UNIVERSITY OF TWENTE.

Formal Methods & Tools.

Quiescent Transition Systems: Model-based Testing with Quiescence

Gerjan Stokkink

March 25, 2012

MBT 2012

Joint work with M. Timmer & M. Stoelinga

Context: model-based testing

Model-based Testing (MBT) of a System Under Test (SUT):

1. Formally modelling the specification of SUT

Context: model-based testing

Model-based Testing (MBT) of a System Under Test (SUT):

1. Formally modelling the specification of SUT
2. Generating test cases from this model

Context: model-based testing

Model-based Testing (MBT) of a System Under Test (SUT):

1. Formally modelling the specification of SUT
2. Generating test cases from this model
3. Running test cases against the SUT

Context: model-based testing

Model-based Testing (MBT) of a System Under Test (SUT):

1. Formally modelling the specification of SUT
2. Generating test cases from this model
3. Running test cases against the SUT
4. Evaluating results

Context: model-based testing

Model-based Testing (MBT) of a System Under Test (SUT):

1. Formally modelling the specification of SUT
2. Generating test cases from this model
3. Running test cases against the SUT
4. Evaluating results

All this can be integrated in a MBT framework, such as `ioco` (input-output conformance).

Context: model-based testing

Model-based Testing (MBT) of a System Under Test (SUT):

1. Formally modelling the specification of SUT
2. Generating test cases from this model
3. Running test cases against the SUT
4. Evaluating results

All this can be integrated in a MBT framework, such as `ioco` (input-output conformance).

`ioco`-based tools: TVEDA, TGV, TestGen, TorX, etc.

Context: model-based testing

Model-based Testing (MBT) of a System Under Test (SUT):

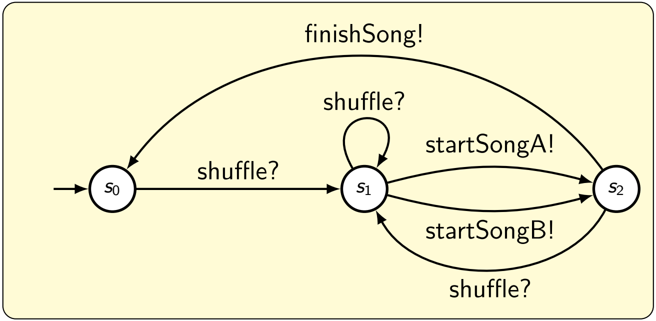
1. Formally modelling the specification of SUT
   - `ioco`: using IOTSs (suspension automata)
2. Generating test cases from this model
   - `ioco`: test cases as IOTSs (suspension automata)
3. Running test cases against the SUT
   - `ioco`: parallel composition
4. Evaluating results
   - `ioco`: using the `ioco` conformance relation

Context: model-based testing

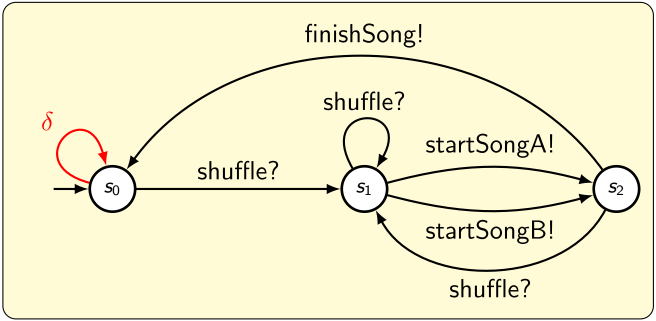
Model-based Testing (MBT) of a System Under Test (SUT):

1. Formally modelling the specification of SUT
   - `ioco`: using IOTSs (suspension automata)
2. Generating test cases from this model
   - `ioco`: test cases as IOTSs (suspension automata)
3. Running test cases against the SUT
   - `ioco`: parallel composition
4. Evaluating results
   - `ioco`: using the `ioco` conformance relation
     - No unexpected outputs.
     - No unexpected quiescence (absence of outputs).

Specification as IOTS.

