Safety Contracts for Timed Reactive Components

Iulia Dragomir, Iulian Ober and Christian Percebois

IRIT - University of Toulouse

March 21, 2013

Outline

1. Motivation
2. Contract-based Reasoning
3. Component framework: Timed Input/Output Automata
4. A toy example
5. Contract framework for Timed Input/Output Automata
6. Applying contract-based reasoning on the toy example
7. Conclusions

Outline

1. Motivation

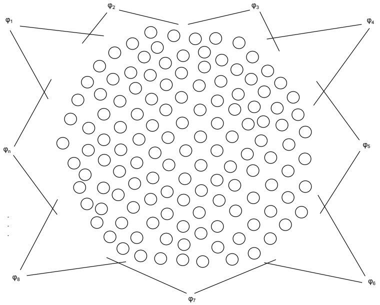
Context & Problematics

Context: development of component-based critical real-time embedded systems

Context & Problematics

Context: development of component-based critical real-time embedded systems

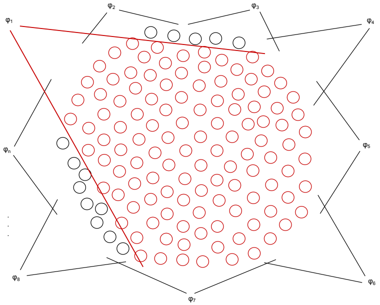
Let S be a component-based system and $\varphi_1, \cdots, \varphi_n$ a set of requirements.



Context & Problematics

Context: development of component-based critical real-time embedded systems

Let S be a component-based system and $\varphi_1, \cdots, \varphi_n$ a set of requirements.

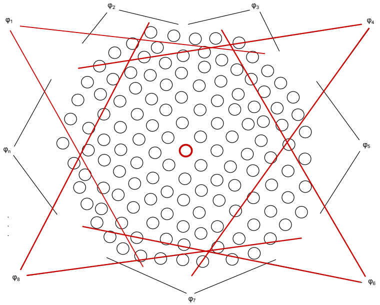


- A requirement is in general satisfied by the collaboration of a set of components

Context & Problematics

Context: development of component-based critical real-time embedded systems

Let S be a component-based system and $\varphi_1, \cdots, \varphi_n$ a set of requirements.

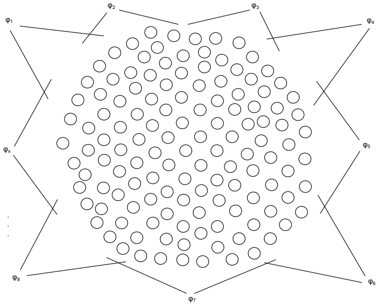


- A requirement is in general satisfied by the collaboration of a set of components
- Each component is involved in the satisfaction of several requirements

Context & Problematics

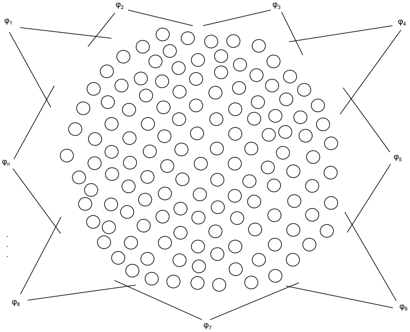
Context: development of component-based critical real-time embedded systems

Let S be a component-based system and $\varphi_1, \cdots, \varphi_n$ a set of requirements.

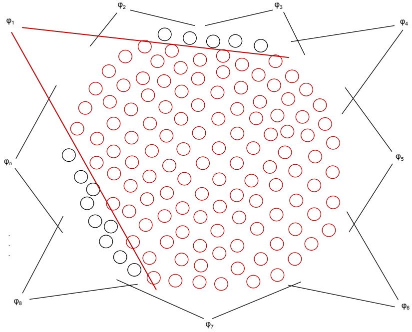


- A requirement is in general satisfied by the collaboration of a set of components
- Each component is involved in the satisfaction of several requirements
  $\Rightarrow$ the need for components abstractions

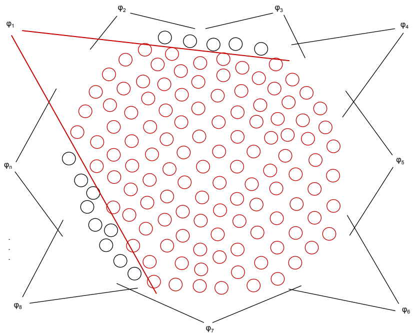
Verification by abstractions



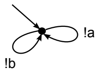
Verification by abstractions



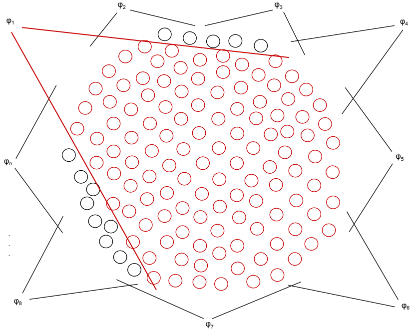
Verification by abstractions

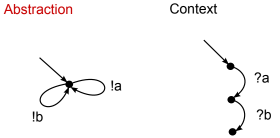


Abstraction

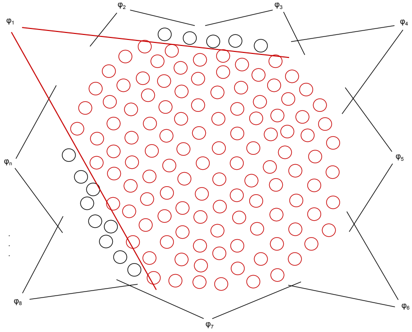


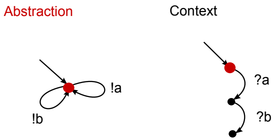
Verification by abstractions



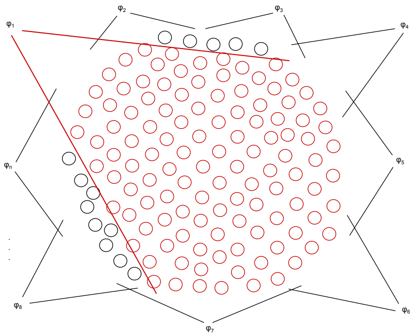


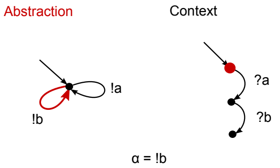
Verification by abstractions



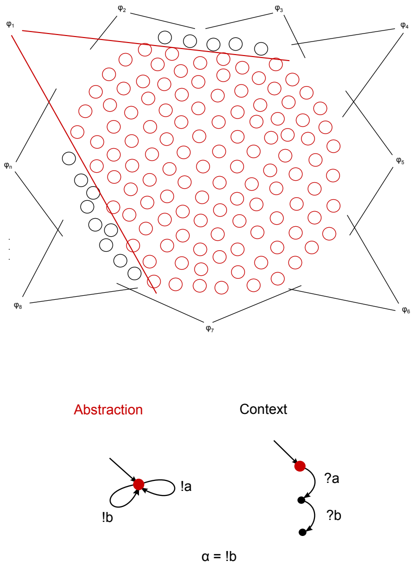


Verification by abstractions



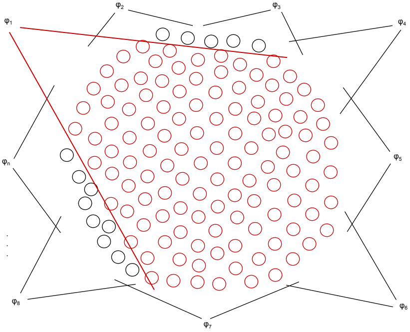


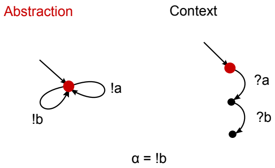
Verification by abstractions



- Deadlock due to the abstraction

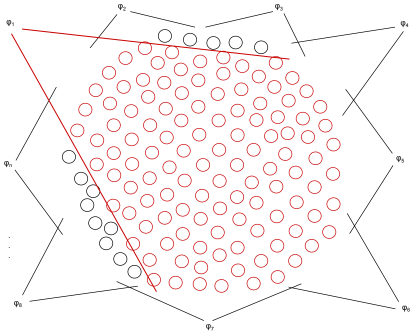
Verification by abstractions

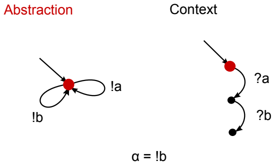




- Deadlock due to the abstraction
  → Not sufficient!

