{q \mid \delta(q) \wedge \mu = \delta\}$
- Q' *after* $\rho = \bigcup_{q \in Q'} q$ *after* ρ



- q_3 *after* ☕!🍩? =

Example for *after*

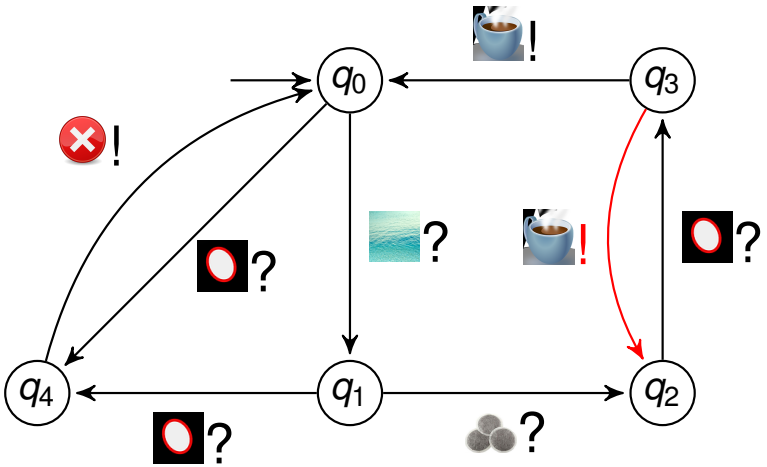
- q *after* $\mu\rho = Q'$ *after* ρ where
  $Q' = \{q' \in Q \mid (q, \mu, q') \in T\} \cup \{q \mid \delta(q) \wedge \mu = \delta\}$
- Q' *after* $\rho = \bigcup_{q \in Q'} q$ *after* ρ



- q_3 *after* coffee! cup? =
  $(\{q' \in Q \mid (q_3, \text{coffee}!, q') \in T\} \cup \{q_3 \mid \delta(q_3) \wedge \text{coffee}! = \delta\})$ *after* cup? =

Example for *after*

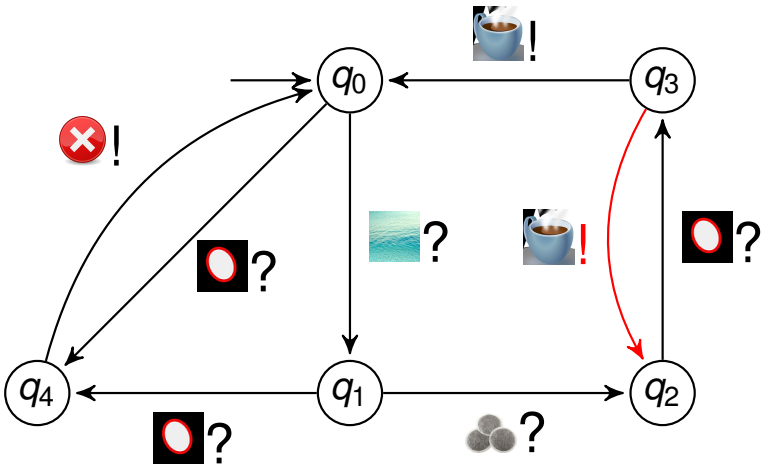
- q *after* $\mu\rho = Q'$ *after* ρ where
  $Q' = \{q' \in Q \mid (q, \mu, q') \in T\} \cup \{q \mid \delta(q) \wedge \mu = \delta\}$
- Q' *after* $\rho = \bigcup_{q \in Q'} q$ *after* ρ



- q_3 *after* ☕!🍩? =
  $(\{q' \in Q \mid (q_3, ☕!, q') \in T\} \cup \{q_3 \mid \delta(q_3) \wedge ☕! = \delta\})$ *after* 🍩? =
  $(\{q_0, q_2\} \cup \emptyset)$ *after* 🍩? =

Example for *after*

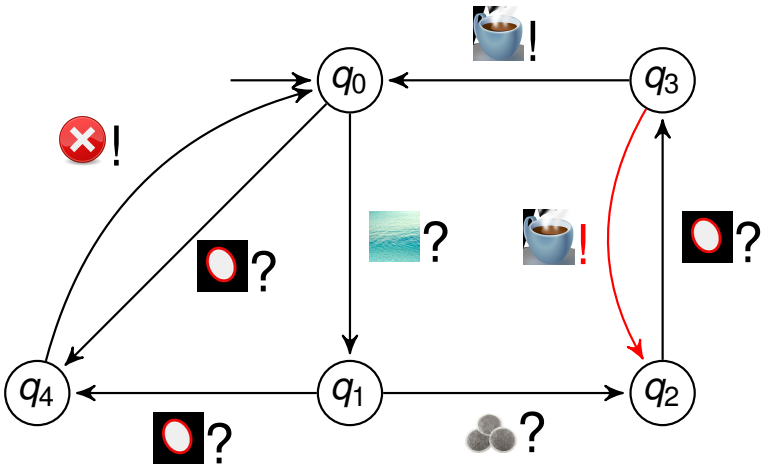
- q *after* $\mu\rho = Q'$ *after* ρ where
  $Q' = \{q' \in Q \mid (q, \mu, q') \in T\} \cup \{q \mid \delta(q) \wedge \mu = \delta\}$
- Q' *after* $\rho = \bigcup_{q \in Q'} q$ *after* ρ



- q_3 *after* ☕!🍩? =
  $(\{q' \in Q \mid (q_3, ☕!, q') \in T\} \cup \{q_3 \mid \delta(q_3) \wedge ☕! = \delta\})$ *after* 🍩? =
  $(\{q_0, q_2\} \cup \emptyset)$ *after* 🍩? =
  (q_0 *after* 🍩?) $\cup$ (q_2 *after* 🍩?) =

Example for *after*

- q *after* $\mu\rho = Q'$ *after* ρ where
  $Q' = \{q' \in Q \mid (q, \mu, q') \in T\} \cup \{q \mid \delta(q) \wedge \mu = \delta\}$
- Q' *after* $\rho = \bigcup_{q \in Q'} q$ *after* ρ

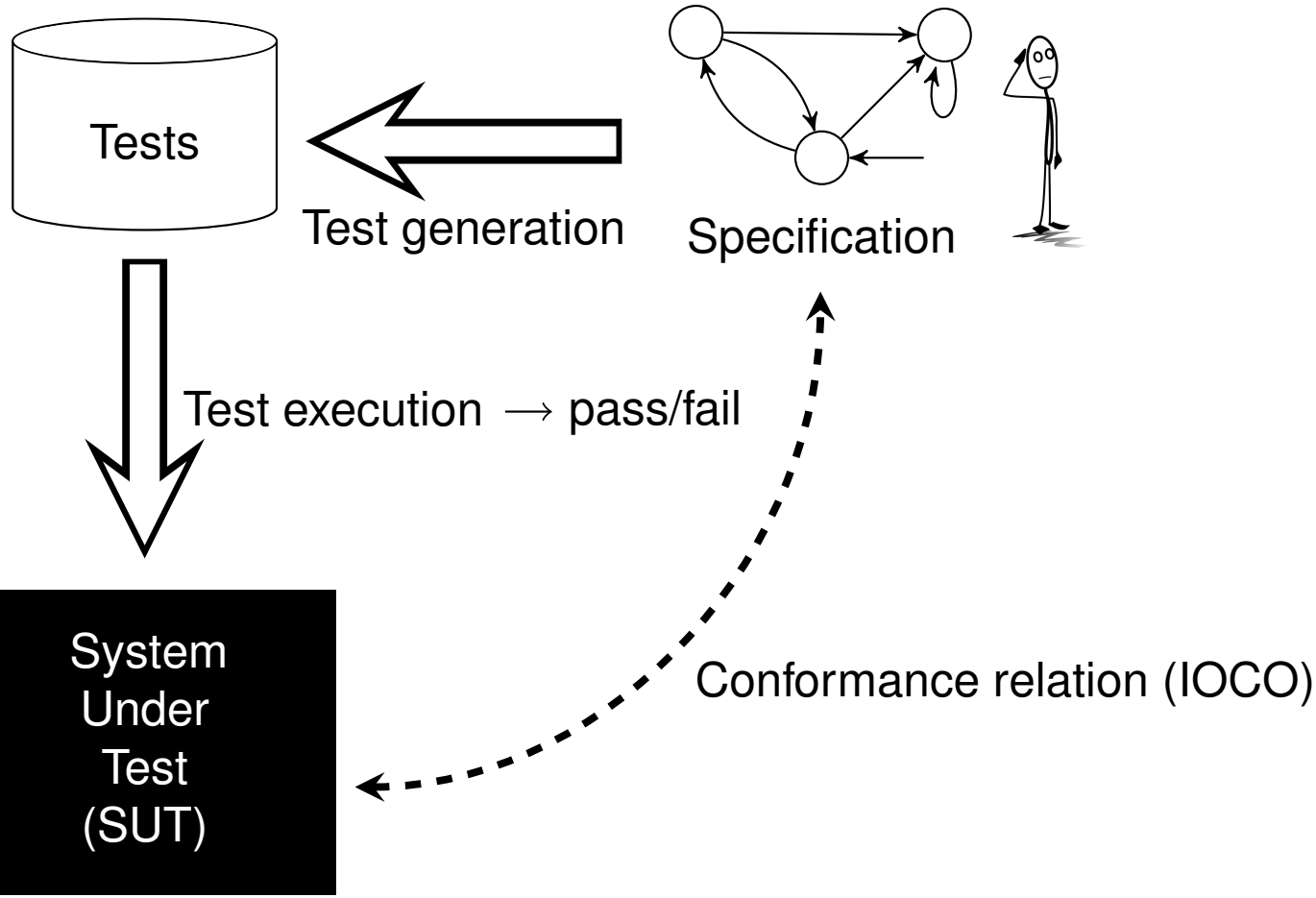


- q_3 *after* coffee! cup? =
  $(\{q' \in Q \mid (q_3, \text{coffee}!, q') \in T\} \cup \{q_3 \mid \delta(q_3) \wedge \text{coffee}! = \delta\})$ *after* cup? =
  $(\{q_0, q_2\} \cup \emptyset)$ *after* cup? =
  $(q_0$ *after* cup?$) \cup (q_2$ *after* cup?$) =$
  $\{q_4\} \cup \{q_2\} = \{q_2, q_4\}$

Formal definition of test cases

- A test case for an LTS S is an LTS $t = (Q^t, \underline{L_I, L_U}, T^t, q_0^t)$ s.t.:
  - $\underline{t \text{ uses the same labels as } S}$
  - There are two special states *Pass*, *Fail*$\in Q^t$
  - States *Pass* and *Fail* have self-loops for all outputs (incl. δ): $\forall x \in L_U \cup \{\delta\} : \textit{Pass after } x = \{\textit{Pass}\}$ and $\textit{Fail after } x = \{\textit{Fail}\}$
  - States *Pass* and *Fail* have no transition for inputs: $\forall a \in L_I : \textit{Pass after } a = \textit{Fail after } a = \emptyset$
  - t has no cycles except those in *Pass* and *Fail*
  - t is deterministic
  - Every state enables all outputs L_U, and either one input or δ: $\forall q \in \underline{Q^t} : (|in(q)| = 0 \wedge out(q) = L_U \cup \{\delta\}) \vee (out(q) = L_U \wedge |in(q)| = 1)$

Model-Based Testing



Test generation (batch)

- Algorithm *batchGen*:
  - **Input:** LTS $S = (Q, L_I, L_U, T, q_0)$, and current states $Q' \subseteq Q$
  - **Output:** a test case for S

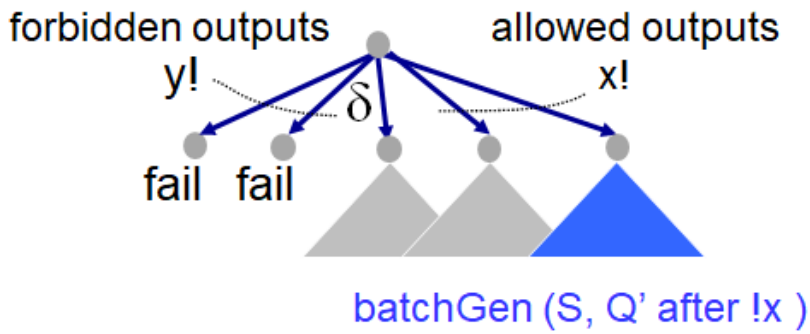
Test generation (batch)

- Algorithm *batchGen*:
  - **Input:** LTS $S = (Q, L_I, L_U, T, q_0)$, and current states $Q' \subseteq Q$
  - **Output:** a test case for S
- $batchGen(S, Q') =$ take either of the following steps:

End test case	pass

Test generation (batch)

- Algorithm *batchGen*:
  - **Input:** LTS $S = (Q, L_I, L_U, T, q_0)$, and current states $Q' \subseteq Q$
  - **Output:** a test case for S
- $batchGen(S, Q') =$ take either of the following steps:

End test case	pass
Observe output	

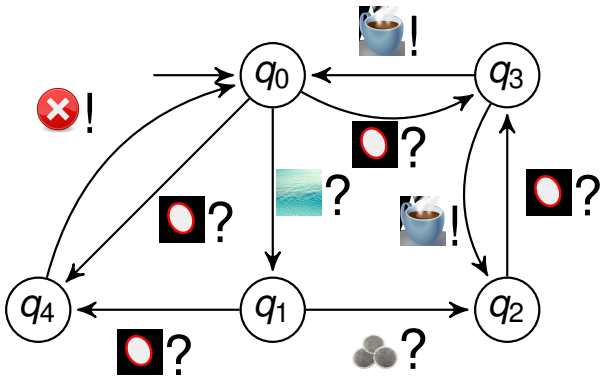
Test generation (batch)

- Algorithm *batchGen*:
  - **Input:** LTS $S = (Q, L_I, L_U, T, q_0)$, and current states $Q' \subseteq Q$
  - **Output:** a test case for S
- $batchGen(S, Q') =$ take either of the following steps:

End test case	pass
Observe output	forbidden outputs y! δ fail fail allowed outputs x! batchGen (S, Q' after !x)
Supply input a? where Q' *after* $a? \neq \emptyset$	forbidden outputs y! a? fail fail allowed outputs x! batchGen(S, Q' after a?)

