Black Box Testing of Finite State Machines

Frits Vaandrager

Radboud University Nijmegen

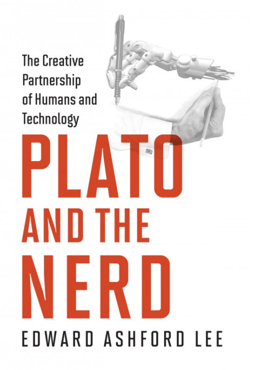
Testing Techniques Course
December 13, 2024

Outline

Learning and Testing

Theme of the last three lectures in TT course:

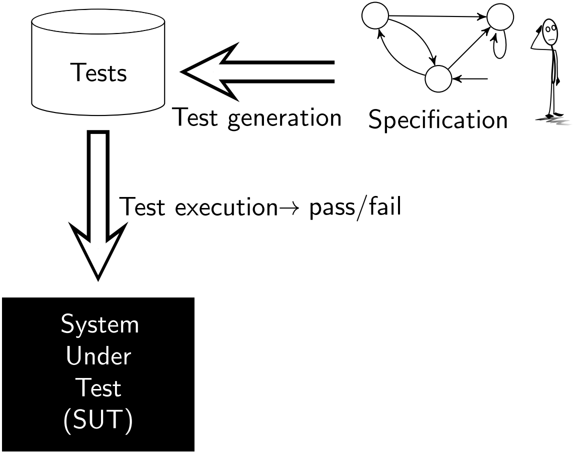
- Duality of learning and testing (Weyuker)
- Science models vs engineering models (Lee)



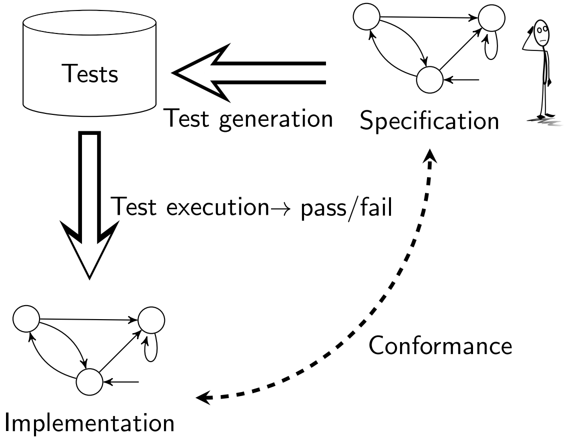
Black Box Testing

A method of software testing that examines the functionality of an application without peering into its internal structures or workings.

Model Based Testing



Model Based Testing



Today We Will Address Three Questions

1. What is a model?
   - Today: a Finite State Machine (FSM)

Today We Will Address Three Questions

1. What is a model?
   - Today: a Finite State Machine (FSM)
2. What is conformance?
   - Today: FSM equivalence

Today We Will Address Three Questions

1. What is a model?
   - Today: a Finite State Machine (FSM)
2. What is conformance?
   - Today: FSM equivalence
3. How can we construct a good test suite?
   - Today: a k-complete test suite for FSMs

Today We Will Address Three Questions

1. What is a model?
   - Today: a Finite State Machine (FSM)
2. What is conformance?
   - Today: FSM equivalence
3. How can we construct a good test suite?
   - Today: a k-complete test suite for FSMs

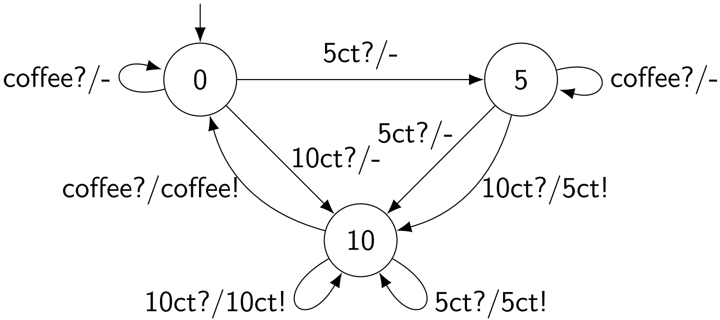
Literature:

- Dorofeeva, R., et al. FSM-based conformance testing methods — A survey annotated with experimental evaluation. *Information and Software Technology*, 2010, 52.12: 1286-1297.
- Ural, H. Formal methods for test sequence generation. *Computer communications*, 1992, 15.5: 311-325.
- Lee, D. & Yannakakis, M. Principles and methods of testing finite state machines. *Proc. IEEE*, 1996, 84.8: 1090-1123.

FSM

An Finite State Machine (FSM) (or Mealy machine) consists of:

- states
- transitions
- inputs
- outputs



What Can Be Modeled With FSMs?

- FSMs model functional behavior of reactive systems
- Examples:
  - communication protocols: TCP, SSH, TLS,...
  - hardware circuits
  - web applications
  - embedded control software within printers, cars, X-ray scanners, lithography systems, elevators, thermostats, ...
  - ...

FSMs are Quite Restrictive!

1. Each input triggers exactly one output
2. Source state and input uniquely determine target state (determinism)
3. Only finitely many states, inputs and outputs
4. No data parameters

Formal Definition

An FSM (Mealy machine) is a 6-tuple $M = (Q, q_0, I, O, \delta, \lambda)$ with:

- Q a finite set of states
- q_0 the initial state
- I a finite set of inputs
- O a finite set of outputs
- $\delta : Q \times I \to Q$ the transition function
- $\lambda : Q \times I \to O$ the output function

Formal Definition

An FSM (Mealy machine) is a 6-tuple $M = (Q, q_0, I, O, \delta, \lambda)$ with:

- Q a finite set of states
- q_0 the initial state
- I a finite set of inputs
- O a finite set of outputs
- $\delta : Q \times I \to Q$ the transition function
- $\lambda : Q \times I \to O$ the output function

$Q \rightarrow$	0		5		10	
$I \downarrow$	λ	δ	λ	δ	λ	δ
5ct	-	5	-	10	5ct	10
10ct	-	10	5ct	10	10ct	10
coffee	-	0	-	5	coffee	0

Extension of Transition and Output Functions

Extend δ and λ to sequences: $\delta^* : Q \times I^* \rightarrow Q$ and $\lambda^* : Q \times I^* \rightarrow O^*$:

$$\begin{aligned} \delta^*(q, \epsilon) &= q \\ \delta^*(q, \mu \cdot \sigma) &= \delta^*(\delta(q, \mu), \sigma) \qquad (\mu \in I \text{ is a single symbol}) \end{aligned}$$

$$\begin{aligned} \lambda^*(q, \epsilon) &= \epsilon \\ \lambda^*(q, \mu \cdot \sigma) &= \lambda(q, \mu) \cdot \lambda^*(\delta(q, \mu), \sigma) \end{aligned}$$

Extension of Transition and Output Functions

Extend δ and λ to sequences: $\delta^* : Q \times I^* \to Q$ and $\lambda^* : Q \times I^* \to O^*$:

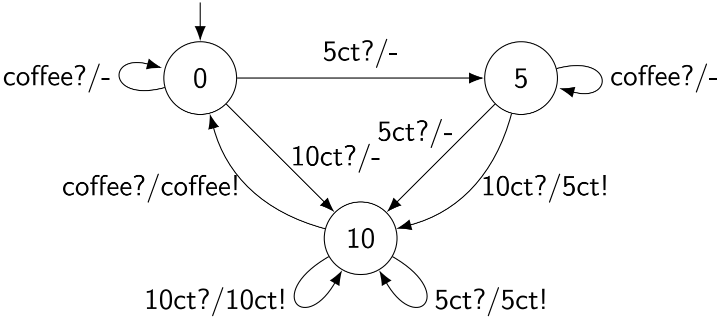
$$\begin{aligned} \delta^*(q, \epsilon) &= q \\ \delta^*(q, \mu \cdot \sigma) &= \delta^*(\delta(q, \mu), \sigma) \qquad (\mu \in I \text{ is a single symbol}) \end{aligned}$$

$$\begin{aligned} \lambda^*(q, \epsilon) &= \epsilon \\ \lambda^*(q, \mu \cdot \sigma) &= \lambda(q, \mu) \cdot \lambda^*(\delta(q, \mu), \sigma) \end{aligned}$$

For FMS M with initial state q_0, we write:

$$\begin{aligned} \delta^*(M, \sigma) &= \delta^*(q_0, \sigma) \\ \lambda^*(M, \sigma) &= \lambda^*(q_0, \sigma) \end{aligned}$$

Extension of Transition and Output Functions



$$\delta^*(M, \text{5ct? 10ct? coffee?}) = 0$$
$$\lambda^*(M, \text{5ct? 10ct? coffee?}) = \text{- 5ct! coffee!}$$

FSM Restrictions

FSMs are:

- deterministic: δ and λ, δ^* and λ^* are functions

FSM Restrictions

FSMs are:

- deterministic: δ and λ, δ^* and λ^* are functions
- completely specified: δ, λ, δ^* and λ^* are complete functions
  - Symbol '-' in the coffee machine is an artificial output

FSM Restrictions

FSMs are:

- deterministic: δ and λ, δ^* and λ^* are functions
- completely specified: δ, λ, δ^* and λ^* are complete functions
  - Symbol '-' in the coffee machine is an artificial output
- connected: from initial state any other state can be reached
  - Every non-connected FSM can be rewritten to a connected FSM

Equivalence

- States q, q' are equivalent if they produce the same output sequence for every input sequence:

$$\forall \sigma \in I^* : \lambda^*(q, \sigma) = \lambda^*(q', \sigma)$$

Equivalence

- States q, q' are equivalent if they produce the same output sequence for every input sequence:

$$\forall \sigma \in I^* : \lambda^*(q, \sigma) = \lambda^*(q', \sigma)$$

- States q, q' are inequivalent if there is a separating sequence:

$$\exists \sigma \in I^* : \lambda^*(q, \sigma) \neq \lambda^*(q', \sigma)$$

Equivalence

- States q, q' are equivalent if they produce the same output sequence for every input sequence:

$$\forall \sigma \in I^* : \lambda^*(q, \sigma) = \lambda^*(q', \sigma)$$

- States q, q' are inequivalent if there is a separating sequence:

$$\exists \sigma \in I^* : \lambda^*(q, \sigma) \neq \lambda^*(q', \sigma)$$

- Two FSMs are equivalent if their initial states are equivalent

Equivalence

- States q, q' are equivalent if they produce the same output sequence for every input sequence:

$$\forall \sigma \in I^* : \lambda^*(q, \sigma) = \lambda^*(q', \sigma)$$

- States q, q' are inequivalent if there is a separating sequence:

$$\exists \sigma \in I^* : \lambda^*(q, \sigma) \neq \lambda^*(q', \sigma)$$

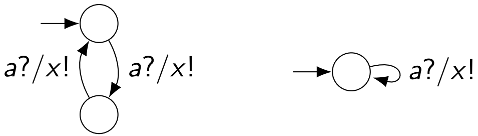
- Two FSMs are equivalent if their initial states are equivalent

For FSMs, we use equivalence as conformance relation

Minimality

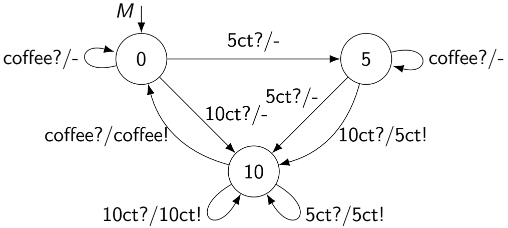
An FSM is minimal if no two states are equivalent.