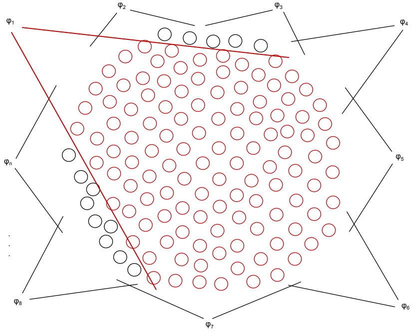
Verification by abstractions

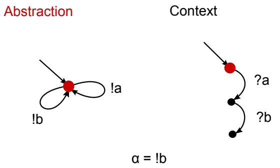




- Deadlock due to the abstraction
  $\rightarrow$ Not sufficient!
- An abstraction has to be *correct* in a *context*

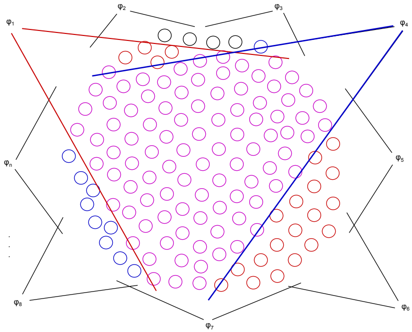
Verification by abstractions





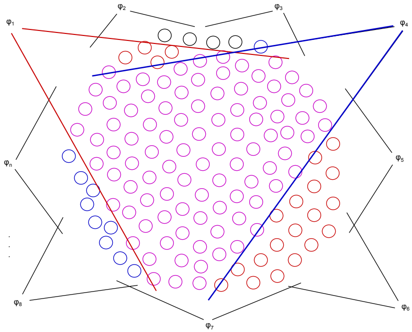
- Deadlock due to the abstraction
  → Not sufficient!
- An abstraction has to be *correct* in a *context*
  → usage of contracts

Introducing contracts



Contract:

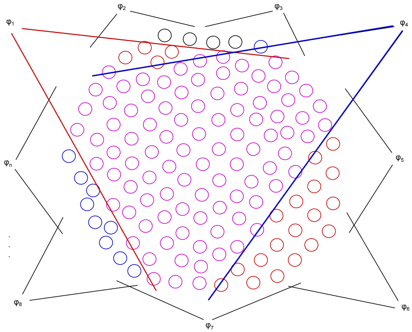
Introducing contracts



Contract:

- defines partial and abstract component specification for one component and one requirement

Introducing contracts



Contract:

- defines partial and abstract component specification for one component and one requirement
- is a pair (*assumption*, *guarantee*)

Using contracts: why?

- Requirement specification and decomposition

Using contracts: why?

- Requirement specification and decomposition
- Mapping and tracing requirements

Using contracts: why?

- Requirement specification and decomposition
- Mapping and tracing requirements
- Model reviews

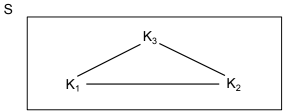
Using contracts: why?

- Requirement specification and decomposition
- Mapping and tracing requirements
- Model reviews
- Verification of system designs (in SysML)

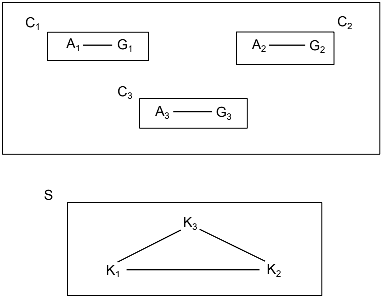
Outline

2. Contract-based Reasoning

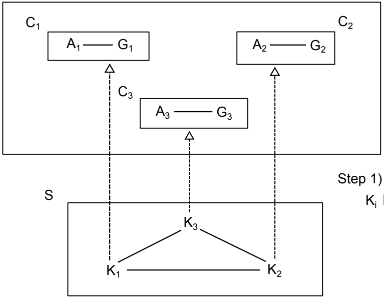
Contract-based Reasoning



Contract-based Reasoning



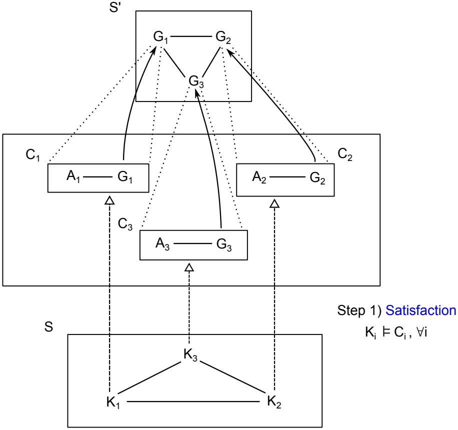
Contract-based Reasoning



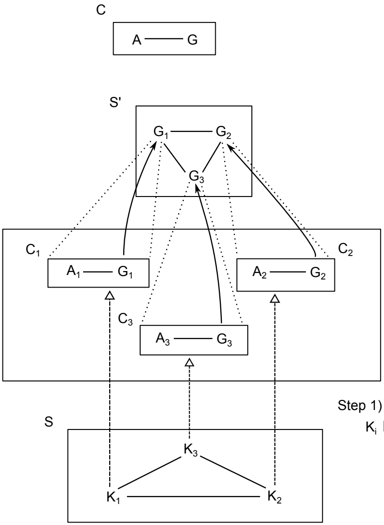
Step 1) Satisfaction

$K_i \models C_i, \forall i$

Contract-based Reasoning



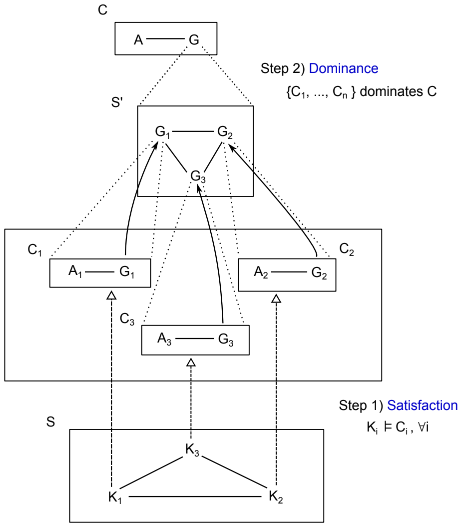
Contract-based Reasoning



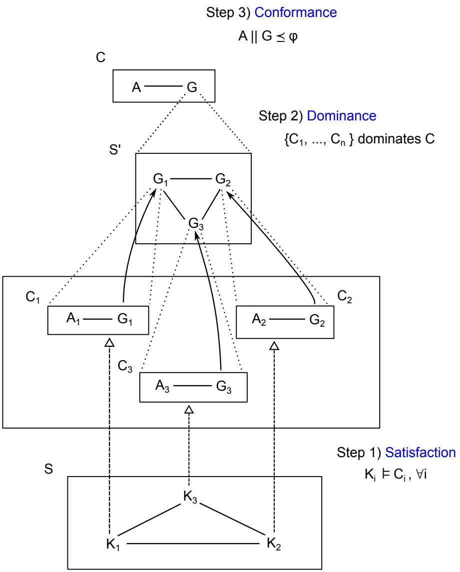
Step 1) Satisfaction

$K_i \models C_i, \forall i$

Contract-based Reasoning



Contract-based Reasoning



Contract-based approach for component-based systems

- Formalization of component framework

Contract-based approach for component-based systems

- Formalization of component framework
- Verification relations

Contract-based approach for component-based systems

- Formalization of component framework
- Verification relations
  1. Contract satisfaction
  2. Dominance between contracts
  3. Conformance

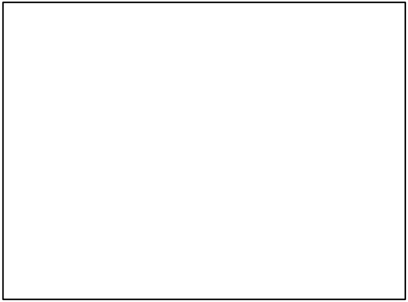
Outline

3. Component framework: Timed Input/Output Automata

Component: Timed Input/Output Automaton

Definition

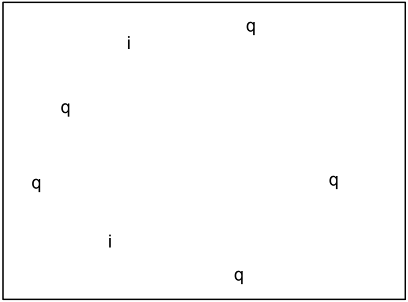
Timed input/output automaton $\mathcal{A}$



Component: Timed Input/Output Automaton

Definition

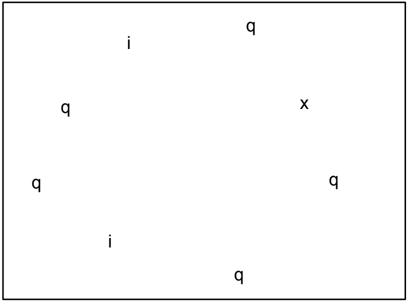
Timed input/output automaton $\mathcal{A} = (X,$



Component: Timed Input/Output Automaton

Definition

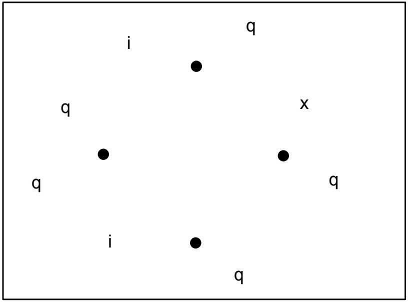
Timed input/output automaton $\mathcal{A} = (X, Clk,$



Component: Timed Input/Output Automaton

Definition

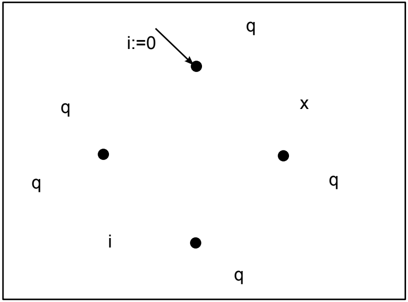
Timed input/output automaton $\mathcal{A} = (X, Clk, Q,$



Component: Timed Input/Output Automaton

Definition

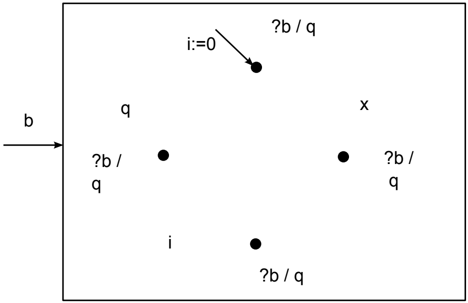
Timed input/output automaton $\mathcal{A} = (X, Clk, Q, \theta,$