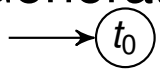
Batch test generation example

- Input LTS:



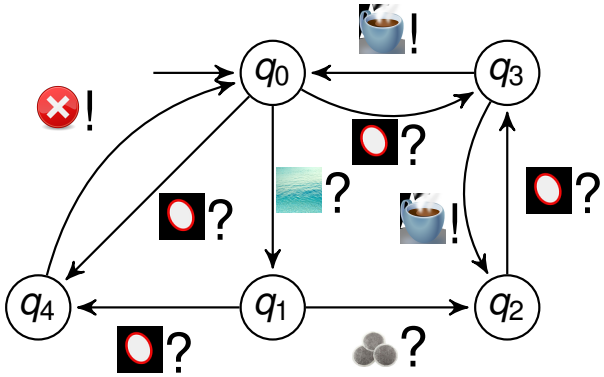
- Current states in t_0: $\{q_0\}$

- Generated test case:



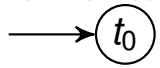
Batch test generation example

- Input LTS:



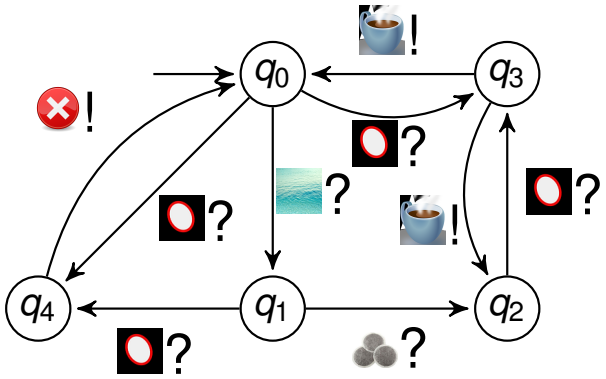
- Current states in t_0: $\{q_0\}$
- Choose input: ?

- Generated test case:



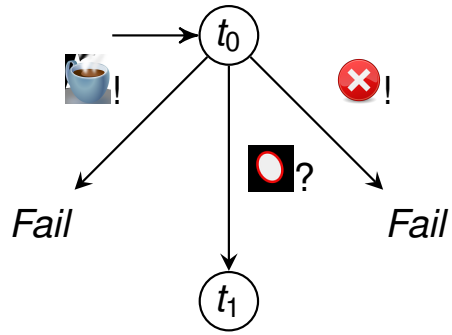
Batch test generation example

- Input LTS:



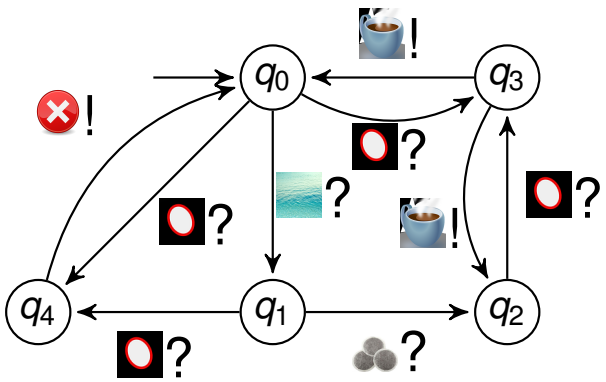
- Current states in t_0: $\{q_0\}$
- Choose input: ?
- Current states in t_1: $\{q_3, q_4\}$

- Generated test case:



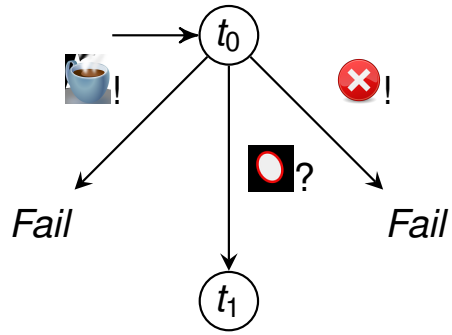
Batch test generation example

- Input LTS:



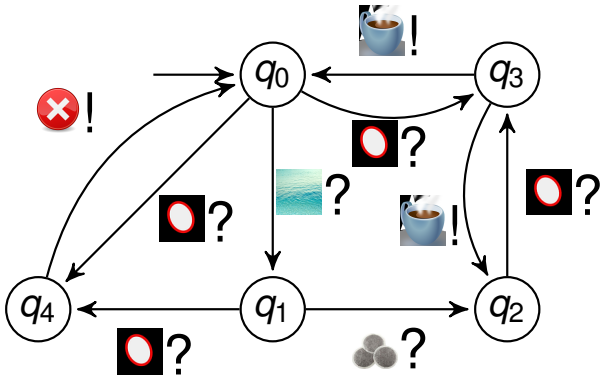
- Current states in t_0: $\{q_0\}$
- Choose input: ?
- Current states in t_1: $\{q_3, q_4\}$
- Observe outputs

- Generated test case:



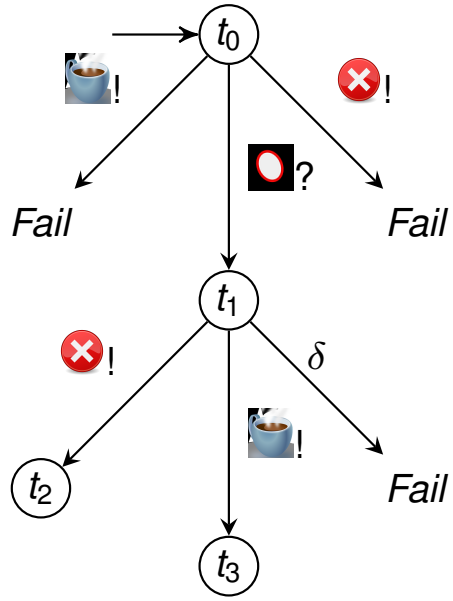
Batch test generation example

- Input LTS:



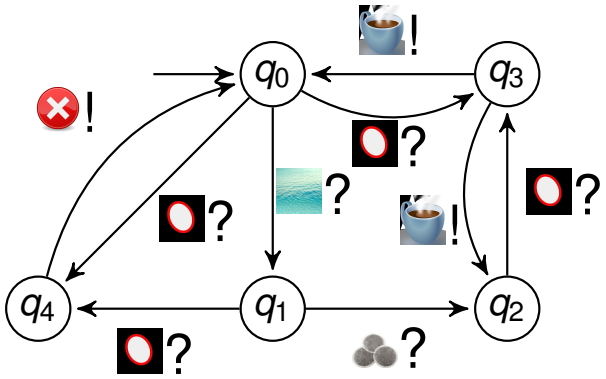
- Current states in t_0: $\{q_0\}$
- Choose input: ?
- Current states in t_1: $\{q_3, q_4\}$
- Observe outputs
- Current states in t_2: $\{q_0\}$

- Generated test case:



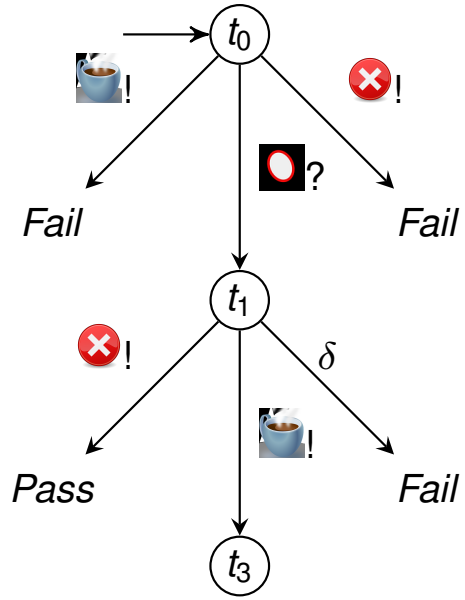
Batch test generation example

- Input LTS:



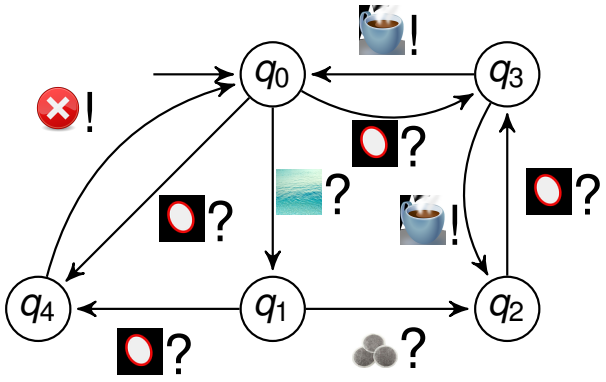
- Current states in t_0: $\{q_0\}$
- Choose input: ?
- Current states in t_1: $\{q_3, q_4\}$
- Observe outputs
- Current states in t_2: $\{q_0\}$
- Stop

- Generated test case:



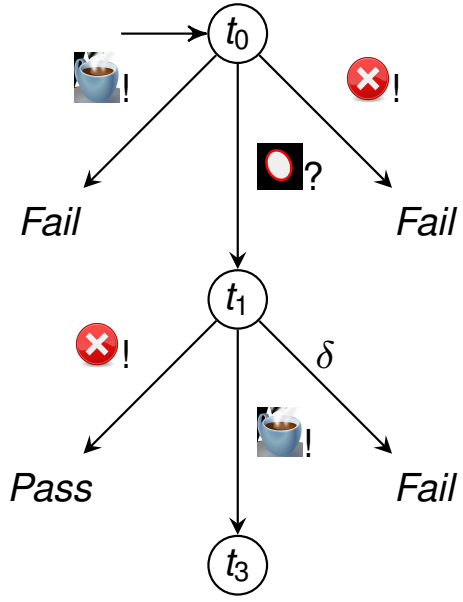
Batch test generation example

- Input LTS:



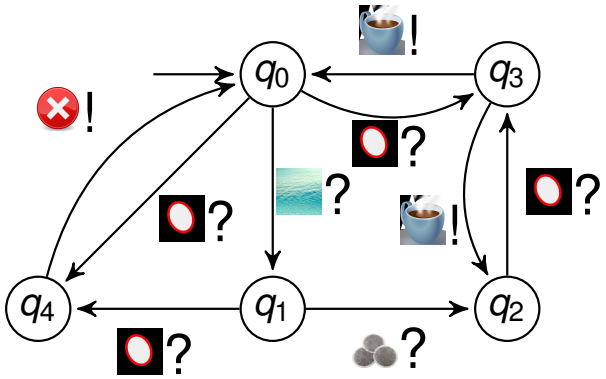
- Current states in t_0: $\{q_0\}$
- Choose input: ?
- Current states in t_1: $\{q_3, q_4\}$
- Observe outputs
- Current states in t_2: $\{q_0\}$
- Stop
- Current states in t_3: $\{q_0, q_2\}$

- Generated test case:



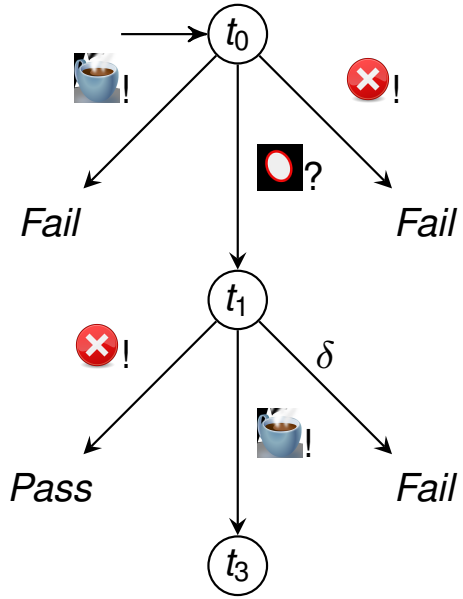
Batch test generation example

- Input LTS:



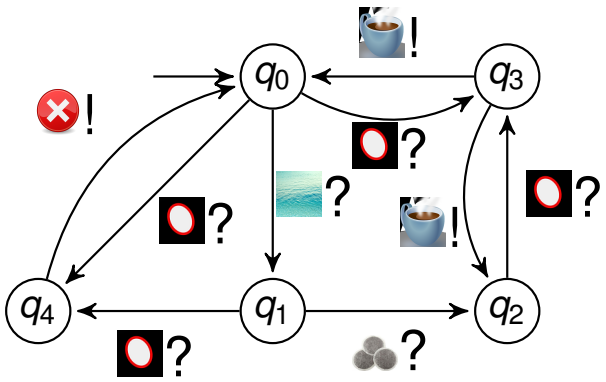
- Current states in t_0: $\{q_0\}$
- Choose input: ?
- Current states in t_1: $\{q_3, q_4\}$
- Observe outputs
- Current states in t_2: $\{q_0\}$
- Stop
- Current states in t_3: $\{q_0, q_2\}$
- Choose input: ?