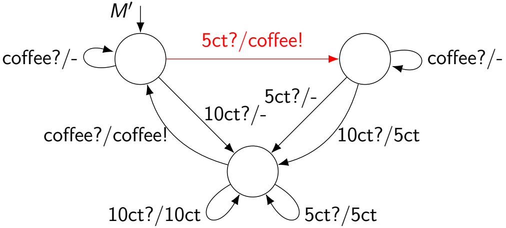
- A non-minimal FSM can be rewritten to an equivalent minimal FSM



Inequivalence Examples

Output fault: transition has wrong output

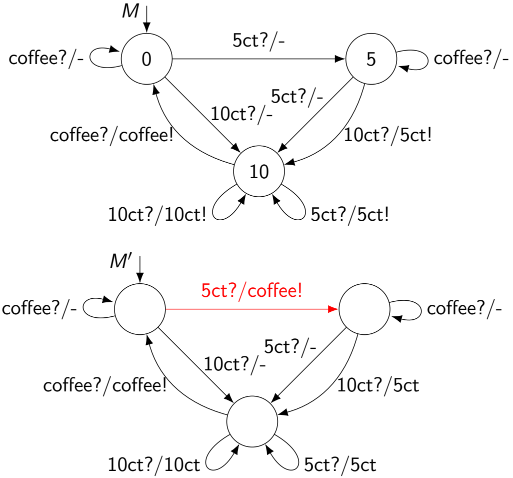




separating sequence?

Inequivalence Examples

Output fault: transition has wrong output

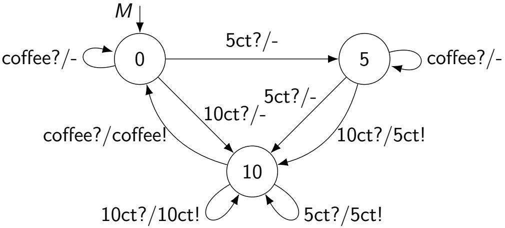


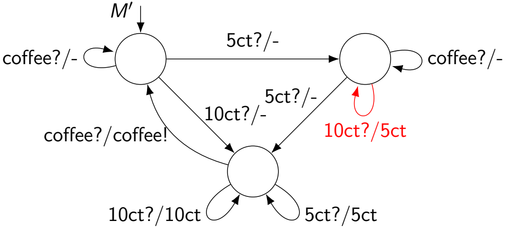
separating sequence?

$\lambda^*(M,5ct?)$ = -
$\lambda^*(M',5ct?)$ = coffee!

Inequivalence Examples

Transition fault: transition goes to wrong state

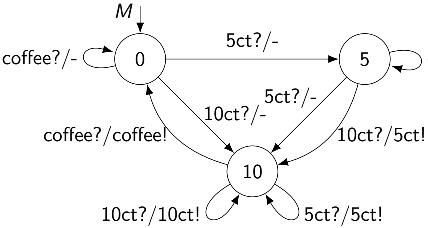


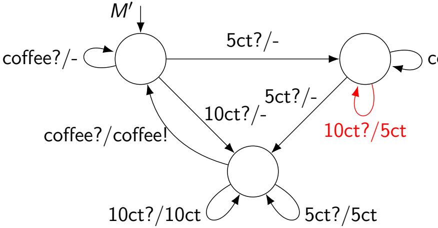


separating sequence?

Inequivalence Examples

Transition fault: transition goes to wrong state





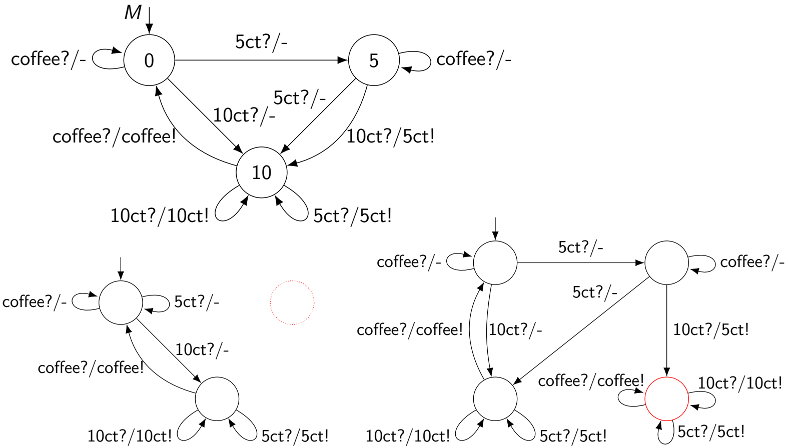
separating sequence?

$\lambda^*(M,$5ct? 10ct? 5ct?$)$ $=$ - 5ct! 5ct!

$\lambda^*(M',$5ct? 10ct? 5ct?$)$ $=$ - 5ct! -

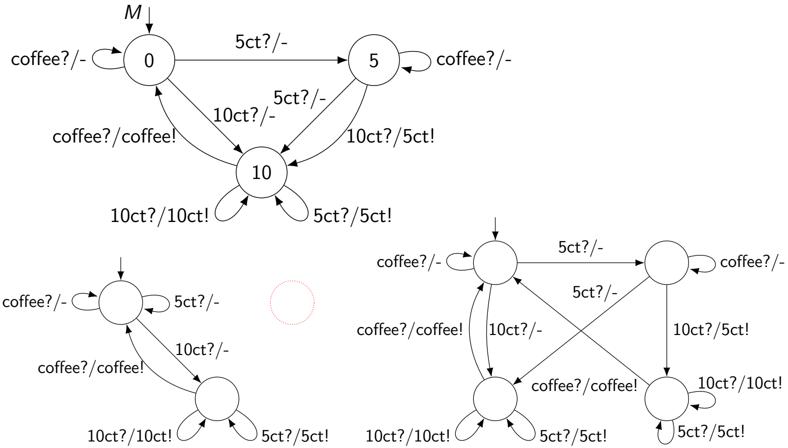
Inequivalence Examples

Missing states and extra states



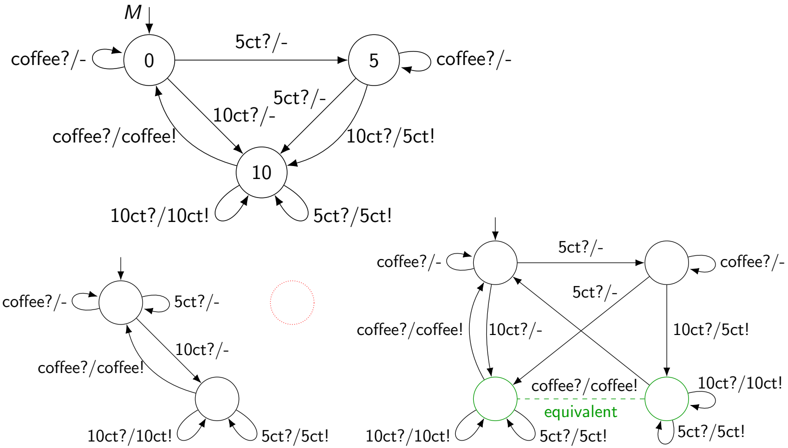
Inequivalence Examples

Missing states and extra states



Inequivalence Examples

Missing states and extra states



Test Suite

Given a specification FSM S and an implementation FSM M:

- A test case is an input sequence $\sigma \in I^*$
  (expected outputs and judgment are implicit from S)

Test Suite

Given a specification FSM S and an implementation FSM M:

- A test case is an input sequence $\sigma \in I^*$
  (expected outputs and judgment are implicit from S)

- M passes σ if $\lambda^*(S, \sigma) = \lambda^*(M, \sigma)$
- M fails σ if $\lambda^*(S, \sigma) \neq \lambda^*(M, \sigma)$

Test Suite

Given a specification FSM S and an implementation FSM M:

- A test case is an input sequence $\sigma \in I^*$
  (expected outputs and judgment are implicit from S)

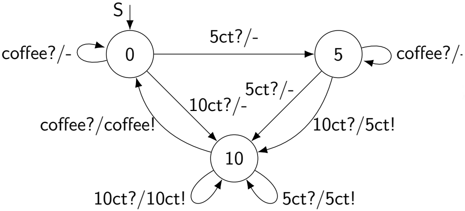
- M passes σ if $\lambda^*(S, \sigma) = \lambda^*(M, \sigma)$
- M fails σ if $\lambda^*(S, \sigma) \neq \lambda^*(M, \sigma)$

- A test suite is a finite set of test cases $T \subseteq I^*$
- A test suite fails if a single test case fails, and passes otherwise

Executing a Test Suite on a Black-Box System

To execute T:

- apply input sequences $\sigma \in T$
- observe output sequences $\lambda^*(M, \sigma)$
  - fail if $\lambda^*(M, \sigma) \neq \lambda^*(S, \sigma)$
- reset system in between tests



Complete Test Suite

- Let S be a specification and T a test suite.
- Then T is complete if for any implementation M:
  M passes T implies M equivalent to S

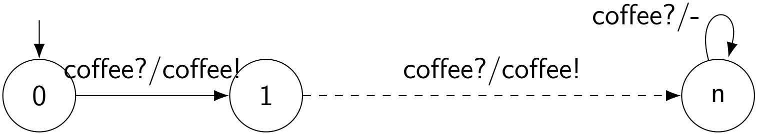
Complete Test Suite

- Let S be a specification and T a test suite.
- Then T is complete if for any implementation M:

  M passes T implies M equivalent to S
- Complete test suites do not exist!

  Specification:

Implementation:



Test cases of length $< n$ will not find this fault, and n can be arbitrarily large

Fault Domains

A fault domain reflects assumptions about faults that may occur in an implementation and that need to be detected during testing:

Definition (Fault domains and $\mathcal{U}$-completeness)

Let $\mathcal{S}$ be a Mealy machine. A fault domain is a set $\mathcal{U}$ of Mealy machines. A test suite T for $\mathcal{S}$ is $\mathcal{U}$-complete if, for each $\mathcal{M} \in \mathcal{U}$, $\mathcal{M}$ passes T implies $\mathcal{M} \approx \mathcal{S}$.

The Most Popular Fault Domain Ever

Based on work of Moore, Hennie, and Chow, hundreds of papers about conformance testing use the following fault domain:

Definition ($\mathcal{U}_m$)

Let $m > 0$. Then $\mathcal{U}_m$ is the set of all Mealy machines with at most m states.