Component: Timed Input/Output Automaton

Definition

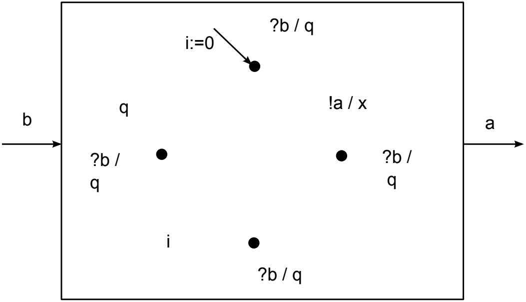
Timed input/output automaton $\mathcal{A} = (X, Clk, Q, \theta, I,$



Component: Timed Input/Output Automaton

Definition

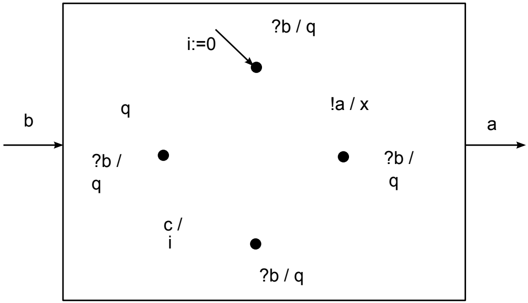
Timed input/output automaton $\mathcal{A} = (X, Clk, Q, \theta, I, O,$



Component: Timed Input/Output Automaton

Definition

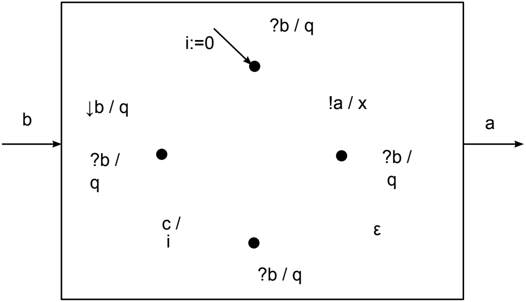
Timed input/output automaton $\mathcal{A} = (X, Clk, Q, \theta, I, O, V,$



Component: Timed Input/Output Automaton

Definition

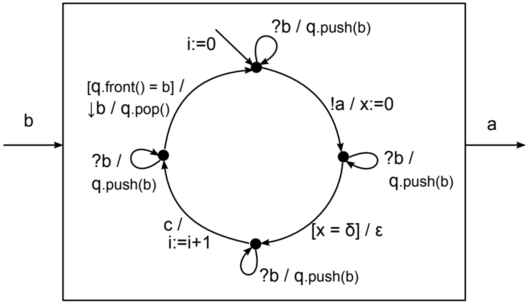
Timed input/output automaton $\mathcal{A} = (X, Clk, Q, \theta, I, O, V, H,$



Component: Timed Input/Output Automaton

Definition

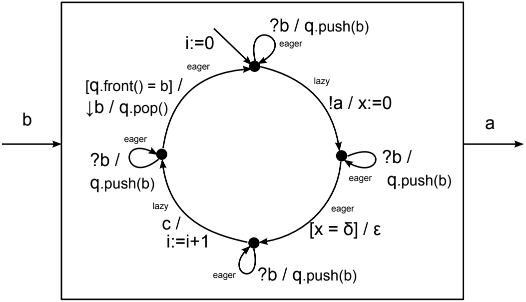
Timed input/output automaton $\mathcal{A} = (X, Clk, Q, \theta, I, O, V, H, D,$



Component: Timed Input/Output Automaton

Definition

Timed input/output automaton $\mathcal{A} = (X, Clk, Q, \theta, I, O, V, H, D, \mathcal{T})$



Properties of a timed input/output automaton

1. Existence of *point trajectories*: $\forall \mathrm{x} \in Q, \gamma(\mathrm{x}) : [0,0] \rightarrow \mathrm{x} \in \mathcal{T}$

Properties of a timed input/output automaton

1. Existence of *point trajectories*: $\forall \mathrm{x} \in Q, \gamma(\mathrm{x}) : [0,0] \rightarrow \mathrm{x} \in \mathcal{T}$
2. Prefix, suffix and concatenation closure of $\mathcal{T}$

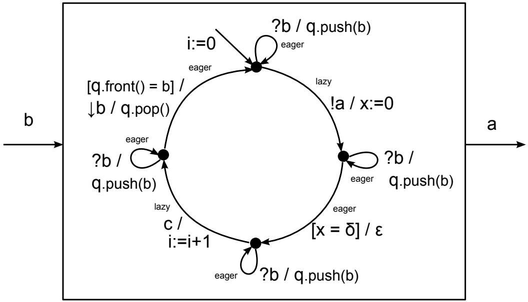
Properties of a timed input/output automaton

1. Existence of *point trajectories*: $\forall \mathrm{x} \in Q, \gamma(\mathrm{x}) : [0,0] \to \mathrm{x} \in \mathcal{T}$
2. Prefix, suffix and concatenation closure of $\mathcal{T}$
3. Input actions enabling: $\forall \mathrm{x} \in Q, \forall a \in I, \exists \mathrm{x}' \in Q$ such that $\mathrm{x} \xrightarrow{?a} \mathrm{x}'$

Properties of a timed input/output automaton

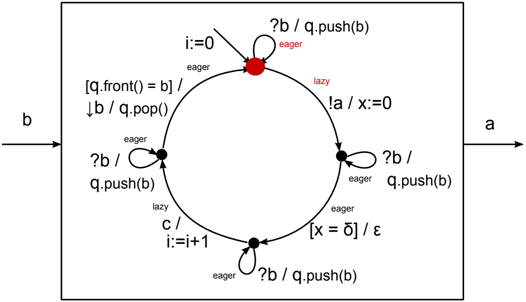
1. Existence of *point trajectories*: $\forall \mathrm{x} \in Q, \gamma(\mathrm{x}) : [0,0] \rightarrow \mathrm{x} \in \mathcal{T}$
2. Prefix, suffix and concatenation closure of $\mathcal{T}$
3. Input actions enabling: $\forall \mathrm{x} \in Q, \forall a \in I, \exists \mathrm{x}' \in Q$ such that $\mathrm{x} \stackrel{?a}{\rightarrow} \mathrm{x}'$
4. Time-passage enabling: $\forall \mathrm{x} \in Q, \exists \tau \in \mathcal{T}$ such that $\tau(0) = \mathrm{x}$ and either
   - $\tau.\mathit{limit\_time} = \infty$ or
   - τ is closed and some $l \in O \cup V \cup H$ is enabled is $\tau(\tau.\mathit{limit\_time})$

TIOA behaviour



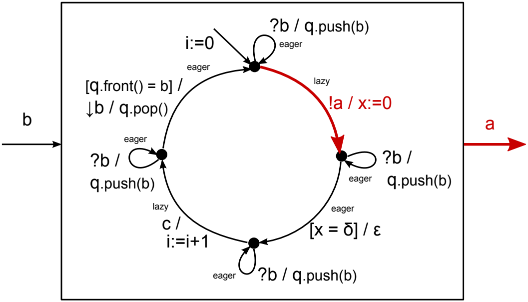
- *Execution fragment*: sequence of trajectories and actions
  Example:

TIOA behaviour



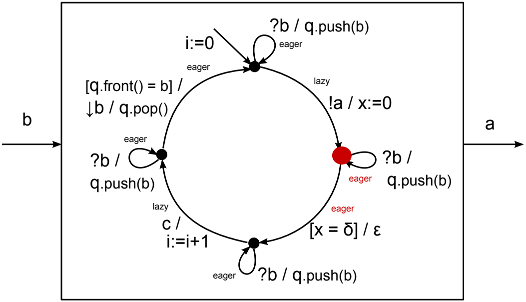
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0, 0]$

TIOA behaviour



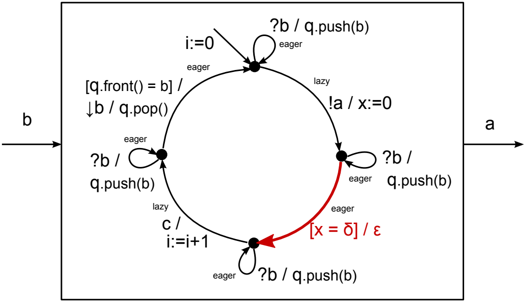
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a$

TIOA behaviour



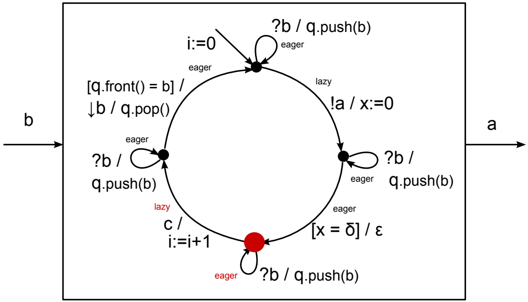
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0, 0]!a[0, \delta]$

TIOA behaviour



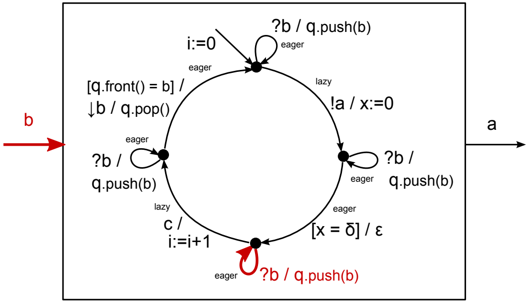
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0, 0]!a[0, \delta]\epsilon$