- Generated test case:



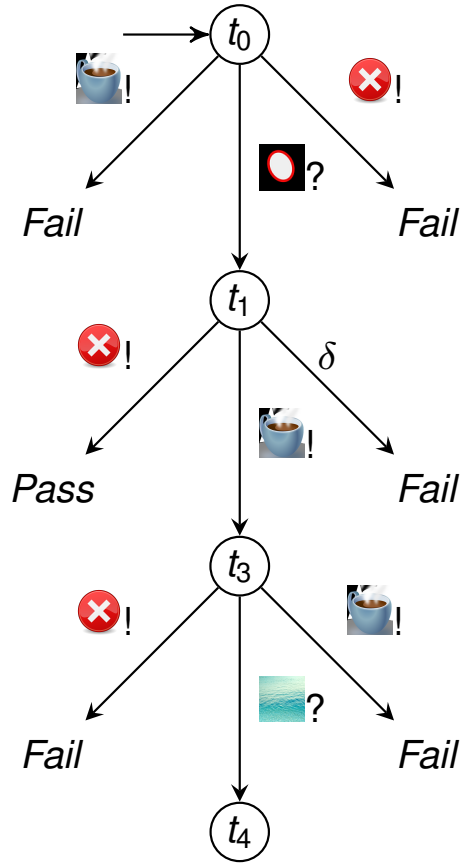
Batch test generation example

- Input LTS:



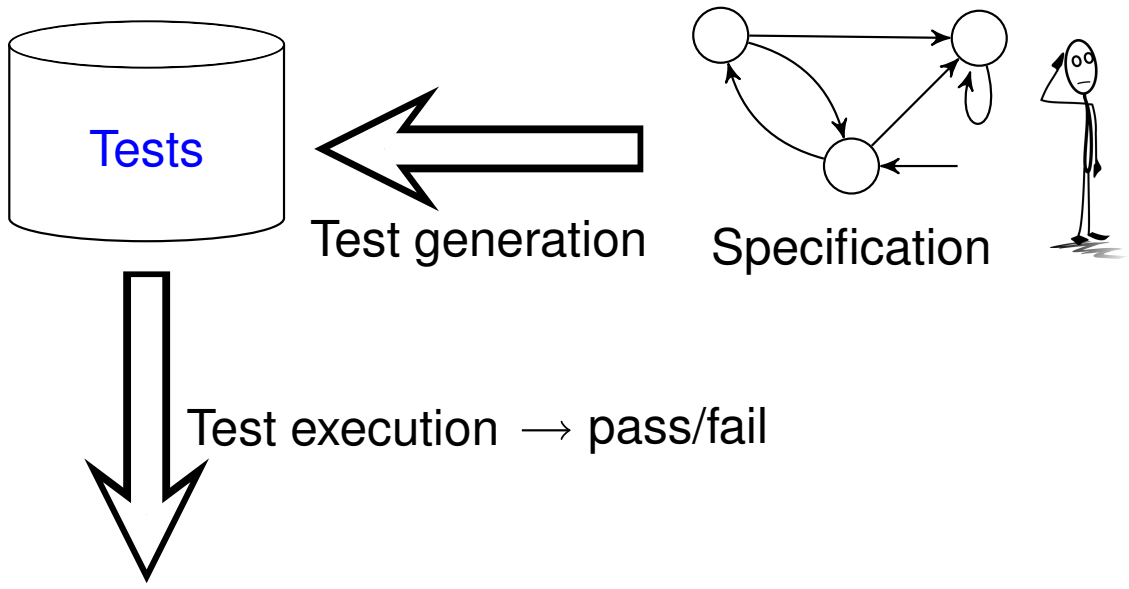
- Current states in t_0: $\{q_0\}$
- Choose input: ?
- Current states in t_1: $\{q_3, q_4\}$
- Observe outputs
- Current states in t_2: $\{q_0\}$
- Stop
- Current states in t_3: $\{q_0, q_2\}$
- Choose input: ?
- Current states in t_4: $\{q_1\}$ and ?

- Generated test case:



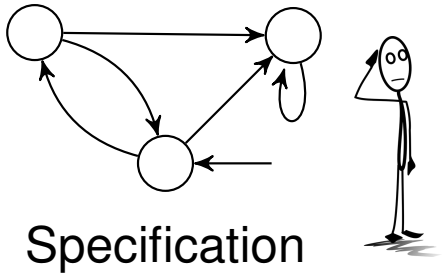
Test generation (batch)

- *batchGen* is a nondeterministic algorithm
  - Outputs different test cases
- Store test cases for execution



On-the-fly test generation

- *OnTheFlyGen* is also a nondeterministic algorithm
  - Outputs different test cases
- Generates a test step and executes it right away



Specification

Test generation and execution

pass/fail

System Under Test (SUT)

On-the-fly test generation

- Algorithm *onTheFlyGen*:
  - **Input:** LTS $S = (Q, L_I, L_U, T, q_0)$, and current states $Q' = \{q_0\}$
  - **Output:** a verdict *Pass* or *Fail*

On-the-fly test generation

- Algorithm *onTheFlyGen*:
  - **Input:** LTS $S = (Q, L_I, L_U, T, q_0)$, and current states $Q' = \{q_0\}$
  - **Output:** a verdict *Pass* or *Fail*
- *onTheFlyGen*$(S, Q') =$ take either of the following steps:

End test case	return verdict *Pass*

On-the-fly test generation

- Algorithm *onTheFlyGen*:
  - **Input:** LTS $S = (Q, L_I, L_U, T, q_0)$, and current states $Q' = \{q_0\}$
  - **Output:** a verdict *Pass* or *Fail*
- *onTheFlyGen*(S, Q') = take either of the following steps:

End test case	return verdict *Pass*
Observe output $x!$	if $x! \in out(Q')$: *onTheFlyGen*$(S, Q'$ *after* $x!)$ else: return *Fail*

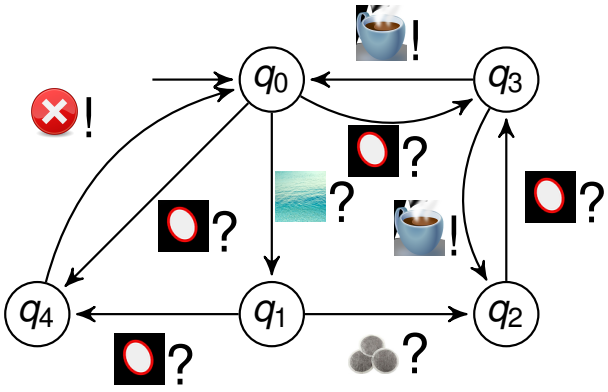
On-the-fly test generation

- Algorithm *onTheFlyGen*:
  - **Input:** LTS $S = (Q, L_I, L_U, T, q_0)$, and current states $Q' = \{q_0\}$
  - **Output:** a verdict *Pass* or *Fail*
- $onTheFlyGen(S, Q') =$ take either of the following steps:

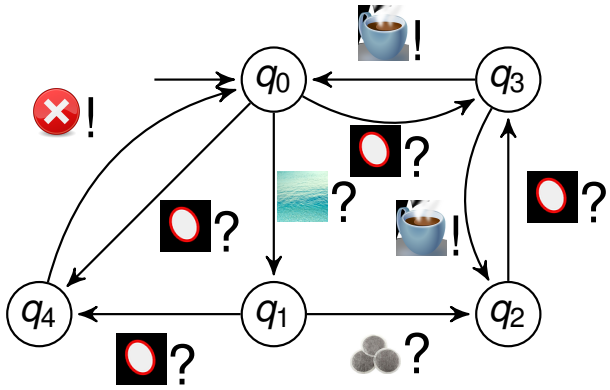
End test case	return verdict *Pass*
Observe output $x!$	if $x! \in out(Q')$: $onTheFlyGen(S, Q'\ after\ x!)$ else: return *Fail*

Choose input $a?$
where $Q'\ after\ a? \neq \emptyset$
	if SUT provides output $x!$ already:
	  handle output observation (as above)
	else:
	  supply $a?$ and continue with
	  $onTheFlyGen(S, Q'\ after\ a?)$

On-the-fly test generation example

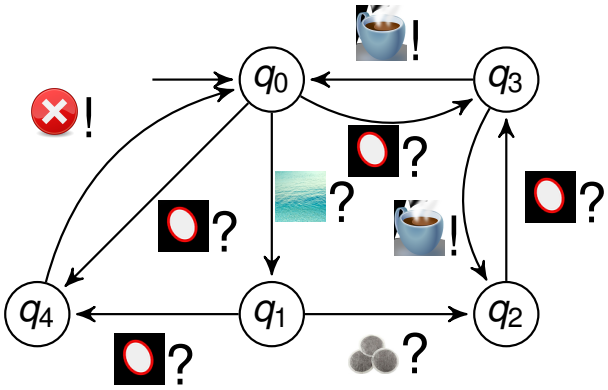


On-the-fly test generation example



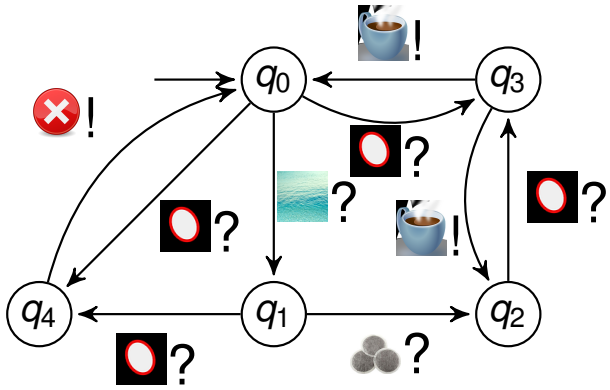
- Choose ?

On-the-fly test generation example



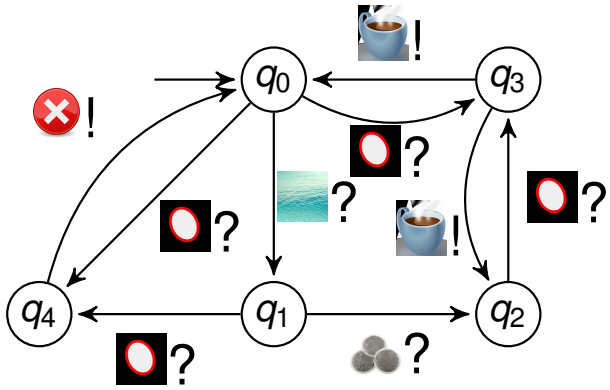
- Choose ?
- SUT accepts input

On-the-fly test generation example



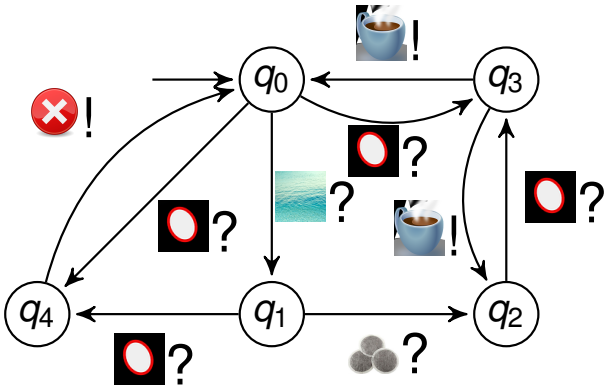
- Choose ?
- SUT accepts input
- Current states: $\{q_3, q_4\}$

On-the-fly test generation example



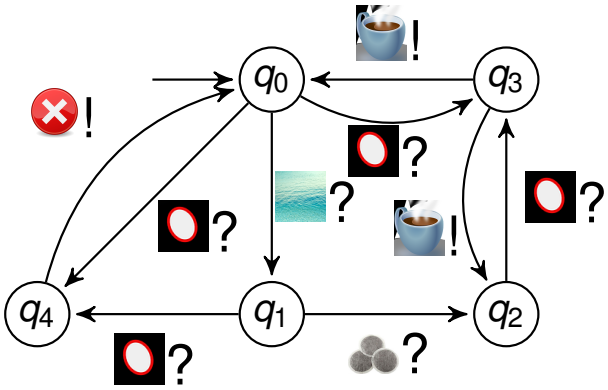
- Choose ?
- SUT accepts input
- Current states: $\{q_3, q_4\}$
- Observe output

On-the-fly test generation example



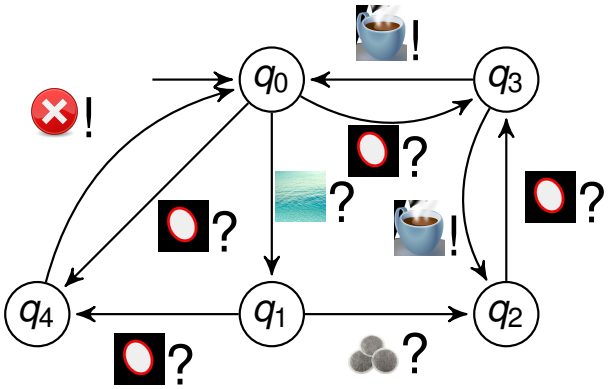
- Choose ?
- SUT accepts input
- Current states: $\{q_3, q_4\}$
- Observe output
- SUT provides !

