Interface-based Design 2

Tom Henzinger
EPFL and UC Berkeley

Partial Composition

Bottom-up Compositionality:

If $f||f'$ defined and $f \cdot g$ and $f' \cdot g'$,
then $g||g'$ defined and $f||f' \cdot g||g'$.

Top-down Compositionality:

If $g||g'$ defined and $g \lhd f$ and $g' \lhd f'$,
then $f||f'$ defined and $g||g' \lhd f||f'$.

Partial Composition

Bottom-up Compositionality:

If $f||f'$ defined and $f \cdot g$ and $f' \cdot g'$,
then $g||g'$ defined and $f||f' \cdot g||g'$.

Processes

Top-down Compositionality:

If $g||g'$ defined and $g \triangleright f$ and $g' \triangleright f'$,
then $f||f'$ defined and $g||g' \triangleright f||f'$.

Interfaces

Principle of independent implementability.

Partial Composition

Bottom-up Compositionality:

If $f||f'$ defined and $f \cdot g$ and $f' \cdot g'$,
then $g||g'$ defined and $f||f' \cdot g||g'$.

Processes

Trace containment, Simuation

Top-down Compositionality:

If $g||g'$ defined and $g \triangleleft f$ and $g' \triangleleft f'$,
then $f||f'$ defined and $g||g' \triangleleft f||f'$.

Interfaces

Subtyping

Principle of independent implementability.

Partial Composition

Bottom-up Compositionality:

If $f||f'$ defined and $f \cdot g$ and $f' \cdot g'$,
then $g||g'$ defined and $f||f' \cdot g||g'$.

Processes

Top-down Compositionality:

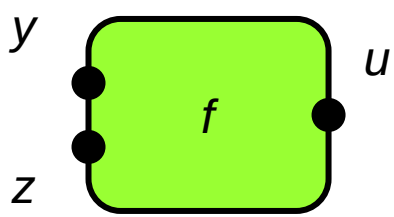
If $g||g'$ defined and $g \multimap f$ and $g' \multimap f'$,
then $f||f'$ defined and $g||g' \multimap f||f'$.

Interfaces

Block Diagram Algebra

-A set *F* of *blocks*.

-A set *P* of *ports*.

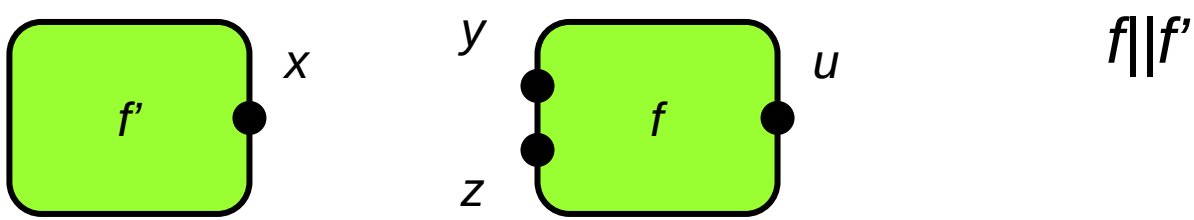


Block Diagram Algebra

-A set *F* of *blocks*.

-A set *P* of *ports*.

-A partial binary function || on blocks, called *composition*.



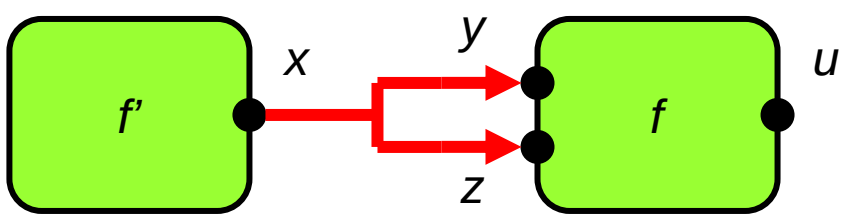
Block Diagram Algebra

-A set *F* of *blocks*.

-A set *P* of *ports*.

-A partial binary function || on blocks, called *composition*.

-A partial function mapping a block $f2F$ and an interconnect $\theta 2P£P$ to a block $P\theta$.



$(f||f')\theta$

$\theta = \{ (x,y), (x,z) \}$

Block Diagram Algebra

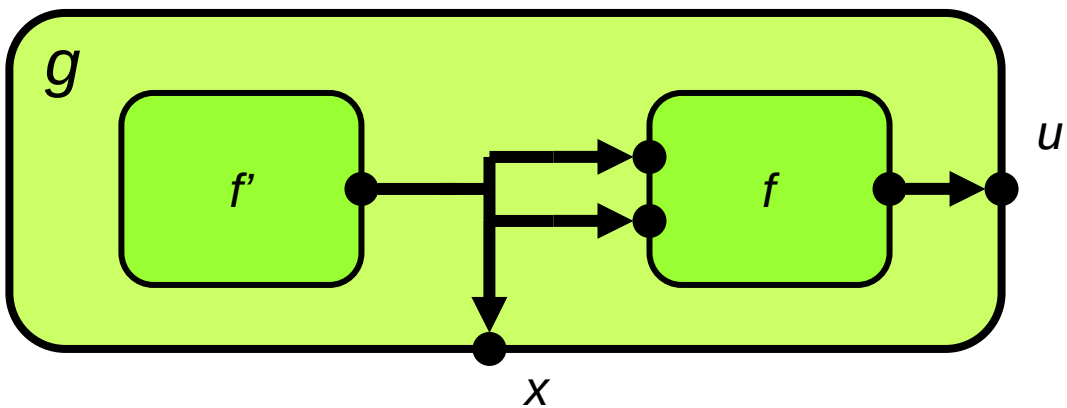
-A set *F* of *blocks*.

-A set *P* of *ports*.

-A partial binary function || on blocks, called *composition*.

-A partial function mapping a block $f \in F$ and an interconnect $\theta \in P \times P$ to a block $P\theta$.

-A binary relation $\cdot$ on blocks, called *hierarchy*.



$(f||f')\theta \cdot g$

A block diagram algebra is an ***interface algebra*** if

1. if $g||h$ defined and $g \triangleright f$, then $f||h$ defined and $g||h \triangleright f||h$
2. if $g\theta$ defined and $g \triangleright f$, then $f\theta$ defined and $g\theta \triangleright f\theta$

A block diagram algebra is a ***process algebra*** if

1. if $f||h$ defined and $f \cdot g$, then $g||h$ defined and $f||h \cdot g||h$
2. if $f\theta$ defined and $f \cdot g$, then $g\theta$ defined and $f\theta \cdot g\theta$

Stateless Input/Output Processes

$f = (I_f, O_f, p_f)$

I_f ... *input ports*

O_f ... *output ports:* disjoint from I_f

p_f ... *input/output relation:* predicate on I_f [O_f such that $(8\ I_f)(9\ O_f)\ p_f$

Stateless Input/Output Processes

fθ *defined* if $(8\, I_{f\theta})(9\, O_{f\theta})(p_f \text{ Æ } p_\theta)$

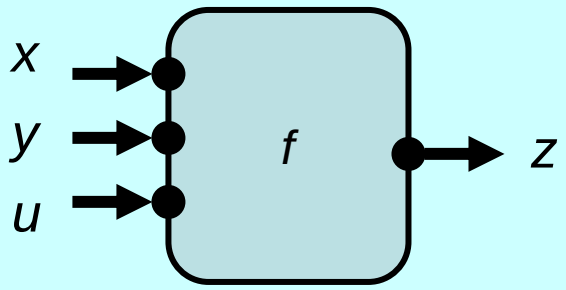
If so, then: $p_{f\theta} = (9\, O_\theta)(p_f \text{ Æ } p_\theta)$

$I_{f\theta} = I_f \setminus O_\theta$
$O_{f\theta} = O_f [\ O_\theta$

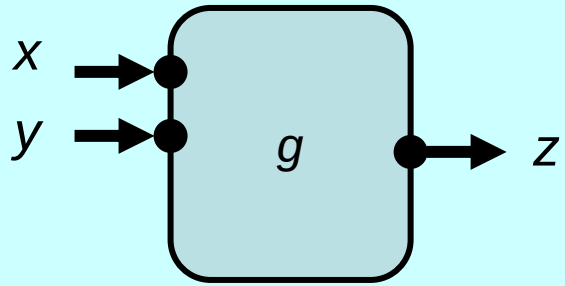
f · *g* if

1. I_f ¶ I_g
2. O_f ¶ O_g
3. p_f) p_g

Stateless Input/Output Processes



.

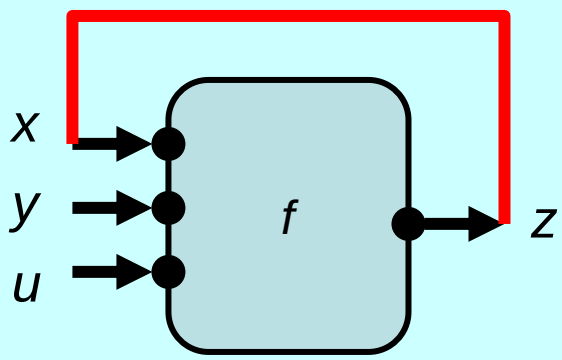


(u=0) z=0) Æ

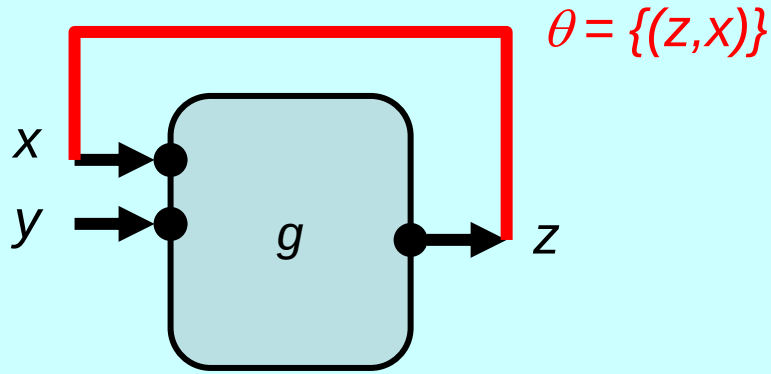
(u≠0) z=x+y)

(z=0 Ç z=x+y)

Stateless Input/Output Processes



.



