How appropriate is optimism for Interface Automata

F. Bujtor W. Vogler

Theoretische Informatik
Universität Augsburg

D-Con 2013

Interface-based design

- Interfaces abstract from details like data
- Focus on behavioural type and communication
- Analysis of errors in communication
- Refinement-/Implementation-relation
  - Refinement of components → refinement of system
  - → precongruence

Interface-based design

- Interfaces abstract from details like data
- Focus on behavioural type and communication
- Analysis of errors in communication
- Refinement-/Implementation-relation
  - Refinement of components → refinement of system
  - → precongruence

Interface-based design

- Interfaces abstract from details like data
- Focus on behavioural type and communication
- Analysis of errors in communication
- Refinement-/Implementation-relation
  - Refinement of components → refinement of system
  - → precongruence

Interface-based design

- Interfaces abstract from details like data
- Focus on behavioural type and communication
- Analysis of errors in communication
- Refinement-/Implementation-relation
  - Refinement of components $\rightarrow$ refinement of system
  - $\rightarrow$ precongruence

Interface-based design

- Interfaces abstract from details like data
- Focus on behavioural type and communication
- Analysis of errors in communication
- Refinement-/Implementation-relation
  - Refinement of components $\rightarrow$ refinement of system
  - $\rightarrow$ precongruence

Interface-based design

- Interfaces abstract from details like data
- Focus on behavioural type and communication
- Analysis of errors in communication
- Refinement-/Implementation-relation
  - Refinement of components $\rightarrow$ refinement of system
  - $\rightarrow$ precongruence

Setting by de Alfaro/Henzinger

- Automata with disjoint inputs, outputs and internal actions
- Composition of 'compatible' Automata ($I_1 \cap I_2 = \emptyset = O_1 \cap O_2$)
- Shared actions: input of the one and output of the other automaton
- Synchronization on all shared actions (with hiding)
- Synchronizing output cannot be matched $\rightarrow$ error state
- Removal of illegal states as part of composition
- Alternating simulation as refinement

Setting by de Alfaro/Henzinger

- Automata with disjoint inputs, outputs and internal actions
- Composition of 'compatible' Automata
  $(I_1 \cap I_2 = \emptyset = O_1 \cap O_2)$
- Shared actions: input of the one and output of the other automaton
- Synchronization on all shared actions (with hiding)
- Synchronizing output cannot be matched $\rightarrow$ error state
- Removal of illegal states as part of composition
- Alternating simulation as refinement

Setting by de Alfaro/Henzinger

- Automata with disjoint inputs, outputs and internal actions
- Composition of 'compatible' Automata
  $(I_1 \cap I_2 = \emptyset = O_1 \cap O_2)$
- Shared actions: input of the one and output of the other automaton
- Synchronization on all shared actions (with hiding)
- Synchronizing output cannot be matched $\rightarrow$ error state
- Removal of illegal states as part of composition
- Alternating simulation as refinement

Setting by de Alfaro/Henzinger

- Automata with disjoint inputs, outputs and internal actions
- Composition of 'compatible' Automata
  $(I_1 \cap I_2 = \emptyset = O_1 \cap O_2)$
- Shared actions: input of the one and output of the other automaton
- Synchronization on all shared actions (with hiding)
- Synchronizing output cannot be matched $\rightarrow$ error state
- Removal of illegal states as part of composition
- Alternating simulation as refinement

Setting by de Alfaro/Henzinger

- Automata with disjoint inputs, outputs and internal actions
- Composition of 'compatible' Automata
  $(I_1 \cap I_2 = \emptyset = O_1 \cap O_2)$
- Shared actions: input of the one and output of the other automaton
- Synchronization on all shared actions (with hiding)
- Synchronizing output cannot be matched $\rightarrow$ error state
- Removal of illegal states as part of composition
- Alternating simulation as refinement

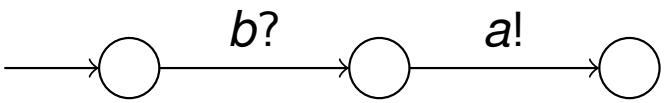
Setting by de Alfaro/Henzinger

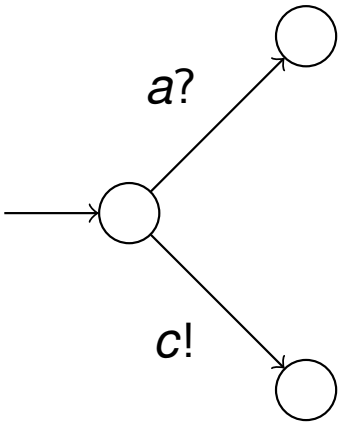
- Automata with disjoint inputs, outputs and internal actions
- Composition of 'compatible' Automata
  $(I_1 \cap I_2 = \emptyset = O_1 \cap O_2)$
- Shared actions: input of the one and output of the other automaton
- Synchronization on all shared actions (with hiding)
- Synchronizing output cannot be matched $\rightarrow$ error state
- Removal of illegal states as part of composition
- Alternating simulation as refinement

Setting by de Alfaro/Henzinger

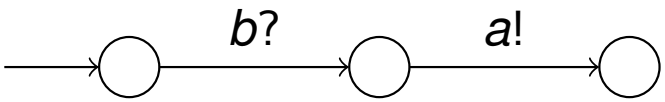
- Automata with disjoint inputs, outputs and internal actions
- Composition of 'compatible' Automata ($I_1 \cap I_2 = \emptyset = O_1 \cap O_2$)
- Shared actions: input of the one and output of the other automaton
- Synchronization on all shared actions (with hiding)
- Synchronizing output cannot be matched $\rightarrow$ error state
- Removal of illegal states as part of composition
- Alternating simulation as refinement