TIOA behaviour



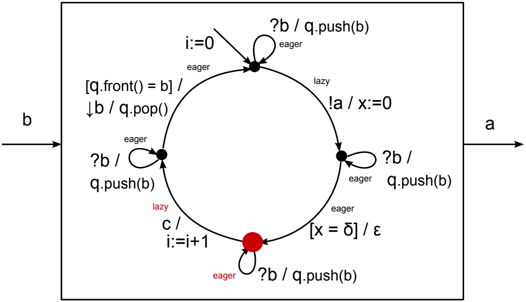
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0, 0]!a[0, \delta]\epsilon[0, \delta]$

TIOA behaviour



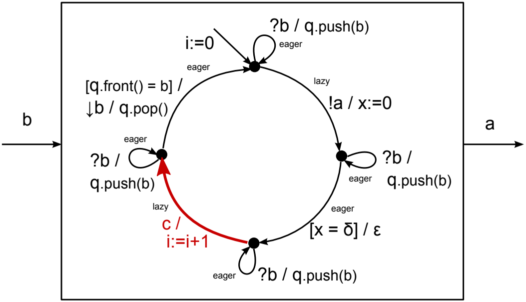
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b$

TIOA behaviour



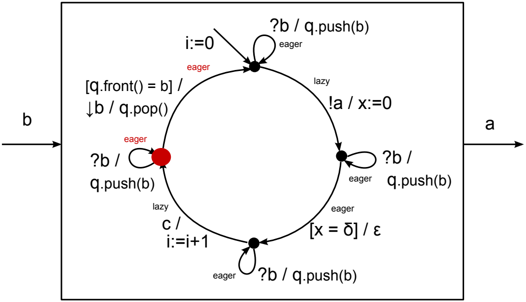
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]$

TIOA behaviour



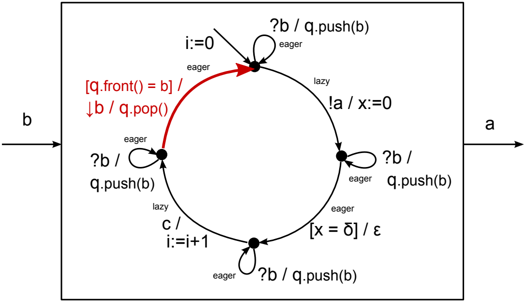
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c$

TIOA behaviour



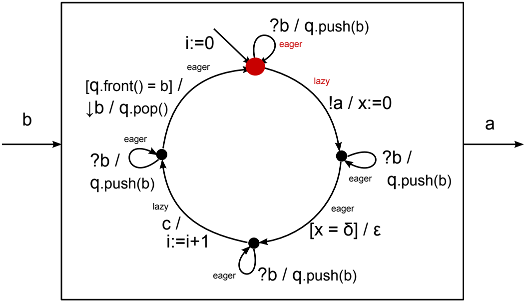
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]$

TIOA behaviour



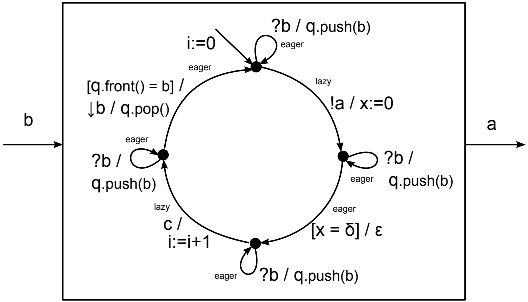
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\downarrow b$

TIOA behaviour



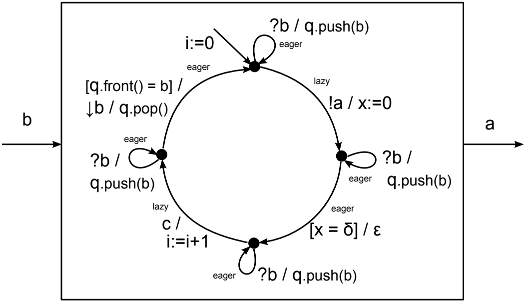
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0, 0]!a[0, \delta]\epsilon[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0]$

TIOA behaviour



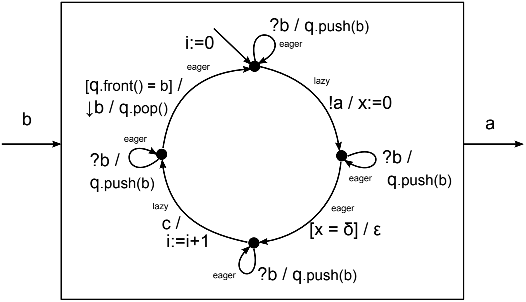
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\downarrow b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) =$

TIOA behaviour



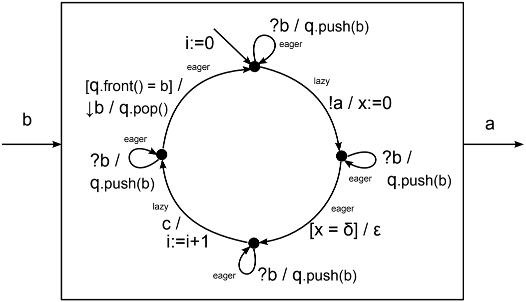
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\downarrow b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0,0]$

TIOA behaviour



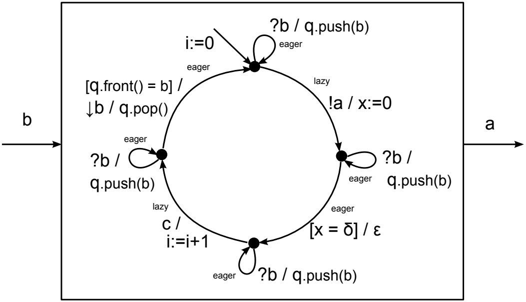
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\downarrow b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0,0]!a$

TIOA behaviour



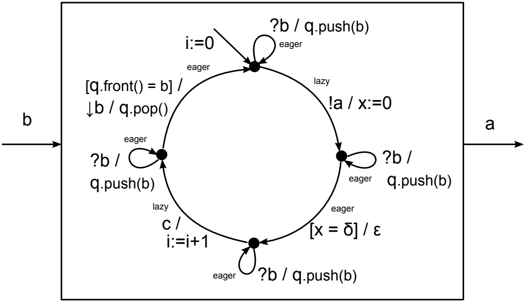
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\downarrow b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0,0]!a[0,\delta]$

TIOA behaviour



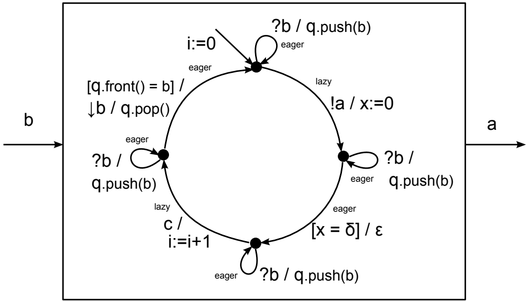
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\downarrow b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0,0]!a[0,\delta]$

TIOA behaviour



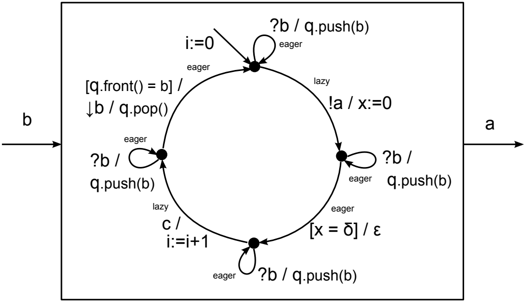
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\downarrow b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0,0]!a[0,\delta][0,\delta]$

TIOA behaviour



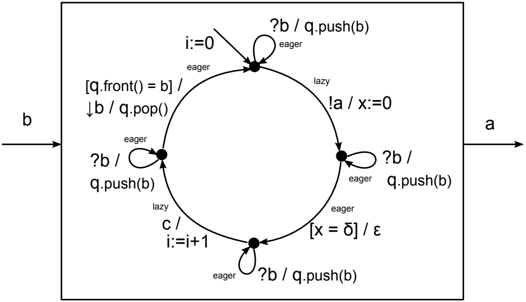
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\!\downarrow b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0,0]!a[0,\delta][0,\delta]$

TIOA behaviour



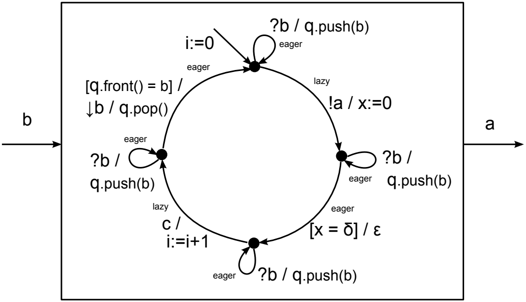
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\downarrow b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0,0]!a[0, 2*\delta]$

TIOA behaviour



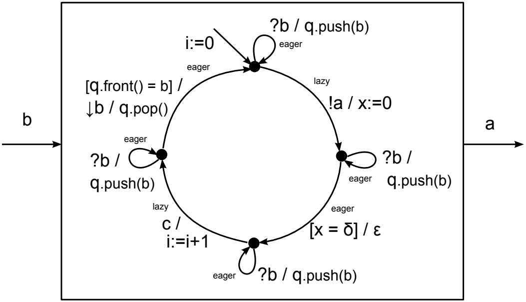
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\downarrow b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0,0]!a[0,2*\delta]?b$