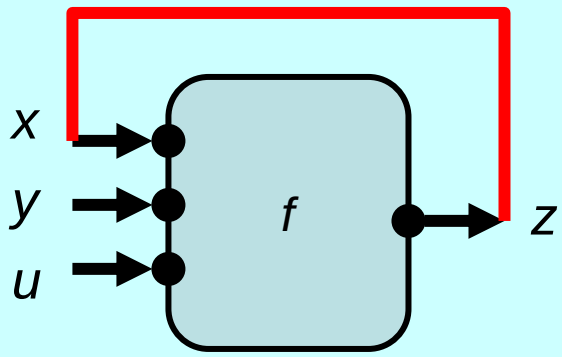
(u=0) z=0) Æ

(u≠0) z=x+y) Æ

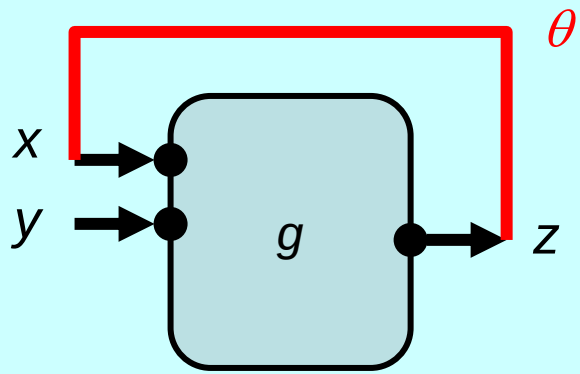
x=z

(z=0 Ç *z=x+y)*
Æ *x=z*

Stateless Input/Output Processes



.



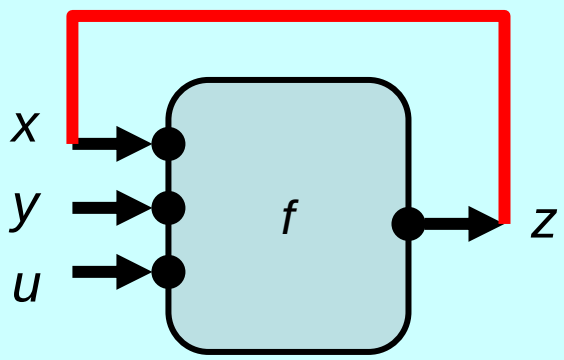
$\forall y,u\ \exists z,x$ $(u=0 \Rightarrow z=0) \wedge$

$(u \neq 0 \Rightarrow z=x+y) \wedge$

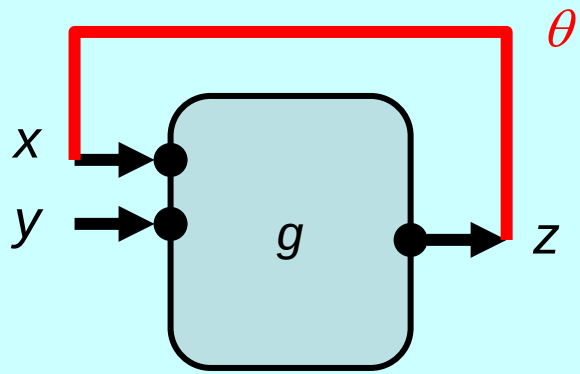
$x=z$

$\forall y\ \exists z,x$ $(z=0 \vee z=x+y)$
$\wedge\ x=z$

Stateless Input/Output Processes



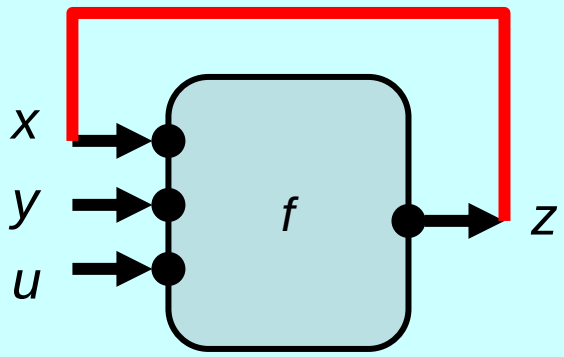
.



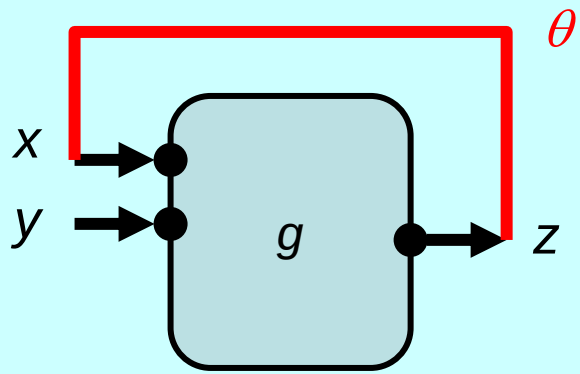
$\forall y,u\ \exists z,x$ $(u=0 \Rightarrow z=0) \wedge$ $(u \neq 0 \Rightarrow z=z+y) \wedge$ $x=z$

$\forall y\ \exists z,x$ $(z=0 \vee z=z+y) \wedge x=z$

Stateless Input/Output Processes



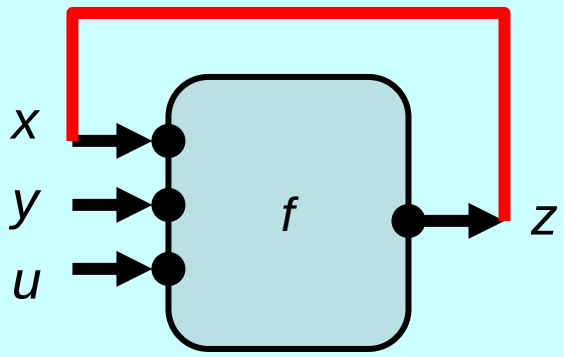
.



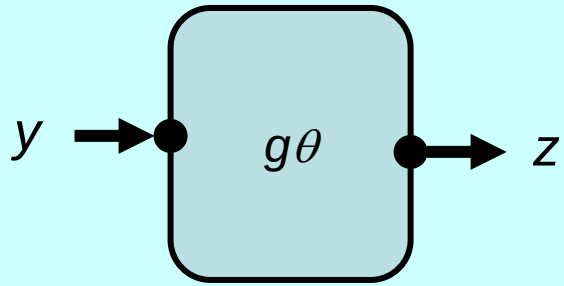
$\forall y,u\ \exists z,x$ $(u=0 \Rightarrow z=0) \wedge$ $(u \neq 0 \Rightarrow z=z+y) \wedge$ $x=z$

$\forall y\ \exists z,x$ $(z=0 \vee z=z+y) \wedge x=z$

Stateless Input/Output Processes



.



$\theta = \{(z,x)\}$

$8\ y,u\ 9\ z,x$ $(u=0 \ ) \ z=0) \ Æ$

$(u \neq 0 \ ) \ z=z+y) \ Æ$

$x=z$

$(z=0 \ Ç \ z=z+y)$

$Æ \ x=z$

Stateless Assume/Guarantee Interfaces

$f = (I_f, O^+_f, O_f, in_f, out_f)$

$O^+_f \P O_f$ ... *available ports:* disjoint from I_f

in_f ... *input assumption*: predicate on I_f
such that $(9\ I_f)\ in_f$

out_f ... *output guarantee*: predicate on O_f
such that $(9\ O_f)\ out_f$

Stateless Assume/Guarantee Interfaces

$f\theta$ *defined* if $(9\, I_{f\theta})(8\, O_{f\theta})((out_f \text{ Æ } p_\theta) \Rightarrow in_f)$

$I_{f\theta} = I_f \setminus O_\theta$
$O_{f\theta} = O_f \cup O_\theta$

If so, then:

$in_{f\theta} = (8\, O_{f\theta})((out_f \text{ Æ } p_\theta) \Rightarrow in_f)$

$out_{f\theta} = out_f \text{ Æ } p_\theta$

Stateless Assume/Guarantee Interfaces

$f\theta$ *defined* if $(9\ I_{f\theta})(8\ O_{f\theta})((out_f \text{ Æ } p_\theta) \text{) } in_f)$