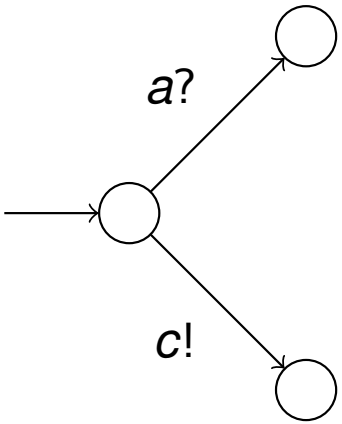
Composition

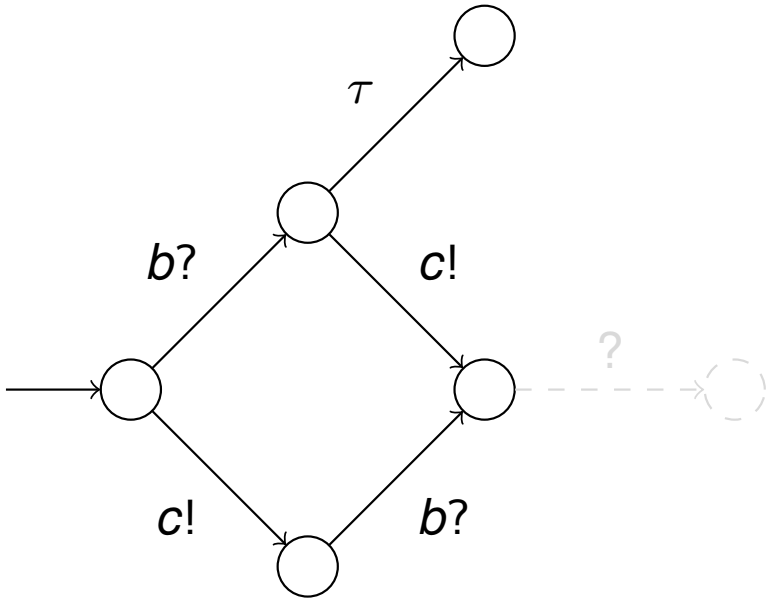




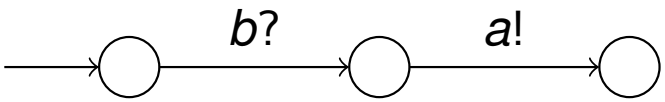
Composition

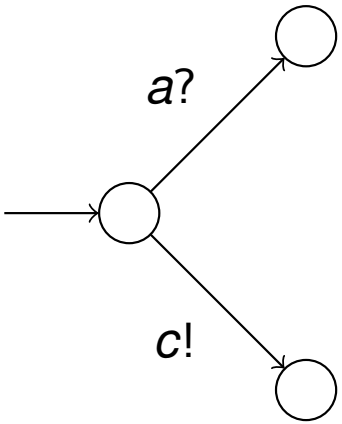


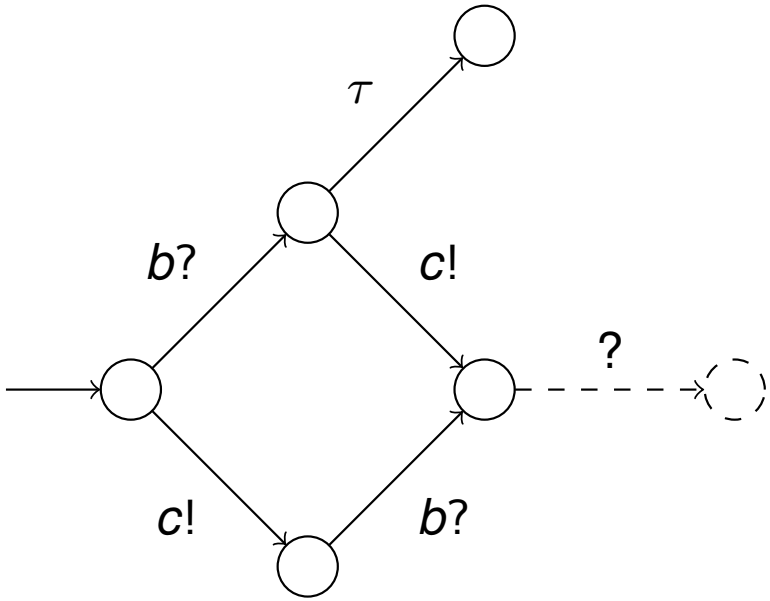




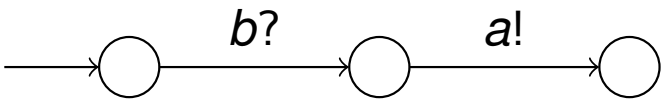
Composition

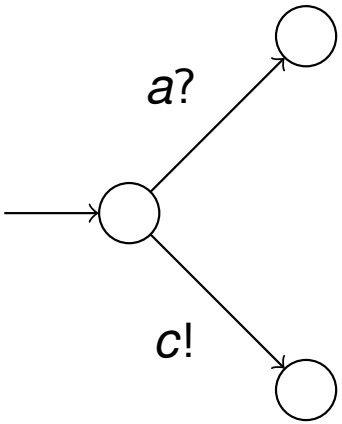


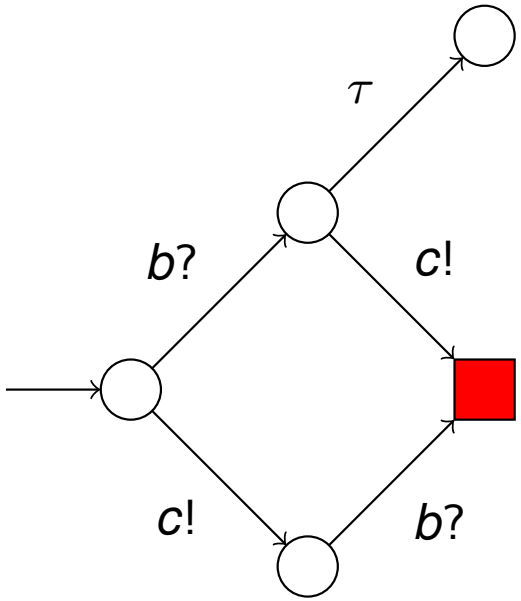




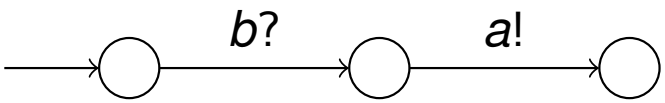
Composition – Errors

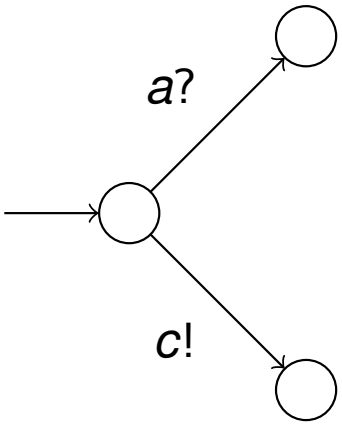


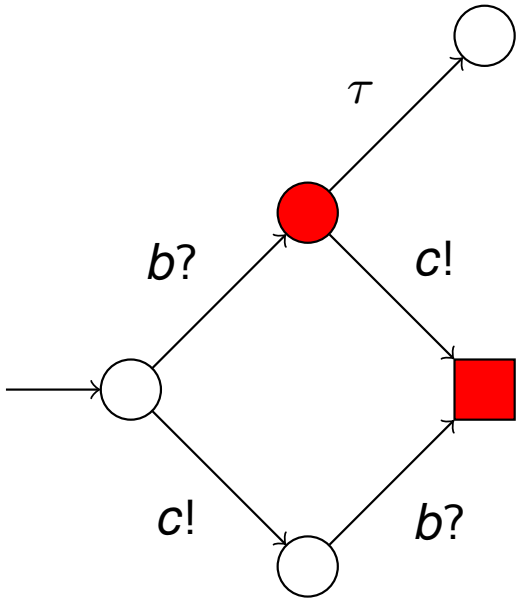




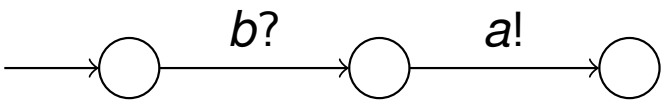
Composition – Errors

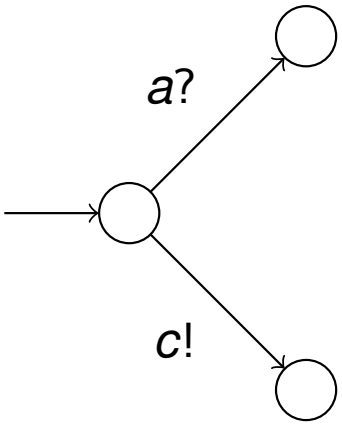


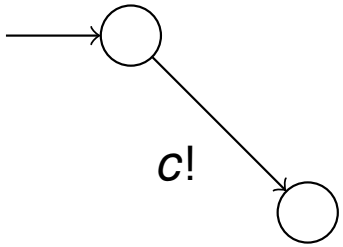




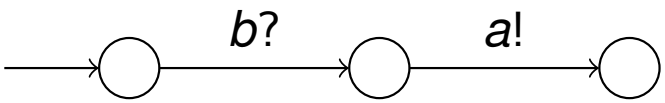
Composition – Errors / Pruning

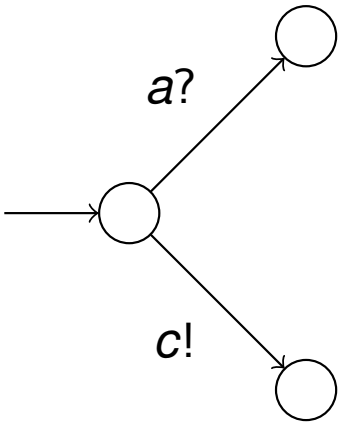


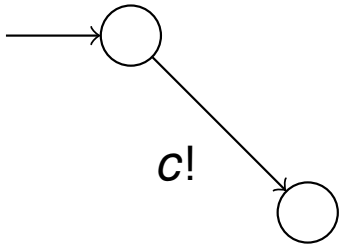




Composition – Errors / Pruning

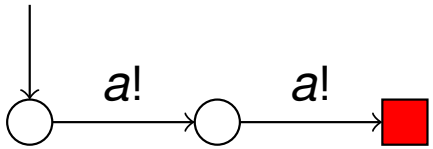




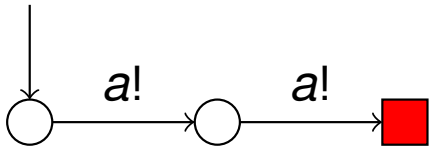


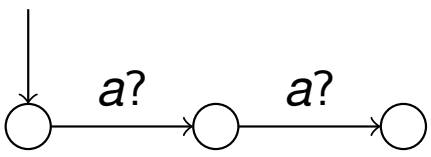
works for input determinate systems

Unavoidable Error Lemma

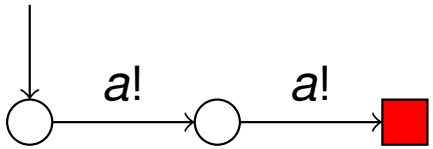


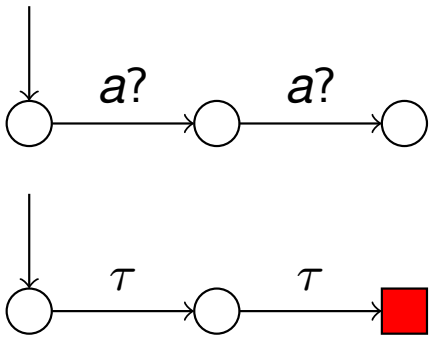
Unavoidable Error Lemma



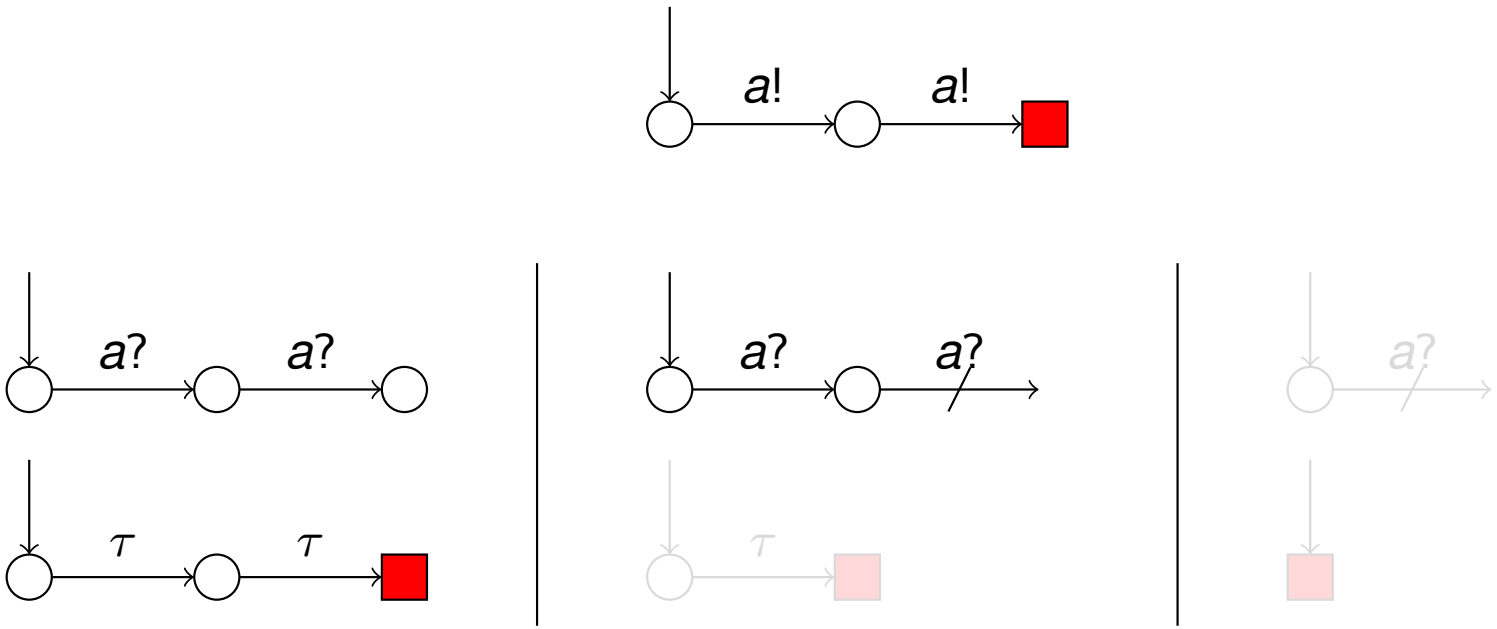


Unavoidable Error Lemma

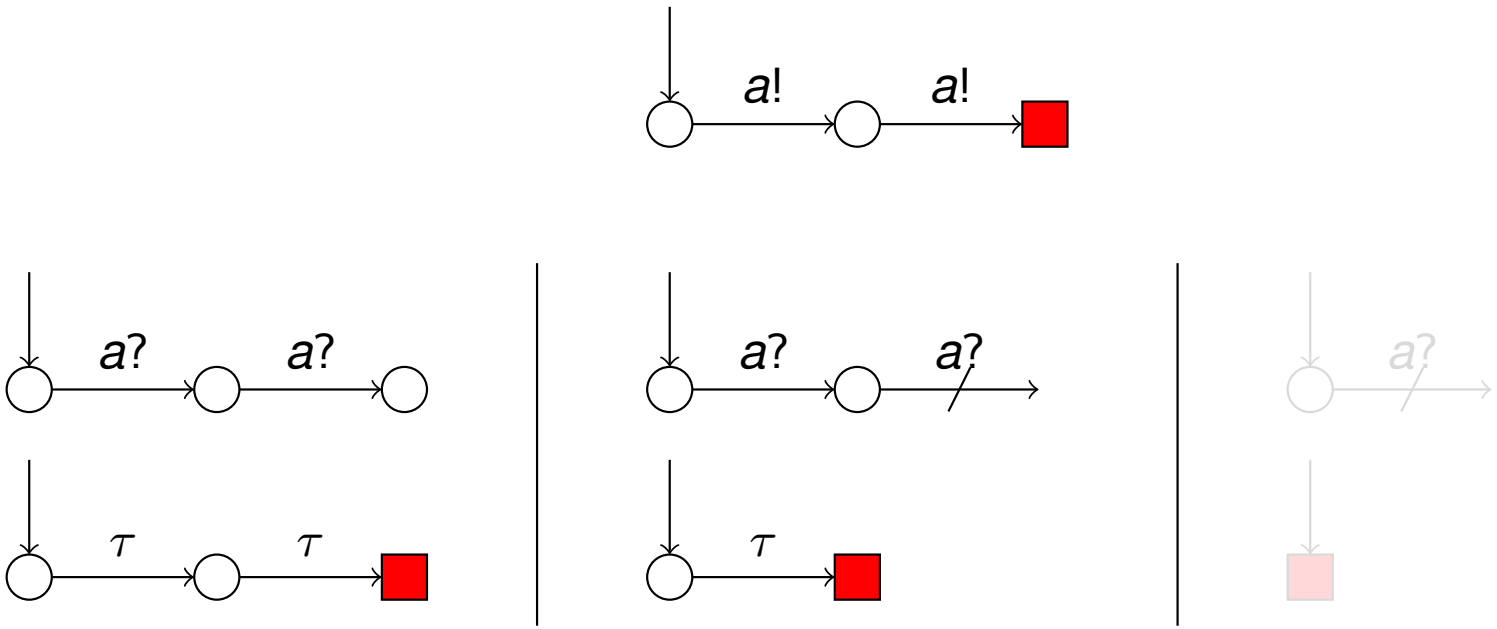




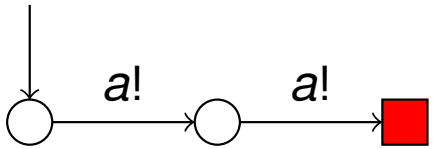
Unavoidable Error Lemma

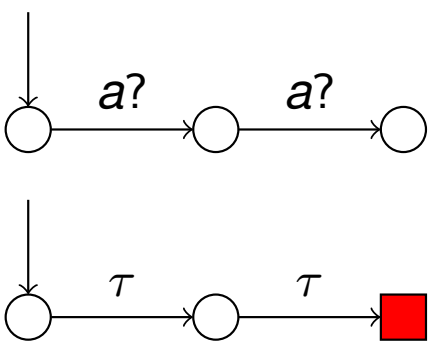


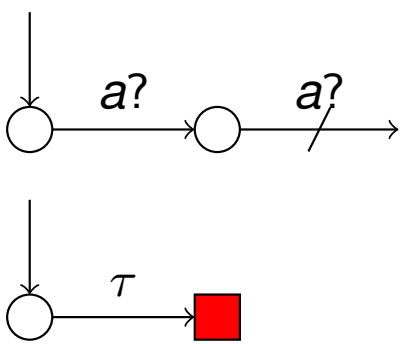
Unavoidable Error Lemma

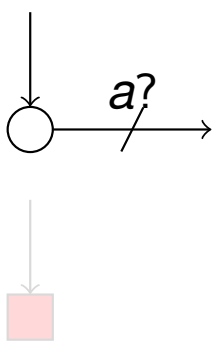


Unavoidable Error Lemma

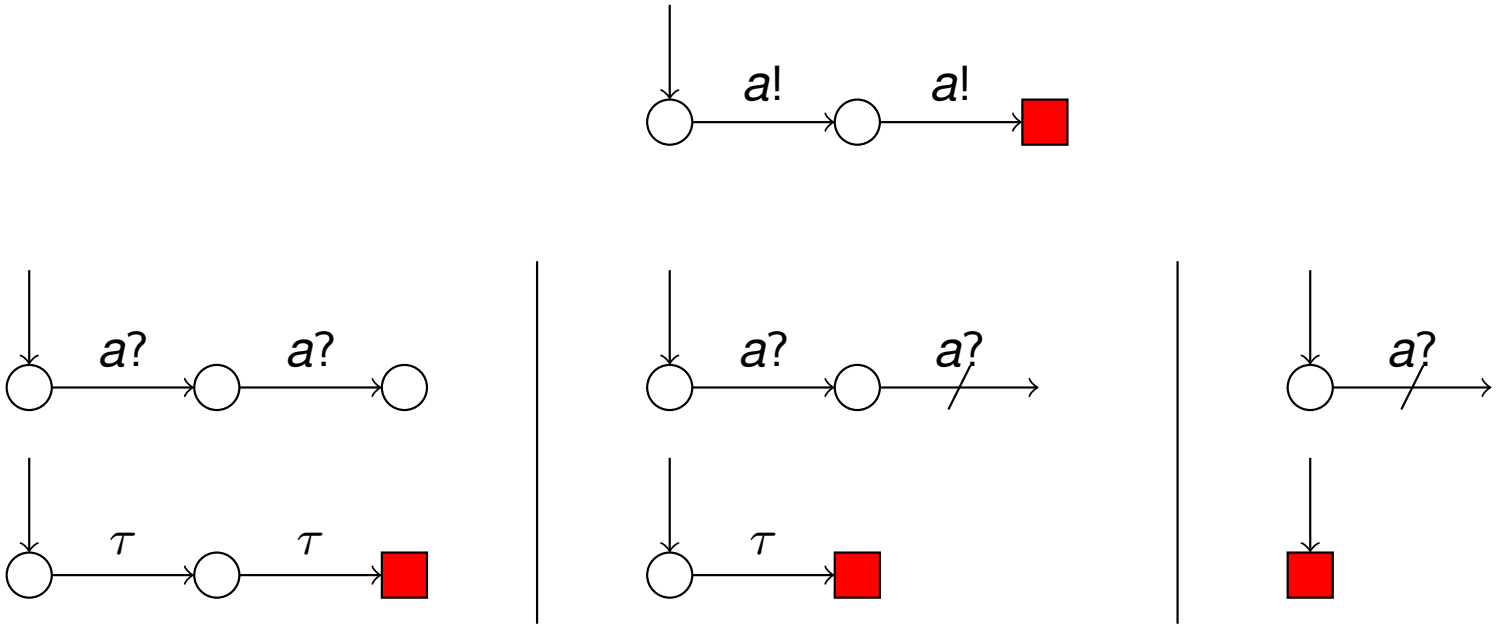








Unavoidable Error Lemma



Setting by de Alfaro/Henzinger

- Automata with disjoint inputs, outputs and internal actions
- Composition of 'compatible' Automata ($I_1 \cap I_2 = \emptyset = O_1 \cap O_2$)
- Shared actions: input of the one and output of the other automaton
- Synchronization on all shared actions (with hiding)
- Synchronizing output cannot be matched $\rightarrow$ error state
- Removal of illegal states as part of composition
- Alternating simulation as refinement
- Explicit treatment of error states
- Find 'most appropriate' refinement (coarsest precongruence)

Adjusted Setting

- Automata with disjoint inputs, outputs and internal actions
- Composition of 'compatible' Automata ($I_1 \cap I_2 = \emptyset = O_1 \cap O_2$)
- Shared actions: input of the one and output of the other automaton
- Synchronization on all shared actions (with hiding)
- Synchronizing output cannot be matched $\rightarrow$ error state
- ~~Removal of illegal states as part of composition~~
- ~~Alternating simulation as refinement~~
- Explicit treatment of error states
- Find 'most appropriate' refinement (coarsest precongruence)

Adjusted Setting

- Automata with disjoint inputs, outputs and internal actions
- Composition of 'compatible' Automata ($I_1 \cap I_2 = \emptyset = O_1 \cap O_2$)
- Shared actions: input of the one and output of the other automaton
- Synchronization on all shared actions (with hiding)
- Synchronizing output cannot be matched $\rightarrow$ error state
- ~~Removal of illegal states as part of composition~~
- ~~Alternating simulation as refinement~~
- Explicit treatment of error states
- Find 'most appropriate' refinement (coarsest precongruence)

Adjusted Setting

- Automata with disjoint inputs, outputs and internal actions
- Composition of 'compatible' Automata ($I_1 \cap I_2 = \emptyset = O_1 \cap O_2$)
- Shared actions: input of the one and output of the other automaton
- Synchronization on all shared actions (with hiding)
- Synchronizing output cannot be matched $\rightarrow$ error state
- ~~Removal of illegal states as part of composition~~
- ~~Alternating simulation as refinement~~
- Explicit treatment of error states
- Find 'most appropriate' refinement (coarsest precongruence)

Precongruence