TIOA behaviour



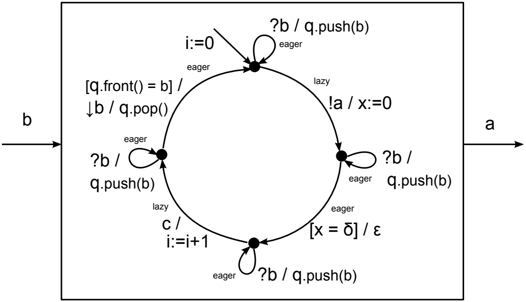
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\downarrow b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0,0]!a[0,2*\delta]?b[0,0]$

TIOA behaviour



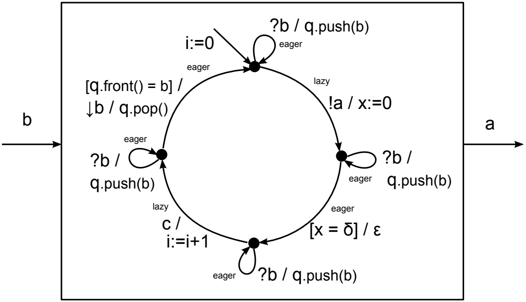
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\downarrow b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0,0]!a[0,2*\delta]?b[0,0]c$

TIOA behaviour



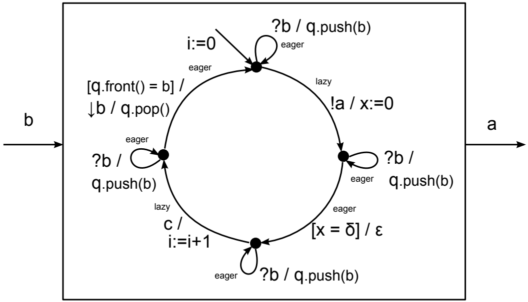
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\downarrow b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0,0]!a[0,2*\delta]?b[0,0]c[0,0]$

TIOA behaviour



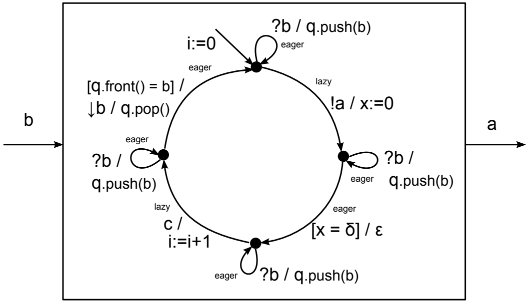
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\downarrow b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0,0]!a[0,2*\delta]?b[0,0]c[0,0]$

TIOA behaviour



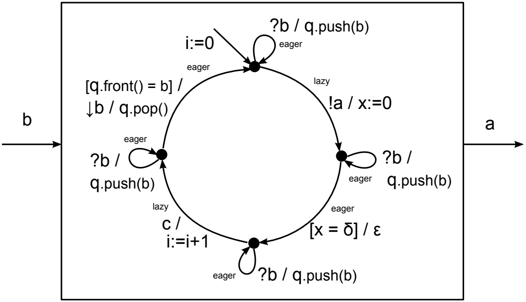
- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]{\downarrow}\, b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0,0]!a[0, 2*\delta]?b[0,0]c[0,0][0,0]$

TIOA behaviour



- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0,0]!a[0,\delta]\epsilon[0,\delta]?b[0,0]c[0,0]\downarrow b[0,0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0,0]!a[0,2*\delta]?b[0,0]c[0,0][0,0]$

TIOA behaviour



- *Execution fragment*: sequence of trajectories and actions
  Example: $\alpha = [0, 0]!a[0, \delta]\epsilon[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0]$
- *Trace*: sequence of time-passage lengths and external actions
  Example: $trace(\alpha) = [0, 0]!a[0, 2 * \delta]?b[0, 0]c[0, 0]$

TIOA composition

Composition compatibility:
$Y_i \cap Y_j = H_i \cap A_j = V_i \cap A_j = O_i \cap O_j = I_i \cap I_j = \emptyset,$ for $i \neq j$

TIOA composition

Composition compatibility:

$Y_i \cap Y_j = H_i \cap A_j = V_i \cap A_j = O_i \cap O_j = I_i \cap I_j = \emptyset,$ for $i \neq j$

Parallel composition:

$$\frac{\mathrm{x}_1 \xrightarrow{a} \mathrm{x}_1'}{(\mathrm{x}_1 \cup \mathrm{x}_2) \xrightarrow{a} (\mathrm{x}_1' \cup \mathrm{x}_2)} (a \in A_1 \setminus A_2)$$

$$\frac{\mathrm{x}_2 \xrightarrow{a} \mathrm{x}_2'}{(\mathrm{x}_1 \cup \mathrm{x}_2) \xrightarrow{a} (\mathrm{x}_1 \cup \mathrm{x}_2')} (a \in A_2 \setminus A_1)$$

TIOA composition

Composition compatibility:
$Y_i \cap Y_j = H_i \cap A_j = V_i \cap A_j = O_i \cap O_j = I_i \cap I_j = \emptyset$, for $i \neq j$

Parallel composition:

$$\frac{\mathrm{x}_1 \xrightarrow{a} \mathrm{x}_1'}{(\mathrm{x}_1 \cup \mathrm{x}_2) \xrightarrow{a} (\mathrm{x}_1' \cup \mathrm{x}_2)} (a \in A_1 \setminus A_2)$$

$$\frac{\mathrm{x}_2 \xrightarrow{a} \mathrm{x}_2'}{(\mathrm{x}_1 \cup \mathrm{x}_2) \xrightarrow{a} (\mathrm{x}_1 \cup \mathrm{x}_2')} (a \in A_2 \setminus A_1)$$

$$\frac{\mathrm{x}_1 \xrightarrow{a} \mathrm{x}_1' \wedge \mathrm{x}_2 \xrightarrow{a} \mathrm{x}_2'}{(\mathrm{x}_1 \cup \mathrm{x}_2) \xrightarrow{a} (\mathrm{x}_1' \cup \mathrm{x}_2')} (a \in (A_1 \cap A_2) \cup (\mathcal{T}_1 \wedge \mathcal{T}_2))$$

TIOA composition

Composition compatibility:

$Y_i \cap Y_j = H_i \cap A_j = V_i \cap A_j = O_i \cap O_j = I_i \cap I_j = \emptyset,$ for $i \neq j$

Parallel composition:

$$\frac{\mathrm{x}_1 \xrightarrow{a} \mathrm{x}_1'}{(\mathrm{x}_1 \cup \mathrm{x}_2) \xrightarrow{a} (\mathrm{x}_1' \cup \mathrm{x}_2)}(a \in A_1 \setminus A_2)$$

$$\frac{\mathrm{x}_2 \xrightarrow{a} \mathrm{x}_2'}{(\mathrm{x}_1 \cup \mathrm{x}_2) \xrightarrow{a} (\mathrm{x}_1 \cup \mathrm{x}_2')}(a \in A_2 \setminus A_1)$$

$$\frac{\mathrm{x}_1 \xrightarrow{a} \mathrm{x}_1' \wedge \mathrm{x}_2 \xrightarrow{a} \mathrm{x}_2'}{(\mathrm{x}_1 \cup \mathrm{x}_2) \xrightarrow{a} (\mathrm{x}_1' \cup \mathrm{x}_2')}(a \in (A_1 \cap A_2) \cup (\mathcal{T}_1 \wedge \mathcal{T}_2))$$

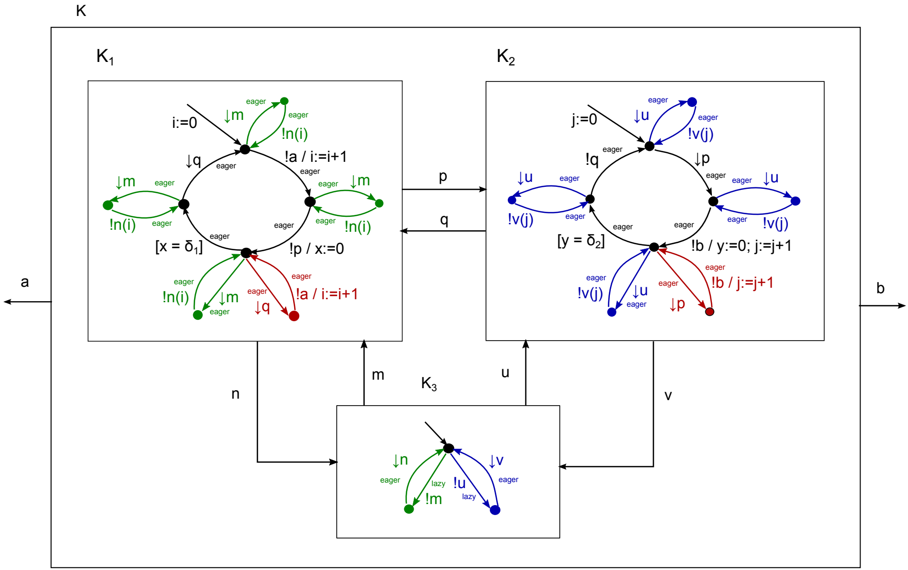
Theorem

The parallel composition operator is commutative and associative.