On-the-fly test generation example



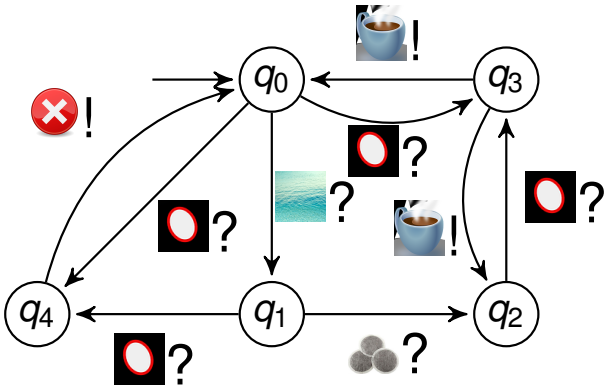
- Choose ?
- SUT accepts input
- Current states: $\{q_3, q_4\}$
- Observe output
- SUT provides !
- Current states: $\{q_0, q_2\}$

On-the-fly test generation example



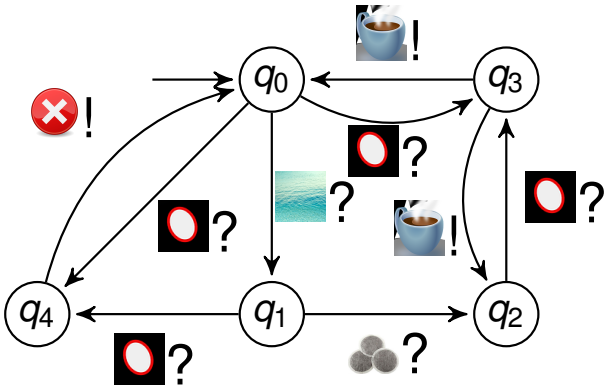
- Choose ?
- SUT accepts input
- Current states: $\{q_3, q_4\}$
- Observe output
- SUT provides !
- Current states: $\{q_0, q_2\}$
- Observe output

On-the-fly test generation example



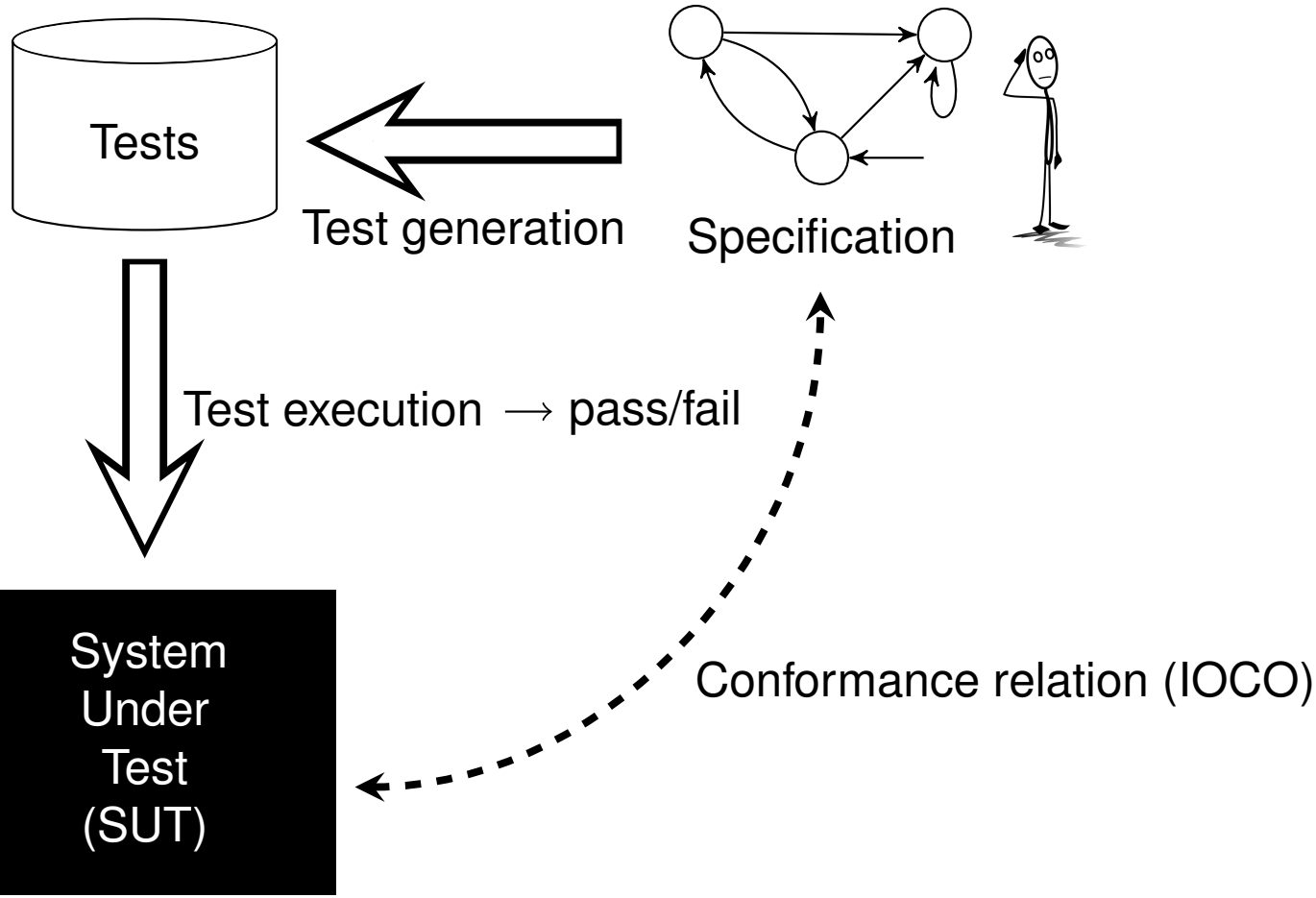
- Choose ?
- SUT accepts input
- Current states: $\{q_3, q_4\}$
- Observe output
- SUT provides !
- Current states: $\{q_0, q_2\}$
- Observe output
- SUT provides !

On-the-fly test generation example

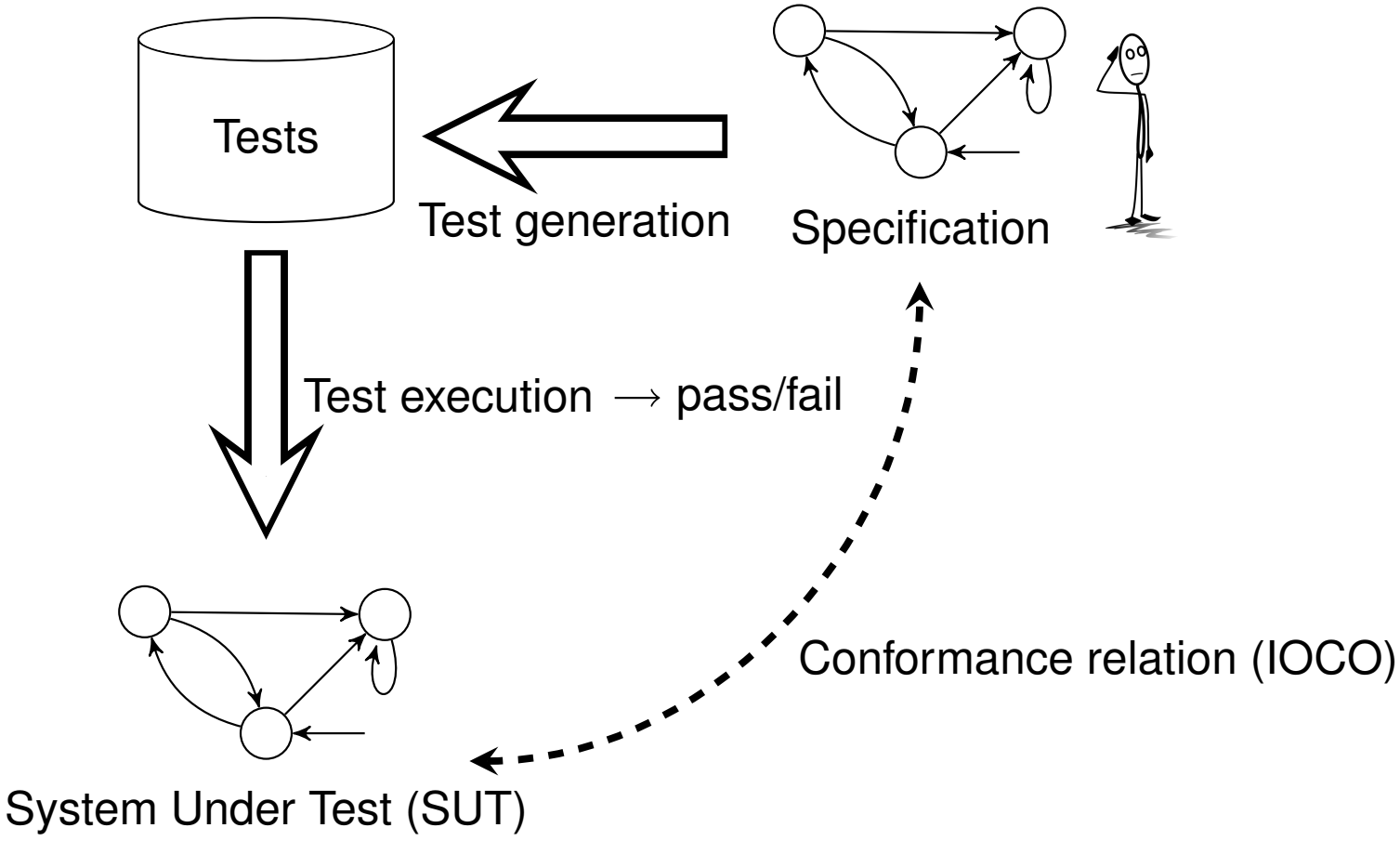


- Choose ?
- SUT accepts input
- Current states: $\{q_3, q_4\}$
- Observe output
- SUT provides !
- Current states: $\{q_0, q_2\}$
- Observe output
- SUT provides !
- Stop with verdict *Fail*

Model-Based Testing

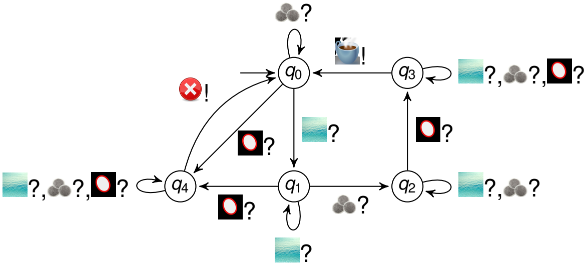


Test hypothesis



Test assumptions on SUT for ioco

- Test hypothesis: the SUT can be modeled as an LTS (or STS)
- The SUT accepts any input (unless it is providing an output)
  - Model SUT with IELTS: input-enabled LTS, e.g.:



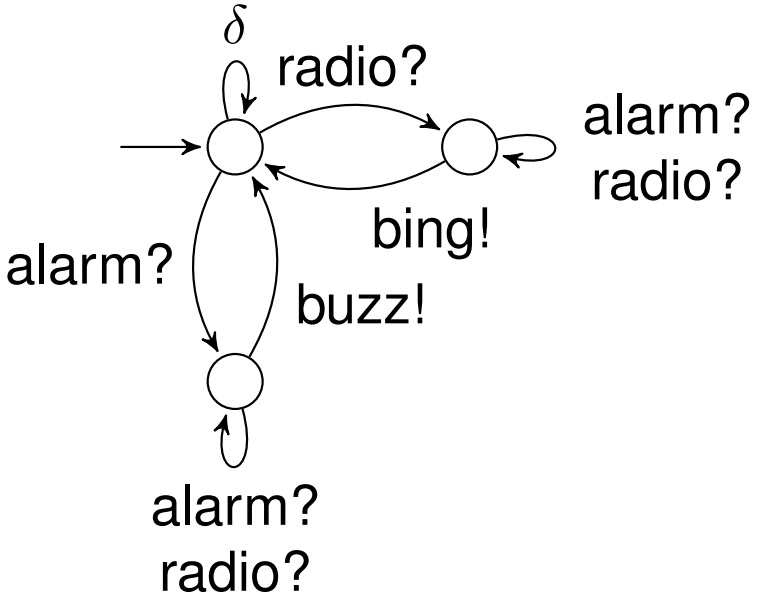
  - Formally: An LTS $S = (Q, L_I, L_U, T, q_0)$ is input-enabled if $\forall q \in Q : in(q) = L_I$
- Optional:
  - Fairness: if the SUT can provide an output, it will eventually

ioco

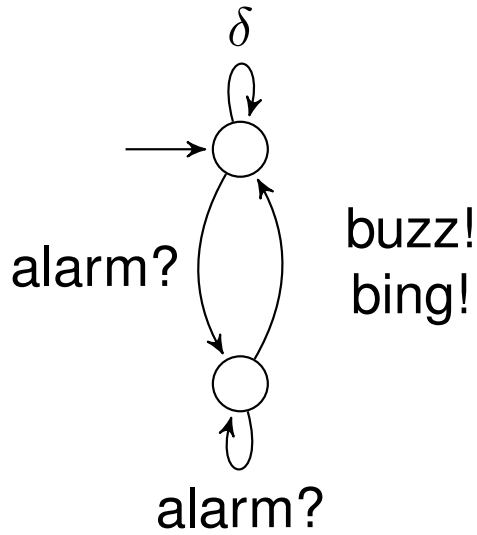
- IOCO: Input Output COnformance
- ioco : IELTS $\times$ LTS $\rightarrow$ Bool
- Write: *I* ioco *S*
- Intuition: what does *I* ioco *S* mean?
  - *I* allows the same or more inputs than *S*
  - *I* provides the same or fewer outputs than *S*

More inputs and fewer outputs for ioco

Implementation



Specification



Write S instead of q_0

- Let $S = (Q, L_I, L_U, T, q_0)$ be an LTS, $\rho \in L^*$
- $Straces(S) = Straces(q_0)$
- $S \text{ after } \rho = q_0 \text{ after } \rho$
- $out(S) = out(q_0)$
- $in(S) = in(q_0)$

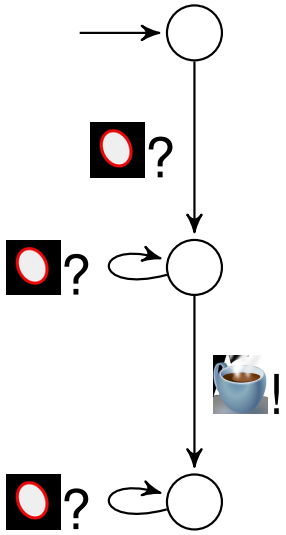
Formal definition for ioco

- Let $S = (Q^s, L_I, L_U, T^s, q_0^s)$ be an LTS
- Let $I = (Q^i, L_I, L_U, T^i, q_0^i)$ be an IELTS
  - Note: same label sets!
- Then ioco : IELTS $\times$ LTS $\rightarrow$ Bool is defined as:
  I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$
  - Note '$\subseteq$': If I can produce an output $x!$ after ρ then S can produce the output $x!$ after ρ as well

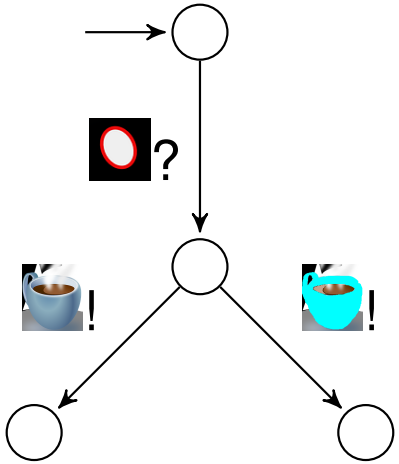
Example 1: ioco?

- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

Implementation I



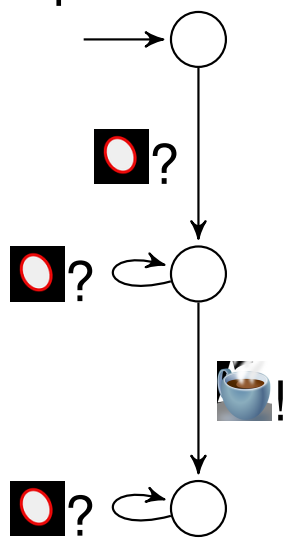
Specification S



Example 1: ioco?

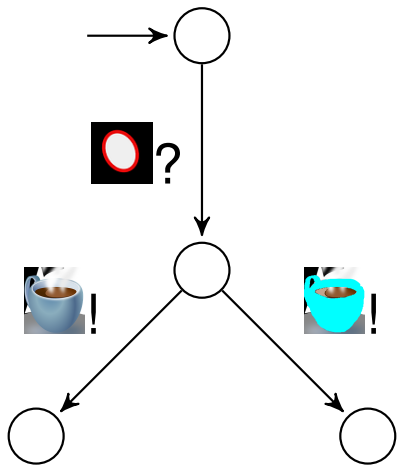
- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

Implementation I



$out(I \text{ after } ?) = \{!\}$

Specification S

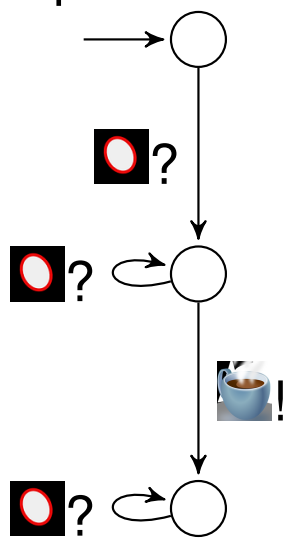


$out(I \text{ after } ?) = \{!, !\}$

Example 1: ioco?

- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

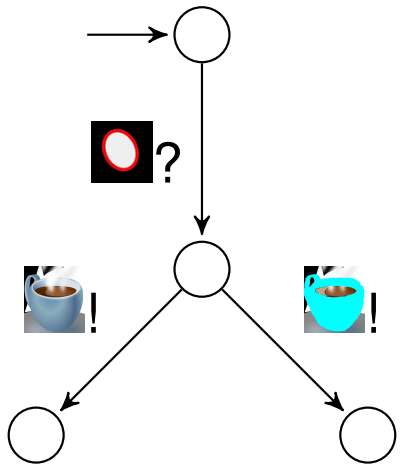
Implementation I



$out(I \text{ after } ?) = \{!\} \subseteq$

ioco

Specification S

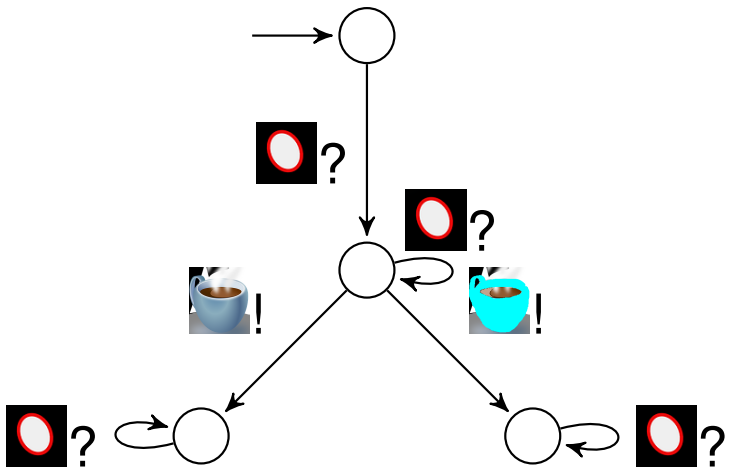


$out(I \text{ after } ?) = \{!, !\}$

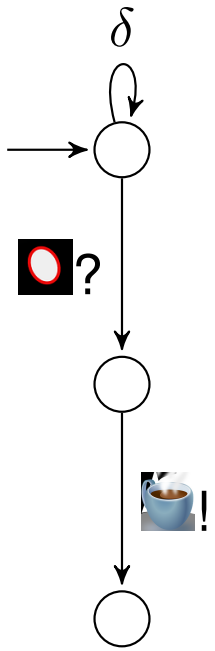
Example 2: ioco?

- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

Implementation I



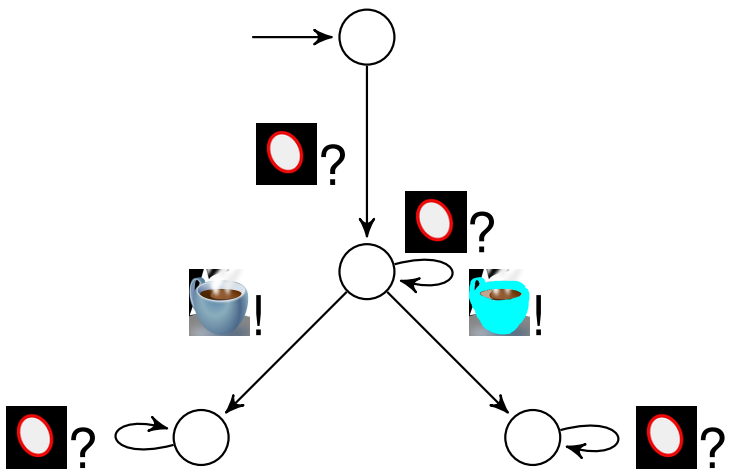
Specification S



Example 2: ioco?

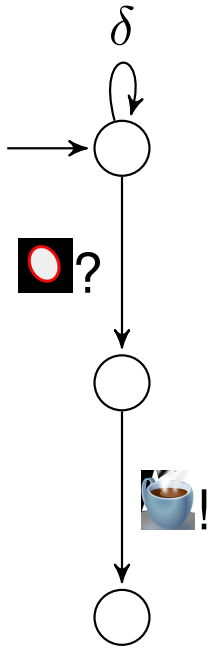
- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

Implementation I



$out(I \text{ after } ?) = \{!, !\}$

Specification S

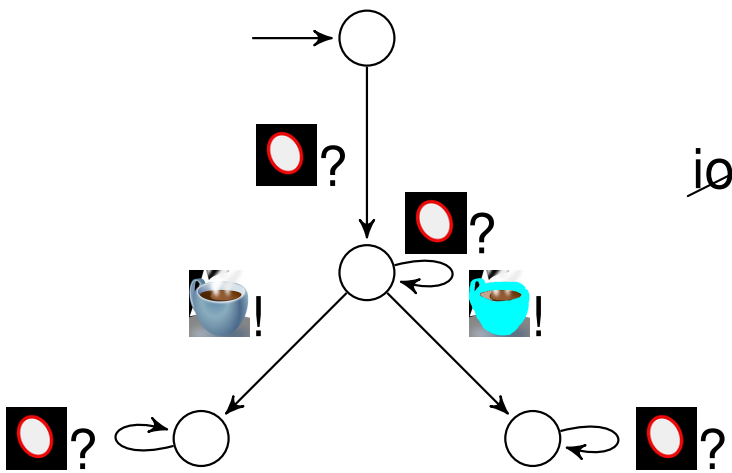


$out(I \text{ after } ?) = \{!\}$

Example 2: ioco?

- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

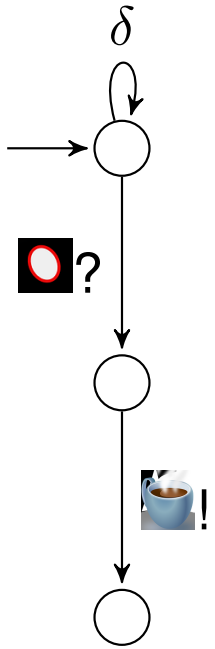
Implementation I



$out(I \text{ after } ?) = \{!, !\} \not\subseteq$

ioco

Specification S

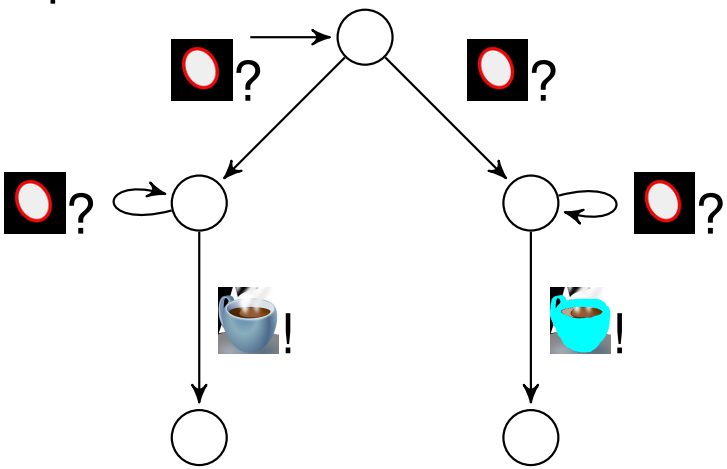


$out(I \text{ after } ?) = \{!\}$

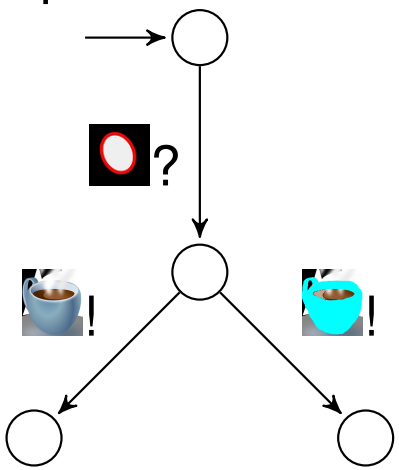
Example 3: ioco?

- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

Implementation I



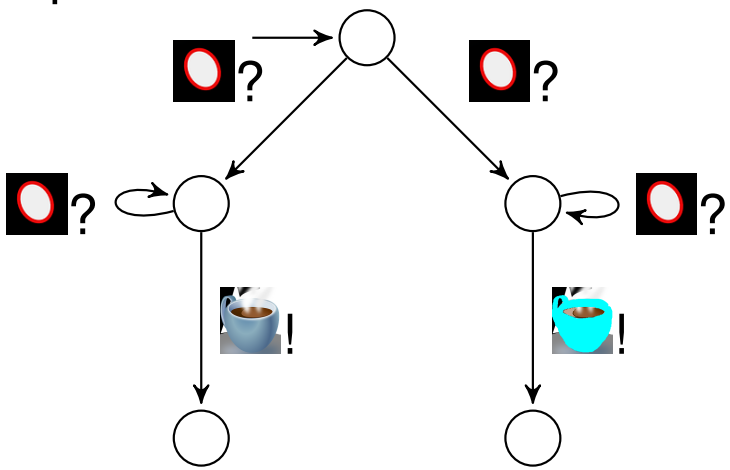
Specification S



Example 3: ioco?

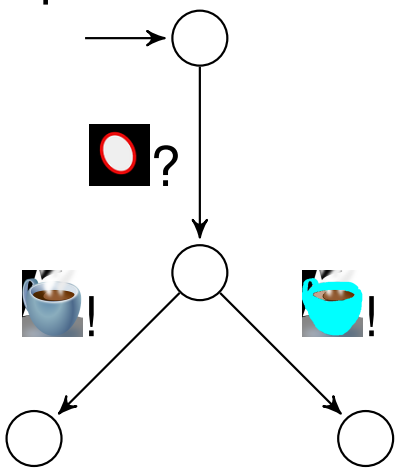
- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

Implementation I



$out(I \text{ after } ?) = \{!, !\}$

Specification S

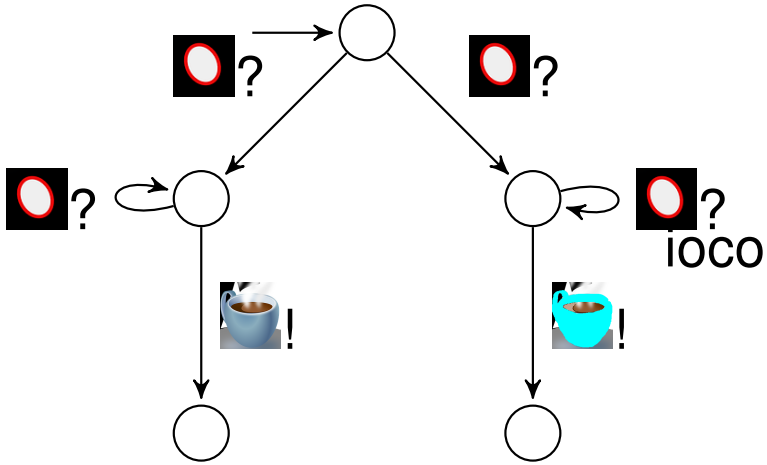


$out(I \text{ after } ?) = \{!, !\}$

Example 3: ioco?

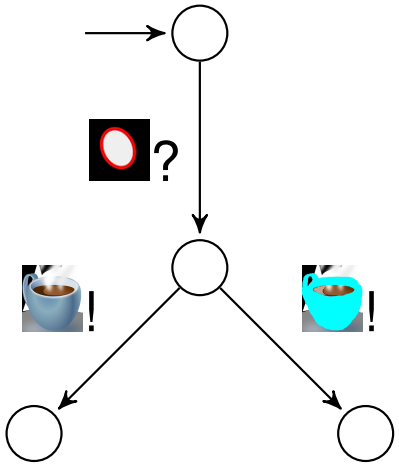
- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

Implementation I



ioco

Specification S

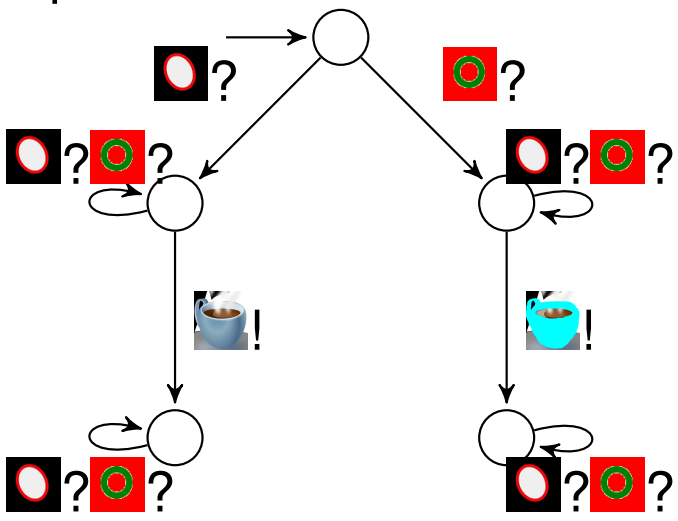


$out(I \text{ after } ?) = \{!, !\} \subseteq out(I \text{ after } ?) = \{!, !\}$

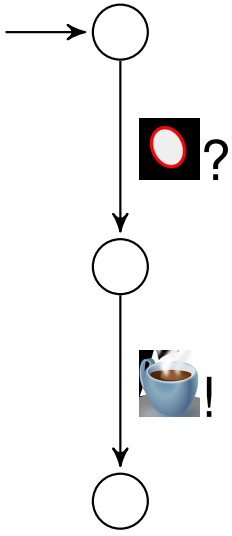
Example 4: ioco?

- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

Implementation I



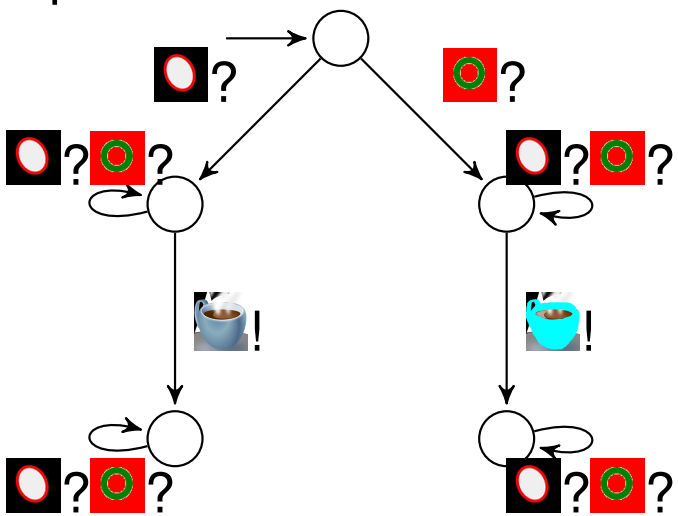
Specification S



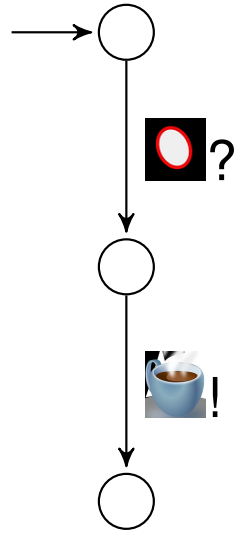
Example 4: ioco?

- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

Implementation I



Specification S



$out(I \text{ after } ?) = \{!\}$

$out(I \text{ after } ?) = \{!\}$

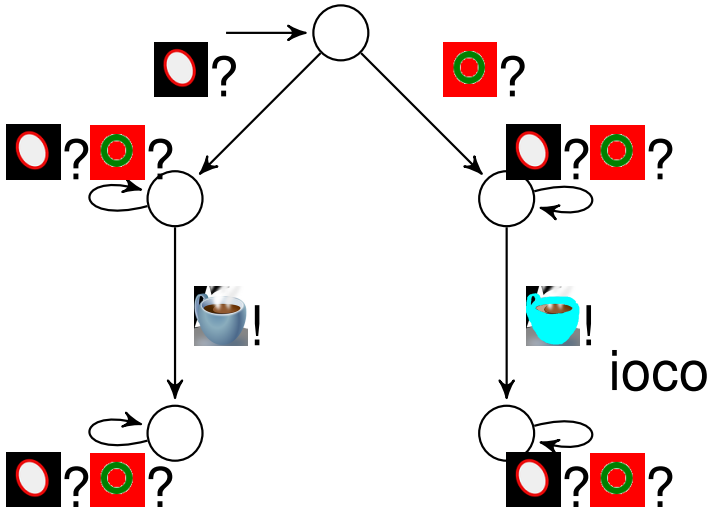
$out(I \text{ after } ?) = \{!\}$

$out(I \text{ after } ?) = \emptyset$

Example 4: ioco?

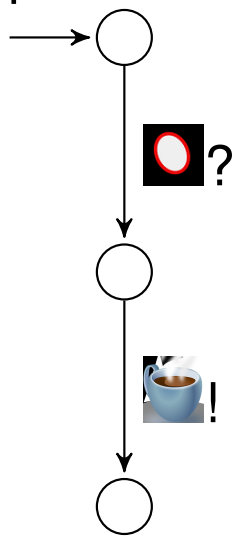
- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

Implementation I



ioco

Specification S



$out(I \text{ after } ?) = \{!\}$

$out(I \text{ after } ?) = \{!\} \not\subseteq$

but ? $\notin Straces(S)$

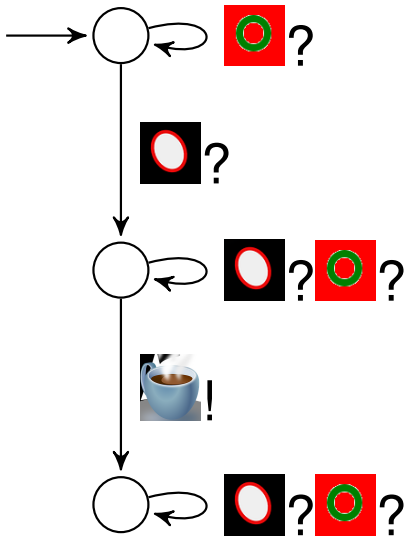
$out(I \text{ after } ?) = \{!\}$

$out(I \text{ after } ?) = \emptyset$

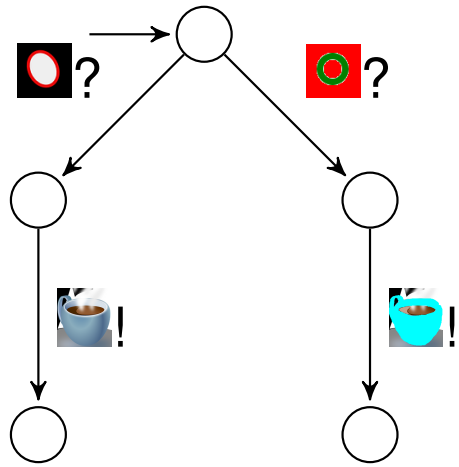
Example 5: ioco?

- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

Implementation I



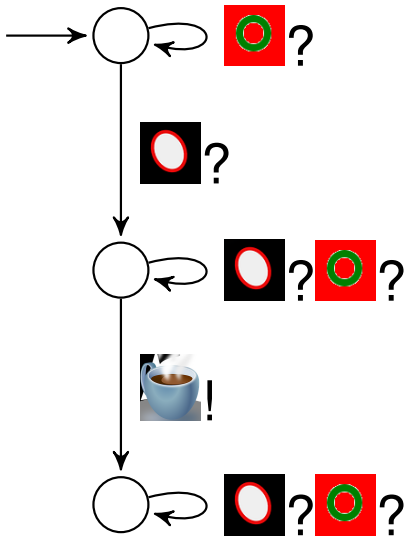
Specification S



Example 5: ioco?

- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

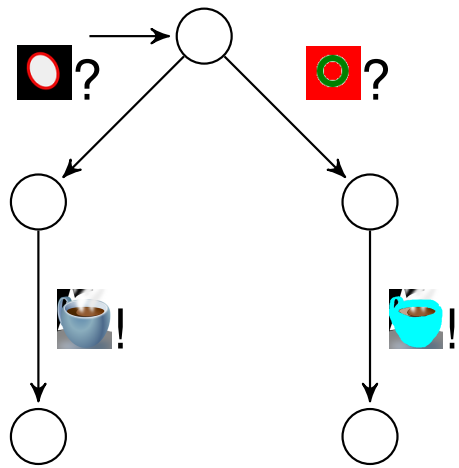
Implementation I



$out(I \text{ after } ?) = \{!\}$

$out(I \text{ after } ?) = \{\delta\}$

Specification S



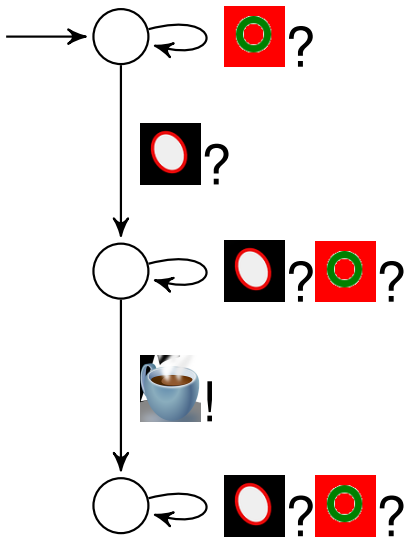
$out(I \text{ after } ?) = \{!\}$

$out(I \text{ after } ?) = \{!\}$

Example 5: ioco?

- I ioco $S = \forall \rho \in Straces(S) : out(I \text{ after } \rho) \subseteq out(S \text{ after } \rho)$

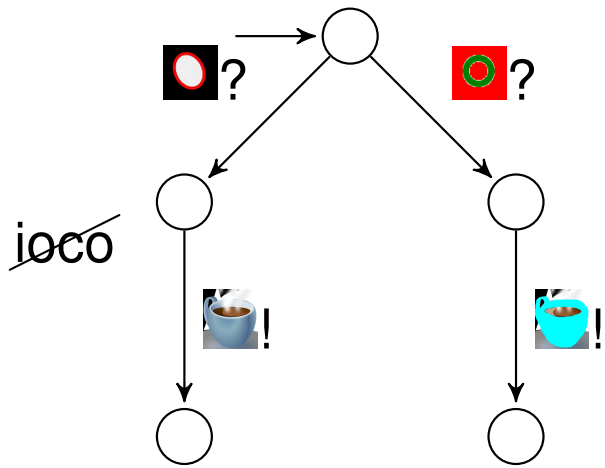
Implementation I



$out(I \text{ after } ?) = \{!\}$

$out(I \text{ after } ?) = \{\delta\} \not\subseteq$

Specification S



$out(I \text{ after } ?) = \{!\}$

$out(I \text{ after } ?) = \{!\}$

Soundness

- When to put *Pass* or *Fail* at end node of test case?

Soundness

- When to put *Pass* or *Fail* at end node of test case?
- A test case t for an LTS S is sound if:
  for any IELTS I:
  I ioco S implies that execution of t on I yields *Pass*

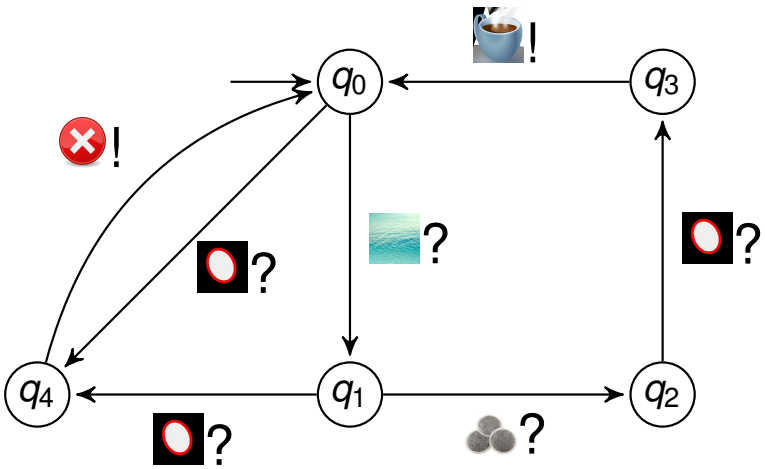
Soundness

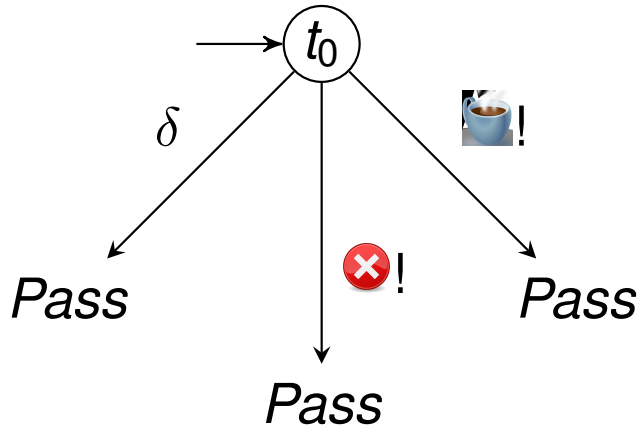
- When to put *Pass* or *Fail* at end node of test case?
- A test case *t* for an LTS *S* is sound if:
  for any IELTS *I*:
    I ioco *S* implies that execution of *t* on *I* yields *Pass*
- Or alternatively formulated as follows:
  for any IELTS *I* :
    execution of *t* on *I* yields *Fail* implies that *I* ~~ioco~~ *S*

Soundness

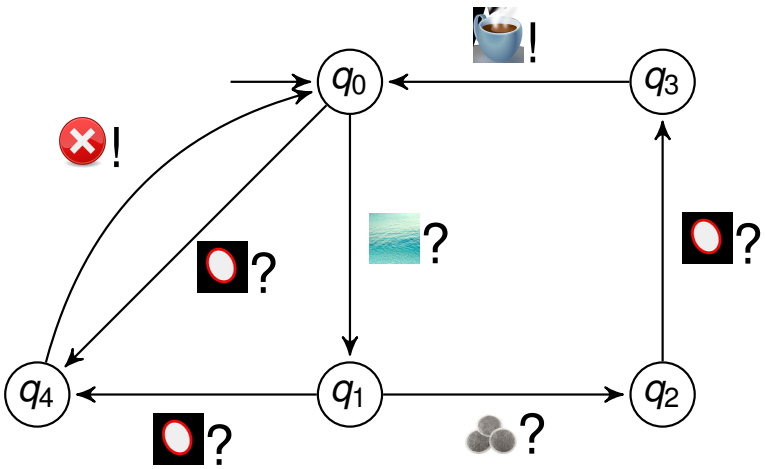
- When to put *Pass* or *Fail* at end node of test case?
- A test case t for an LTS S is sound if:
  for any IELTS I:
  I ioco S implies that execution of t on I yields *Pass*
- Or alternatively formulated as follows:
  for any IELTS I :
  execution of t on I yields *Fail* implies that $I \not\text{ioco}\ S$
- A test suite TS is sound if all its test cases $t \in TS$ are sound

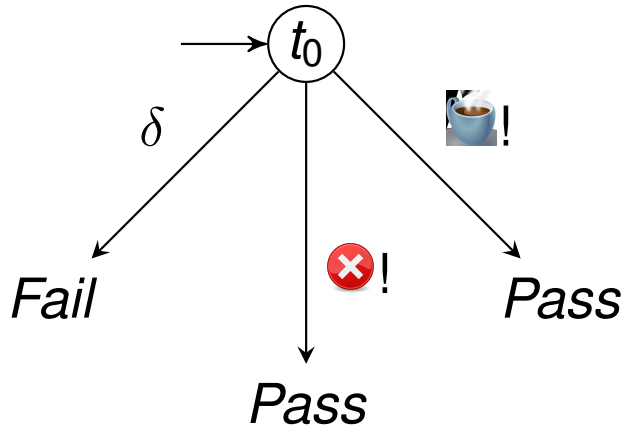
Is this test case sound?





Is this test case sound?





Exhaustiveness

- A test suite *TS* for LTS *S* is exhaustive if:
  for any IELTS *I* :
  I ~~ioco~~ *S* implies that:
  there is a $t \in TS$ such that execution of *t* on *I* yields *Fail*

Exhaustiveness

- A test suite *TS* for LTS *S* is exhaustive if:
  for any IELTS *I* :
  I ~~ioco~~ *S* implies that:
  there is a $t \in TS$ such that execution of *t* on *I* yields *Fail*
- Q: how many test cases in an exhaustive test suite?

Exhaustiveness

- A test suite *TS* for LTS *S* is exhaustive if:
  for any IELTS *I* :
  I ~~ioco~~ *S* implies that:
  there is a $t \in TS$ such that execution of *t* on *I* yields *Fail*
- Q: how many test cases in an exhaustive test suite?
  - Consider implementation with trace $(?!)^n \; ?\delta$ for any $n \in \mathbb{N}$

Exhaustiveness

- A test suite *TS* for LTS *S* is exhaustive if:
  for any IELTS *I* :
  I ~~ioco~~ *S* implies that:
  there is a $t \in TS$ such that execution of *t* on *I* yields *Fail*
- Q: how many test cases in an exhaustive test suite?
  - Consider implementation with trace $(?!)^n \; ?\delta$ for any $n \in \mathbb{N}$
- A test suite *TS* is complete for LTS *S* if it is sound and exhaustive for *S*

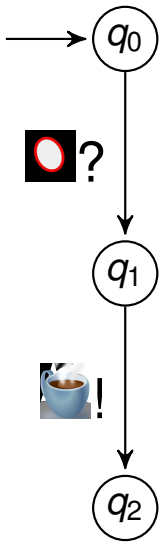
Making underspecification explicit

- Let $S = (Q, L_I, L_U, T, q_0)$ be an LTS.
- Then $a? \in L_I$ is underspecified in $q \in Q$ if $a \notin in(q)$

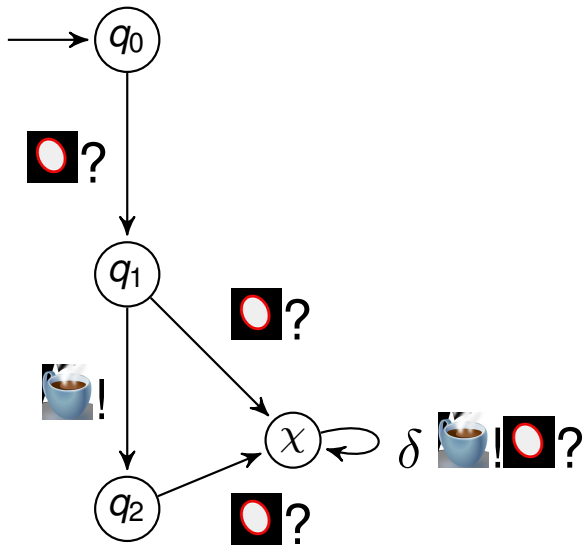
Making underspecification explicit

- Let $S = (Q, L_I, L_U, T, q_0)$ be an LTS.
- Then $a? \in L_I$ is underspecified in $q \in Q$ if $a \notin in(q)$
- Demonic completion:
  - Add χ-state (chaos, where anything is allowed)
  - Add a transition to χ for all underspecified inputs

Demonic completion example



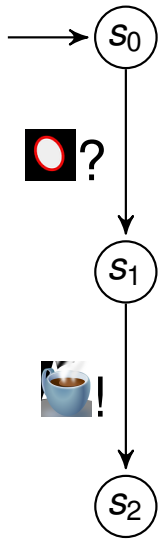
Demonic completion example



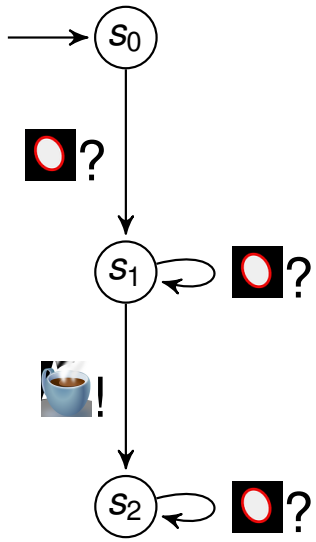
Angelic completion

- Add a self-loop for all underspecified inputs
- Does not preserve ioco!