$I_{f\theta} = I_f \setminus O_\theta$
$O_{f\theta} = O_f [O_\theta$

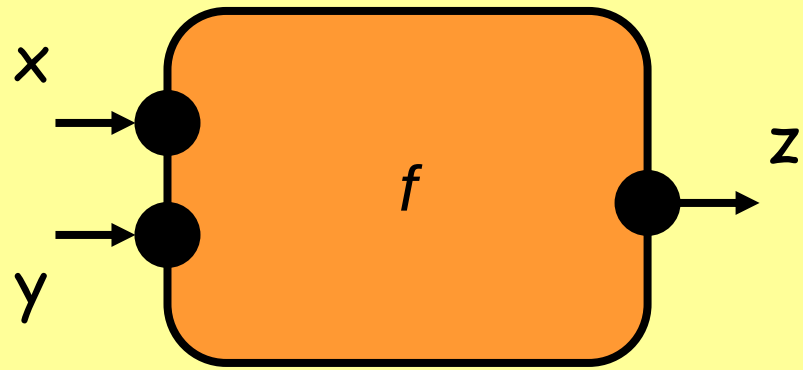
If so, then:

$in_{f\theta} = (8\ O_{f\theta})((out_f \text{ Æ } p_\theta) \text{) } in_f)$

$out_{f\theta} = out_f \text{ Æ } p_\theta$

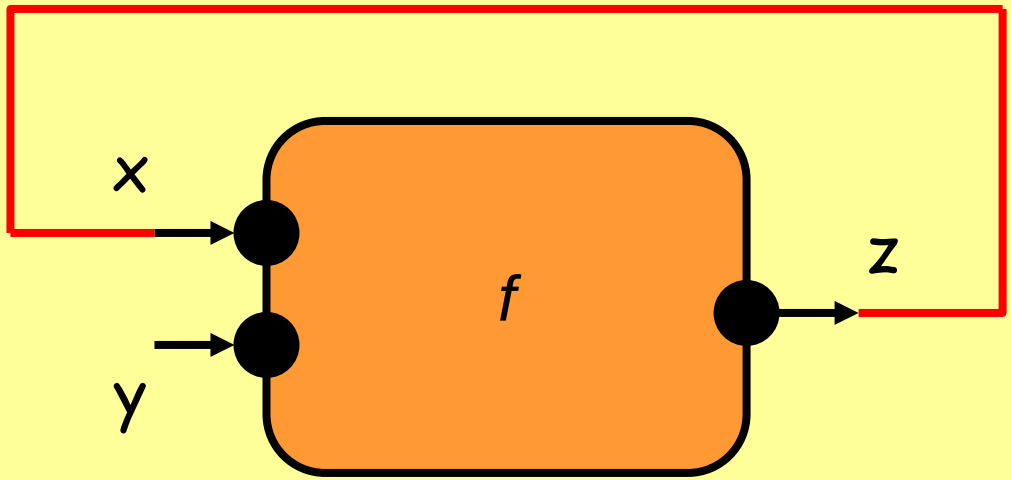
Stateless Input/Output Processes

$f\theta$ *defined* if $(8\ I_{f\theta})(9\ O_{f\theta})(p_f \text{ Æ } p_\theta)$



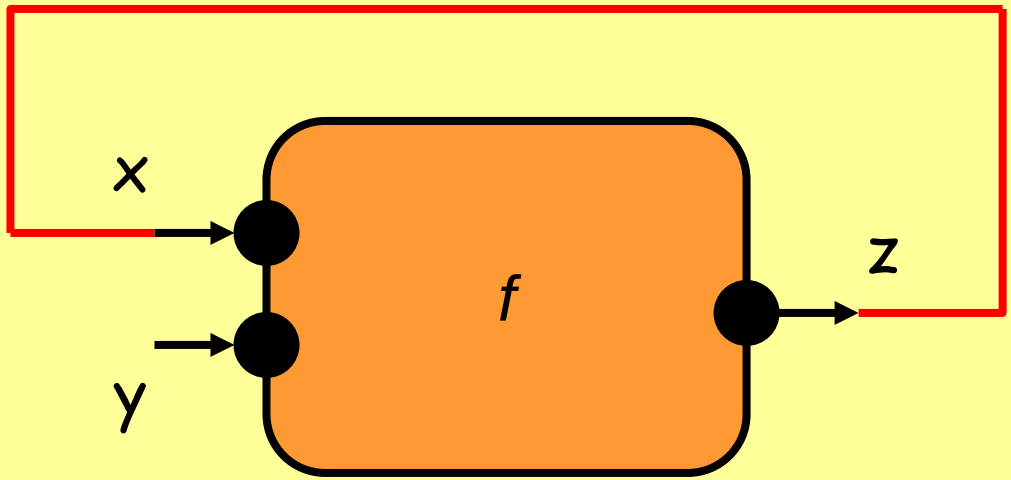
$x=0 \Rightarrow y=0$ true

$\theta = \{(z,x)\}$



$x=0 \Rightarrow y=0$ true

$\theta = \{(z,x)\}$

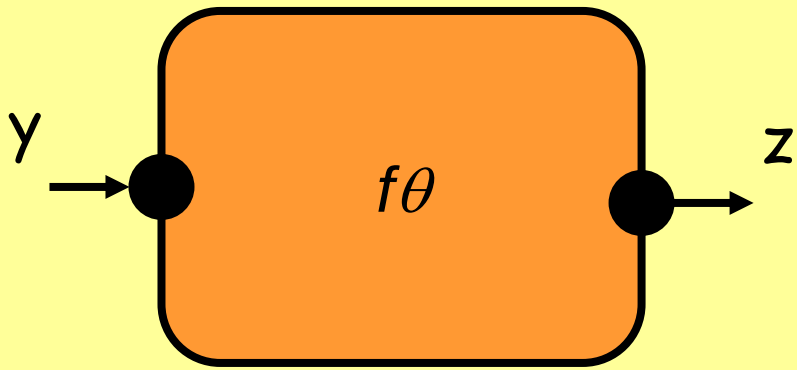


$x=0 \Rightarrow y=0$ true

$y = 0$

$(\forall x,z)\ ((\ \text{true} \wedge x=z\ ) \Rightarrow (\ x=0 \Rightarrow y=0\ ))$