Outline

4. A toy example

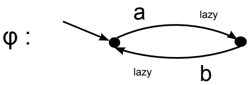
Running example



Property φ to be checked

Property

Given $\delta_1 < \delta_2$, the subsystem doesn't emit consecutive *a*'s or *b*'s.



if $\delta_1 < \delta_2$

Outline

5. Contract framework for Timed Input/Output Automata

Formal contract

Component K: a timed input/output automaton

Formal contract

Component K: a timed input/output automaton

Closed component: $I = O = \emptyset$
Open component: it is not closed

Formal contract

Component K: a timed input/output automaton

Closed component: $I = O = \emptyset$
Open component: it is not closed

Environment E for K: a timed input/output automaton compatible with K such that $I_E \subseteq O_K$ and $O_E \subseteq I_K$

Formal contract

Component K: a timed input/output automaton

Closed component: $I = O = \emptyset$
Open component: it is not closed

Environment E for K: a timed input/output automaton compatible with K such that $I_E \subseteq O_K$ and $O_E \subseteq I_K$

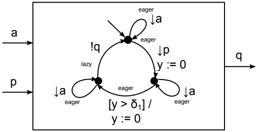
Definition

A *contract* for a component K is a pair (A, G) of TIOA such that $I_A = O_G$ and $O_A = I_G$ (i.e. the composition is a closed system) and $I_G \subseteq I_K$ and $O_G \subseteq O_K$ (i.e. the interface of K is a refinement of that of G).

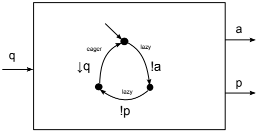
Contracts for the running example

$C_1 = (A_1, G_1)$ contract for K_1

A_1



G_1

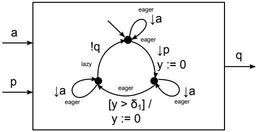


Contracts for the running example

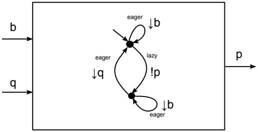
$C_1 = (A_1, G_1)$ contract for K_1

$C_2 = (A_2, G_2)$ contract for K_2

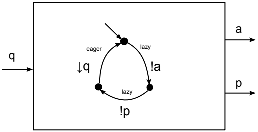
A_1



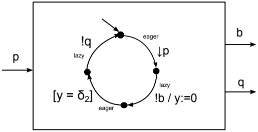
A_2



G_1



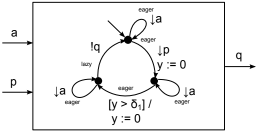
G_2



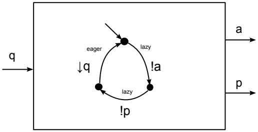
Contracts for the running example

$C_1 = (A_1, G_1)$ contract for K_1

A_1

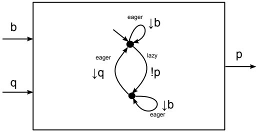


G_1

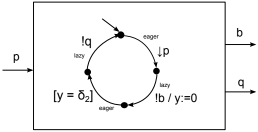


$C_2 = (A_2, G_2)$ contract for K_2

A_2



G_2



$C_3 = (A_3, G_3)$ contract for K_3

A_3

G_3

Conformance relation

Conformance relation

Definition

Let K_1 and K_2 be two comparable components (i.e. having the same external interface). $K_1 \preceq K_2$ if $traces_{K_1} \subseteq traces_{K_2}$.

Conformance relation

Definition

Let K_1 and K_2 be two comparable components (i.e. having the same external interface). $K_1 \preceq K_2$ if $traces_{K_1} \subseteq traces_{K_2}$.

Theorem

Conformance is a preorder relation.

Conformance relation

Definition

Let K_1 and K_2 be two comparable components (i.e. having the same external interface). $K_1 \preceq K_2$ if $traces_{K_1} \subseteq traces_{K_2}$.

Theorem

Conformance is a preorder relation.

Theorem

Let K_1 and K_2 be two comparable components with $K_1 \preceq K_2$ and E a component compatible with both K_1 and K_2. Then $K_1 \parallel E \preceq K_2 \parallel E$.

Refinement under context relation

Definition

Let K_1 and K_2 be two components such that $I_{K_2} \subseteq I_{K_1} \cup V_{K_1}$, $O_{K_2} \subseteq O_{K_1} \cup V_{K_1}$ and $V_{K_2} \subseteq V_{K_1}$. Let E be an environment for K_1 compatible with both K_1 and K_2. We say that K_1 *refines* K_2 *in the context of* E, denoted $K_1 \sqsubseteq_E K_2$, if

$$K_1 \parallel E \parallel E' \preceq K_2 \parallel E \parallel K' \parallel E'$$

where K' and E' are defined such that both members of the conformance relation are comparable and closed.

Refinement under context relation

Definition

Let K_1 and K_2 be two components such that $I_{K_2} \subseteq I_{K_1} \cup V_{K_1}$, $O_{K_2} \subseteq O_{K_1} \cup V_{K_1}$ and $V_{K_2} \subseteq V_{K_1}$. Let E be an environment for K_1 compatible with both K_1 and K_2. We say that K_1 *refines* K_2 *in the context of* E, denoted $K_1 \sqsubseteq_E K_2$, if

$$K_1 \parallel E \parallel E' \preceq K_2 \parallel E \parallel K' \parallel E'$$

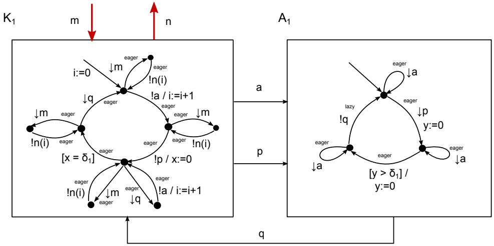
where K' and E' are defined such that both members of the conformance relation are comparable and closed.

Definition

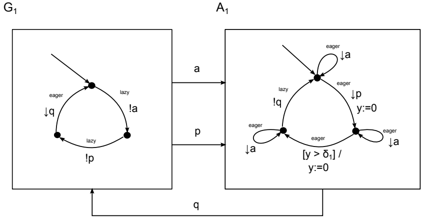
$K \models C = (A, G) \Leftrightarrow K \sqsubseteq_A G$