A preorder relation $\sqsubseteq$ is a precongruence w.r.t. $\mid$ if

$$\mathit{Impl} \sqsubseteq \mathit{Spec} \quad \Rightarrow \quad \mathit{Impl} \mid \mathit{Env} \sqsubseteq \mathit{Spec} \mid \mathit{Env}$$

Refinement / Implementation

Basic requirement:
If the specification is error free, then the implementation has to be error free as well

Coarsest precongruence: $\sqsubseteq^{C}$

- Fulfilling the basic requirement
- Precongruence w.r.t. $\|$
- Largest possible such relation

Refinement / Implementation

Basic requirement:
If the specification is error free, then the implementation has to be error free as well

Coarsest precongruence: $\sqsubseteq^{C}$

- Fulfilling the basic requirement
- Precongruence w.r.t. $|$
- Largest possible such relation

Refinement / Implementation

Basic requirement:
If the specification is error free, then the implementation has to be error free as well

Coarsest precongruence: $\sqsubseteq^C$

- Fulfilling the basic requirement
- Precongruence w.r.t. $|$
- Largest possible such relation

Refinement / Implementation

Basic requirement:
If the specification is error free, then the implementation has to be error free as well

Coarsest precongruence: $\sqsubseteq^{C}$

- Fulfilling the basic requirement
- Precongruence w.r.t. $|$
- Largest possible such relation

Refinement / Implementation

Basic requirement:
If the specification is error free, then the implementation has to be error free as well

Coarsest precongruence: $\sqsubseteq^{C}$

- Fulfilling the basic requirement
- Precongruence w.r.t. $|$
- Largest possible such relation

Refinement / Implementation

Basic requirement:
If the specification is error free, then the implementation has to be error free as well

Notions of having an error:

1. An error state is reachable by locally controlled actions (internal and output)
2. An error state is reachable by internal actions
3. An error state is reachable at all

Refinement / Implementation

Basic requirement:
If the specification is error free, then the implementation has to be error free as well

Notions of having an error:

1. An error state is reachable by locally controlled actions (internal and output)
2. An error state is reachable by internal actions
3. An error state is reachable at all

Refinement / Implementation

Basic requirement:
If the specification is error free, then the implementation has to be error free as well

Notions of having an error:

1. An error state is reachable by locally controlled actions (internal and output)
2. An error state is reachable by internal actions
3. An error state is reachable at all

Refinement / Implementation

Basic requirement:
If the specification is error free, then the implementation has to be error free as well

Notions of having an error:

1. An error state is reachable by locally controlled actions (internal and output)
2. An error state is reachable by internal actions
3. An error state is reachable at all

Refinement / Implementation

Basic requirement:
If the specification is error free, then the implementation has to be error free as well

Notions of having an error:

1. An error state is reachable by locally controlled actions (internal and output)
2. An error state is reachable by internal actions
3. An error state is reachable at all

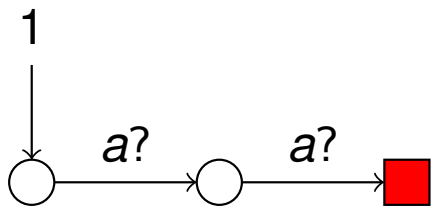
Functions

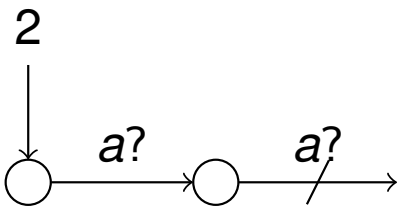
- $prune: \Sigma^* \to \Sigma^*, w \mapsto u$, where $w = uv$, $u = \varepsilon \vee u \in \Sigma^* \cdot I$ and $v \in O^*$
- $cont: \Sigma^* \to \mathfrak{P}(\Sigma^*), w \mapsto \{wu \mid u \in \Sigma^*\}$

Functions

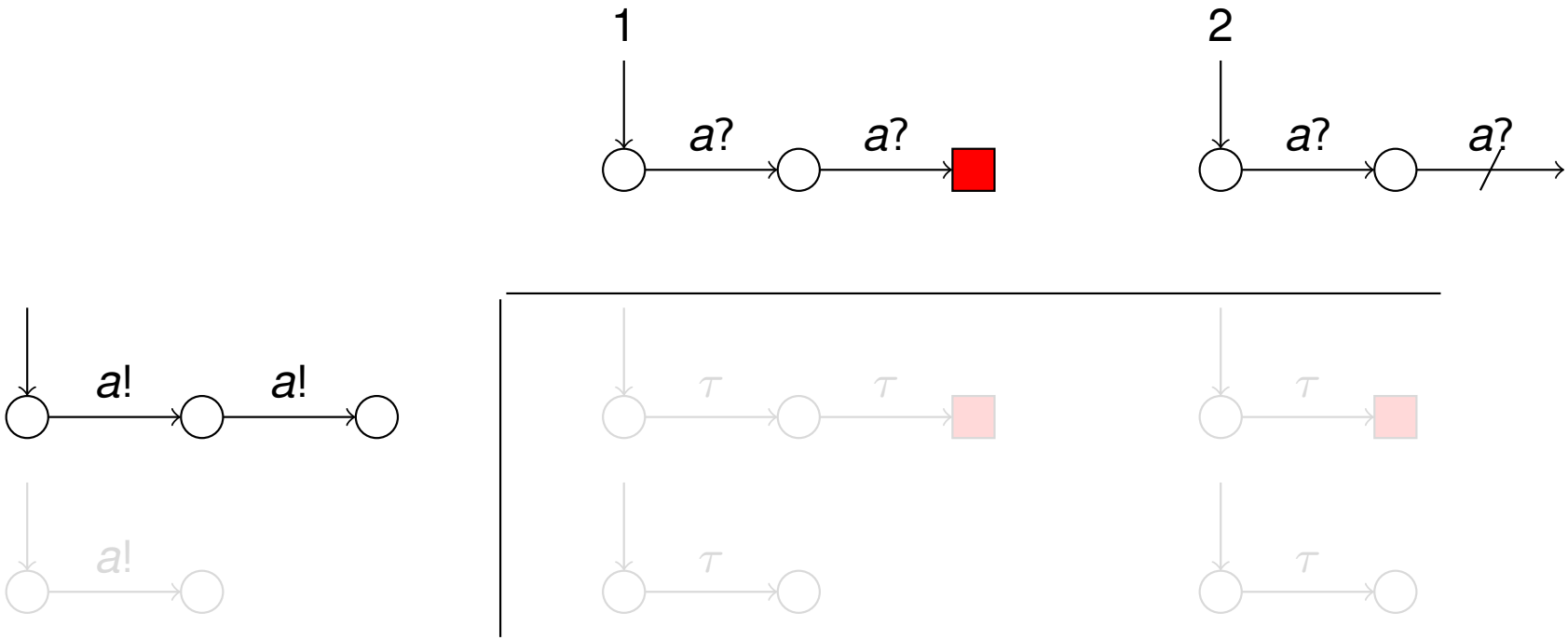
- $prune : \Sigma^* \to \Sigma^*, w \mapsto u$, where $w = uv$, $u = \varepsilon \vee u \in \Sigma^* \cdot I$ and $v \in O^*$
- $cont : \Sigma^* \to \mathfrak{P}(\Sigma^*), w \mapsto \{wu \mid u \in \Sigma^*\}$