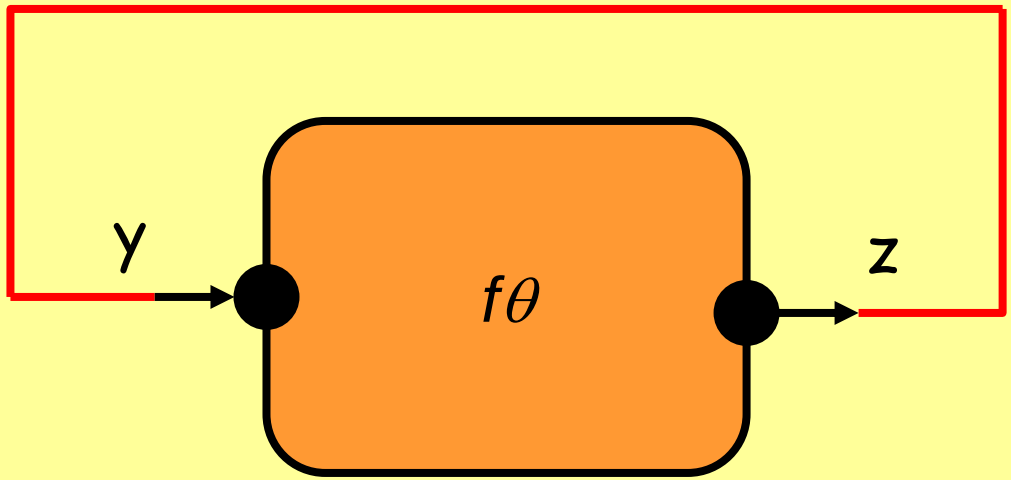
$\theta = \{(z,x)\}$



y = 0

true

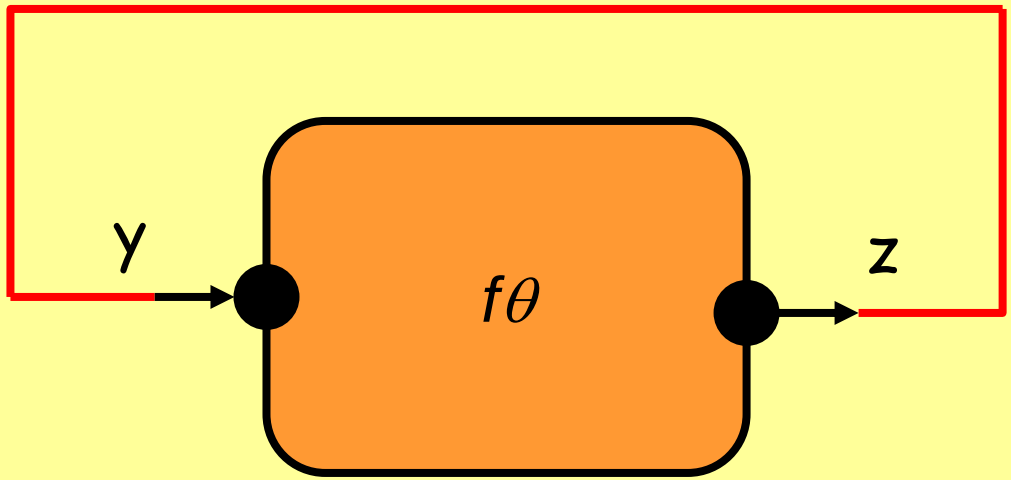
{(z,y)}



y = 0

true

{(z,y)}

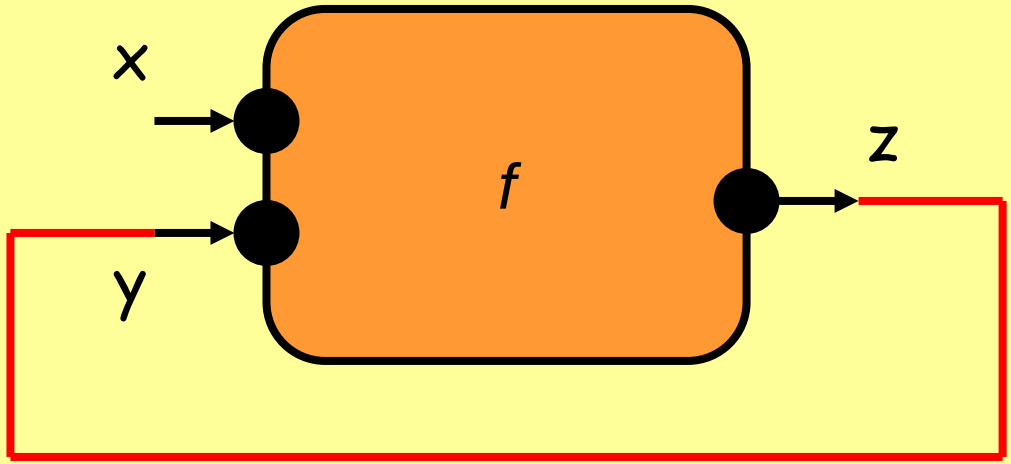


y = 0

true

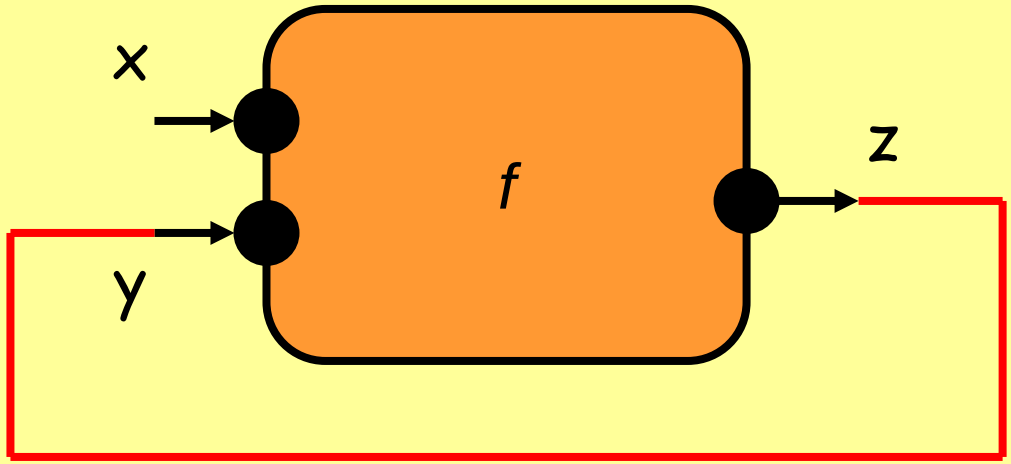
Undefined.

$\theta' = \{(z,y)\}$



$x=0 \Rightarrow y=0$ true

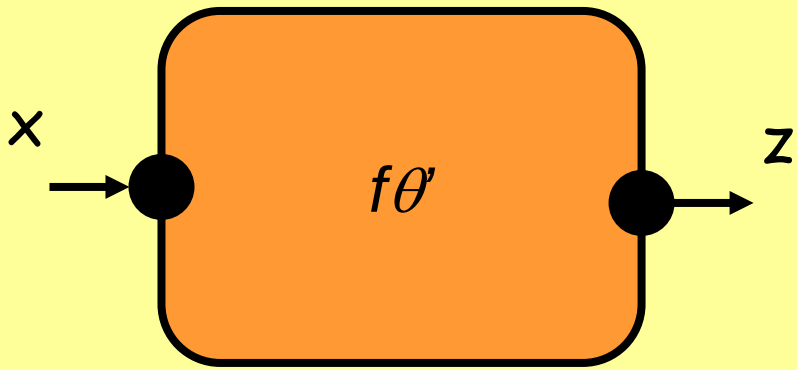
$\theta' = \{(z,y)\}$



$x=0 \Rightarrow y=0$ true

$x \neq 0$

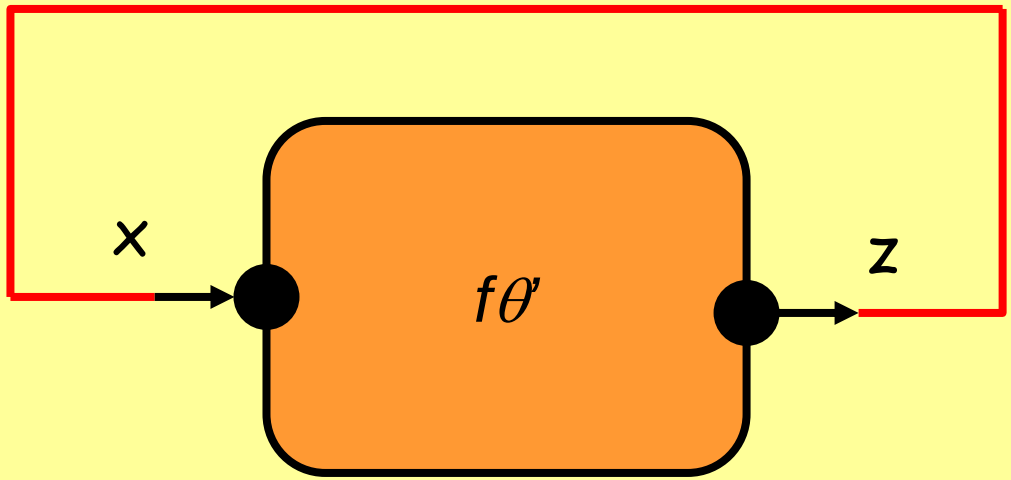
$\theta' = \{(z,y)\}$



$x \neq 0$

true

{(z,y)}



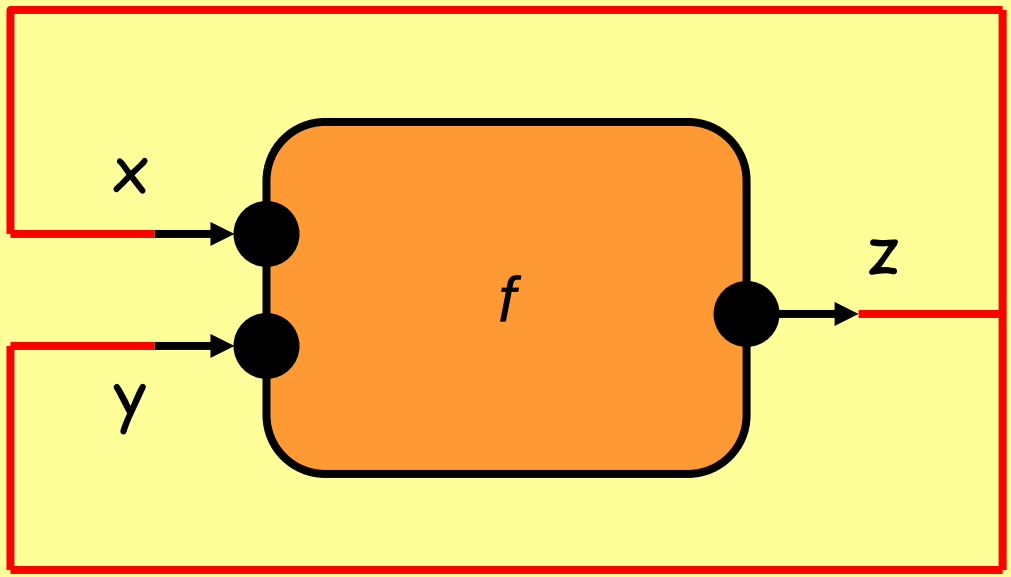
$x \neq 0$

true

Undefined.

Interconnects must be *Sets* of Links:

$\theta'' = \{(z,x),(z,y)\}$

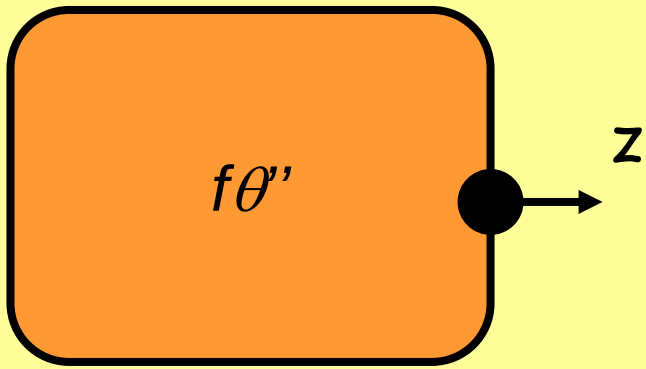


$x=0 \Rightarrow y=0$ true

true

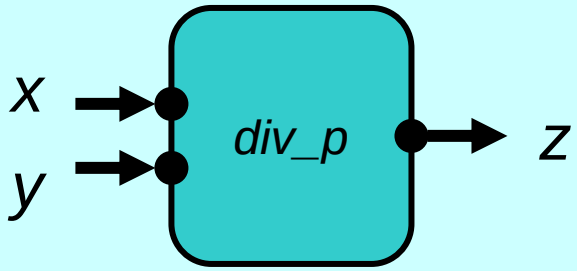
Interconnects must be *Sets* of Links:

$\theta'' = \{(z,x),(z,y)\}$



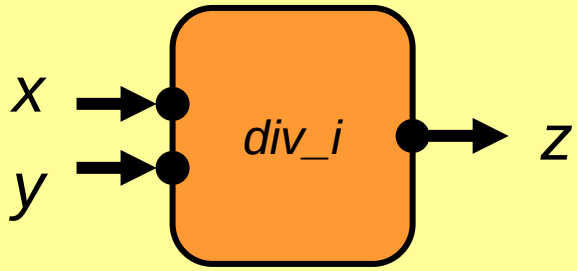
true

Process



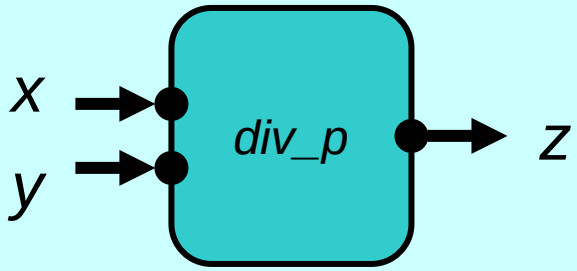
$y \neq 0$) $z = x/y$

Interface



$y \neq 0$ true

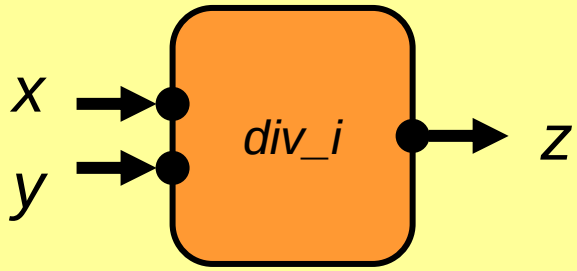
Process



$y \neq 0$) $z = x/y$

-in any environment
-e.g. executable

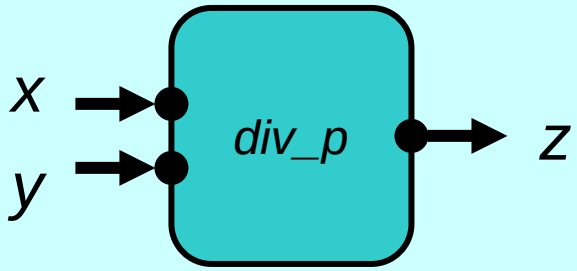
Interface



$y \neq 0$ true

-constrains environment
-e.g. typed source

Process

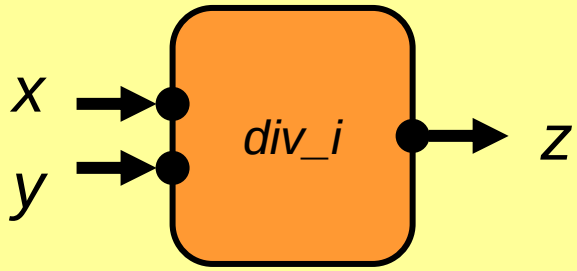


$y \neq 0 \;) z = x/y$

-in any environment
-e.g. executable

Well-formed if *(8 x,y)(9 z)*

Interface

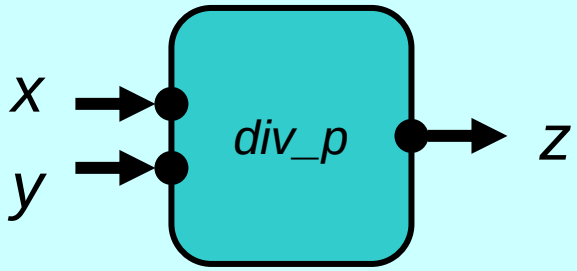


$y \neq 0$ true

-constrains environment
-e.g. typed source

Well-formed if *(9 x,y)(9 z)*

Process

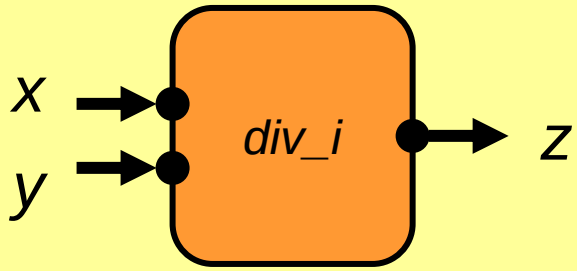


$y \neq 0 \;) \; z \neq 0$

-in any environment
-e.g. executable

Well-formed if *(8 x,y)(9 z)*

Interface



$y \neq 0$ $\qquad$ $z \neq 0$

-constrains environment
-e.g. typed source

Well-formed if *(9 x,y)(9 z)*

Stateless Assume/Guarantee Interfaces

$f \cdot g$ if

1. $I_f \mu I_g$
2. $O^+_f \mu O^+_g$
3. $O_f \P O_g$
4. $in_f (in_g$
5. $out_f) out_g$

Stateless Assume/Guarantee Interfaces

$f \cdot g$ if

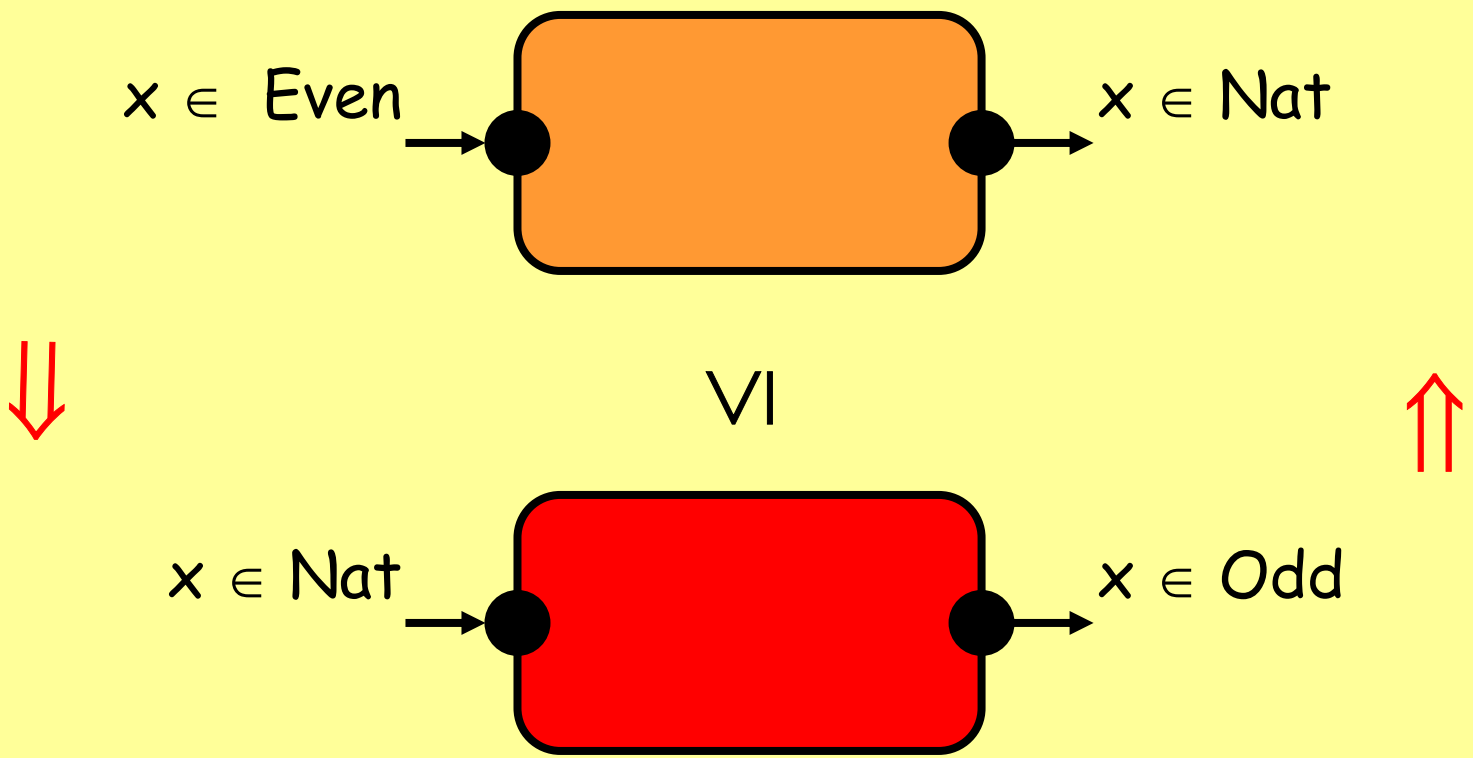
1. $I_f \mu I_g$
2. $O^+_f \mu O^+_g$
3. $O_f \P O_g$
4. $in_f (in_g$
5. $out_f) out_g$