Example: $K_1 \sqsubseteq_{A_1} G_1$

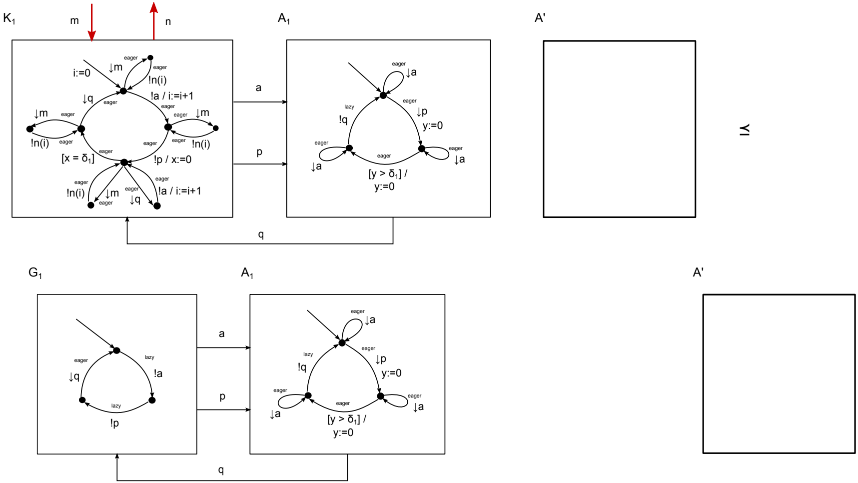
Example: $K_1 \sqsubseteq_{A_1} G_1$



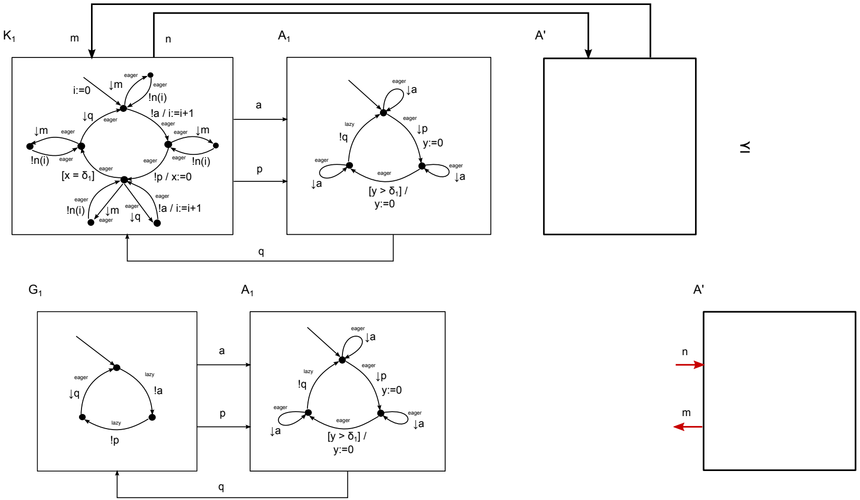
$\preceq$



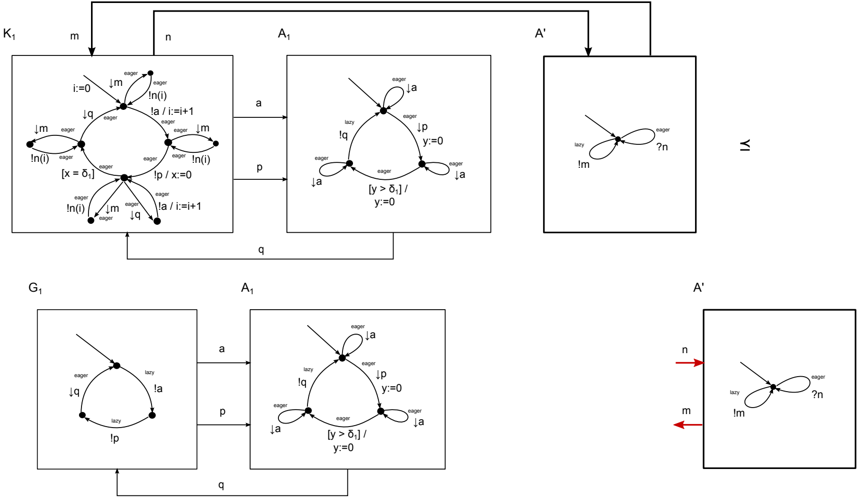
Example: $K_1 \sqsubseteq_{A_1} G_1$



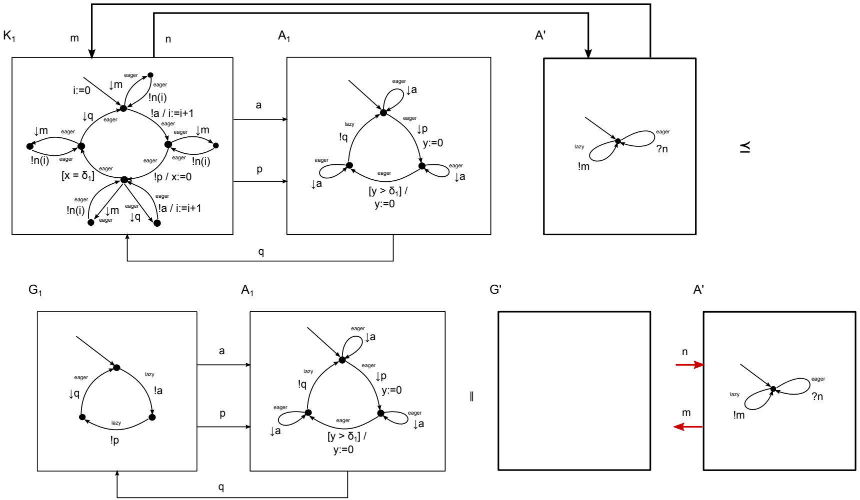
Example: $K_1 \sqsubseteq_{A_1} G_1$



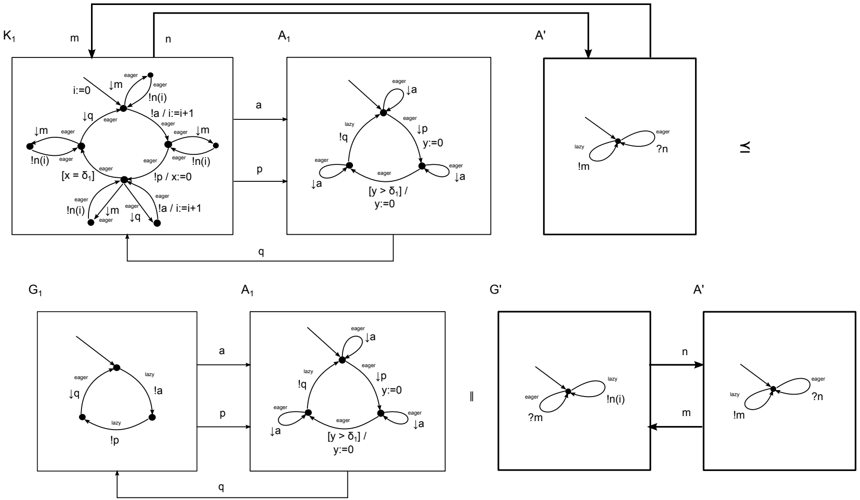
Example: $K_1 \sqsubseteq_{A_1} G_1$



Example: $K_1 \sqsubseteq_{A_1} G_1$



Example: $K_1 \sqsubseteq_{A_1} G_1$



Properties of refinement under context

Theorem

Given a set $\mathcal{K}$ of comparable components and a fixed environment E for that interface, the refinement under context relation $\sqsubseteq_E$ is a preorder over $\mathcal{K}$.

Properties of refinement under context

Theorem

Given a set $\mathcal{K}$ of comparable components and a fixed environment E for that interface, the refinement under context relation $\sqsubseteq_E$ is a preorder over $\mathcal{K}$.

Theorem

Let K_1 and K_2 be two components and E an environment compatible with both K_1 and K_2 such that $E = E_1 \parallel E_2$.

$$K_1 \sqsubseteq_{E_1 \parallel E_2} K_2 \Leftrightarrow K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1$$

Properties of refinement under context

Theorem

Given a set $\mathcal{K}$ of comparable components and a fixed environment E for that interface, the refinement under context relation $\sqsubseteq_E$ is a preorder over $\mathcal{K}$.

Theorem

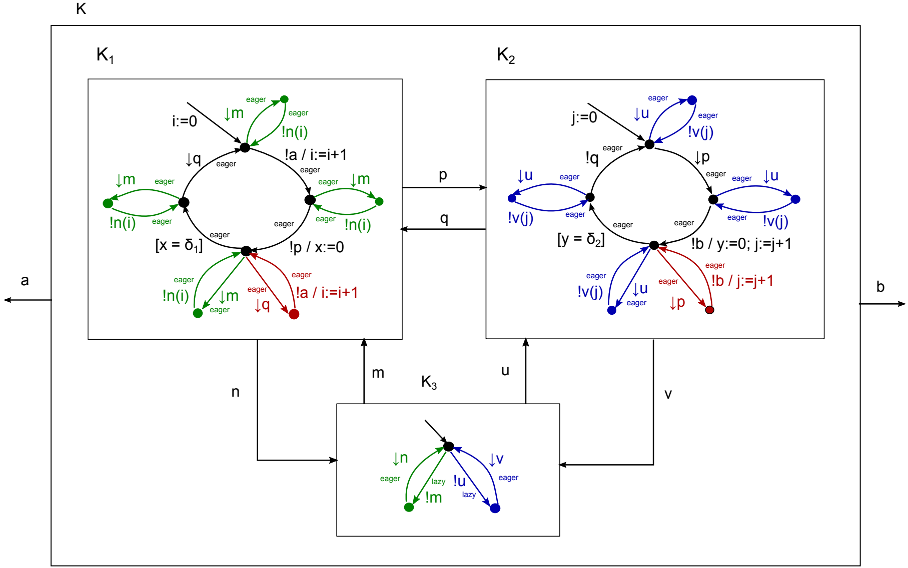
Let K_1 and K_2 be two components and E an environment compatible with both K_1 and K_2 such that $E = E_1 \parallel E_2$.

$$K_1 \sqsubseteq_{E_1 \parallel E_2} K_2 \Leftrightarrow K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1$$

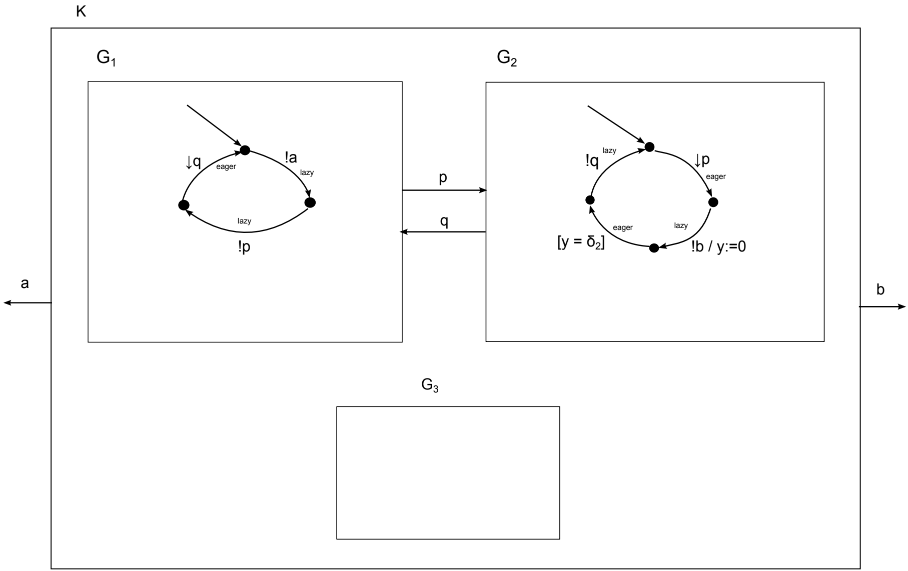
Theorem

Let K be a component, E its environment and $C = (A, G)$ the contract for K such that K and G are compatible with each of E and A. If (1) $traces_G$ is closed under limits, (2) $traces_G$ is closed under time-extension, (3) $K \sqsubseteq_A G$ and (4) $E \sqsubseteq_G A$ then $K \sqsubseteq_E G$.

Abstract system

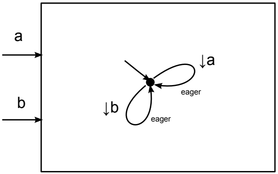


Abstract system

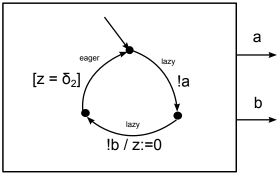


Top contract for the abstract system

A



G



Contract dominance

Definition

$\{C_i\}_{i=1}^n$ *dominates* C iff $\forall \{K_i\}_{i=1}^n$ such that, $\forall i$, $K_i \models C_i$, we have $(K_1 \parallel K_2 \parallel \cdots \parallel K_n) \models C$.