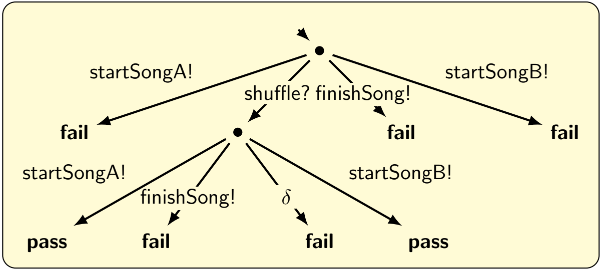


Specification as *suspension automaton*
(= 'observation automaton').

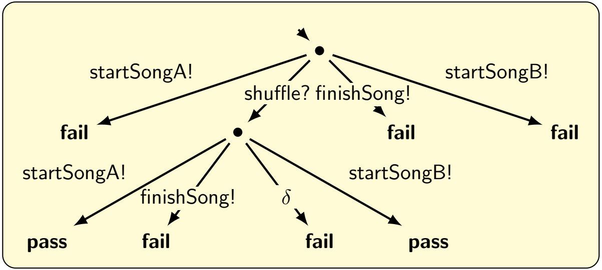


δ = observation of quiescence

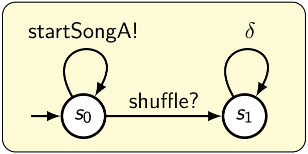
ioco: test case and test execution



(a) Test case

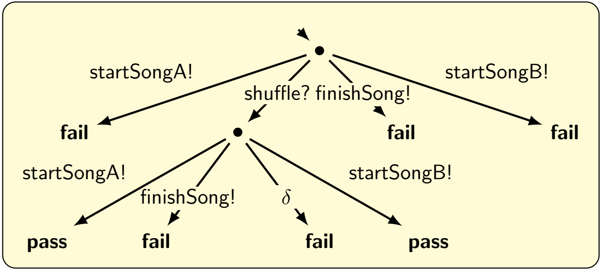


(a) Test case

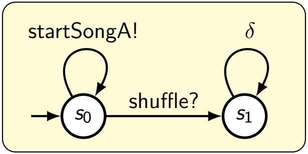


(b) Erroneous implementation

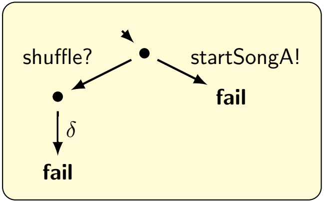
ioco: test case and test execution



(a) Test case



(b) Erroneous implementation



(c) Test execution

Limitations of suspension automata

- Suspension automata are not first-class citizens.

Limitations of suspension automata

- Suspension automata are not first-class citizens.
  - What properties does a suspension automaton have?
  - How do suspension automata behave under various operations?
  - Well-formedness?

Limitations of suspension automata

- Suspension automata are not first-class citizens.
  - What properties does a suspension automaton have?
  - How do suspension automata behave under various operations?
  - Well-formedness?
- Suspension automata must be convergent.
  - Divergence is assumed not to occur.
  - However, in practice it does occur.

Limitations of suspension automata

- Suspension automata are not first-class citizens.
  - What properties does a suspension automaton have?
  - How do suspension automata behave under various operations?
  - Well-formedness?
- Suspension automata must be convergent.
  - Divergence is assumed not to occur.
  - However, in practice it does occur.
- Suspension automata must be input-enabled.
  - Underspecification desirable for specifications.
  - Non-input-enabled suspension automata violate IOTS requirements.

Quiescent Transition Systems (QTSs)

Quiescent Transition Systems:

- First-class citizens.
- Fully formalised theory.

Quiescent Transition Systems (QTSs)

Quiescent Transition Systems:

- First-class citizens.
- Fully formalised theory.
  - Well-formedness formally defined.
  - Operations formally defined.
  - Closure and commutativity properties investigated.