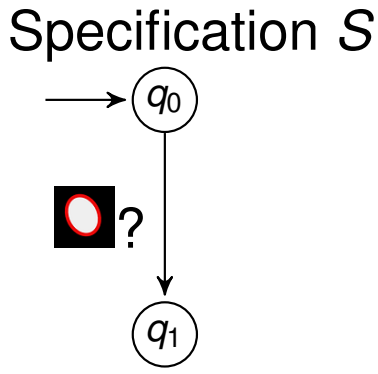
Angelic completion example



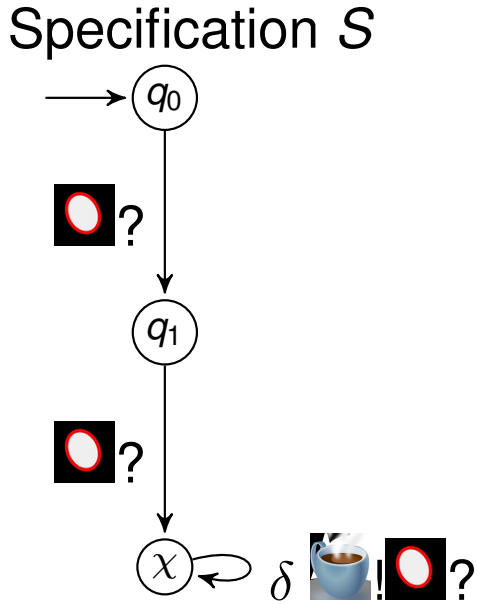
Angelic completion example



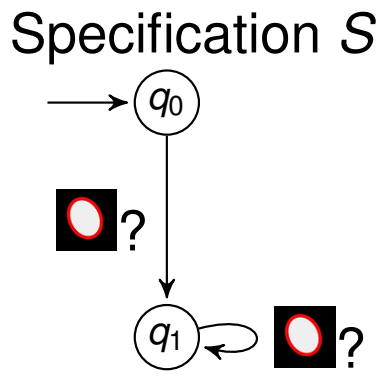
Angelic completion counterexample for ioco



Angelic completion counterexample for ioco

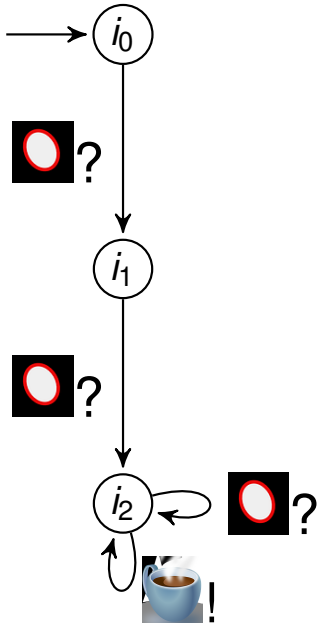


Angelic completion counterexample for ioco

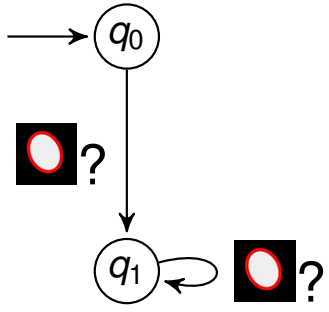


Angelic completion counterexample for ioco

Implementation *I*

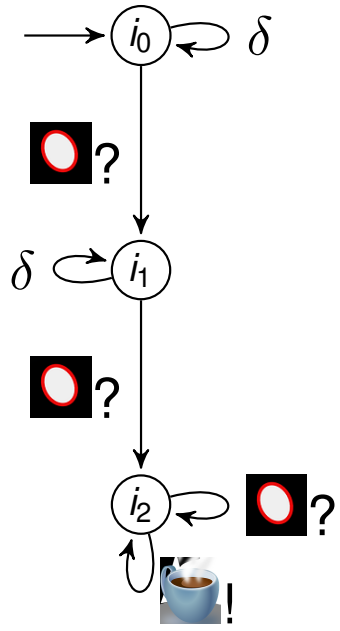


Specification *S*

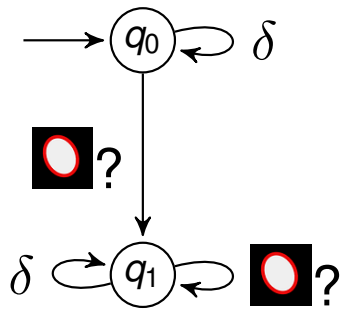


Angelic completion counterexample for ioco

Implementation *I*

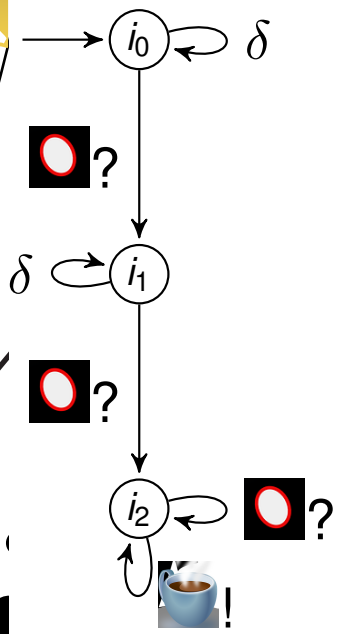


Specification *S*

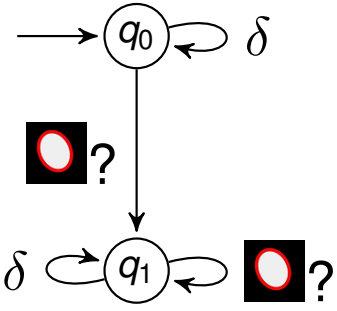


Angelic completion counterexample for ioco

Implementation *I*



Specification *S*



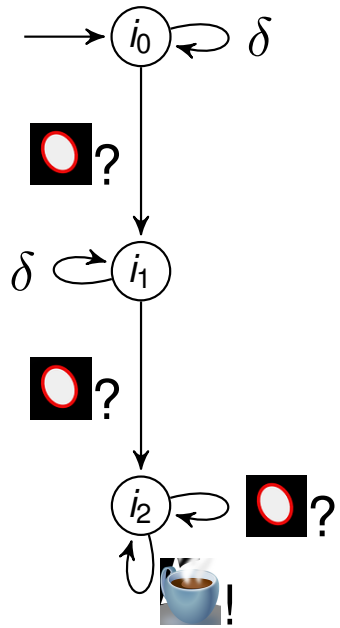
~~ioco~~

$out(S \text{ after } ??) = \{!\}$

$out(S \text{ after } ??) = \{\delta\}$

Angelic completion counterexample for ioco

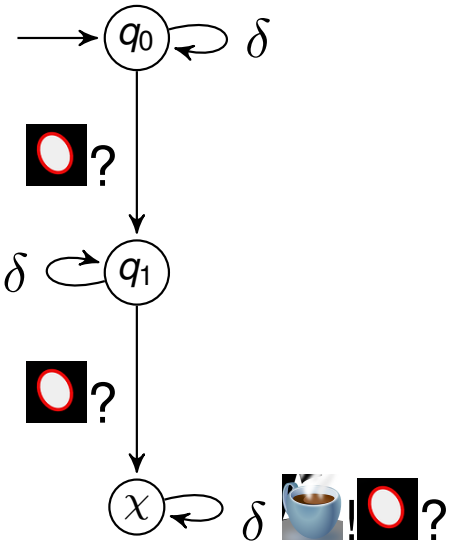
Implementation *I*



$out(S \text{ after } ??) = \{$!$\}$

ioco

Specification *S*



$out(S \text{ after } ??) = \{\delta,$!$\}$

ioco is not input-universal [1]

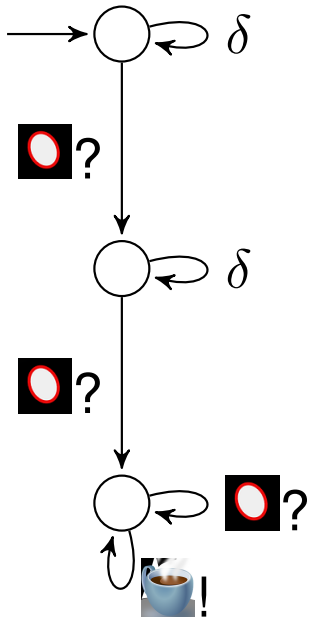
- Edge case of ioco: combine nondeterminism and underspecification

[1] R. Janssen, F. W. Vaandrager, J. Tretmans: Relating Alternating Relations for Conformance and Refinement. https://doi.org/10.1007/978-3-030-34968-4_14

ioco is not input-universal [1]

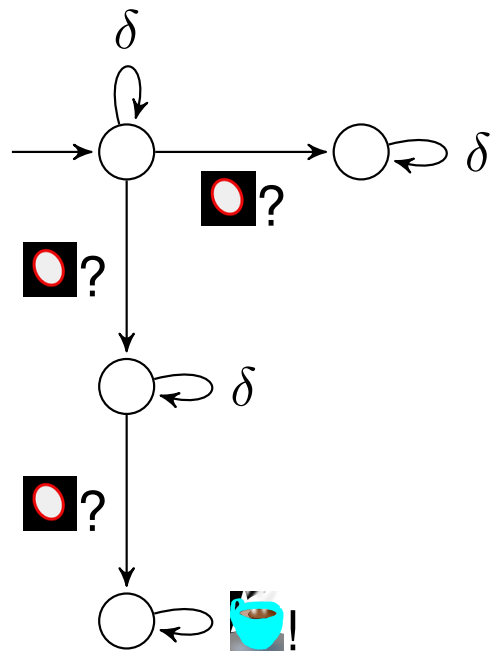
- Edge case of ioco: combine nondeterminism and underspecification

Implementation *I*



$out(S \text{ after } ??) = \{!\}$

Specification *S*



~~ioco~~
uioco

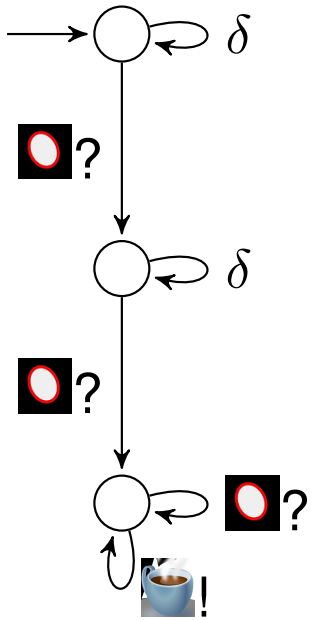
$out(S \text{ after } ??) = \{!\}$

[1] R. Janssen, F. W. Vaandrager, J. Tretmans: Relating Alternating Relations for Conformance and Refinement. https://doi.org/10.1007/978-3-030-34968-4_14

ioco is not input-universal [1]

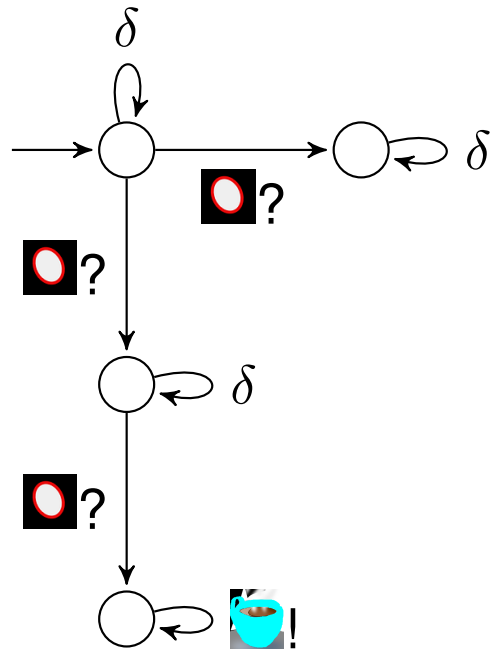
- Edge case of ioco: combine nondeterminism and underspecification

Implementation *I*



out(*S after* ??) = {!}

Specification *S*



~~ioco~~

out(*S after* ??) = {!}

- *I uioco S* = $\forall \rho \in$ *Utraces*(*S*) :*out*(*I after* ρ) $\subseteq$ *out*(*S after* ρ)
- ?? $\notin$ *Utraces*(*S*)

[1] R. Janssen, F. W. Vaandrager, J. Tretmans: Relating Alternating Relations for Conformance and Refinement. https://doi.org/10.1007/978-3-030-34968-4_14

Next times

- Tutorial next Monday
- Lecture next Wednesday: Symbolic Transition Systems