Missing Inputs

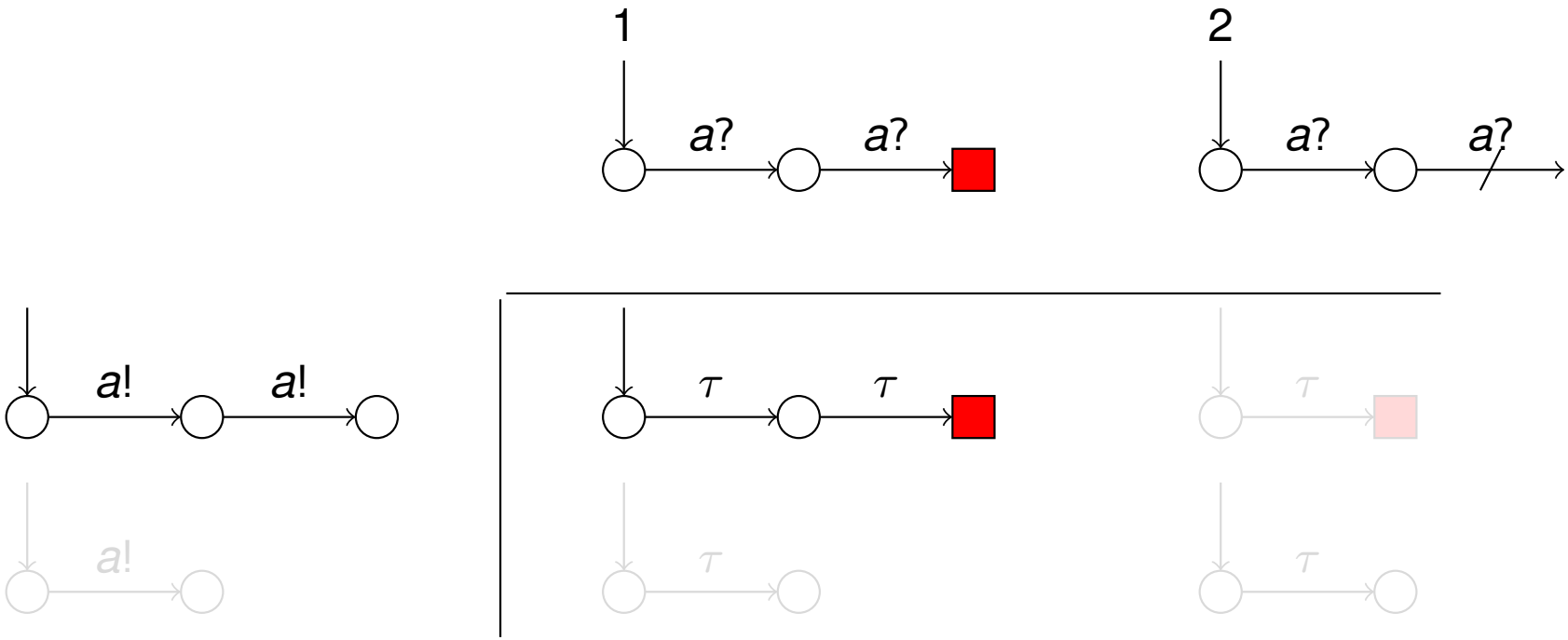




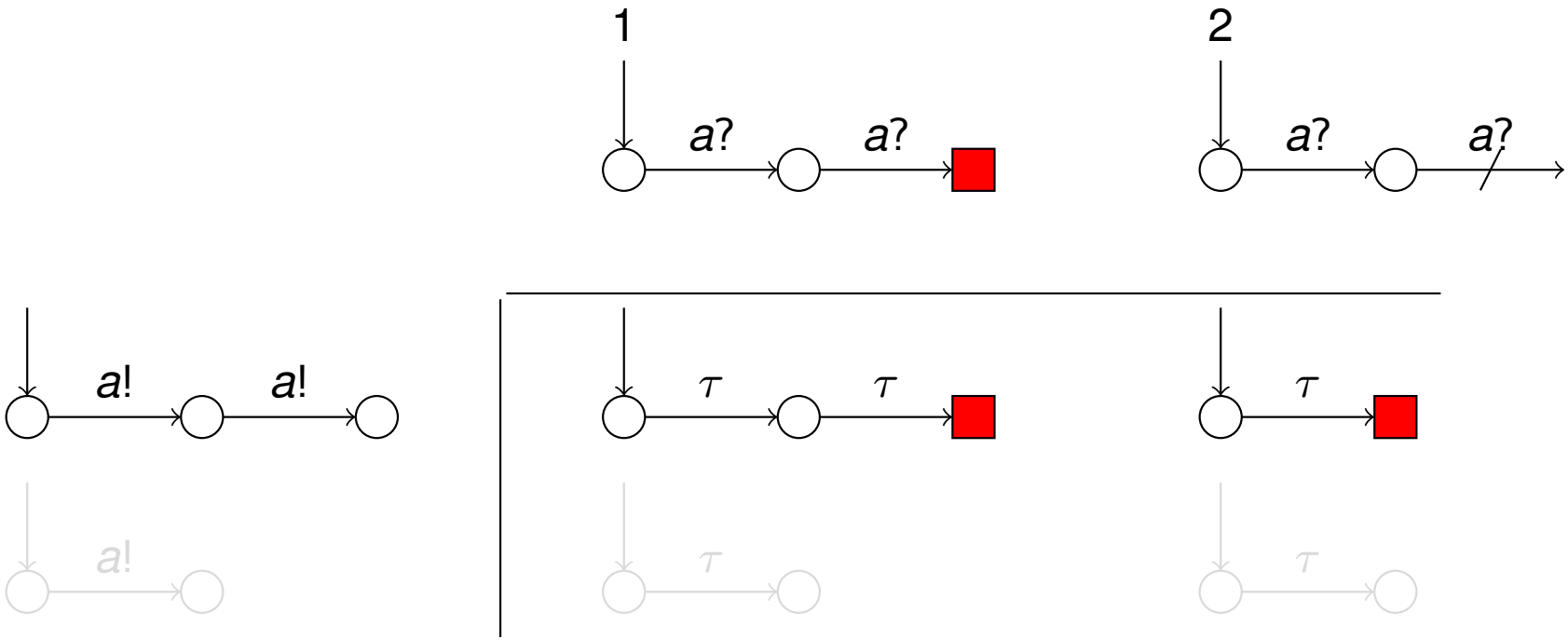
Missing Inputs



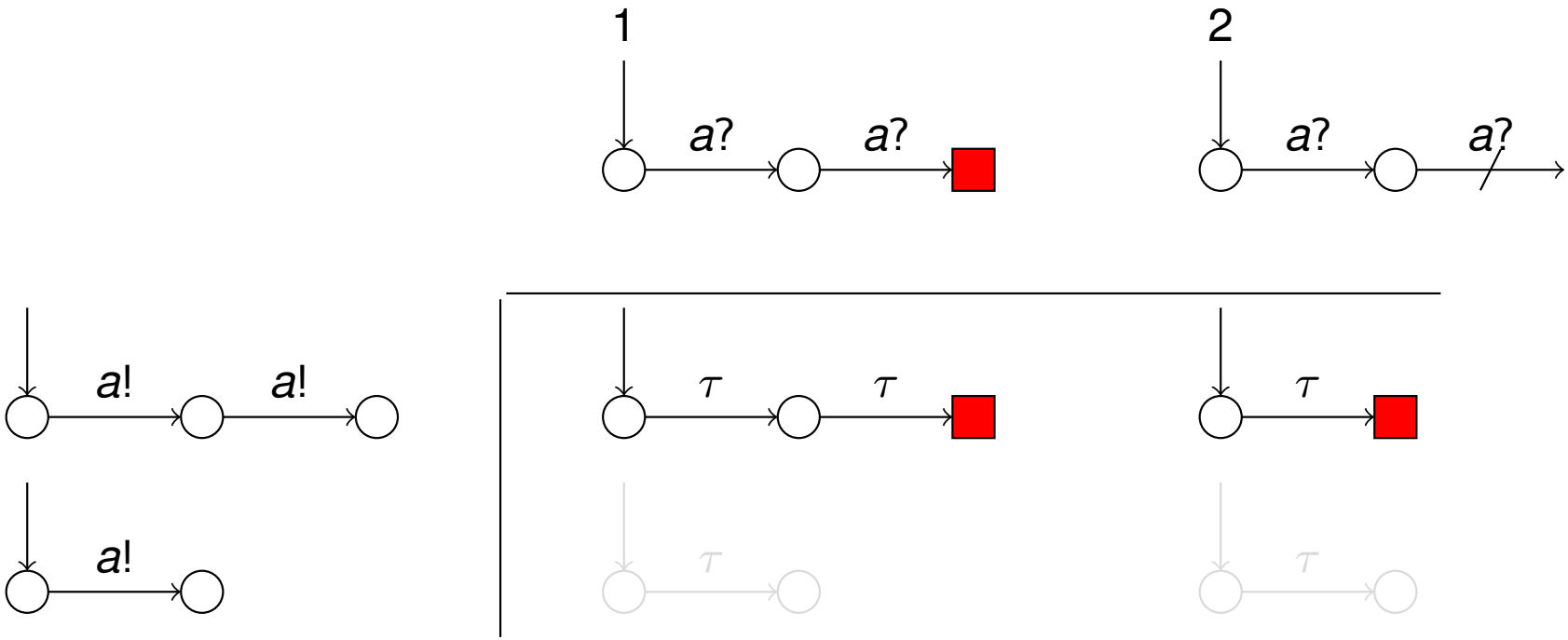
Missing Inputs



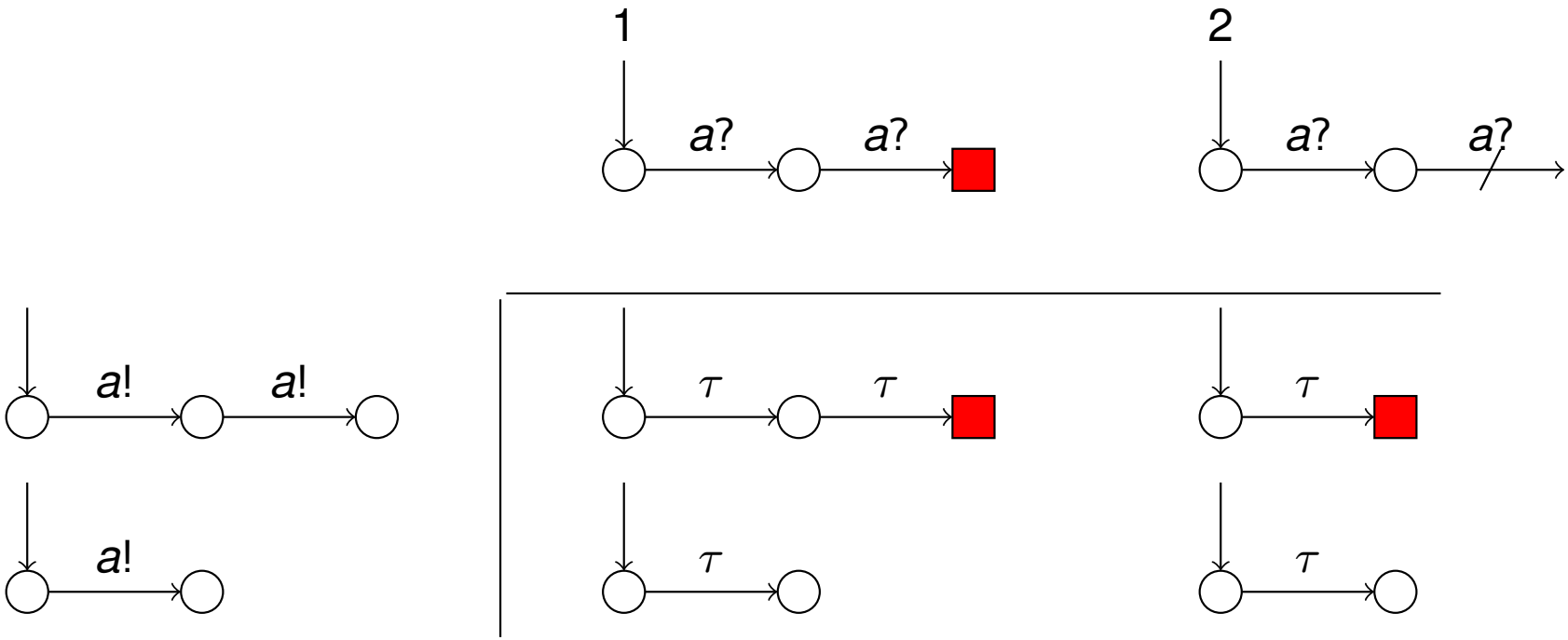
Missing Inputs



Missing Inputs



Missing Inputs



Traces and Semantics

- Strict error traces: $StS(S) = \{w \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow} q \in E\}$
- Pruned error traces: $PrT(S) = \{prune(w) \mid w \in StT(S)\}$
- Missing-input traces:
$$MIT(S) = \{wa \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow} q \wedge a \in I \wedge q \stackrel{a}{\not\rightarrow}\}$$
- Basic language: L(S) = $\{w \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow}\}$