The Most Popular Fault Domain Ever

Based on work of Moore, Hennie, and Chow, hundreds of papers about conformance testing use the following fault domain:

Definition ($\mathcal{U}_m$)

Let $m > 0$. Then $\mathcal{U}_m$ is the set of all Mealy machines with at most m states.

Suppose T is a test suite for a specification S with n states. Let $k \geq 0$. Then T is k-complete for S if it is $\mathcal{U}_{n+k}$-complete.

The Most Popular Fault Domain Ever

Based on work of Moore, Hennie, and Chow, hundreds of papers about conformance testing use the following fault domain:

Definition ($\mathcal{U}_m$)

Let $m > 0$. Then $\mathcal{U}_m$ is the set of all Mealy machines with at most m states.

Suppose T is a test suite for a specification S with n states. Let $k \geq 0$. Then T is k-complete for S if it is $\mathcal{U}_{n+k}$-complete.
Exists! Based on access sequences and characterization sets
(a.k.a. the W-method).

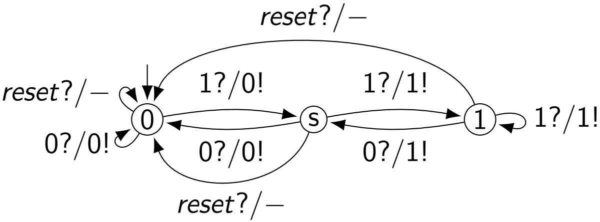
Building Block: Access sequences

Let S be a specification FSM with states Q and initial state q_0.

- An access sequence for state $q \in Q$ is any sequence σ with $\delta^*(q_0, \sigma) = q$.
- An access sequence set $A \subseteq I^*$ for Q contains an access sequence for all states in Q; we require $\epsilon \in A$.

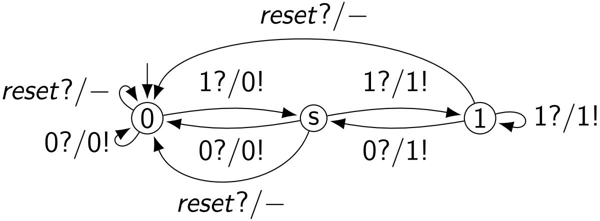
Executing A ensures that we reach all states in Q.

Access Sequences Example



$A = ?$

Access Sequences Example



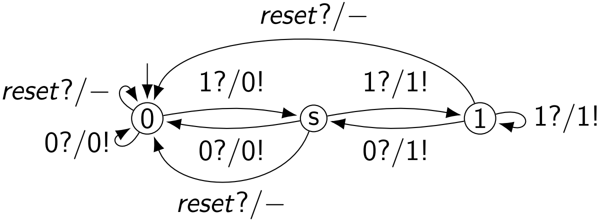
$$A = \{\epsilon,\ 1?,\ 1?1?\}$$

Building Block: Characterization Sets

Let S be a minimal specification FSM with states Q.

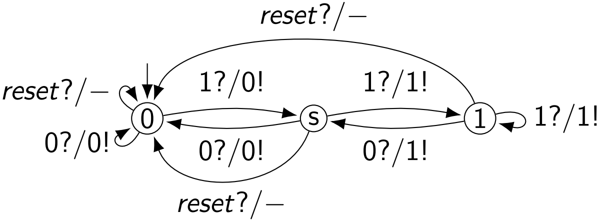
- A characterization set $C \subseteq I^*$ for Q contains a separating sequence for every pair of states $q, q' \in Q$ (with $q \neq q'$).

Characterisation Set Example



$C = ?$

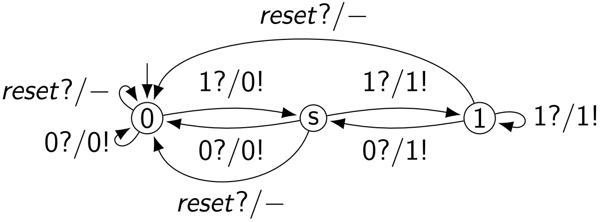
Characterisation Set Example



$C = \{0?, 1?\}$

Why is this a characterization set?

Characterisation Set Example

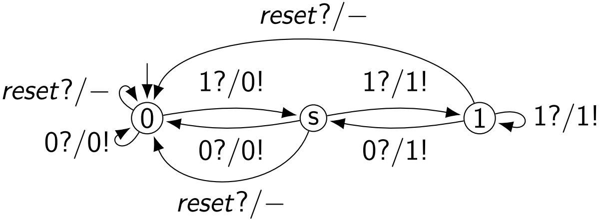


$C = \{0?, 1?\}$

Why is this a characterization set?

$\lambda^*(0, 1?) = 0! \neq 1! = \lambda^*(s, 1?)$ (C separates 0 and s)

Characterisation Set Example

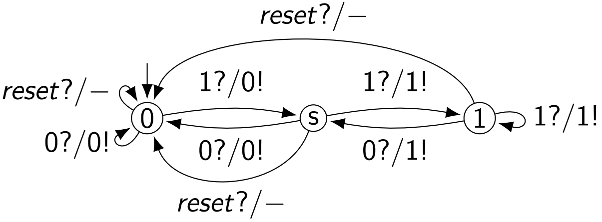


$C = \{0?, 1?\}$

Why is this a characterization set?

$\lambda^*(0, 1?) = 0! \neq 1! = \lambda^*(s, 1?)$ (C separates 0 and s)
$\lambda^*(0, 0?) = 0! \neq 1! = \lambda^*(1, 0?)$ (C separates 0 and 1)

Characterisation Set Example

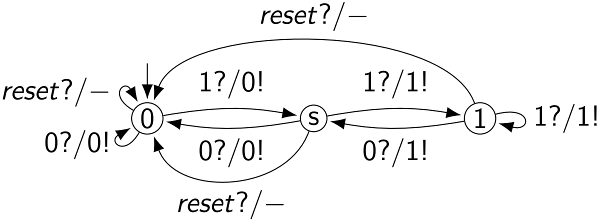


$C = \{0?, 1?\}$

Why is this a characterization set?

$\lambda^*(0, 1?) = 0! \neq 1! = \lambda^*(s, 1?)$ (C separates 0 and s)
$\lambda^*(0, 0?) = 0! \neq 1! = \lambda^*(1, 0?)$ (C separates 0 and 1)
$\lambda^*(1, 0?) = 1! \neq 0! = \lambda^*(s, 0?)$ (C separates 1 and s)

Characterisation Set Example



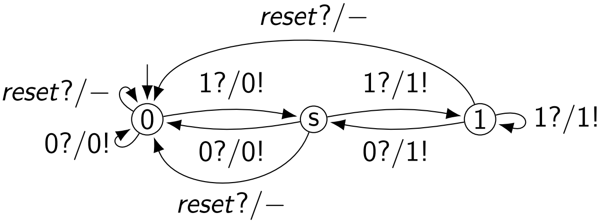
$C = \{0?, 1?\}$

Why is this a characterization set?

$$
\begin{array}{lr}
\lambda^*(0, 1?) = 0! \neq 1! = \lambda^*(s, 1?) & (C \text{ separates } 0 \text{ and } s) \\
\lambda^*(0, 0?) = 0! \neq 1! = \lambda^*(1, 0?) & (C \text{ separates } 0 \text{ and } 1) \\
\lambda^*(1, 0?) = 1! \neq 0! = \lambda^*(s, 0?) & (C \text{ separates } 1 \text{ and } s)
\end{array}
$$

(why is ?*reset* useless in a characterization set?)

Characterisation Set Example



$C = \{0?, 1?\}$

λ^*	0?	1?
0	0!	0!
s	0!	1!
1	1!	1!

All rows of this table (λ^* for Q and C) are different: state identification

Building Blocks for 0-Complete Test Suite

- Check that the implementation has at least as many states as the specification (no missing states)
- Check that each implementation state is correct:
  - outgoing transitions have a correct output (no output fault), and
  - lead to correct specification state (no transition fault)

Building Blocks for 0-Complete Test Suite

- Check that the implementation has at least as many states as the specification (no missing states)
- Check that each implementation state is correct:
  - outgoing transitions have a correct output (no output fault), and
  - lead to correct specification state (no transition fault)

Assumption: 0 extra states w.r.t. S (no extra states)

Building Block 1: No Missing States

Check that M has at least as many states as specification S:

- Execute all input sequences of $A \cdot C$ on M
- For every $a, a' \in A$, execution of $a \cdot C$ and $a' \cdot C$ shows that a and a' reach different specification states

Building Block 1: No Missing States

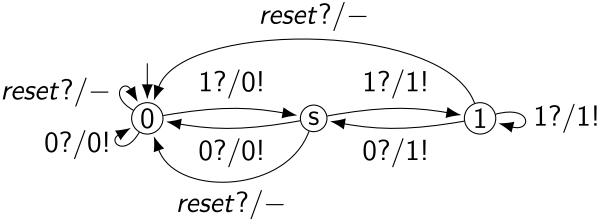
Check that M has at least as many states as specification S:

- Execute all input sequences of $A \cdot C$ on M
- For every $a, a' \in A$, execution of $a \cdot C$ and $a' \cdot C$ shows that a and a' reach different specification states

$A = \{\epsilon,\ 1?,\ 1?1?\}$
$C = \{0?,\ 1?\}$
$A \cdot C = \{0?,\ 1?,\ 1?0?,\ 1?1?,\ 1?1?0?,\ 1?1?1?\}$

Building Block 1: No Missing States

$$A \cdot C = \{0?,\ 1?,\ 1?0?,\ 1?1?,\ 1?1?0?,\ 1?1?1?\}$$



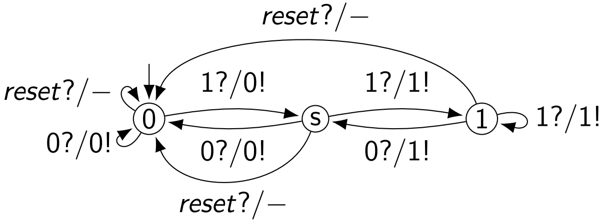
Passing implementation must have at least 3 states, reached by A:

$a \cdot c$	0?	1?	1?0?	1?1?	1?1?0?	1?1?1?
$\lambda^*(M, a \cdot c)$	0!	0!	0!0!	0!1!	0!1!1!	0!1!1!

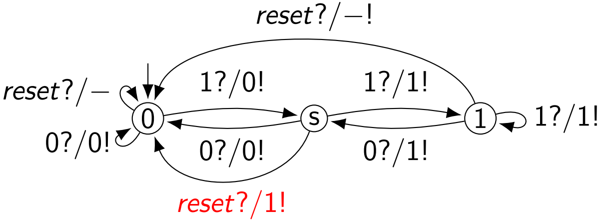
Building Block 1: No Missing States

$A \cdot C$ does not find all output faults yet!