Quiescent Transition Systems (QTSs)

Quiescent Transition Systems:

- First-class citizens.
- Fully formalised theory.
    - Well-formedness formally defined.
    - Operations formally defined.
    - Closure and commutativity properties investigated.
- Work in progress: divergence and non-input-enabledness.

Overview

1. Definition of QTSs
2. Well-formedness
3. Operations on well-formed QTSs
4. From IOTS to well-formed QTS: deltafication
5. Properties of well-formed QTSs
6. Conclusions and future work

QTSs: definition

Based on IOTSs.

Definition (Quiescent Transition Systems)

A *Quiescent Transition System* (QTS) $= \langle S, S^0, L^{\mathrm{I}}, L^{\mathrm{O}}, \rightarrow \rangle$:

- S is a non-empty set of states;
- S^0 is a non-empty set of initial states;
- L^{I} and L^{O} are disjoint sets of inputs and outputs; $L = L^{\mathrm{I}} \cup L^{\mathrm{O}}$
- Two special labels:
  - $\tau \notin L$ is the internal (unobservable) action;
  - $\delta \notin L$ denotes the observation of quiescence;
- $\rightarrow \; \subseteq S \times (L \cup \{ \tau, \delta \}) \times S$ is the transition relation.

Well-formedness

A QTS is *well-formed*, if:

Well-formedness

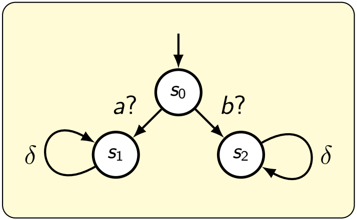
A QTS is *well-formed*, if:

1. R1: every quiescent state has an outgoing δ-transition.

Well-formedness

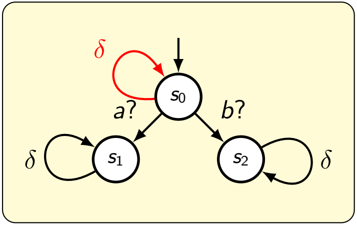
A QTS is *well-formed*, if:

1. R1: every quiescent state has an outgoing δ-transition.



A QTS is *well-formed*, if:

1. R1: every quiescent state has an outgoing δ-transition.



Well-formedness

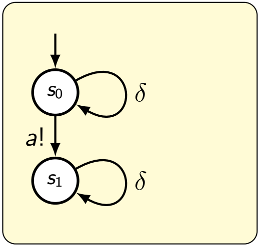
A QTS is *well-formed*, if:

1. R1: every quiescent state has an outgoing δ-transition.
2. R2: after a δ-transition, the new state is quiescent.

Well-formedness

A QTS is *well-formed*, if:

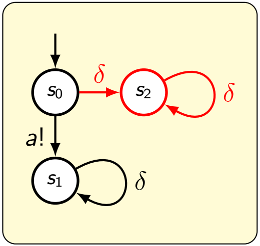
1. R1: every quiescent state has an outgoing δ-transition.
2. R2: after a δ-transition, the new state is quiescent.



Well-formedness

A QTS is *well-formed*, if:

1. R1: every quiescent state has an outgoing δ-transition.
2. R2: after a δ-transition, the new state is quiescent.



Well-formedness

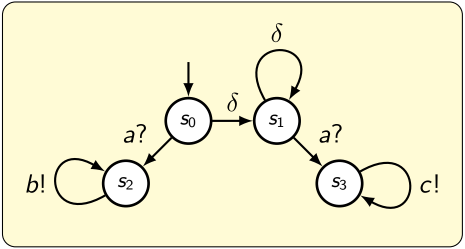
A QTS is *well-formed*, if:

1. R1: every quiescent state has an outgoing δ-transition.
2. R2: after a δ-transition, the new state is quiescent.
3. R3: quiescence does not introduce new behaviour.

Well-formedness

A QTS is *well-formed*, if:

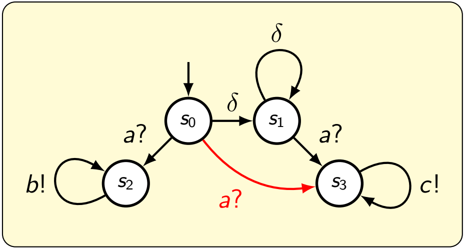
1. R1: every quiescent state has an outgoing δ-transition.
2. R2: after a δ-transition, the new state is quiescent.
3. R3: quiescence does not introduce new behaviour.