Local Error Semantics:

- Error traces: $ET(S) = cont(PrT(S)) \cup cont(MIT(S))$
- Flooded language: $EL(S) = L(S) \cup ET(S)$

Traces and Semantics

- Strict error traces: $StS(S) = \{w \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow} q \in E\}$
- Pruned error traces: $PrT(S) = \{prune(w) \mid w \in StT(S)\}$
- Missing-input traces:
$$MIT(S) = \{wa \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow} q \wedge a \in I \wedge q \stackrel{a}{\not\rightarrow}\}$$
- Basic language: L(S) = $\{w \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow}\}$

Local Error Semantics:

- Error traces: $ET(S) = cont(PrT(S)) \cup cont(MIT(S))$
- Flooded language: $EL(S) = L(S) \cup ET(S)$

Traces and Semantics

- Strict error traces: $StS(S) = \{w \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow} q \in E\}$
- Pruned error traces: $PrT(S) = \{prune(w) \mid w \in StT(S)\}$
- Missing-input traces:
$$MIT(S) = \{wa \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow} q \wedge a \in I \wedge q \stackrel{a}{\not\Rightarrow}\}$$
- Basic language: $L(S) = \{w \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow}\}$

Local Error Semantics:

- Error traces: $ET(S) = cont(PrT(S)) \cup cont(MIT(S))$
- Flooded language: $EL(S) = L(S) \cup ET(S)$

Traces and Semantics

- Strict error traces: $StS(S) = \{w \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow} q \in E\}$
- Pruned error traces: $PrT(S) = \{prune(w) \mid w \in StT(S)\}$
- Missing-input traces:
$$MIT(S) = \{wa \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow} q \wedge a \in I \wedge q \stackrel{a}{\not\rightarrow}\}$$
- Basic language: L(S) = $\{w \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow}\}$

Local Error Semantics:

- Error traces: $ET(S) = cont(PrT(S)) \cup cont(MIT(S))$
- Flooded language: $EL(S) = L(S) \cup ET(S)$

Traces and Semantics

- Strict error traces: $StS(S) = \{w \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow} q \in E\}$
- Pruned error traces: $PrT(S) = \{prune(w) \mid w \in StT(S)\}$
- Missing-input traces:
$$MIT(S) = \{wa \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow} q \wedge a \in I \wedge q \stackrel{a}{\not\Rightarrow}\}$$
- Basic language: L(S) = $\{w \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow}\}$

Local Error Semantics:

- Error traces: $ET(S) = cont(PrT(S)) \cup cont(MIT(S))$
- Flooded language: $EL(S) = L(S) \cup ET(S)$

Traces and Semantics

- Strict error traces: $StS(S) = \{w \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow} q \in E\}$
- Pruned error traces: $PrT(S) = \{prune(w) \mid w \in StT(S)\}$
- Missing-input traces:
$$MIT(S) = \{wa \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow} q \wedge a \in I \wedge q \stackrel{a}{\not\Rightarrow}\}$$
- Basic language: L(S) = $\{w \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow}\}$

Local Error Semantics:

- Error traces: $ET(S) = cont(PrT(S)) \cup cont(MIT(S))$
- Flooded language: $EL(S) = L(S) \cup ET(S)$

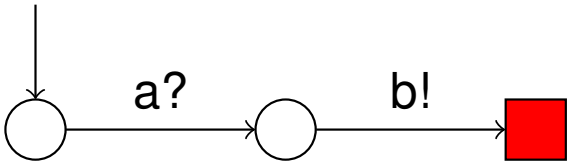
Traces and Semantics

- Strict error traces: $StS(S) = \{w \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow} q \in E\}$
- Pruned error traces: $PrT(S) = \{prune(w) \mid w \in StT(S)\}$
- Missing-input traces:
$$MIT(S) = \{wa \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow} q \wedge a \in I \wedge q \stackrel{a}{\not\Rightarrow}\}$$
- Basic language: L(S) = $\{w \in \Sigma^* \mid q_0 \stackrel{w}{\Rightarrow}\}$

Local Error Semantics:

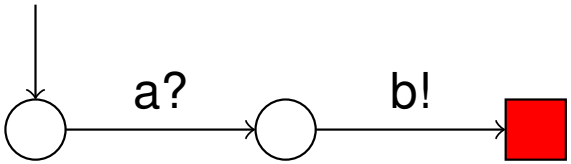
- Error traces: $ET(S) = cont(PrT(S)) \cup cont(MIT(S))$
- Flooded language: $EL(S) = L(S) \cup ET(S)$

Traces – Example



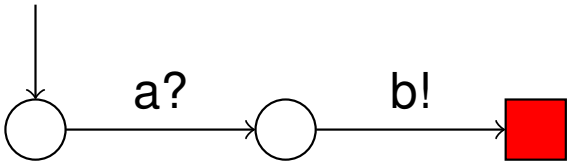
- $StT(S) = \{ab\}$
- $PrT(S) = \{a\}$
- $MIT(S) = \{aa, aba\}$
- $ET(S) = cont(\{a, aa, aba\}) = \{a\}\Sigma^*$
- $EL(S) = \{\tau\} \cup \{a\}\Sigma^*$

Traces – Example



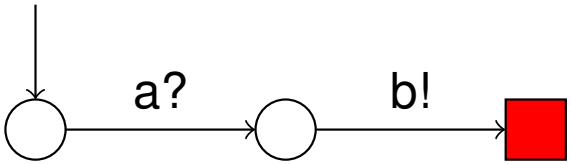
- $StT(S) = \{ab\}$
- $PrT(S) = \{a\}$
- $MIT(S) = \{aa, aba\}$
- $ET(S) = cont(\{a, aa, aba\}) = \{a\}\Sigma^*$
- $EL(S) = \{\tau\} \cup \{a\}\Sigma^*$

Traces – Example



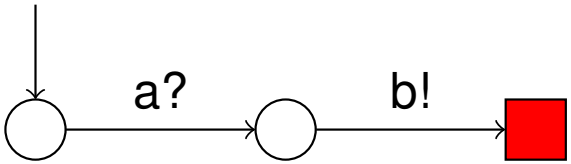
- $StT(S) = \{ab\}$
- $PrT(S) = \{a\}$
- $MIT(S) = \{aa, aba\}$
- $ET(S) = cont(\{a, aa, aba\}) = \{a\}\Sigma^*$
- $EL(S) = \{\tau\} \cup \{a\}\Sigma^*$

Traces – Example



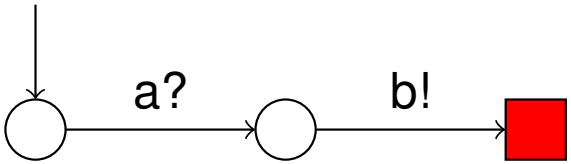
- $StT(S) = \{ab\}$
- $PrT(S) = \{a\}$
- $MIT(S) = \{aa, aba\}$
- $ET(S) = cont(\{a, aa, aba\}) = \{a\}\Sigma^*$
- $EL(S) = \{\tau\} \cup \{a\}\Sigma^*$

Traces – Example



- $StT(S) = \{ab\}$
- $PrT(S) = \{a\}$
- $MIT(S) = \{aa, aba\}$
- $ET(S) = cont(\{a, aa, aba\}) = \{a\}\Sigma^*$
- $EL(S) = \{\tau\} \cup \{a\}\Sigma^*$

Traces – Example



- $StT(S) = \{ab\}$
- $PrT(S) = \{a\}$
- $MIT(S) = \{aa, aba\}$
- $ET(S) = cont(\{a, aa, aba\}) = \{a\}\Sigma^*$
- $EL(S) = \{\tau\} \cup \{a\}\Sigma^*$

Local Error Semantics

- Error traces: $ET(S) = cont(PrT(S)) \cup cont(MIT(S))$
- Flooded language: $EL(S) = L(S) \cup ET(S)$

Refinement preorder:

$$Impl \sqsubseteq_{loc} Spec :\Leftrightarrow \begin{cases} ET(Impl) \subseteq ET(Spec) \\ EL(Impl) \subseteq EL(Spec) \end{cases}$$

Notion of error: $\varepsilon \in ET(S)$
$\Rightarrow$ Basic requirement satisfied

Local Error Semantics

- Error traces: $ET(S) = cont(PrT(S)) \cup cont(MIT(S))$
- Flooded language: $EL(S) = L(S) \cup ET(S)$

Refinement preorder:

$$Impl \sqsubseteq_{loc} Spec :\Leftrightarrow \begin{cases} ET(Impl) \subseteq ET(Spec) \\ EL(Impl) \subseteq EL(Spec) \end{cases}$$

Notion of error: $\varepsilon \in ET(S)$

$\Rightarrow$ Basic requirement satisfied

Local Error Semantics

- Error traces: $ET(S) = cont(PrT(S)) \cup cont(MIT(S))$
- Flooded language: $EL(S) = L(S) \cup ET(S)$

Refinement preorder:

$$Impl \sqsubseteq_{loc} Spec :\Leftrightarrow \begin{cases} ET(Impl) \subseteq ET(Spec) \\ EL(Impl) \subseteq EL(Spec) \end{cases}$$

Notion of error: $\varepsilon \in ET(S)$
$\Rightarrow$ Basic requirement satisfied

Compositionality

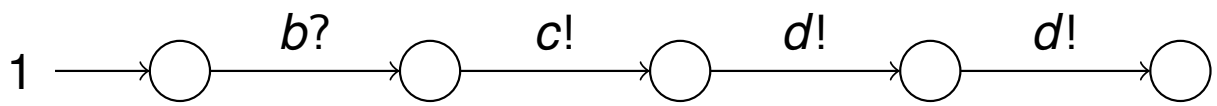
- $ET_{12} = cont\Big(prune\Big((ET_1 \mid EL_2) \cup (EL_1 \mid ET_2)\Big)\Big)$
- $EL_{12} = (EL_1 \mid EL_2) \cup ET_{12}$

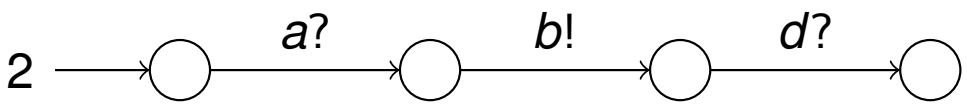
Compositionality

- $ET_{12} = cont\Big(prune\Big((ET_1 \mid EL_2) \cup (EL_1 \mid ET_2)\Big)\Big)$
- $EL_{12} = (EL_1 \mid EL_2) \cup ET_{12}$

Compositionality – Example

$$ET_{12} = cont\Big(prune\Big(\big(ET_1 \mid EL_2\big) \cup \big(EL_1 \mid ET_2\big)\Big)\Big)$$



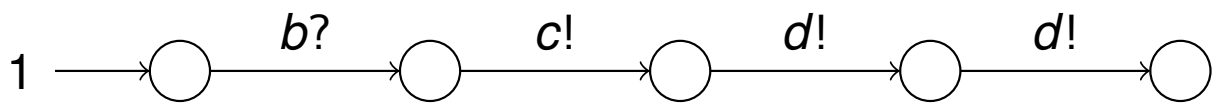


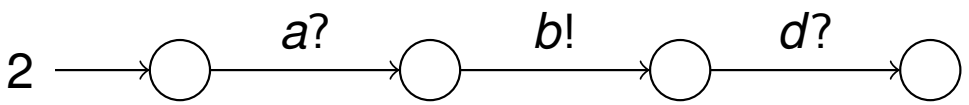
$bcdd \in L_1 \subseteq EL_1$ and $abdd \in MIT_2 \subseteq ET_2$