$S =$



$M =$



Building Block 2: No Output Faults

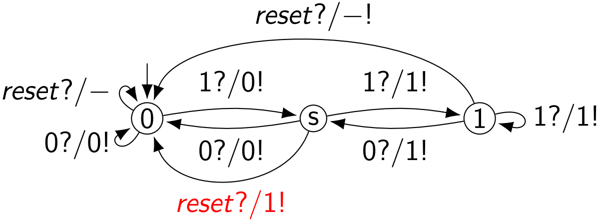
Solution: also test $A \cdot I =$
$\{0?,\ 1?,\ \mathit{reset}?,\ 1?0?,\ 1?1?,\ 1?\mathit{reset}?,\ 1?1?0?,\ 1?1?1?,\ 1?1?\mathit{reset}?\}$

This works, because A reaches all *implementation* states

Building Block 2: No Output Faults

Solution: also test $A \cdot I =$
$\{0?,\ 1?,\ reset?,\ 1?0?,\ 1?1?,\ 1?reset?,\ 1?1?0?,\ 1?1?1?,\ 1?1?reset?\}$

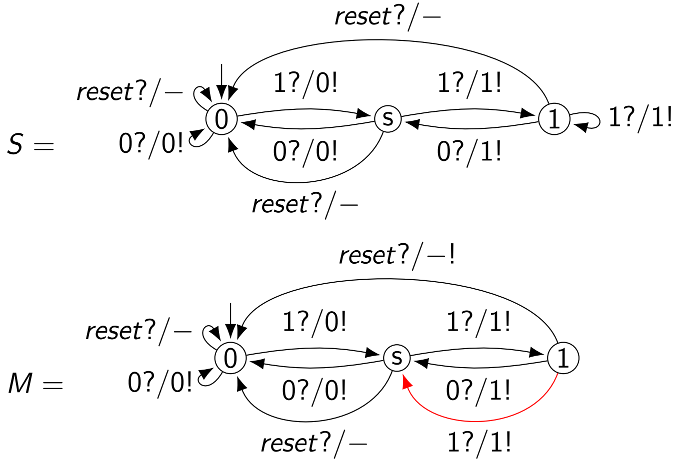
This works, because A reaches all *implementation* states



$\lambda^*(S, 1?reset?) = 0!-$
$\lambda^*(M, 1?reset?) = 0!1!$

Building Block 2: No Transition Faults

$A \cdot C + A \cdot I$ does not detect transition faults yet!



Building Block 2: No Transition Faults

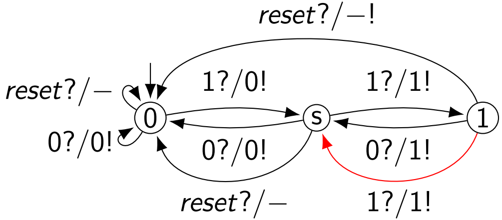
$A \cdot C + A \cdot I$ does not detect transition faults yet!
Solution: also test $A \cdot I \cdot C$

- C tests whether the right state is reached after $A \cdot I$

Building Block 2: No Transition Faults

$A \cdot C + A \cdot I$ does not detect transition faults yet!
Solution: also test $A \cdot I \cdot C$

- C tests whether the right state is reached after $A \cdot I$

$M =$ 

$\lambda(S,\ 1?1?1?0?) = 0!1!1!1!$
$\lambda(M,\ 1?1?1?0?) = 0!1!1!0!$

(access sequence 1?1?; faulty transition 1?; separating sequence 0? for states s and 1)

0-Complete Test Suite

Full 0-complete test suite is
$\mathbf{T} = \mathbf{A} \cdot \mathbf{C} + \mathbf{A} \cdot \mathbf{I} + \mathbf{A} \cdot \mathbf{I} \cdot \mathbf{C}$

0-Complete Test Suite

Full 0-complete test suite is
$A \cdot C + A \cdot I + A \cdot I \cdot C$

If $C \neq \emptyset$ then $A \cdot I$ contains only prefixes of $A \cdot I \cdot C$ so we can leave it out:
$T = \mathbf{A} \cdot \mathbf{C} + \mathbf{A} \cdot \mathbf{I} \cdot \mathbf{C}$

0-Complete Test Suite

Full 0-complete test suite is
$A \cdot C + A \cdot I + A \cdot I \cdot C$

If $C \neq \emptyset$ then $A \cdot I$ contains only prefixes of $A \cdot I \cdot C$ so we can leave it out:
$A \cdot C + A \cdot I \cdot C$

or simply
$\mathbf{T} = \mathbf{A} \cdot \mathbf{I}^{\leq 1} \cdot \mathbf{C}$
($I^{\leq 1}$ means all sequences in I^* up to length 1)

0-Complete Test Suite

Full 0-complete test suite is
$A \cdot C + A \cdot I + A \cdot I \cdot C$

If $C \neq \emptyset$ then $A \cdot I$ contains only prefixes of $A \cdot I \cdot C$ so we can leave it out:
$A \cdot C + A \cdot I \cdot C$

or simply
$\mathbf{T} = \mathbf{A} \cdot \mathbf{I}^{\leq 1} \cdot \mathbf{C}$
($I^{\leq 1}$ means all sequences in I^* up to length 1)

Note: many possible sets A and C!

Correctness

Theorem (W method) Let S be a minimal FSM with set of access sequences A, set of inputs I, and nonempty characterization set C. Then $T = A \cdot I^{\leq 1} \cdot C$ is 0-complete.

Correctness

Theorem (W method) Let S be a minimal FSM with set of access sequences A, set of inputs I, and nonempty characterization set C. Then $T = A \cdot I^{\leq 1} \cdot C$ is 0-complete.

Proof: We use the concept of a bisimulation.

Bisimulation

Definition Let M_1 and M_2 be FSMs with inputs I. A **bisimulation** between M_1 and M_2 is a relation $R \subseteq Q_1 \times Q_2$ such that $(q_0^1, q_0^2) \in R$ and, for all $(q, r) \in R$ and $i \in I$,

1. $\lambda_1(q, i) = \lambda_2(r, i)$,
2. $(\delta_1(q, i), \delta_2(r, i)) \in R$.

Bisimulation (cnt)

Lemma If there exists a bisimulation R between M_1 and M_2, then M_1 and M_2 are equivalent.

Proof: Assume $(q, r) \in R$ and $\sigma \in I^*$. By induction on the length of σ we prove that $\lambda_1^*(q, \sigma) = \lambda_2^*(r, \sigma)$.

- Base. Trivial since $\lambda_1^*(q, \epsilon) = \epsilon = \lambda_1^*(r, \epsilon)$.
- Induction step. Let $\sigma = i\ \rho$. By definition,

$$\begin{aligned} \lambda_1^*(q, \sigma) &= \lambda_1(q, i)\ \lambda_1^*(\delta_1(q, i), \rho), \\ \lambda_2^*(r, \sigma) &= \lambda_2(r, i)\ \lambda_2^*(\delta_2(r, i), \rho). \end{aligned}$$

  By condition (1) for bisimulations $\lambda_1(q, i) = \lambda_2(r, i)$.
  By condition (2) for bisimulations $(\delta_1(q, i), \delta_2(r, i)) \in R$.
  Therefore, by induction hypothesis,
  $\lambda_1^*(\delta_1(q, i), \rho) = \lambda_2^*(\delta_2(r, i), \rho)$.
  This implies that $\lambda_1^*(q, \sigma) = \lambda_2^*(r, \sigma)$, as required.

From this property the lemma follows since $(q_0^1, q_0^2) \in R$.

Correctness (cnt)

Theorem Let S be a minimal FSM with set of access sequences A, set of inputs I, and nonempty characterization set C. Then $T = A \cdot I^{\leq 1} \cdot C$ is 0-complete.

Proof: Let M be an FSM with at most as many states as S such that M passes tests T. By the previous lemma, it suffices to show that the following relation R is a bisimulation between M and S:

$$(q, r) \in R \quad \Leftrightarrow \quad \forall \sigma \in C : \lambda^*_M(q, \sigma) = \lambda^*_S(r, \sigma)$$

Because we require $\epsilon \in A$ we have $C \subseteq T$. Therefore, since M passes T, $\forall \sigma \in C : \lambda^*_M(q_0^M, \sigma) = \lambda^*_S(q_0^S, \sigma)$. This implies $(q_0^M, q_0^S) \in R$, as required.

Correctness (cnt)

Suppose r_1 and r_2 are distinct states of S with access sequences ρ_1 and ρ_2, respectively. Then there is a separating sequence $\sigma \in C$ for r_1 and r_2. Let q_1 and q_2 be the states of M reached by access sequences ρ_1 and ρ_2. Then, since M passes $A \cdot C$, σ is also a separating sequence for q_1 and q_2. Since all states of S can be reached and pairwise be separated by C, M has at least as many states as S, that can pairwise be separated by C.
Since we assume that M has at most as many states as S, we conclude that M and S have the same number of states.
Since M passes $A \cdot C$, we know that for each pair $(q, r) \in R$ there exists an access sequence $\rho \in A$ such that $\delta_M(q_0^M, \rho) = q$ and $\delta_S(q_0^S, \rho) = r$.

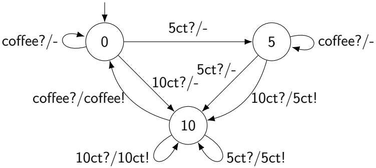
Correctness (cnt)

Now suppose that $(q, r) \in R$ and $i \in I$. Let ρ be an access sequence for q and r. Then, since M passes tests $\rho\ i\ C$ we may conclude

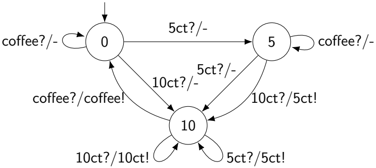
1. $\lambda_M(q, i) = \lambda_S(r, i)$,
2. $(\delta_M(q, i), \delta_S(r, i)) \in R$.

Therefore R is a bisimulation between M and S.