Well-formedness

A QTS is *well-formed*, if:

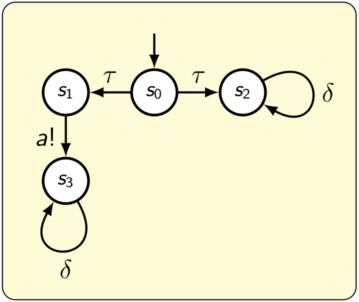
1. R1: every quiescent state has an outgoing δ-transition.
2. R2: after a δ-transition, the new state is quiescent.
3. R3: quiescence does not introduce new behaviour.



Well-formedness

A QTS is *well-formed*, if:

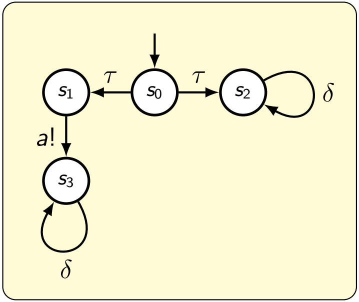
1. R1: every quiescent state has an outgoing δ-transition.
2. R2: after a δ-transition, the new state is quiescent.
3. R3: quiescence does not introduce new behaviour.

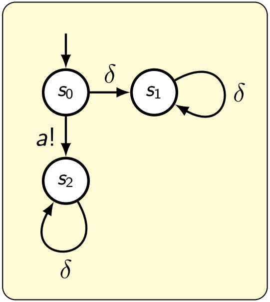


Well-formedness

A QTS is *well-formed*, if:

1. R1: every quiescent state has an outgoing δ-transition.
2. R2: after a δ-transition, the new state is quiescent.
3. R3: quiescence does not introduce new behaviour.





Well-formedness

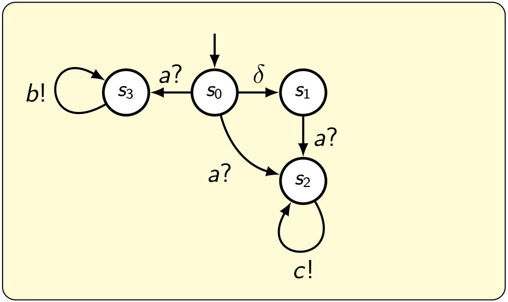
A QTS is *well-formed*, if:

1. R1: every quiescent state has an outgoing δ-transition.
2. R2: after a δ-transition, the new state is quiescent.
3. R3: quiescence does not introduce new behaviour.
4. R4: continued quiescence preserves behaviour.

Well-formedness

A QTS is *well-formed*, if:

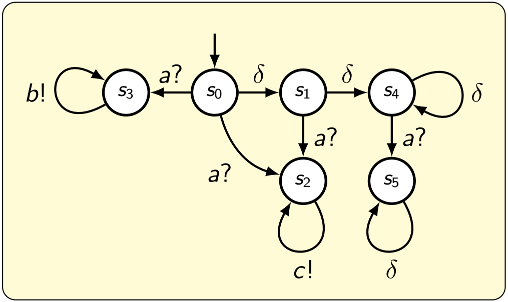
1. R1: every quiescent state has an outgoing δ-transition.
2. R2: after a δ-transition, the new state is quiescent.
3. R3: quiescence does not introduce new behaviour.
4. R4: continued quiescence preserves behaviour.



Well-formedness

A QTS is *well-formed*, if:

1. R1: every quiescent state has an outgoing δ-transition.
2. R2: after a δ-transition, the new state is quiescent.
3. R3: quiescence does not introduce new behaviour.
4. R4: continued quiescence preserves behaviour.