Stateless Input/Output Processes

$f \cdot g$ if

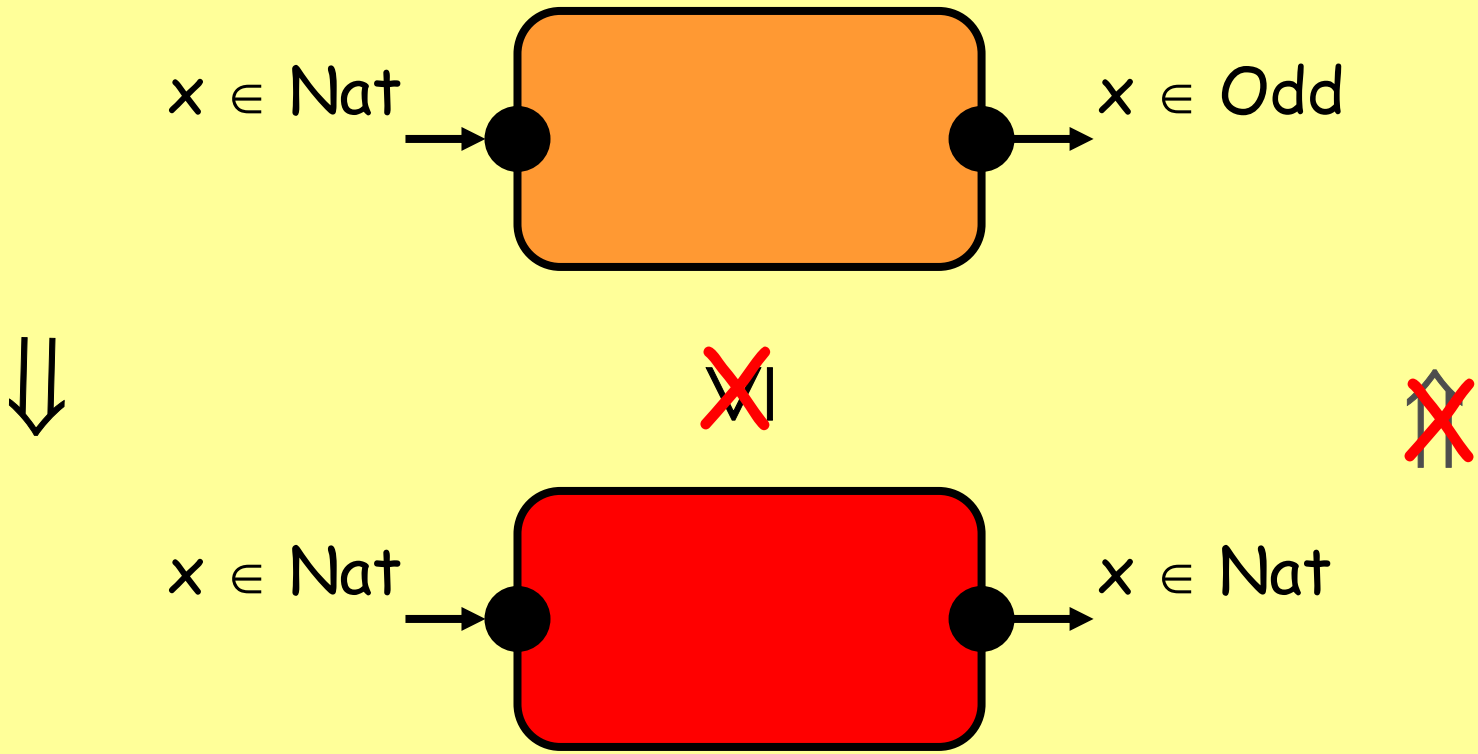
1. $I_f \P I_g$
2. $O_f \P O_g$
3. $p_f) p_g$

Assume/Guarantee Interface Refinement



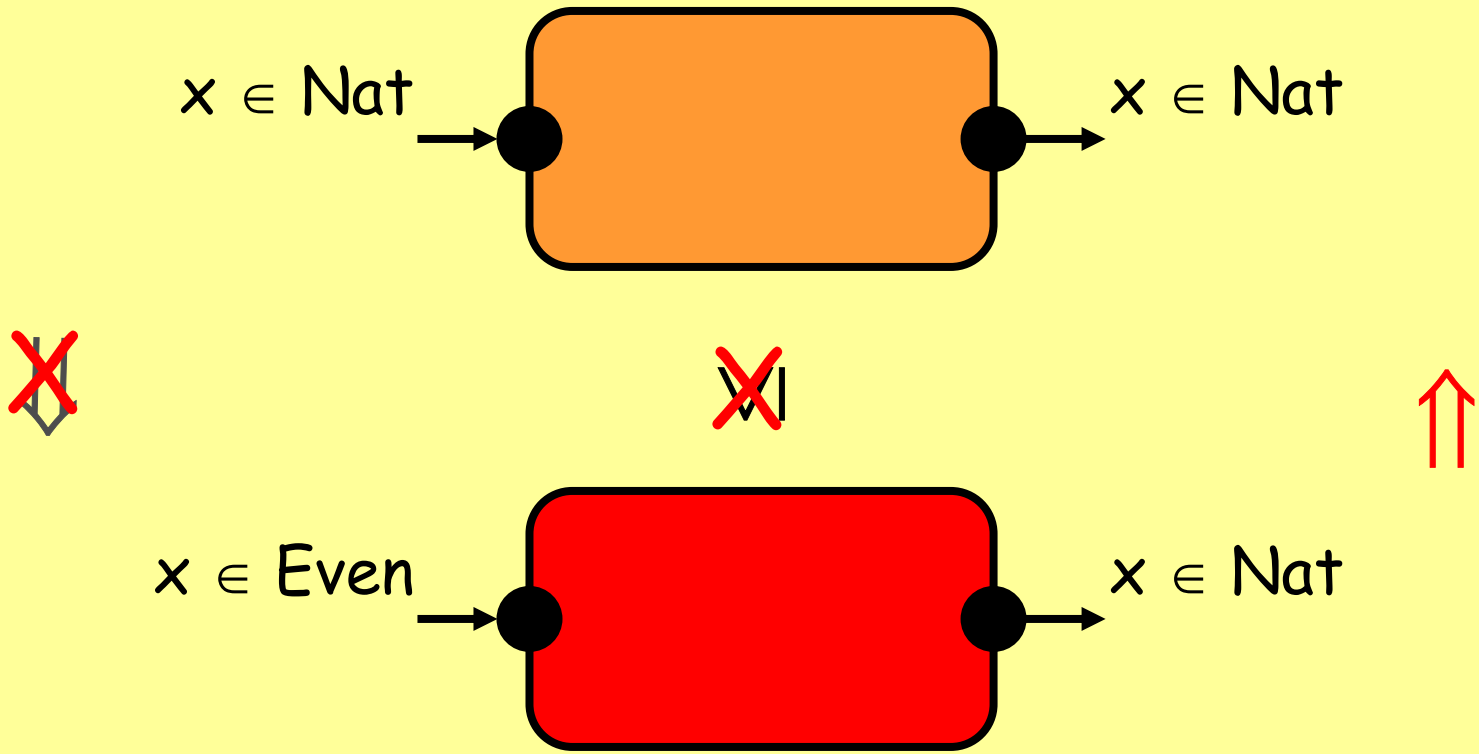
Like subtyping, interface refinement is I/O contravariant.

Assume/Guarantee Interface Refinement



1. Implementation must obey output guarantee.

Assume/Guarantee Interface Refinement



2. Implementation must accept all permissible inputs.

$f\theta$ defined:

$(8\ I_{f\theta})(9\ O_{f\theta})(p_f \text{ Æ } p_\theta)$

Show:

if $f \cdot g$ and $f\theta$ defined, then $g\theta$ defined.

Use:

$f \cdot g$ if p_f) p_g

$g\theta$ defined:

$(9\ I_{g\theta})(8\ O_{g\theta})((out_g \text{ Æ } p_\theta)\ )\ in_g)$

Show:

if g ¸ f and $g\theta$ defined,
then $f\theta$ defined.

Use:

g ¸ f if $in_g\ )\ in_f$

$out_g\ (\ out_f$

$f\theta$ defined:

$(8\ I_{f\theta})(9\ O_{f\theta})(p_f \text{ Æ } p_\theta)$

Show:

if $f \cdot g$ and $f\theta$ defined,
then $g\theta$ defined.

Use:

$f \cdot g$ if $p_f\ )\ p_g$

Lesson 2:

If you want independent implementability, refinement *must* be I/O contravariant.

From Stateless Assume/Guarantee Interfaces

-Type interfaces

-Assertional interfaces

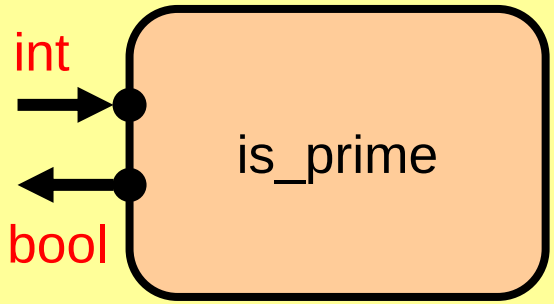
To Stateful Assume/Guarantee Interfaces

-Interface automata

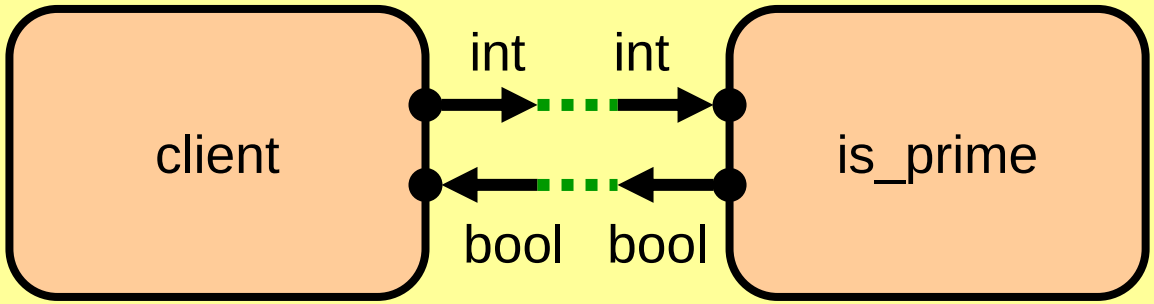
A Type Interface

This interface constrains the client's data.