Contract dominance

Definition

$\{C_i\}_{i=1}^{n}$ *dominates* C iff $\forall \{K_i\}_{i=1}^{n}$ such that, $\forall i$, $K_i \models C_i$, we have $(K_1 \parallel K_2 \parallel \cdots \parallel K_n) \models C$.

Theorem

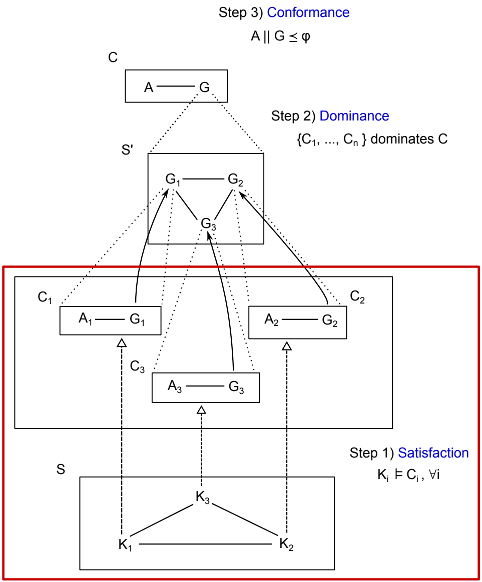
$\{C_i\}_{i=1}^{n}$ dominates C if, $\forall i$, $traces_{A_i}$, $traces_{G_i}$, $traces_A$ and $traces_G$ are closed under limits and under time-extension and

$$\begin{cases} G_1 \parallel \ldots \parallel G_n \sqsubseteq_A G \\ A \parallel G_1 \parallel \ldots \parallel G_{i-1} \parallel G_{i+1} \parallel \ldots \parallel G_n \sqsubseteq_{G_i} A_i, \ \forall i \end{cases}$$

Outline

6. Applying contract-based reasoning on the toy example

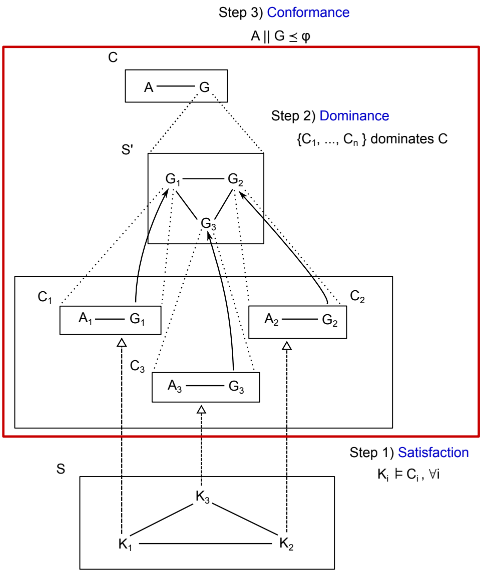
Verifying Step 1



Verifying Step 1

1. $K_1 \models C_1$
2. $K_2 \models C_2$
3. $K_3 \models C_3$

Verifying Step 2



Verifying Step 2

$\{C_1, C_2, C_3\}$ dominates C:

Verifying Step 2

$\{C_1, C_2, C_3\}$ dominates C:

1. $traces_A$, $traces_{A_1}$, $traces_{A_2}$, $traces_{A_3}$ are closed under limits and under time-extension
2. $traces_G$, $traces_{G_1}$, $traces_{G_2}$, $traces_{G_3}$ are closed under limits and under time-extension

Verifying Step 2

$\{C_1, C_2, C_3\}$ dominates C:

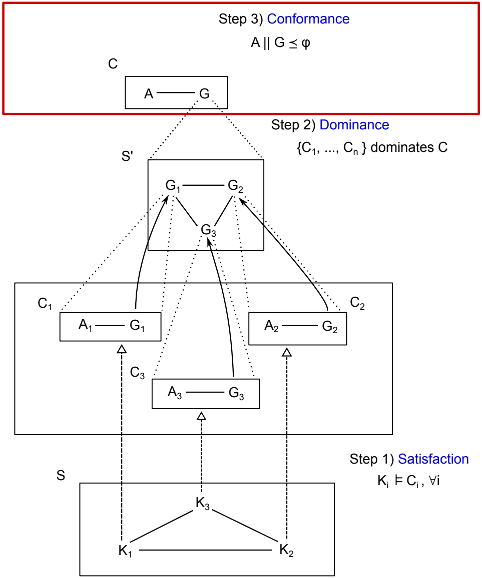
1. $traces_A$, $traces_{A_1}$, $traces_{A_2}$, $traces_{A_3}$ are closed under limits and under time-extension
2. $traces_G$, $traces_{G_1}$, $traces_{G_2}$, $traces_{G_3}$ are closed under limits and under time-extension
3. $G_1 \parallel G_2 \sqsubseteq_A G$

Verifying Step 2

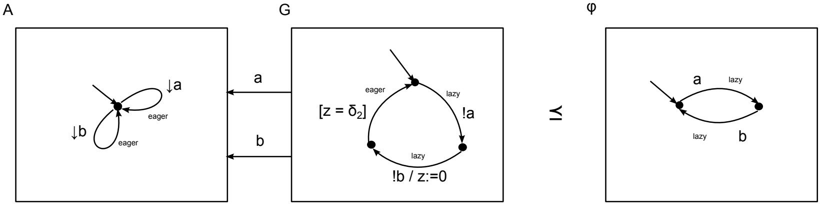
$\{C_1, C_2, C_3\}$ dominates C:

1. $traces_A$, $traces_{A_1}$, $traces_{A_2}$, $traces_{A_3}$ are closed under limits and under time-extension
2. $traces_G$, $traces_{G_1}$, $traces_{G_2}$, $traces_{G_3}$ are closed under limits and under time-extension
3. $G_1 \parallel G_2 \sqsubseteq_A G$
4. $A \parallel G_2 \sqsubseteq_{G_1} A_1$
5. $A \parallel G_1 \sqsubseteq_{G_2} A_2$
6. $A \parallel G_1 \parallel G_2 \sqsubseteq_{G_3} A_3$

Verifying Step 3



Verifying Step 3



Outline

7. Conclusions

Results

- TIOA component framework

Results

- TIOA component framework
- Formal contract with interface refinement

Results

- TIOA component framework
- Formal contract with interface refinement
- Refinement relations based on *trace inclusion*

Results

- TIOA component framework
- Formal contract with interface refinement
- Refinement relations based on *trace inclusion*
- Applied on a toy example

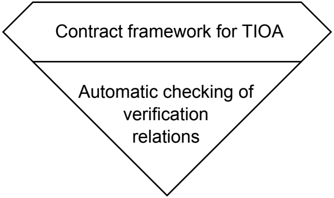
Related work

- (Timed) Interface Automata
- Interface Input/Output Automata
- Timed Input/Output Automata in ECDAR

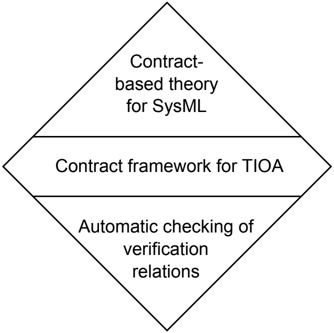
Future work

Contract framework for TIOA

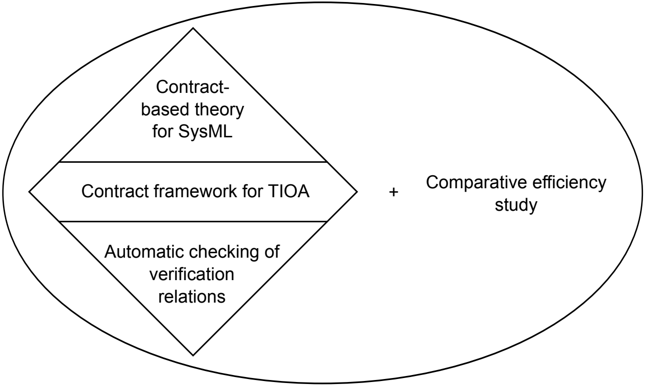
Future work


Contract framework for TIOA
Automatic checking of verification relations

Future work


Contract-
based theory
for SysML
Contract framework for TIOA
Automatic checking of
verification
relations

Future work


Contract-
based theory
for SysML
Contract framework for TIOA
Automatic checking of
verification
relations
+
Comparative efficiency
study

Perspectives

1. How to build contracts?

Perspectives

1. How to build contracts?
   - Solve for G

Perspectives

1. How to build contracts?
    - Solve for G
    - Automatically generate A

Perspectives

1. How to build contracts?
   - Solve for G
   - Automatically generate A
2. Automation and integration within a development process

References:

1. I. Dragomir, I. Ober, D. Lesens. *A Case Study in Formal System Engineering with SysML*. ICECCS 2012, Paris, 18/07/2012 - 20/07/2012, IEEE, p. 189-198, july 2012
2. I. Dragomir, I.Ober, C. Percebois. *Safety Contracts for Timed Reactive Systems*. Technical report, IRIT/RT-2013-11-FR, february 2013. Available at http://www.irit.fr/~Iulian.Ober/docs/TR-Contracts.pdf

Thank you!

Any questions?