Well-formedness

A QTS is *well-formed*, if:

1. R1: every quiescent state has an outgoing δ-transition.
2. R2: after a δ-transition, the new state is quiescent.
3. R3: quiescence does not introduce new behaviour.
4. R4: continued quiescence preserves behaviour.

Every suspension automaton is a well-formed QTS, and vice versa.

Operations on well-formed QTSs

- Determinisation
- Hiding of actions
- Parallel composition

Operations on well-formed QTSs

- Determinisation
  - The same as for LTSs.
- Hiding of actions
- Parallel composition

Operations on well-formed QTSs

- Determinisation
  - The same as for LTSs.
- Hiding of actions
  - Similar to hiding for IOTSs.
  - Only outputs can be hidden.
  - δ cannot be hidden.
- Parallel composition

Operations on well-formed QTSs

- Determinisation
  - The same as for LTSs.
- Hiding of actions
  - Similar to hiding for IOTSs.
  - Only outputs can be hidden.
  - δ cannot be hidden.
- Parallel composition
  - Similar to parallel composition for IOTSs.
  - Synchronise on shared inputs.
  - Synchronise on complementary input-output pairs.
  - Synchronise on δ-transitions.

From IOTS to well-formed QTS: deltafication

Specification often modelled as IOTSs; how to convert these to well-formed QTSs?

From IOTS to well-formed QTS: deltafication

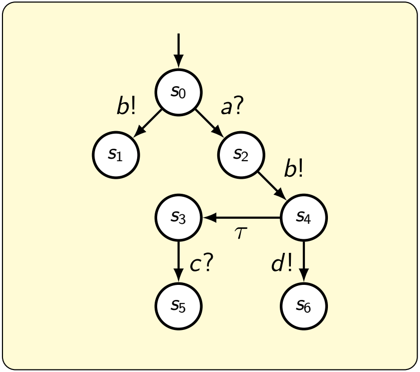
Specification often modelled as IOTSs; how to convert these to well-formed QTSs?

Deltafication: add a δ-labelled self-loop to all quiescent states in the IOTS.

From IOTS to well-formed QTS: deltafication

Specification often modelled as IOTSs; how to convert these to well-formed QTSs?

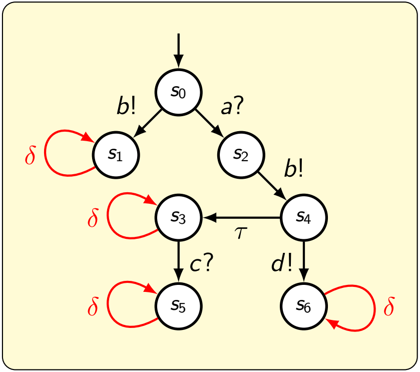
Deltafication: add a δ-labelled self-loop to all quiescent states in the IOTS.



From IOTS to well-formed QTS: deltafication

Specification often modelled as IOTSs; how to convert these to well-formed QTSs?

Deltafication: add a δ-labelled self-loop to all quiescent states in the IOTS.



From IOTS to well-formed QTS: deltafication

Specification often modelled as IOTSs; how to convert these to well-formed QTSs?

Deltafication: add a δ-labelled self-loop to all quiescent states in the IOTS.

Theorem

Given an IOTS $\mathcal{A}$, the deltafication $\delta(\mathcal{A})$ is a well-formed QTS.

From IOTS to well-formed QTS: deltafication

Specification often modelled as IOTSs; how to convert these to well-formed QTSs?

Deltafication: add a δ-labelled self-loop to all quiescent states in the IOTS.

Theorem

Given an IOTS $\mathcal{A}$, the deltafication $\delta(\mathcal{A})$ is a well-formed QTS.

Thus, given an IOTS $\mathcal{A}$, the deltafication $\delta(\mathcal{A})$ satisfies rules R1, R2, R3 and R4.

Properties of well-formed QTSs

Mainly interested in two kinds of properties:

- Closure properties.

- Commutativity of deltafication.

Properties of well-formed QTSs

Mainly interested in two kinds of properties:

- Closure properties.
  - Closed under deltafication?
  - Closed under action hiding?
  - Closed under parallel composition?
  - Closed under determinisation?
- Commutativity of deltafication.