E.g. typed programming languages.



Type Interface Compatibility

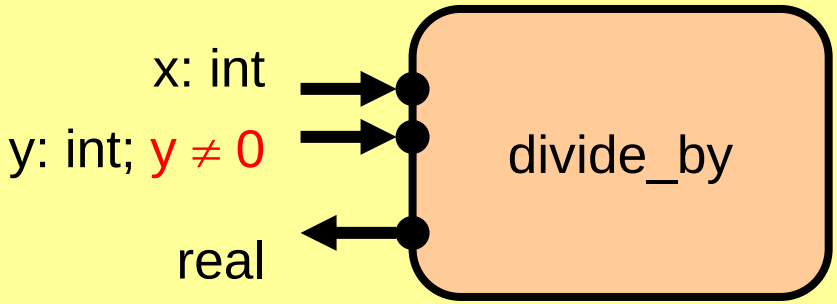


Type checking.

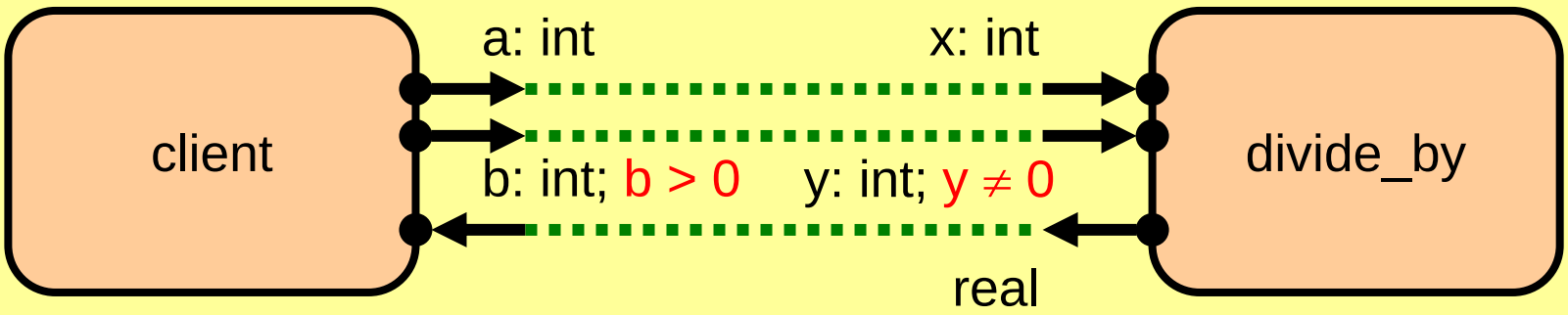
An Assertional Interface

This interface still constrains the client's data.

E.g. pre- and postconditions.



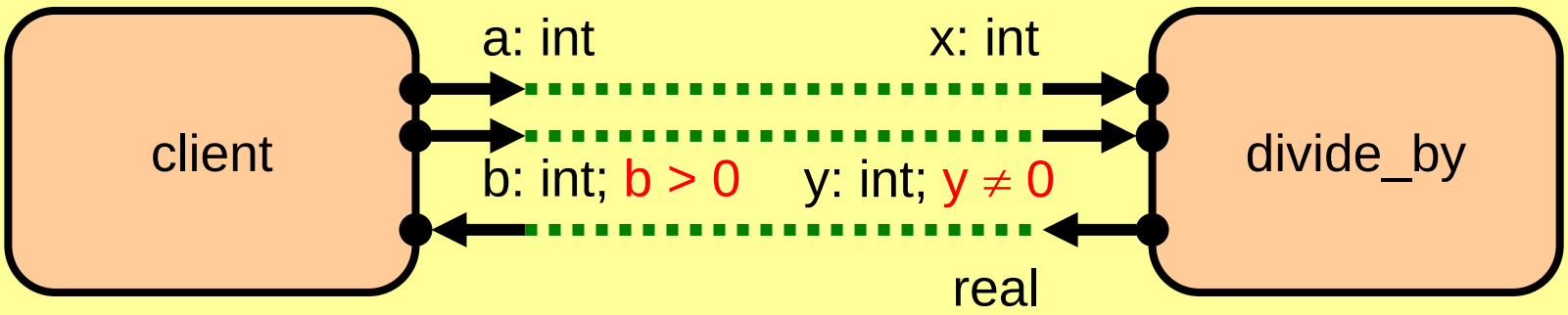
Assertional Interface Compatibility



(8 b,y) ((b > 0 Æ y = b)) y ≠ 0)

Extended static checking.

Assertional Interface Compatibility



(8 b,y) ((b > 0 Æ y = b)) y ≠ 0)

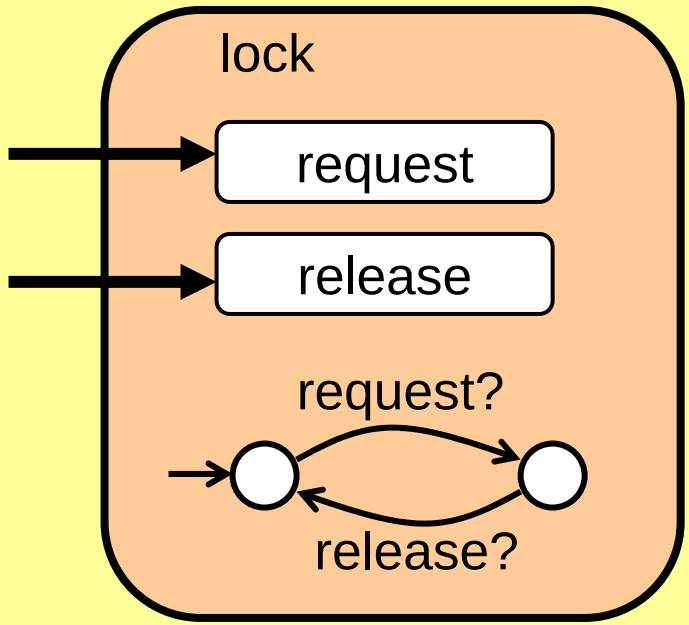
Extended static checking.

Preconditions are assumptions on the input.
Postconditions are guarantees on the output.

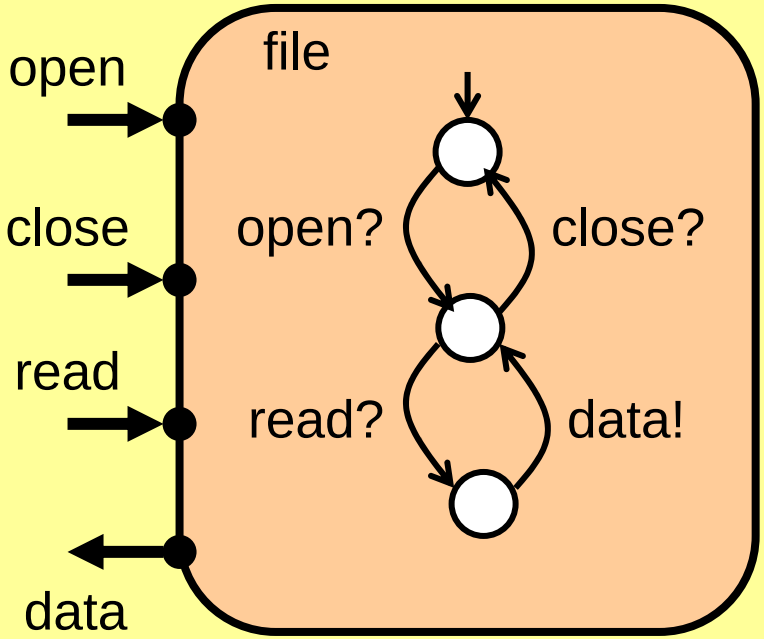
An Automaton Interface

This interface constrains the client's control.