Example: 0-Complete Test Suite Coffee Machine

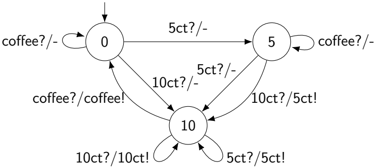


Example: 0-Complete Test Suite Coffee Machine



$$A = \{\epsilon,\ 5ct?,\ 10ct?\}$$

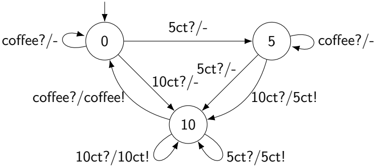
Example: 0-Complete Test Suite Coffee Machine



$$A = \{\epsilon,\ 5ct?,\ 10ct?\}$$

$$I^{\leq 1} = \{\epsilon,\ 5ct?,\ 10ct?,\ coffee?\}$$

Example: 0-Complete Test Suite Coffee Machine

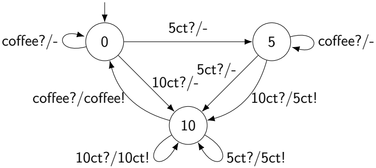


$A = \{\epsilon,\ 5ct?,\ 10ct?\}$

$I^{\leq 1} = \{\epsilon,\ 5ct?,\ 10ct?,\ coffee?\}$

$C = \{10ct?\}$

Example: 0-Complete Test Suite Coffee Machine



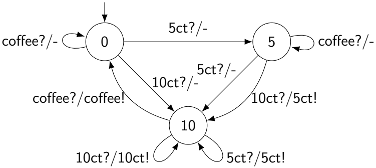
$A = \{\epsilon,\ 5ct?,\ 10ct?\}$

$I^{\leq 1} = \{\epsilon,\ 5ct?,\ 10ct?,\ coffee?\}$

$C = \{10ct?\}$

$A \cdot I^{\leq 1} \cdot C =$
$\{$

Example: 0-Complete Test Suite Coffee Machine



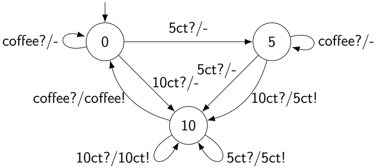
$A = \{\epsilon,\ 5ct?,\ 10ct?\}$

$I^{\leq 1} = \{\epsilon,\ 5ct?,\ 10ct?,\ coffee?\}$

$C = \{10ct?\}$

$A \cdot I^{\leq 1} \cdot C =$
$\{10ct?,$

Example: 0-Complete Test Suite Coffee Machine



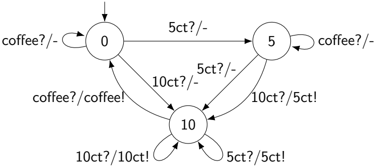
$$A = \{\epsilon,\ 5ct?,\ 10ct?\}$$

$$I^{\leq 1} = \{\epsilon,\ 5ct?,\ 10ct?,\ coffee?\}$$

$$C = \{10ct?\}$$

$$A \cdot I^{\leq 1} \cdot C =$$
$$\{10ct?,\ 5ct?10ct?,$$

Example: 0-Complete Test Suite Coffee Machine



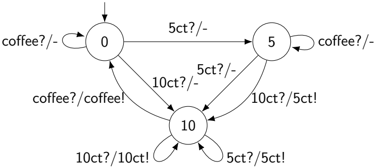
$A = \{\epsilon,\ 5ct?,\ 10ct?\}$

$I^{\leq 1} = \{\epsilon,\ 5ct?,\ 10ct?,\ coffee?\}$

$C = \{10ct?\}$

$A \cdot I^{\leq 1} \cdot C =$
$\{10ct?,\ 5ct?10ct?,\ 10ct?10ct?,$

Example: 0-Complete Test Suite Coffee Machine



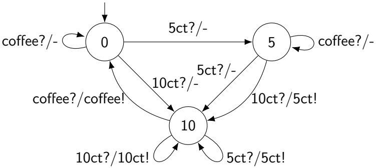
$A = \{\epsilon,\ 5ct?,\ 10ct?\}$

$I^{\leq 1} = \{\epsilon,\ 5ct?,\ 10ct?,\ coffee?\}$

$C = \{10ct?\}$

$A \cdot I^{\leq 1} \cdot C =$
$\{10ct?,\ 5ct?10ct?,\ 10ct?10ct?,\ coffee?10ct?,$

Example: 0-Complete Test Suite Coffee Machine



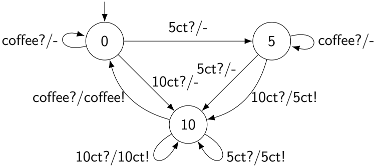
$A = \{\epsilon,\ 5ct?,\ 10ct?\}$

$I^{\leq 1} = \{\epsilon,\ 5ct?,\ 10ct?,\ coffee?\}$

$C = \{10ct?\}$

$A \cdot I^{\leq 1} \cdot C =$
$\{10ct?,\ 5ct?10ct?,\ 10ct?10ct?,\ coffee?10ct?,$
$5ct?10ct?,$

Example: 0-Complete Test Suite Coffee Machine



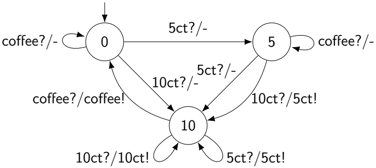
$A = \{\epsilon,\ 5ct?,\ 10ct?\}$

$I^{\leq 1} = \{\epsilon,\ 5ct?,\ 10ct?,\ coffee?\}$

$C = \{10ct?\}$

$A \cdot I^{\leq 1} \cdot C =$
$\{10ct?,\ 5ct?10ct?,\ 10ct?10ct?,\ coffee?10ct?,$
$5ct?10ct?,\ 5ct?5ct?10ct?,\ 5ct?10ct?10ct?,\ 5ct?coffee?10ct?,$

Example: 0-Complete Test Suite Coffee Machine



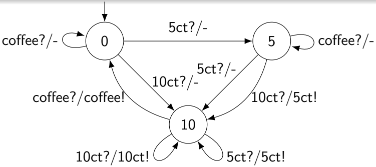
$A = \{\epsilon,\ 5ct?,\ 10ct?\}$

$I^{\leq 1} = \{\epsilon,\ 5ct?,\ 10ct?,\ coffee?\}$

$C = \{10ct?\}$

$A \cdot I^{\leq 1} \cdot C =$
$\{10ct?,\ 5ct?10ct?,\ 10ct?10ct?,\ coffee?10ct?,$
$5ct?10ct?,\ 5ct?5ct?10ct?,\ 5ct?10ct?10ct?,\ 5ct?coffee?10ct?,$
$10ct?10ct?,\ 10ct?5ct?10ct?,\ 10ct?10ct?10ct?,\ 10ct?coffee?10ct?\ \}$

Example: 0-Complete Test Suite Coffee Machine



$A = \{\epsilon,\ 5ct?,\ 10ct?\}$

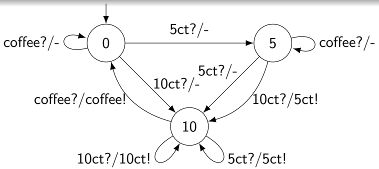
$I^{\leq 1} = \{\epsilon,\ 5ct?,\ 10ct?,\ coffee?\}$

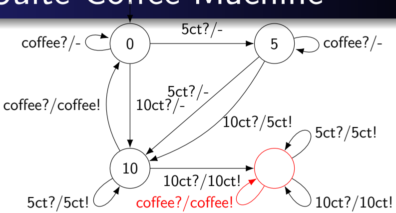
$C = \{10ct?\}$

$A \cdot I^{\leq 1} \cdot C =$
$\{\cancel{10ct?},\ \cancel{5ct?10ct?},\ \cancel{10ct?10ct?},\ coffee?10ct?,$
$\cancel{5ct?10ct?},\ 5ct?5ct?10ct?,\ 5ct?10ct?10ct?,\ 5ct?coffee?10ct?,$
$\cancel{10ct?10ct?},\ 10ct?5ct?10ct?,\ 10ct?10ct?10ct?,\ 10ct?coffee?10ct?\ \}$

(remove redundant prefixes)

Example: 0-Complete Test Suite Coffee Machine





$A = \{\epsilon,\ 5ct?,\ 10ct?\}$

$I^{\leq 1} = \{\epsilon,\ 5ct?,\ 10ct?,\ coffee?\}$

$C = \{10ct?\}$

$A \cdot I^{\leq 1} \cdot C =$
$\{\cancel{10ct?},\ \cancel{5ct?10ct?},\ \cancel{10ct?10ct?},\ coffee?10ct?,$
$\cancel{5ct?10ct?},\ 5ct?5ct?10ct?,\ 5ct?10ct?10ct?,\ 5ct?coffee?10ct?,$
$\cancel{10ct?10ct?},\ 10ct?5ct?10ct?,\ 10ct?10ct?10ct?,\ 10ct?coffee?10ct?\ \}$

(remove redundant prefixes)

k-Complete Test Suite

What if $k > 0$?

- We should detect up to k extra states.

k-Complete Test Suite

What if $k > 0$?

- We should detect up to k extra states.
- $A \cdot I^{\leq k}$ reaches all implementation states!
- replace A in the 0-complete test suite by $A \cdot I^{\leq k}$

k-Complete Test Suite

What if $k > 0$?

- We should detect up to k extra states.
- $A \cdot I^{\leq k}$ reaches all implementation states!
- replace A in the 0-complete test suite by $A \cdot I^{\leq k}$

An k-complete test suite:
$T = (\mathbf{A} \cdot \mathbf{I}^{\leq \mathbf{k}}) \cdot \mathbf{I}^{\leq \mathbf{1}} \cdot \mathbf{C}$

k-Complete Test Suite

What if $k > 0$?

- We should detect up to k extra states.
- $A \cdot I^{\leq k}$ reaches all implementation states!
- replace A in the 0-complete test suite by $A \cdot I^{\leq k}$

An k-complete test suite:
$(A \cdot I^{\leq k}) \cdot I^{\leq 1} \cdot C$

or simply
$\mathbf{T} = \mathbf{A} \cdot \mathbf{I}^{\leq \mathbf{k}+1} \cdot \mathbf{C}$

Large Characterisation Sets

- Remember: set $C \subseteq I^*$ is a characterisation set for specification S if:
  - For each pair of distinct states q and q' of S there is a $c \in C$ such that $\lambda^*(q, c) \neq \lambda^*(q', c)$
- Upper bound on the size of C is $(\frac{|S|^2 - |S|}{2})$ elements.

Special (Smaller) Characterisation Sets

- A sequence $c \in C$ is a Unique Input Output sequence (UIO) for some state q if:
  - for all other states q' of S: $\lambda^*(q, c) \neq \lambda^*(q', c)$
- Hence, a characterisation set of UIOs needs only $|S| - 1$ elements.

Special (Smaller) Characterisation Sets

- A sequence $c \in C$ is a Unique Input Output sequence (UIO) for some state q if:
  - for all other states q' of S: $\lambda^*(q, c) \neq \lambda^*(q', c)$
- Hence, a characterisation set of UIOs needs only $|S| - 1$ elements.

- A sequence $c \in C$ is a Distinguishing Sequence (DS) for S if:
  - For all states q, q' (with $q \neq q'$) of S: $\lambda^*(q, c) \neq \lambda^*(q', c)$
- Hence, a DS gives a singleton characterization set!

Special (Smaller) Characterisation Sets

- A sequence $c \in C$ is a Unique Input Output sequence (UIO) for some state q if:
  - for all other states q' of S: $\lambda^*(q, c) \neq \lambda^*(q', c)$
- Hence, a characterisation set of UIOs needs only $|S| - 1$ elements.

- A sequence $c \in C$ is a Distinguishing Sequence (DS) for S if:
  - For all states q, q' (with $q \neq q'$) of S: $\lambda^*(q, c) \neq \lambda^*(q', c)$
- Hence, a DS gives a singleton characterization set!