Properties of well-formed QTSs

Mainly interested in two kinds of properties:

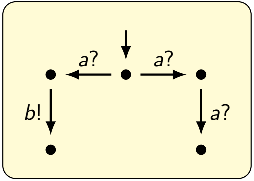
- Closure properties.
  - Closed under deltafication? ✓
  - Closed under action hiding? ✓
  - Closed under parallel composition? ✓
  - Closed under determinisation? ✓
- Commutativity of deltafication.

Properties of well-formed QTSs

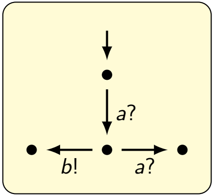
Mainly interested in two kinds of properties:

- Closure properties.
  - Closed under deltafication? ✓
  - Closed under action hiding? ✓
  - Closed under parallel composition? ✓
  - Closed under determinisation? ✓
- Commutativity of deltafication.
  - Commutative with action hiding? ✓
  - Commutative with parallel composition? ✓
  - Commutative with determinisation? ✗

Deltafication and determinisation do not commute

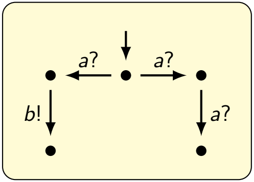


(a) $\mathcal{A}$

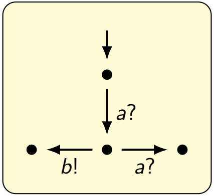


(b) $det(\mathcal{A})$

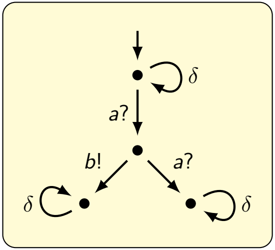
Deltafication and determinisation do not commute



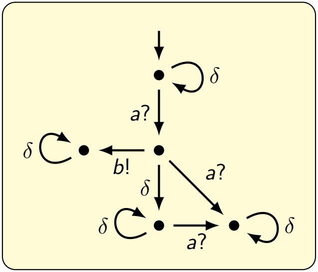
(a) $\mathcal{A}$



(b) $det(\mathcal{A})$

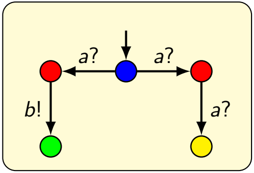


(c) $\delta(det(\mathcal{A}))$

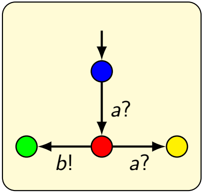


(d) $det(\delta(\mathcal{A}))$

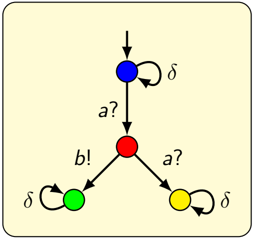
Deltafication and determinisation do not commute



(a) $\mathcal{A}$

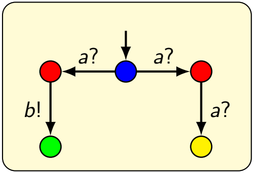


(b) $det(\mathcal{A})$

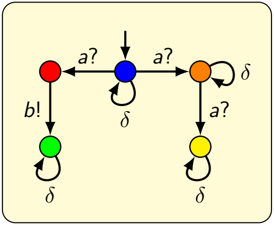


(c) $\delta(det(\mathcal{A}))$

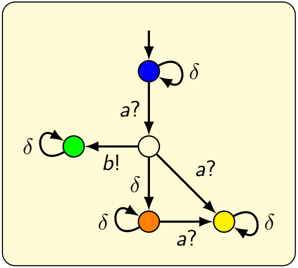
Deltafication and determinisation do not commute



(a) $\mathcal{A}$



(b) $\delta(\mathcal{A})$



(c) $det(\delta(\mathcal{A}))$

Conclusions

- QTSs: new, rigorous theory.
- Many desirable properties regarding composition.
- Drop-in replacement for suspension automata.
- QTSs offer a solid basis for `ioco`.