$\Rightarrow \{a\}\Sigma^* \subseteq ET_{12}$

Compositionality – Example

$$ET_{12} = cont\Big(prune\Big(\big(ET_1 \mid EL_2\big) \cup \big(EL_1 \mid ET_2\big)\Big)\Big)$$

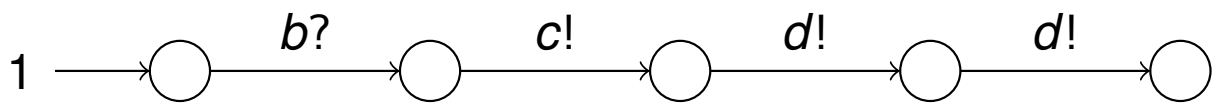


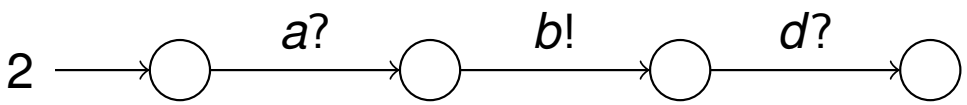


$bcdd \in L_1 \subseteq EL_1$ and $abdd \in MIT_2 \subseteq ET_2$
$\Rightarrow \{a\}\Sigma^* \subseteq ET_{12}$

Compositionality – Example

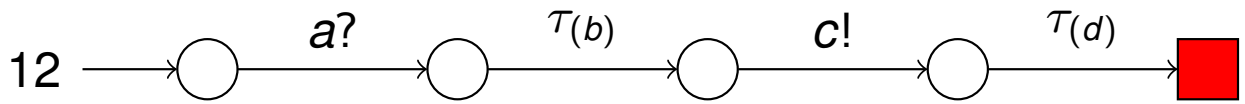
$$ET_{12} = cont\Big(prune\Big(\big(ET_1 \mid EL_2\big) \cup \big(EL_1 \mid ET_2\big)\Big)\Big)$$





$bcdd \in L_1 \subseteq EL_1$ and $abdd \in MIT_2 \subseteq ET_2$

$\Rightarrow \{a\}\Sigma^* \subseteq ET_{12}$



Compositionality

- $ET_{12} = cont\Big(prune\Big((ET_1 \mid EL_2) \cup (EL_1 \mid ET_2)\Big)\Big)$
- $EL_{12} = (EL_1 \mid EL_2) \cup ET_{12}$

$\Rightarrow \sqsubseteq_{loc}$ is a precongruence (monotony)

Compositionality

- $ET_{12} = cont\Big(prune\Big((ET_1 \mid EL_2) \cup (EL_1 \mid ET_2)\Big)\Big)$
- $EL_{12} = (EL_1 \mid EL_2) \cup ET_{12}$

$\Rightarrow \sqsubseteq_{loc}$ is a precongruence (monotony)

Coarsest Precongruence

Proove $Impl \sqsubseteq^{c}_{loc} Spec \Rightarrow Impl \sqsubseteq_{loc} Spec$

- Assume $Impl \sqsubseteq^{c}_{loc} Spec$
- Thus $Impl \mid Env \sqsubseteq^{c}_{loc} Spec \mid Env$
- Thus $Impl \mid Env \sqsubseteq^{B}_{loc} Spec \mid Env$
- $Impl \mid Env$ has an error $\Rightarrow$ $Spec \mid Env$ has an error.

- Suitable test environment Env for every $w \in ET(Impl)$
- $Spec \mid Env$ has an error as well
- Show that this error is due to $w \in ET(Spec)$

Coarsest Precongruence

Proove $Impl \sqsubseteq^{c}_{loc} Spec \Rightarrow Impl \sqsubseteq_{loc} Spec$

- Assume $Impl \sqsubseteq^{c}_{loc} Spec$
- Thus $Impl \mid Env \sqsubseteq^{c}_{loc} Spec \mid Env$
- Thus $Impl \mid Env \sqsubseteq^{B}_{loc} Spec \mid Env$
- $Impl \mid Env$ has an error $\Rightarrow$ $Spec \mid Env$ has an error.

- Suitable test environment Env for every $w \in ET(Impl)$
- $Spec \mid Env$ has an error as well
- Show that this error is due to $w \in ET(Spec)$

Coarsest Precongruence

Proove $Impl \sqsubseteq^{c}_{loc} Spec \Rightarrow Impl \sqsubseteq_{loc} Spec$

- Assume $Impl \sqsubseteq^{c}_{loc} Spec$
- Thus $Impl \mid Env \sqsubseteq^{c}_{loc} Spec \mid Env$
- Thus $Impl \mid Env \sqsubseteq^{B}_{loc} Spec \mid Env$
- $Impl \mid Env$ has an error $\Rightarrow$ $Spec \mid Env$ has an error.
- Suitable test environment Env for every $w \in ET(Impl)$
- $Spec \mid Env$ has an error as well
- Show that this error is due to $w \in ET(Spec)$

Coarsest Precongruence

Proove $Impl \sqsubseteq^{c}_{loc} Spec \Rightarrow Impl \sqsubseteq_{loc} Spec$

- Assume $Impl \sqsubseteq^{c}_{loc} Spec$
- Thus $Impl \mid Env \sqsubseteq^{c}_{loc} Spec \mid Env$
- Thus $Impl \mid Env \sqsubseteq^{B}_{loc} Spec \mid Env$
- $Impl \mid Env$ has an error $\Rightarrow$ $Spec \mid Env$ has an error.
- Suitable test environment Env for every $w \in ET(Impl)$
- $Spec \mid Env$ has an error as well
- Show that this error is due to $w \in ET(Spec)$

Coarsest Precongruence

Proove $Impl \sqsubseteq^{c}_{loc} Spec \Rightarrow Impl \sqsubseteq_{loc} Spec$

- Assume $Impl \sqsubseteq^{c}_{loc} Spec$
- Thus $Impl \mid Env \sqsubseteq^{c}_{loc} Spec \mid Env$
- Thus $Impl \mid Env \sqsubseteq^{B}_{loc} Spec \mid Env$
- $Impl \mid Env$ has an error $\Rightarrow$ $Spec \mid Env$ has an error.

- Suitable test environment Env for every $w \in ET(Impl)$
- $Spec \mid Env$ has an error as well
- Show that this error is due to $w \in ET(Spec)$

Coarsest Precongruence

Proove $Impl \sqsubseteq^{c}_{loc} Spec \Rightarrow Impl \sqsubseteq_{loc} Spec$

- Assume $Impl \sqsubseteq^{c}_{loc} Spec$
- Thus $Impl \mid Env \sqsubseteq^{c}_{loc} Spec \mid Env$
- Thus $Impl \mid Env \sqsubseteq^{B}_{loc} Spec \mid Env$
- $Impl \mid Env$ has an error $\Rightarrow$ $Spec \mid Env$ has an error.
- Suitable test environment Env for every $w \in ET(Impl)$
- $Spec \mid Env$ has an error as well
- Show that this error is due to $w \in ET(Spec)$

Coarsest Precongruence

Proove $Impl \sqsubseteq^{c}_{loc} Spec \Rightarrow Impl \sqsubseteq_{loc} Spec$

- Assume $Impl \sqsubseteq^{c}_{loc} Spec$
- Thus $Impl \mid Env \sqsubseteq^{c}_{loc} Spec \mid Env$
- Thus $Impl \mid Env \sqsubseteq^{B}_{loc} Spec \mid Env$
- $Impl \mid Env$ has an error $\Rightarrow$ $Spec \mid Env$ has an error.

- Suitable test environment Env for every $w \in ET(Impl)$
- $Spec \mid Env$ has an error as well
- Show that this error is due to $w \in ET(Spec)$

Coarsest Precongruence

Proove $Impl \sqsubseteq^{c}_{loc} Spec \Rightarrow Impl \sqsubseteq_{loc} Spec$

- Assume $Impl \sqsubseteq^{c}_{loc} Spec$
- Thus $Impl \mid Env \sqsubseteq^{c}_{loc} Spec \mid Env$
- Thus $Impl \mid Env \sqsubseteq^{B}_{loc} Spec \mid Env$
- $Impl \mid Env$ has an error $\Rightarrow$ $Spec \mid Env$ has an error.

- Suitable test environment Env for every $w \in ET(Impl)$
- $Spec \mid Env$ has an error as well
- Show that this error is due to $w \in ET(Spec)$

Local Errors – recap

- Characterisation of the coarsest precongruence
- Associativity w.r.t. $=_{loc}$
- Pruning on systems is correct
- Alternating refinement is correct, albeit too strict

Local Errors – recap

- Characterisation of the coarsest precongruence
- Associativity w.r.t. $=_{loc}$
- Pruning on systems is correct
- Alternating refinement is correct, albeit too strict

Local Errors – recap

- Characterisation of the coarsest precongruence
- Associativity w.r.t. $=_{loc}$
- Pruning on systems is correct
- Alternating refinement is correct, albeit too strict

Local Errors – recap

- Characterisation of the coarsest precongruence
- Associativity w.r.t. $=_{loc}$
- Pruning on systems is correct
- Alternating refinement is correct, albeit too strict

Local Errors – questions

- Is pruning useful because of the special role of outputs in the basic requirement?
- How would the precongruence change if the notion of error changes?

Local Errors – questions

- Is pruning useful because of the special role of outputs in the basic requirement?
- How would the precongruence change if the notion of error changes?

Notions of Error

Basic requirement:
If the specification is error free, then the implementation has to be error free as well

Notions of having an error:

1. An error state is reachable by locally controlled actions (internal and output)
2. An error state is reachable by internal actions
3. An error state is reachable at all

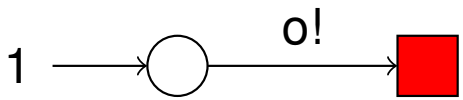
Internal errors

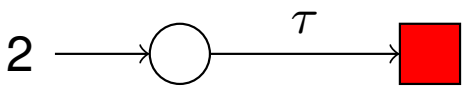
S has an error iff $q_0 \stackrel{\varepsilon}{\Rightarrow} q \in E$

- $ET_1 = EL_1 = \Sigma^* = ET_2 = EL_2$
- Local semantics do not suffice

Internal errors

S has an error iff $q_0 \stackrel{\varepsilon}{\Rightarrow} q \in E$





- $ET_1 = EL_1 = \Sigma^* = ET_2 = EL_2$
- Local semantics do not suffice

Internal errors

S has an error iff $q_0 \stackrel{\varepsilon}{\Rightarrow} q \in E$

- $ET_1 = EL_1 = \Sigma^* = ET_2 = EL_2$
- Local semantics do not suffice

Internal errors

S has an error iff $q_0 \stackrel{\varepsilon}{\Rightarrow} q \in E$

- $ET_1 = EL_1 = \Sigma^* = ET_2 = EL_2$
- Local semantics do not suffice

Pairs

- Strict error pairs:
  $StP(S) = \{(w, X) \mid w \in StT(S), out(w) = X\}$
- Pruned error pairs:
  $PrP(S) = \{prune(w, X) \mid (w, X) \in StP(S)\}$
- Missing-input pairs:
  $MIP(S) = \{(w, X) \mid w \in MIT(S), out(w) = X\}$

- Error pairs: $EP(S) = cont(PrP(S)) \cup cont(MIP(S))$
- Error pair traces: $EPT(S) = \{w \mid (w, out(w)) \in EP(S)\}$
- Error pair language: $EPL(S) = L(S) \cup EPT(S)$

Pairs

- Strict error pairs:
  $StP(S) = \{(w, X) \mid w \in StT(S), out(w) = X\}$
- Pruned error pairs:
  $PrP(S) = \{prune(w, X) \mid (w, X) \in StP(S)\}$
- Missing-input pairs:
  $MIP(S) = \{(w, X) \mid w \in MIT(S), out(w) = X\}$

- Error pairs: $EP(S) = cont(PrP(S)) \cup cont(MIP(S))$
- Error pair traces: $EPT(S) = \{w \mid (w, out(w)) \in EP(S)\}$
- Error pair language: $EPL(S) = L(S) \cup EPT(S)$