- Note:
  - A distinguishing sequence is for an entire specification
  - UIOs are per state
  - Separating sequences are per pair of states

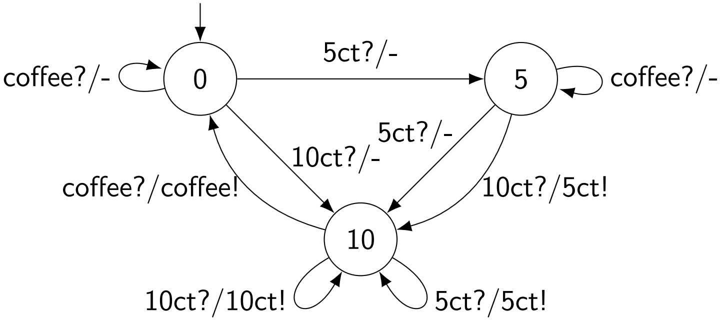
Special (Smaller) Characterisation Sets

- A sequence $c \in C$ is a Unique Input Output sequence (UIO) for some state q if:
  - for all other states q' of S: $\lambda^*(q, c) \neq \lambda^*(q', c)$
- Hence, a characterisation set of UIOs needs only $|S| - 1$ elements.

- A sequence $c \in C$ is a Distinguishing Sequence (DS) for S if:
  - For all states q, q' (with $q \neq q'$) of S: $\lambda^*(q, c) \neq \lambda^*(q', c)$
- Hence, a DS gives a singleton characterization set!

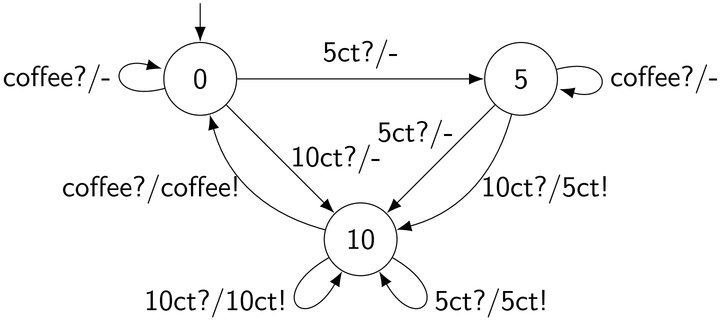
- Note:
  - A distinguishing sequence is for an entire specification
  - UIOs are per state
  - Separating sequences are per pair of states
- UIOs and DSs do not always exist...

Example: SS, UIO, or DSs?



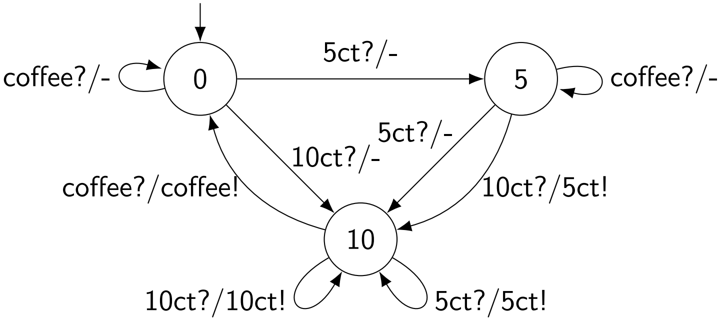
- 10ct?

Example: SS, UIO, or DSs?



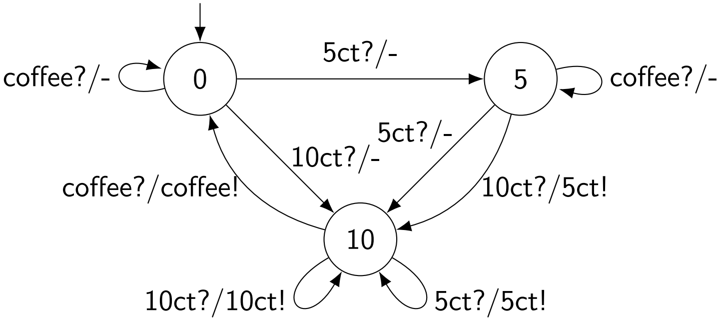
- 10ct? **DS**
- 5ct? coffee?

Example: SS, UIO, or DSs?



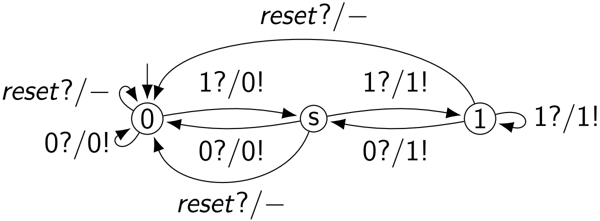
- 10ct? **DS**
- 5ct? coffee? **DS**
- coffee?

Example: SS, UIO, or DSs?



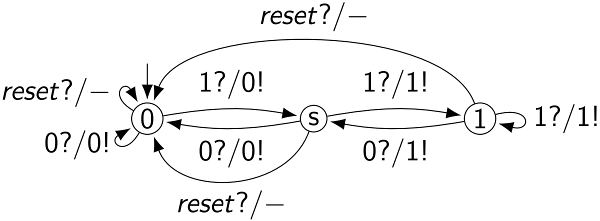
- 10ct? **DS**
- 5ct? coffee? **DS**
- coffee? **UIO for state 10**

Example: Do DSes or UIOs Exist?



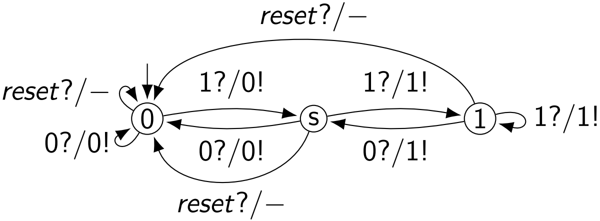
- Any DS?

Example: Do DSes or UIOs Exist?



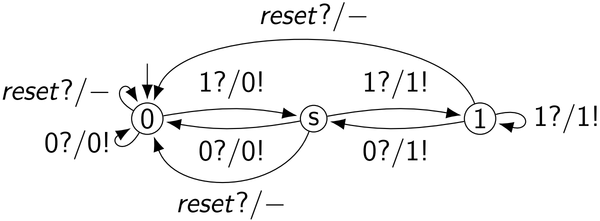
- Any DS? **no**
- Does 0 have an UIO?

Example: Do DSes or UIOs Exist?



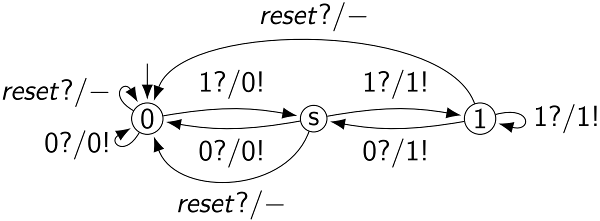
- Any DS? **no**
- Does 0 have an UIO? **yes,** sequence 1?.
- Does s have an UIO?

Example: Do DSes or UIOs Exist?



- Any DS? **no**
- Does 0 have an UIO? **yes,** sequence 1?.
- Does s have an UIO? **no**
- Does 1 have an UIO?

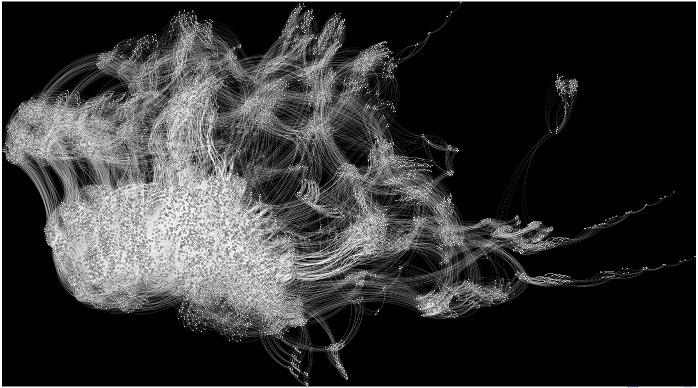
Example: Do DSes or UIOs Exist?



- Any DS? **no**
- Does 0 have an UIO? **yes,** sequence 1?.
- Does s have an UIO? **no**
- Does 1 have an UIO? **yes,** sequence 0?.

A More Realistic Example

- $\pm$ 10.000 states and $\pm$ 150 inputs
- Test suite from this lecture: $\pm 5,0 \cdot 10^8$ inputs
- Smarter test suite (adaptive DS + SS): $\pm 1.5 \cdot 10^8$ inputs



Algorithm for Finding Separating Sequences

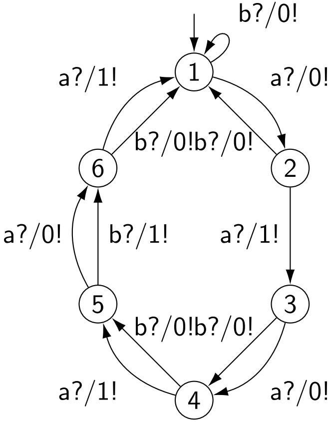
- Using breadth-first search for each pair of states: $O(pn^3)$
- Do it all at once (next slides): $O(pn^2)$
- Optimal (Hopcroft): $O(pn \log n)$

(n = number of states, p = number of inputs)

Algorithm for Finding Separating Sequences

- Use partition refinement
- Initially, all states are not separated: one block
- Gradually separate states: refine partitions
    - A block is split if we find a separating sequence

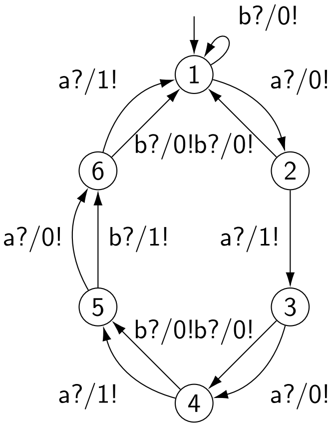
Algorithm for Finding Separating Sequences



Use a splitting tree:

$\{1,\ldots,6\}$

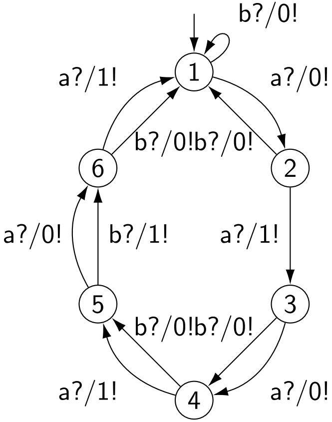
Algorithm for Finding Separating Sequences



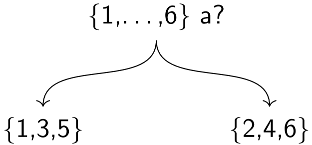
Use a splitting tree:

$\{1,\ldots,6\}$ a?