Conclusions

- QTSs: new, rigorous theory.
- Many desirable properties regarding composition.
- Drop-in replacement for suspension automata.
- QTSs offer a solid basis for `ioco`.
- Can easily be extended.

Future work

Extend QTS theory (work in progress):

- No input-enabledness requirement.
- Divergence allowed (i.e., `ioco` with divergence possible).
- Same well-formedness definition.
- Same properties satisfied.

Future work

Extend QTS theory (work in progress):

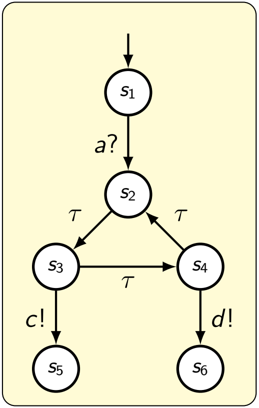
- No input-enabledness requirement.
- Divergence allowed (i.e., `ioco` with divergence possible).
- Same well-formedness definition.
- Same properties satisfied.

New paper coming soon!

Overview

1. Quiescent Transition Systems (QTSs)
2. Well-formedness
3. Operations on well-formed QTSs
4. From IOTS to well-formed QTS: deltafication
5. Properties of well-formed QTSs
6. Conclusions and future work

The trouble with divergence



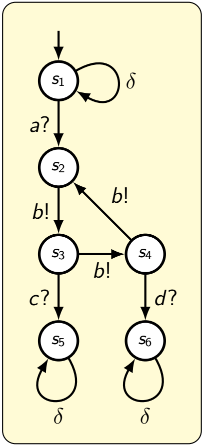
Clearly, states s_1, s_5 and s_6 are quiescent. But what about s_2, s_3 and s_4?

Depends whether an execution corresponding to path $s_2 \tau s_3 \tau s_4 \tau s_2 \ldots$ can actually occur!

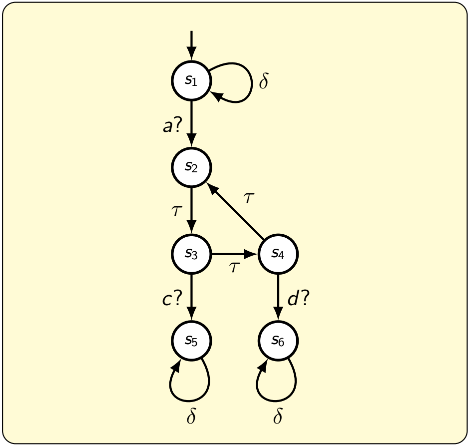
We need some notion of fairness for this.

Borrow locally controlled actions partitioning from Input/Output Automata.

Deltafication and divergence

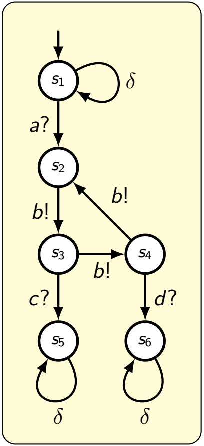


(a) $\mathcal{A}$

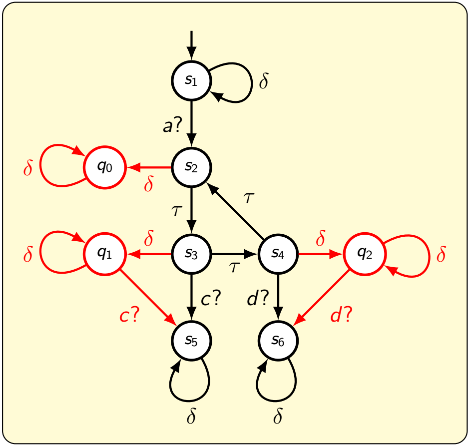


(b) $\mathcal{A} \setminus \{ b \}$

Deltafication and divergence



(a) $\mathcal{A}$



(b) $\mathcal{A} \setminus \{ b \}$