Pairs

- Strict error pairs:
  $StP(S) = \{(w, X) \mid w \in StT(S), out(w) = X\}$
- Pruned error pairs:
  $PrP(S) = \{prune(w, X) \mid (w, X) \in StP(S)\}$
- Missing-input pairs:
  $MIP(S) = \{(w, X) \mid w \in MIT(S), out(w) = X\}$

- Error pairs: $EP(S) = cont(PrP(S)) \cup cont(MIP(S))$
- Error pair traces: $EPT(S) = \{w \mid (w, out(w)) \in EP(S)\}$
- Error pair language: $EPL(S) = L(S) \cup EPT(S)$

Pairs

- Strict error pairs:
  $StP(S) = \{(w, X) \mid w \in StT(S), out(w) = X\}$
- Pruned error pairs:
  $PrP(S) = \{prune(w, X) \mid (w, X) \in StP(S)\}$
- Missing-input pairs:
  $MIP(S) = \{(w, X) \mid w \in MIT(S), out(w) = X\}$

- Error pairs: $EP(S) = cont(PrP(S)) \cup cont(MIP(S))$
- Error pair traces: $EPT(S) = \{w \mid (w, out(w)) \in EP(S)\}$
- Error pair language: $EPL(S) = L(S) \cup EPT(S)$

Pairs

- Strict error pairs:
  $StP(S) = \{(w, X) \mid w \in StT(S), out(w) = X\}$
- Pruned error pairs:
  $PrP(S) = \{prune(w, X) \mid (w, X) \in StP(S)\}$
- Missing-input pairs:
  $MIP(S) = \{(w, X) \mid w \in MIT(S), out(w) = X\}$

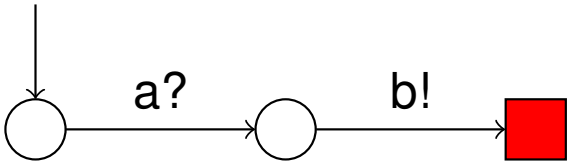
- Error pairs: $EP(S) = cont(PrP(S)) \cup cont(MIP(S))$
- Error pair traces: $EPT(S) = \{w \mid (w, out(w)) \in EP(S)\}$
- Error pair language: $EPL(S) = L(S) \cup EPT(S)$

Pairs

- Strict error pairs:
  $StP(S) = \{(w, X) \mid w \in StT(S), out(w) = X\}$
- Pruned error pairs:
  $PrP(S) = \{prune(w, X) \mid (w, X) \in StP(S)\}$
- Missing-input pairs:
  $MIP(S) = \{(w, X) \mid w \in MIT(S), out(w) = X\}$

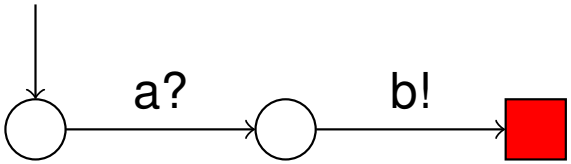
- Error pairs: $EP(S) = cont(PrP(S)) \cup cont(MIP(S))$
- Error pair traces: $EPT(S) = \{w \mid (w, out(w)) \in EP(S)\}$
- Error pair language: $EPL(S) = L(S) \cup EPT(S)$

Pairs – Example



- $StT(S) = \{ab\}$
- $PrT(S) = \{a\}$
- $PrP(S) = \{(a, \{b\})\}$

Pairs – Example



- $StT(S) = \{ab\}$
- $PrT(S) = \{a\}$
- $PrP(S) = \{(a, \{b\})\}$

Internal Error Semantics

- Error pairs: $EP(S) = cont(PrP(S)) \cup cont(MIP(S))$
- Error pair traces: $EPT(S) = \{w \mid (w, out(w)) \in EP(S)\}$
- Error pair language: $EPL(S) = L(S) \cup EPT(S)$

Refinement preorder:

$$Impl \sqsubseteq_{int} Spec :\Leftrightarrow \begin{cases} EP(Impl) \subseteq EP(Spec) \\ EPL(Impl) \subseteq EPL(Spec) \end{cases}$$

Notion of error: $(\varepsilon, \emptyset) \in EP(S)$

Internal Error Semantics

- Error pairs: $EP(S) = cont(PrP(S)) \cup cont(MIP(S))$
- Error pair traces: $EPT(S) = \{w \mid (w, out(w)) \in EP(S)\}$
- Error pair language: $EPL(S) = L(S) \cup EPT(S)$

Refinement preorder:

$$Impl \sqsubseteq_{int} Spec :\Leftrightarrow \begin{cases} EP(Impl) \subseteq EP(Spec) \\ EPL(Impl) \subseteq EPL(Spec) \end{cases}$$

Notion of error: $(\varepsilon, \emptyset) \in EP(S)$

Internal Error Semantics

- Error pairs: $EP(S) = cont(PrP(S)) \cup cont(MIP(S))$
- Error pair traces: $EPT(S) = \{w \mid (w, out(w)) \in EP(S)\}$
- Error pair language: $EPL(S) = L(S) \cup EPT(S)$

Refinement preorder:

$$Impl \sqsubseteq_{int} Spec :\Leftrightarrow \begin{cases} EP(Impl) \subseteq EP(Spec) \\ EPL(Impl) \subseteq EPL(Spec) \end{cases}$$

Notion of error: $(\varepsilon, \emptyset) \in EP(S)$

Internal errors

- $ET_1 = EL_1 = \Sigma^* = ET_2 = EL_2$
- Local semantics do not suffice
- $EP_1 = cont\{(\varepsilon, \{o\})\} \subsetneq cont\{(\varepsilon, \emptyset)\} = EP_2$
- $(\varepsilon, \emptyset) \notin EP_1$, but $(\varepsilon, \emptyset) \in EP_2$

Internal errors

- $ET_1 = EL_1 = \Sigma^* = ET_2 = EL_2$
- Local semantics do not suffice
- $EP_1 = cont\{(\varepsilon, \{o\})\} \subsetneq cont\{(\varepsilon, \emptyset)\} = EP_2$
- $(\varepsilon, \emptyset) \notin EP_1$, but $(\varepsilon, \emptyset) \in EP_2$

Internal errors

- $ET_1 = EL_1 = \Sigma^* = ET_2 = EL_2$
- Local semantics do not suffice
- $EP_1 = cont\{(\varepsilon, \{o\})\} \subsetneq cont\{(\varepsilon, \emptyset)\} = EP_2$
- $(\varepsilon, \emptyset) \notin EP_1$, but $(\varepsilon, \emptyset) \in EP_2$

Internal precongruence

- $EP_{12} = cont\Big(prune\Big(\big(EP_1 \mid EPL_2\big) \cup \big(EPL_1 \mid EP_2\big)\Big)\Big)$
- $EPL_{12} = \big(EPL_1 \mid EPL_2\big) \cup EPT_{12}$

  $\Rightarrow \sqsubseteq_{int}$ is a precongruence (monotony)
- Coarsest precongruence proof is analogous to local one

Internal precongruence

- $EP_{12} = cont\Big(prune\Big(\big(EP_1 \mid EPL_2\big) \cup \big(EPL_1 \mid EP_2\big)\Big)\Big)$
- $EPL_{12} = \big(EPL_1 \mid EPL_2\big) \cup EPT_{12}$

  $\Rightarrow \sqsubseteq_{int}$ is a precongruence (monotony)
- Coarsest precongruence proof is analogous to local one

Internal precongruence

- $EP_{12} = cont\Big(prune\Big(\big(EP_1 \mid EPL_2\big) \cup \big(EPL_1 \mid EP_2\big)\Big)\Big)$
- $EPL_{12} = \big(EPL_1 \mid EPL_2\big) \cup EPT_{12}$

  $\Rightarrow \sqsubseteq_{int}$ is a precongruence (monotony)
- Coarsest precongruence proof is analogous to local one

Internal precongruence

- $EP_{12} = cont\Big(prune\Big(\big(EP_1 \mid EPL_2\big) \cup \big(EPL_1 \mid EP_2\big)\Big)\Big)$
- $EPL_{12} = \big(EPL_1 \mid EPL_2\big) \cup EPT_{12}$

  $\Rightarrow \sqsubseteq_{int}$ is a precongruence (monotony)
- Coarsest precongruence proof is analogous to local one

Internal vs. Local

- Both approaches use pruning
- Internal approach is strictly finer than local one: $Impl \sqsubseteq_{int} Spec$ implies $Impl \sqsubseteq_{loc} Spec$, but not the other way round

Internal vs. Local

- Both approaches use pruning
- Internal approach is strictly finer than local one: $Impl \sqsubseteq_{int} Spec$ implies $Impl \sqsubseteq_{loc} Spec$, but not the other way round

Notions of Error

Basic requirement:
If the specification is error free, then the implementation has to be error free as well

Notions of having an error:

1. An error state is reachable by locally controlled actions (internal and output)
2. An error state is reachable by internal actions
3. An error state is reachable at all

Missing inputs

Again local semantics do not suffice: $ET_1 = \{aa\}\Sigma^* = ET_2$

Pessimistic Semantics

- Continued pruned traces:
$$CPT(S) = cont(prune(StT(S)));$$
- Flooded missing-input traces:
$$MIC(S) = MIT(S) \cup CPT(S);$$
- CPT-flooded language: $LCP(S) = L(S) \cup CPT(S)$.

Refinement preorder:

$$Impl \sqsubseteq_{act} Spec :\Leftrightarrow \begin{cases} CPT(Impl) \subseteq CPT(Spec) \\ MIC(Impl) \subseteq MIC(Spec) \\ LCP(Impl) \subseteq LCP(Spec) \end{cases}$$

Notion of error: $\exists w : \ w \in CPT(S)$

Pessimistic Semantics

- Continued pruned traces:
$$CPT(S) = cont(prune(StT(S)));$$
- Flooded missing-input traces:
$$MIC(S) = MIT(S) \cup CPT(S);$$
- CPT-flooded language: $LCP(S) = L(S) \cup CPT(S)$.

Refinement preorder:

$$Impl \sqsubseteq_{act} Spec :\Leftrightarrow \begin{cases} CPT(Impl) \subseteq CPT(Spec) \\ MIC(Impl) \subseteq MIC(Spec) \\ LCP(Impl) \subseteq LCP(Spec) \end{cases}$$

Notion of error: $\exists w : \ w \in CPT(S)$

Pessimistic Semantics

- Continued pruned traces:
$$CPT(S) = cont(prune(StT(S)));$$
- Flooded missing-input traces:
$$MIC(S) = MIT(S) \cup CPT(S);$$
- CPT-flooded language: $LCP(S) = L(S) \cup CPT(S)$.

Refinement preorder:

$$Impl \sqsubseteq_{act} Spec :\Leftrightarrow \begin{cases} CPT(Impl) \subseteq CPT(Spec) \\ MIC(Impl) \subseteq MIC(Spec) \\ LCP(Impl) \subseteq LCP(Spec) \end{cases}$$

Notion of error: $\exists w : \ w \in CPT(S)$

Pessimistic Semantics

- Continued pruned traces:
$$CPT(S) = cont(prune(StT(S)));$$
- Flooded missing-input traces:
$$MIC(S) = MIT(S) \cup CPT(S);$$
- CPT-flooded language: $LCP(S) = L(S) \cup CPT(S)$.