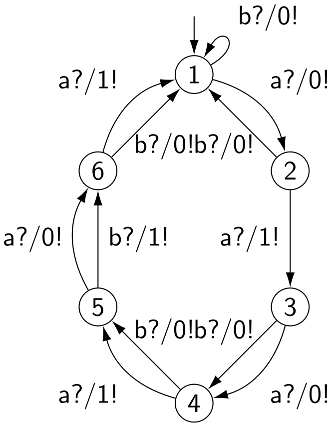
Algorithm for Finding Separating Sequences



Use a splitting tree:



Algorithm for Finding Separating Sequences



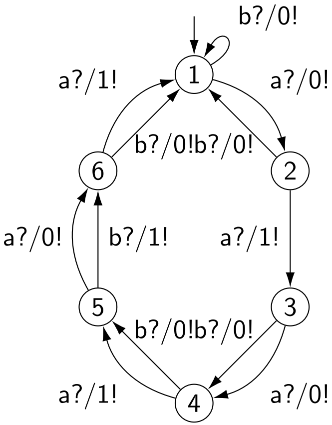
Use a splitting tree:

{1,. . . ,6} a?

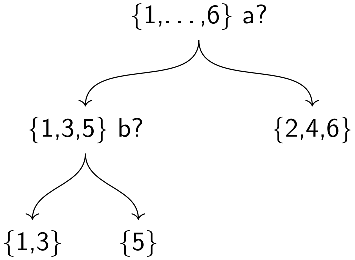
{1,3,5} b?

{2,4,6}

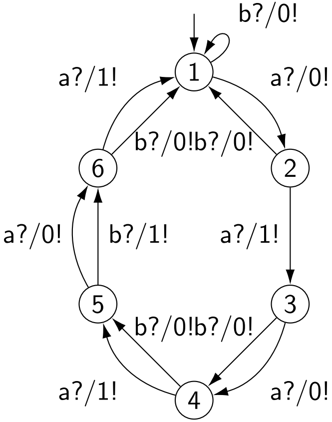
Algorithm for Finding Separating Sequences



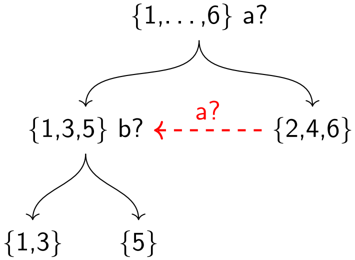
Use a splitting tree:



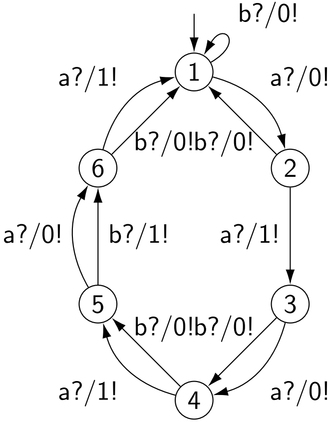
Algorithm for Finding Separating Sequences



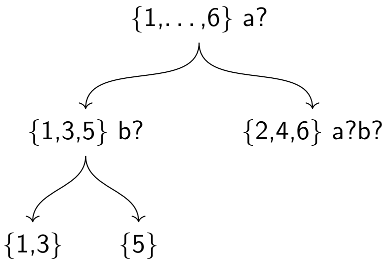
Use a splitting tree:



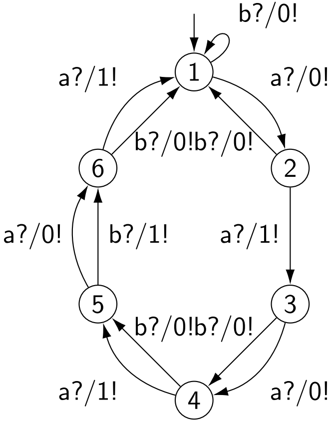
Algorithm for Finding Separating Sequences



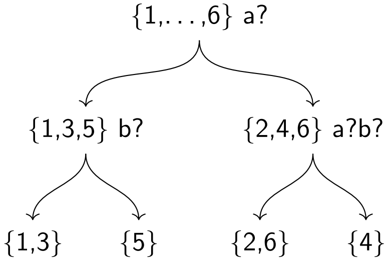
Use a splitting tree:



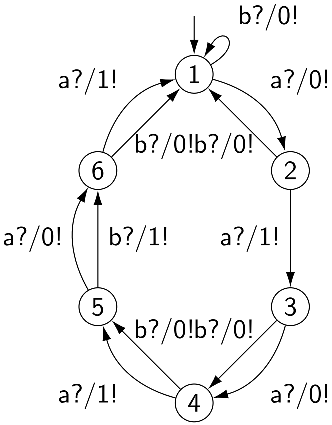
Algorithm for Finding Separating Sequences



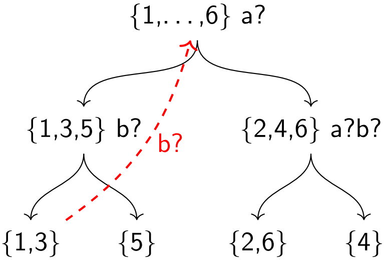
Use a splitting tree:



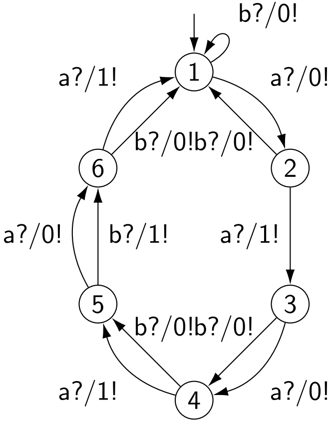
Algorithm for Finding Separating Sequences



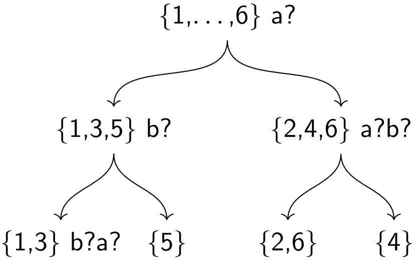
Use a splitting tree:



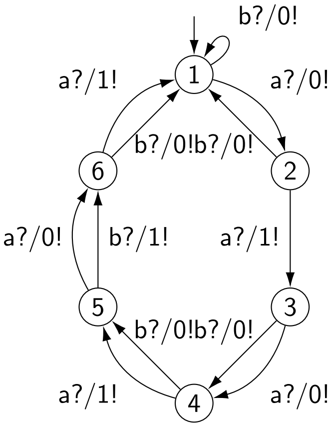
Algorithm for Finding Separating Sequences



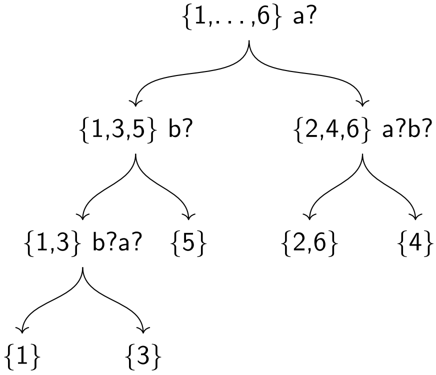
Use a splitting tree:



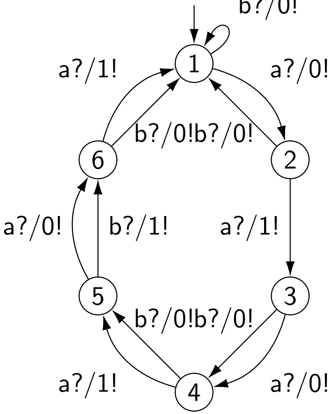
Algorithm for Finding Separating Sequences



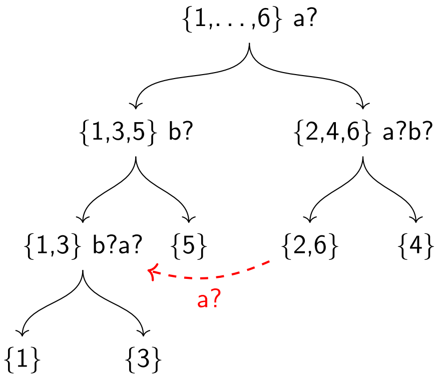
Use a splitting tree:



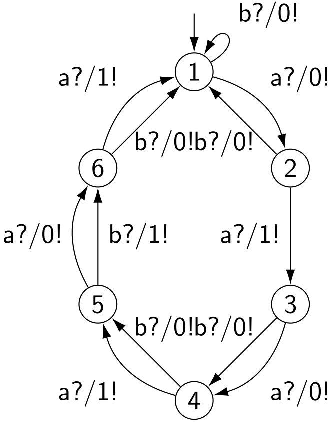
Algorithm for Finding Separating Sequences



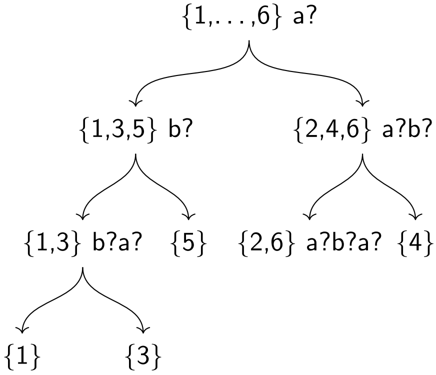
Use a splitting tree:



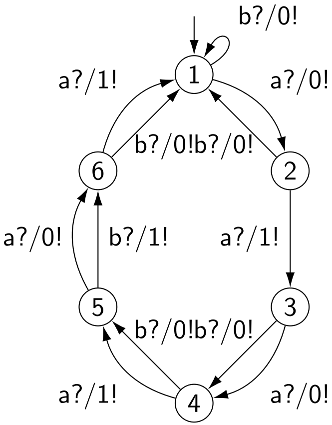
Algorithm for Finding Separating Sequences



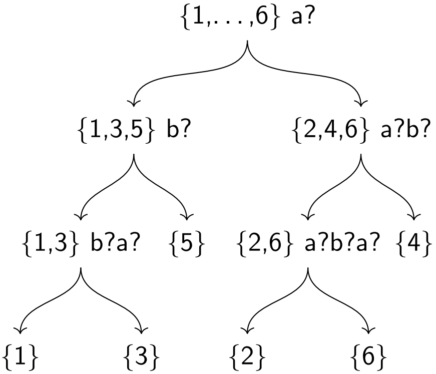
Use a splitting tree:



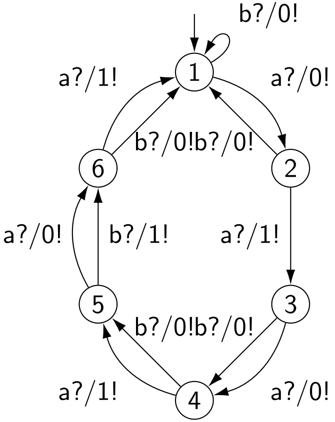
Algorithm for Finding Separating Sequences



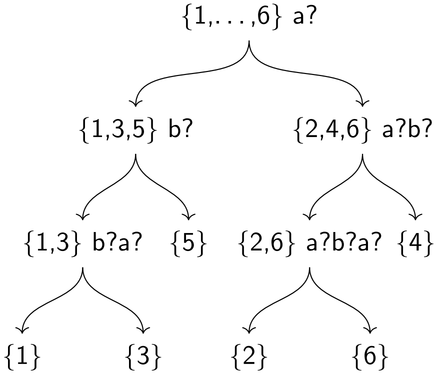
Use a splitting tree:



Algorithm for Finding Separating Sequences

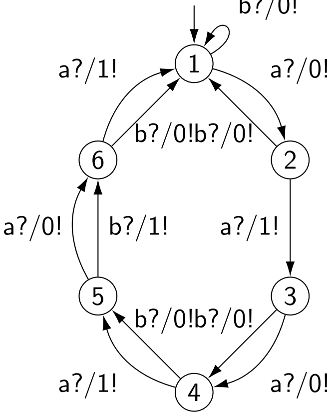


Use a splitting tree:

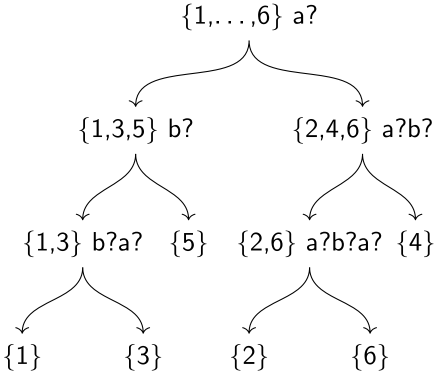


$$C = \{a?, b?, a?b?, b?a?, a?b?a?\}$$

Algorithm for Finding Separating Sequences

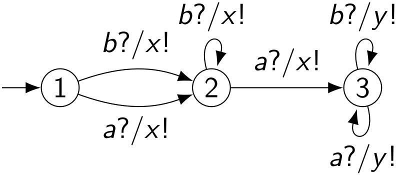


Use a splitting tree:

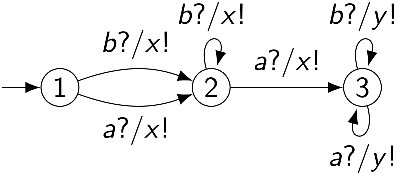


$C = \{\cancel{a?}, \cancel{b?}, \cancel{a?b?}, b?a?, a?b?a?\}$

Splitting node: Separate States by Input

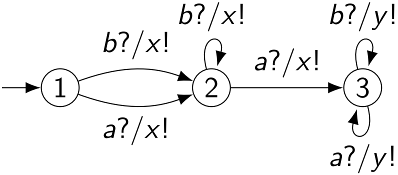


Splitting node: Separate States by Input



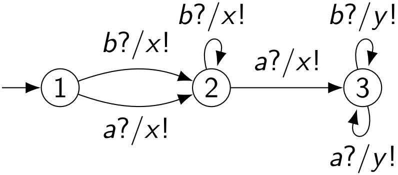
$\{1,2,3\}$

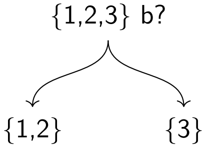
Splitting node: Separate States by Input



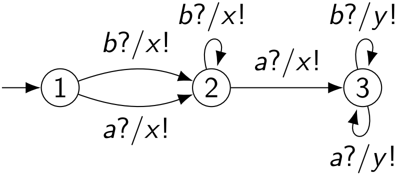
{1,2,3} b?

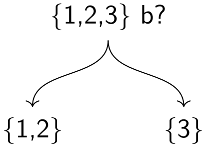
Splitting node: Separate States by Input





Splitting node: Separate States by Input

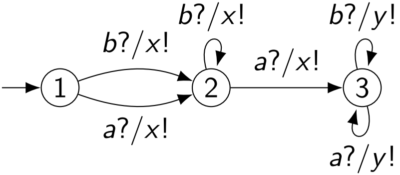


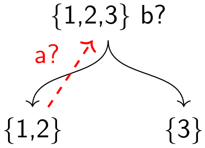


$\{1, 2\}$ can be split based on $a?$ and the split of $\{1, 2, 3\}$, because

- $\delta(1, a?) = 2$ and $\delta(2, a?) = 3$, and
- states 2 and 3 are already split in node $\{1, 2, 3\}$ (they are in different children of $\{1, 2, 3\}$)

Splitting node: Separate States by Input

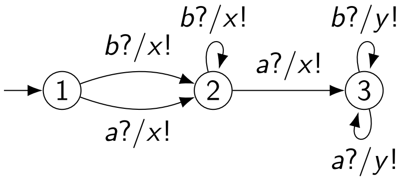


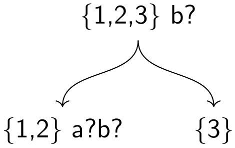


$\{1,2\}$ can be split based on $a?$ and the split of $\{1,2,3\}$, because

- $\delta(1, a?) = 2$ and $\delta(2, a?) = 3$, and
- states 2 and 3 are already split in node $\{1,2,3\}$ (they are in different children of $\{1,2,3\}$)

Splitting node: Separate States by Input

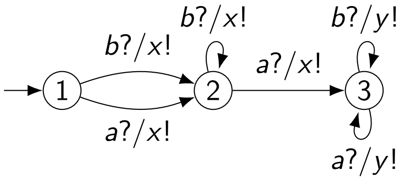


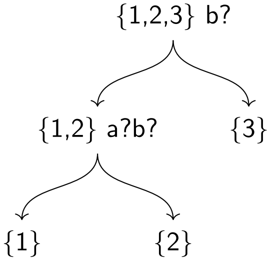


$\{1,2\}$ can be split based on $a?$ and the split of $\{1,2,3\}$, because

- $\delta(1,a?) = 2$ and $\delta(2,a?) = 3$, and
- states 2 and 3 are already split in node $\{1,2,3\}$ (they are in different children of $\{1,2,3\}$)

Splitting node: Separate States by Input





$\{1, 2\}$ can be split based on $a?$ and the split of $\{1, 2, 3\}$, because

- $\delta(1, a?) = 2$ and $\delta(2, a?) = 3$, and
- states 2 and 3 are already split in node $\{1, 2, 3\}$ (they are in different children of $\{1, 2, 3\}$)

$C = \{b?,\ a?b?\}$

Pseudo-Algorithm: What Did We Do?

Initialisation: create root with all states

Pseudo-Algorithm: What Did We Do?

Initialisation: create root with all states
repeat until no more splits can be made:
  pick any leaf N and input i:
    if λ gives different outputs for i, for different states in N
      split with N with i

Pseudo-Algorithm: What Did We Do?

Initialisation: create root with all states
repeat until no more splits can be made:
  pick any leaf N and input i:
    if λ gives different outputs for i, for different states in N
      split with N with i
repeat until finished:
  pick any leaf N and input i:
    if δ brings us to states already split with sequence σ
      split N with i
      append $i \cdot \sigma$ to N

Pseudo-Algorithm: What Did We Do?

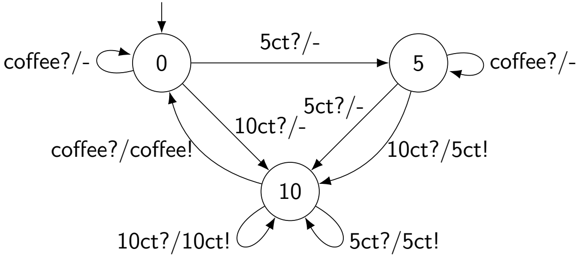
Initialisation: create root with all states
repeat until no more splits can be made:
    pick any leaf N and input i:
        if λ gives different outputs for i, for different states in N
            split with N with i
repeat until finished:
    pick any leaf N and input i:
        if δ brings us to states already split with sequence σ
            split N with i
            append $i \cdot \sigma$ to N

A split for node N and input i partitions N into multiple smaller parts

Testing Without Reset

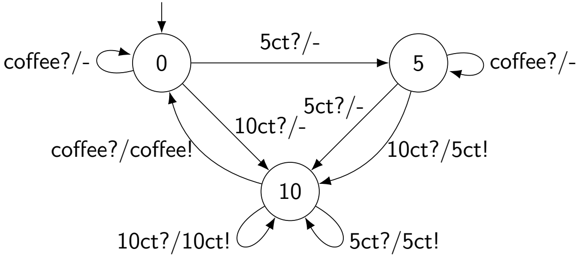
- To execute multiple tests a reset is needed!
- What if the SUT has no reset?
- Use a synchronising sequence:
  - A sequence which always ends in the same state
  - May not exist!
  - Instead of reset, synchronize to initial state
- (Synchronizing sequences are not k-complete!)

Synchronising Sequence



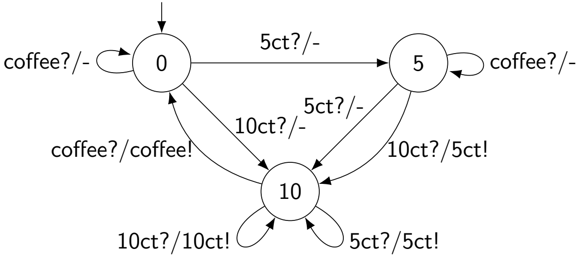
- to state 10:

Synchronising Sequence



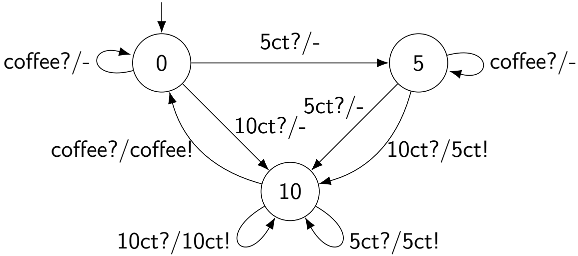
- to state 10: 10ct?
- to state 0:

Synchronising Sequence



- to state 10: 10ct?
- to state 0: 10ct? coffee?
- to state 5:

Synchronising Sequence

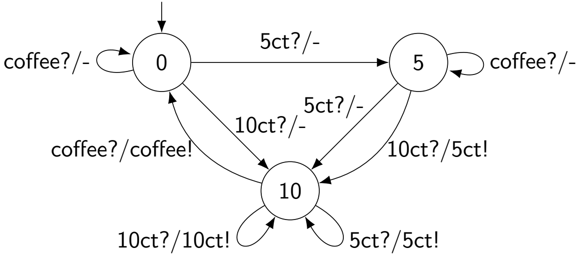


- to state 10: 10ct?
- to state 0: 10ct? coffee?
- to state 5: 10ct? coffee? 5ct?

Transition Tour

Alternative: make a transition tour

- long sequence visiting all transitions ending in initial state
- Can only detect output faults



coffee? 5ct? coffee? 5ct? 5ct? 10ct? coffee?
10ct? coffee?
5ct? 10ct? coffee?

Recap

- Finite state machines
- Equivalence
- k-complete test suite $= \mathbf{A} \cdot \mathbf{I}^{\leq \mathbf{k+1}} \cdot \mathbf{C}$ with
  - Access sequences A
  - Characterization set C, built up from
    - Separating sequences
    - Unique input output sequences (UIO)
    - Distinguishing sequence (DS)
- Algorithm for finding separating sequences
- No reset: transition tour or synchronising sequence

Questions?