Refinement preorder:

$$Impl \sqsubseteq_{act} Spec :\Leftrightarrow \begin{cases} CPT(Impl) \subseteq CPT(Spec) \\ MIC(Impl) \subseteq MIC(Spec) \\ LCP(Impl) \subseteq LCP(Spec) \end{cases}$$

Notion of error: $\exists w : \ w \in CPT(S)$

Pessimistic Semantics

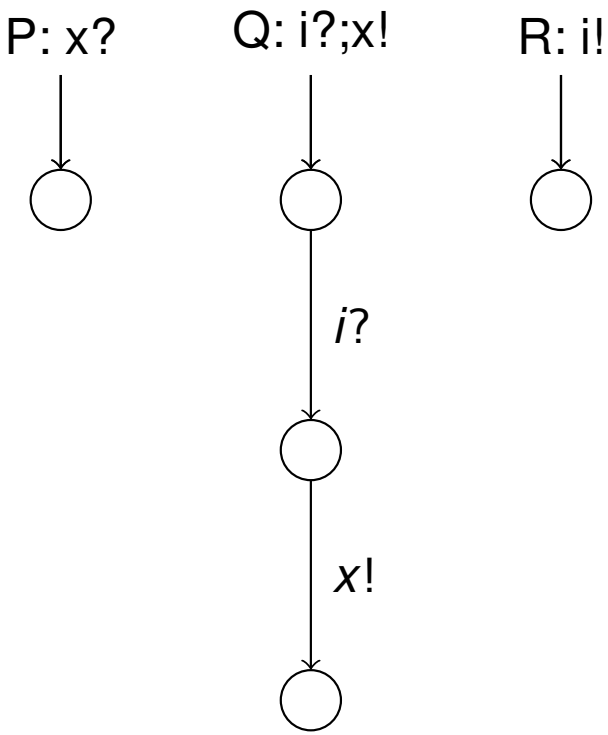
- Continued pruned traces:
$$CPT(S) = cont(prune(StT(S)));$$
- Flooded missing-input traces:
$$MIC(S) = MIT(S) \cup CPT(S);$$
- CPT-flooded language: $LCP(S) = L(S) \cup CPT(S)$.

Refinement preorder:

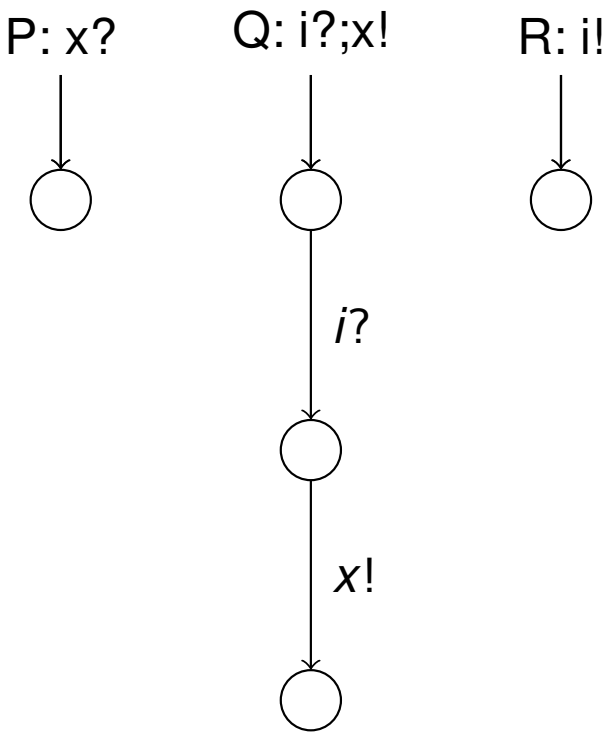
$$Impl \sqsubseteq_{act} Spec :\Leftrightarrow \begin{cases} CPT(Impl) \subseteq CPT(Spec) \\ MIC(Impl) \subseteq MIC(Spec) \\ LCP(Impl) \subseteq LCP(Spec) \end{cases}$$

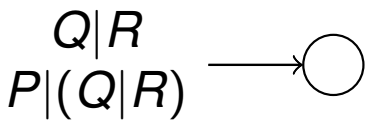
Notion of error: $\exists w : \ w \in CPT(S)$

Associativity

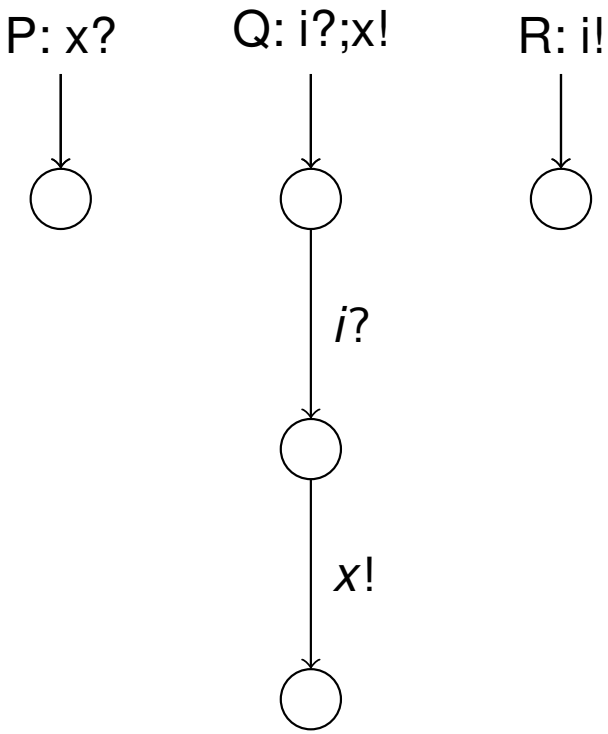


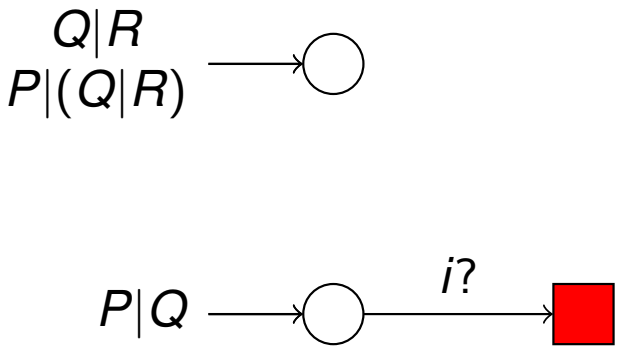
Associativity



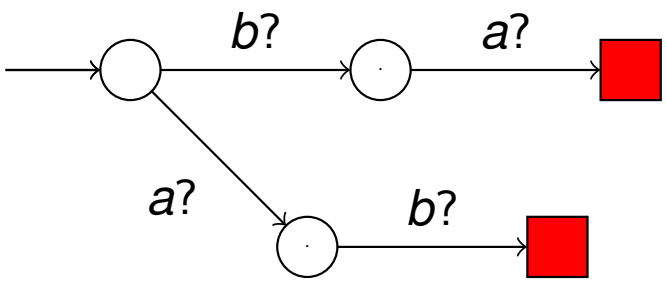


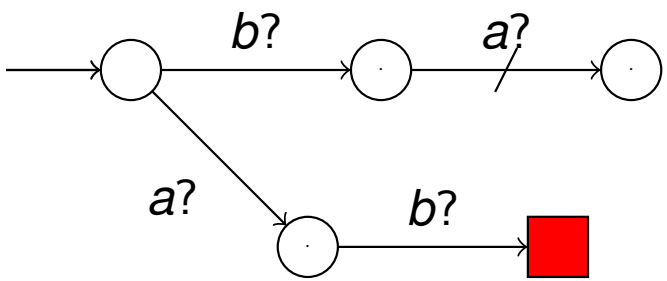
Associativity





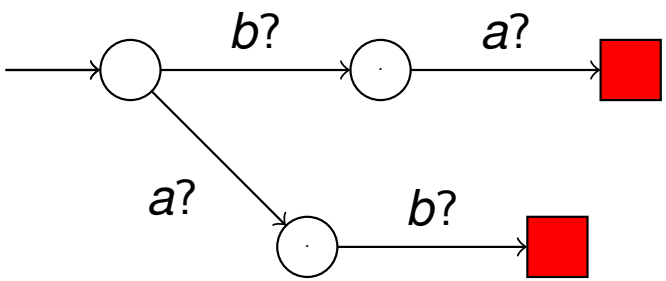
Non-Coarsestness

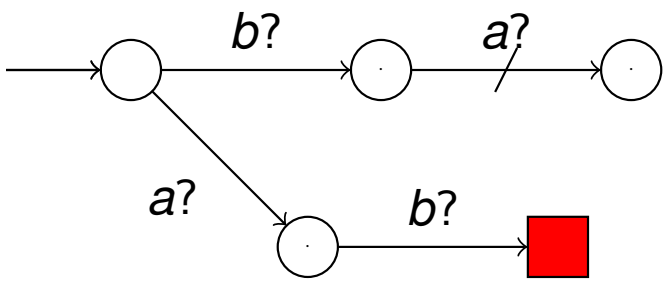




- $a, b \notin O_{Env}$: no change
- $a \in O_{Env}, b \notin O_{Env}$: $Env \overset{wav}{\not\Rightarrow}$ or error after $b?$
- $a, b \in O_{Env}$: no errors, or same errors

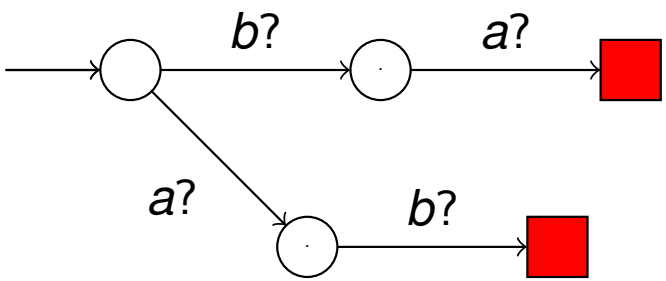
Non-Coarsestness

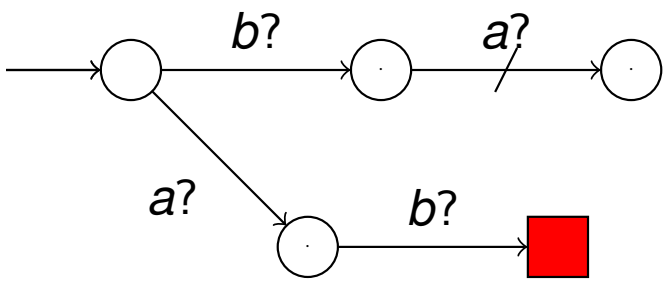




- $a, b \notin O_{Env}$: no change
- $a \in O_{Env}, b \notin O_{Env}$: $Env \overset{wav}{\Rightarrow}$ or error after b?
- $a, b \in O_{Env}$: no errors, or same errors

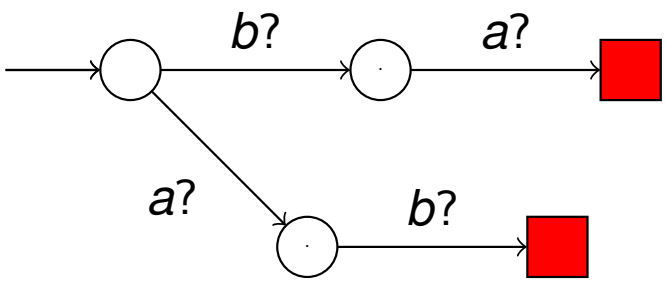
Non-Coarsestness

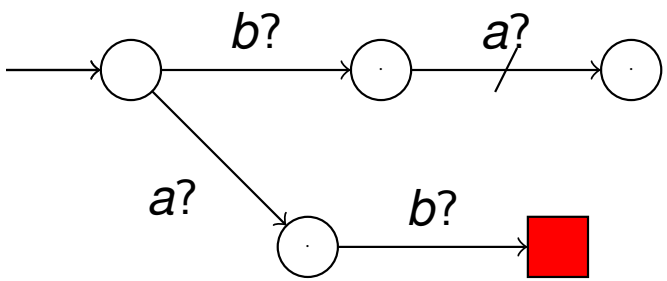




- $a, b \notin O_{Env}$: no change
- $a \in O_{Env}, b \notin O_{Env}$: $Env \overset{wav}{\Rightarrow}$ or error after $b?$
- $a, b \in O_{Env}$: no errors, or same errors

Non-Coarsestness





- $a, b \notin O_{Env}$: no change
- $a \in O_{Env}, b \notin O_{Env}$: $Env \overset{wav}{\Rightarrow}$ or error after $b?$
- $a, b \in O_{Env}$: no errors, or same errors

Pessimistic vs. local

- Loss of associativity
- Strictly finer than local approach
- Unneccesarily complex

Pessimistic vs. local

- Loss of associativity
- Strictly finer than local approach
- Unneccesarily complex

Pessimistic vs. local

- Loss of associativity
- Strictly finer than local approach
- Unneccesarily complex

Conclusion

- Optimistic/local approach corresponds to intuition
- Internal and pessimistic are finer
- Optimistic/local approach is characterized on closed systems

Conclusion

- Optimistic/local approach corresponds to intuition
- Internal and pessimistic are finer
- Optimistic/local approach is characterized on closed systems

Conclusion

- Optimistic/local approach corresponds to intuition
- Internal and pessimistic are finer
- Optimistic/local approach is characterized on closed systems

Be optimistic!

Thank you