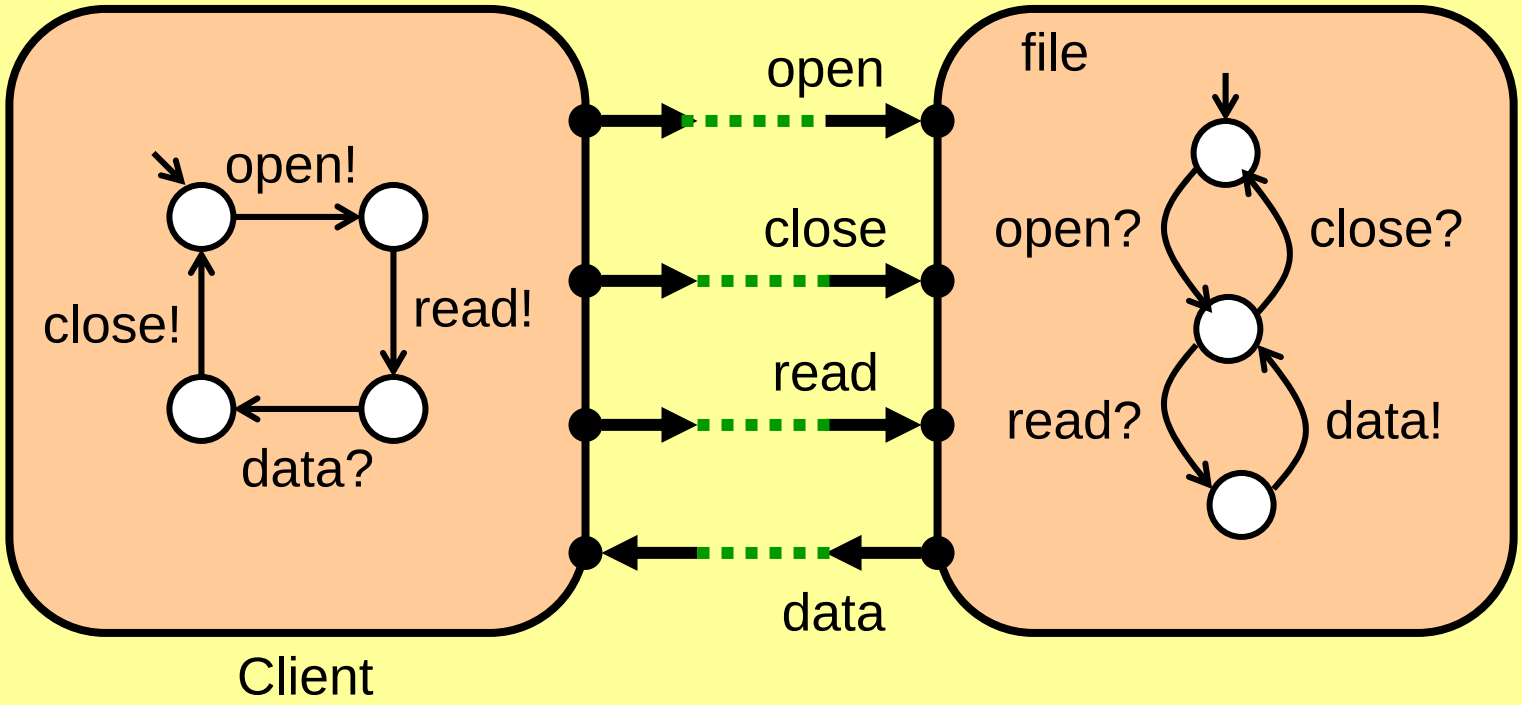
E.g. type-state checking.



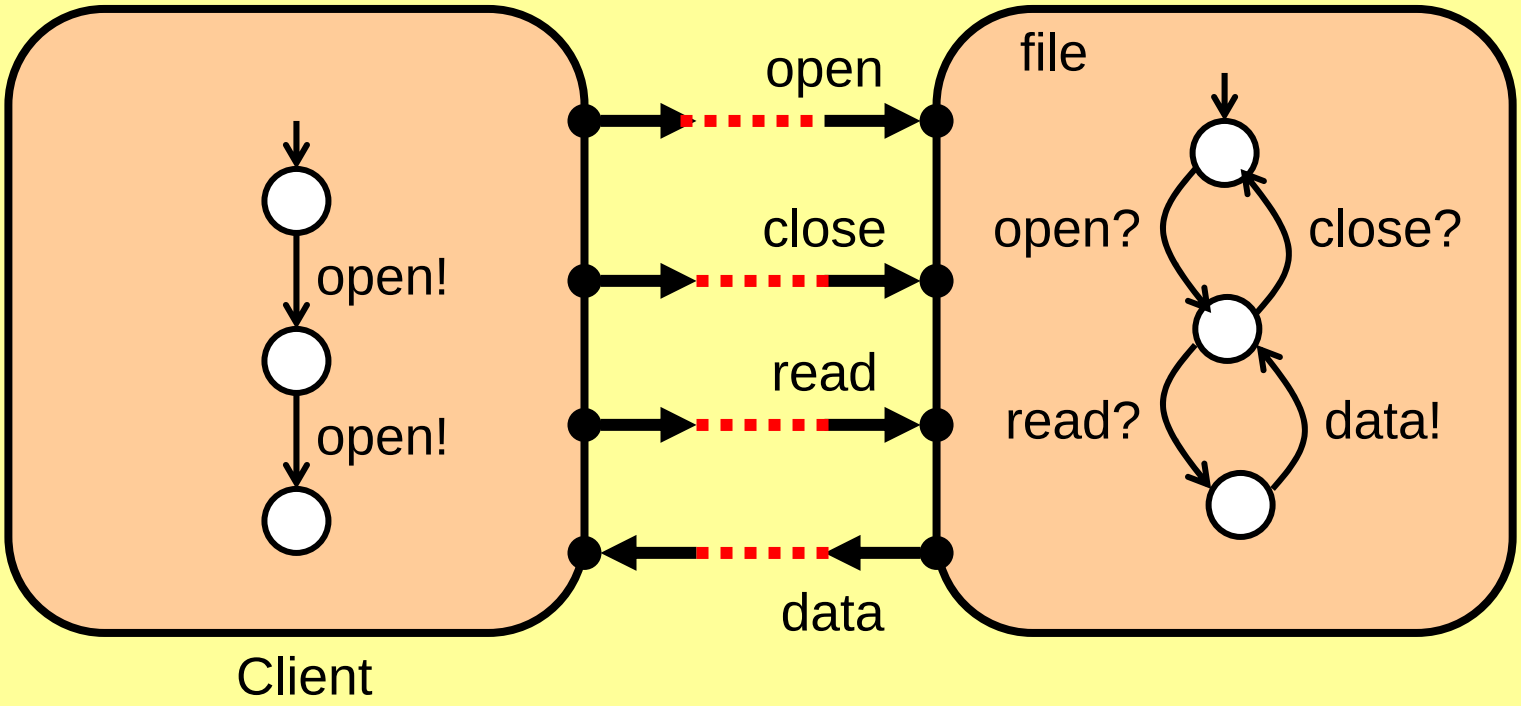
Another Automaton Interface



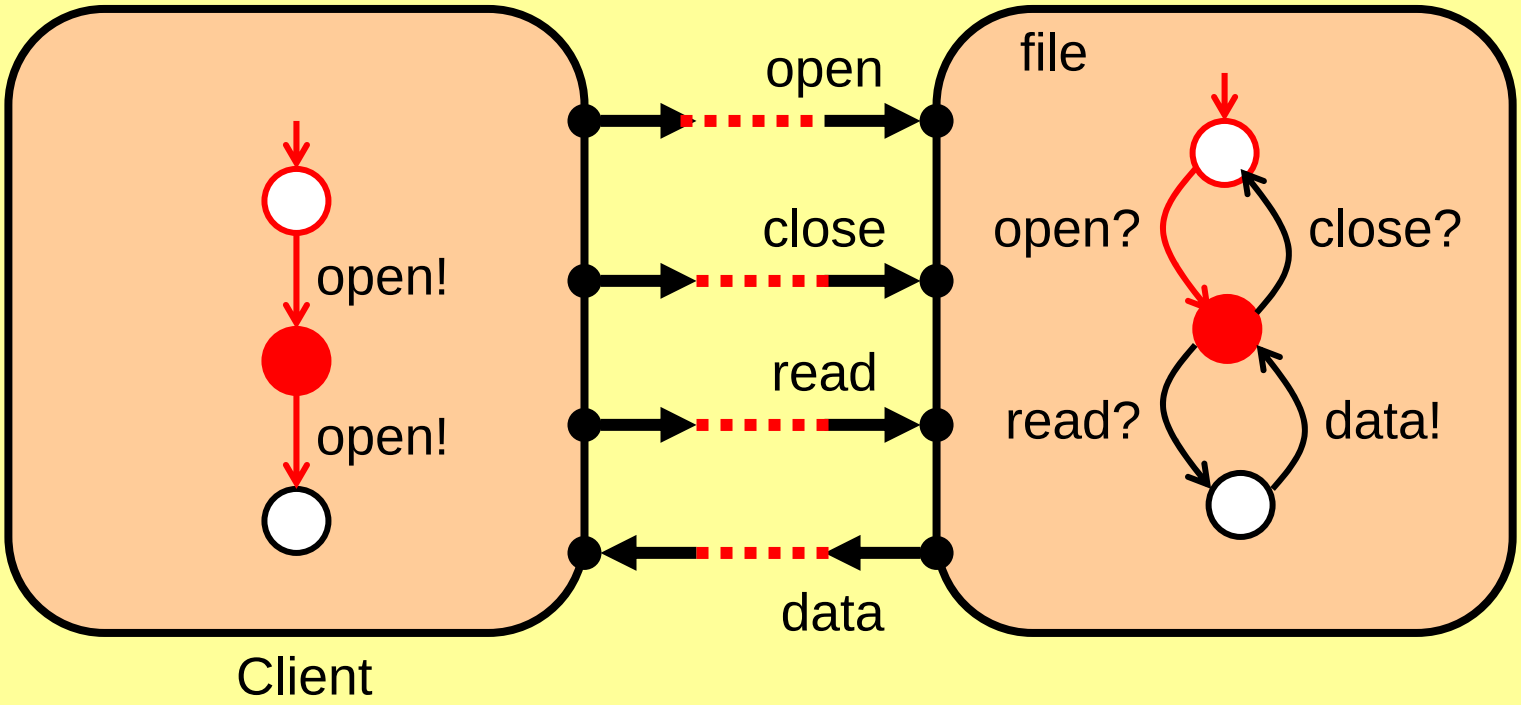
Automaton Interface Compatibility



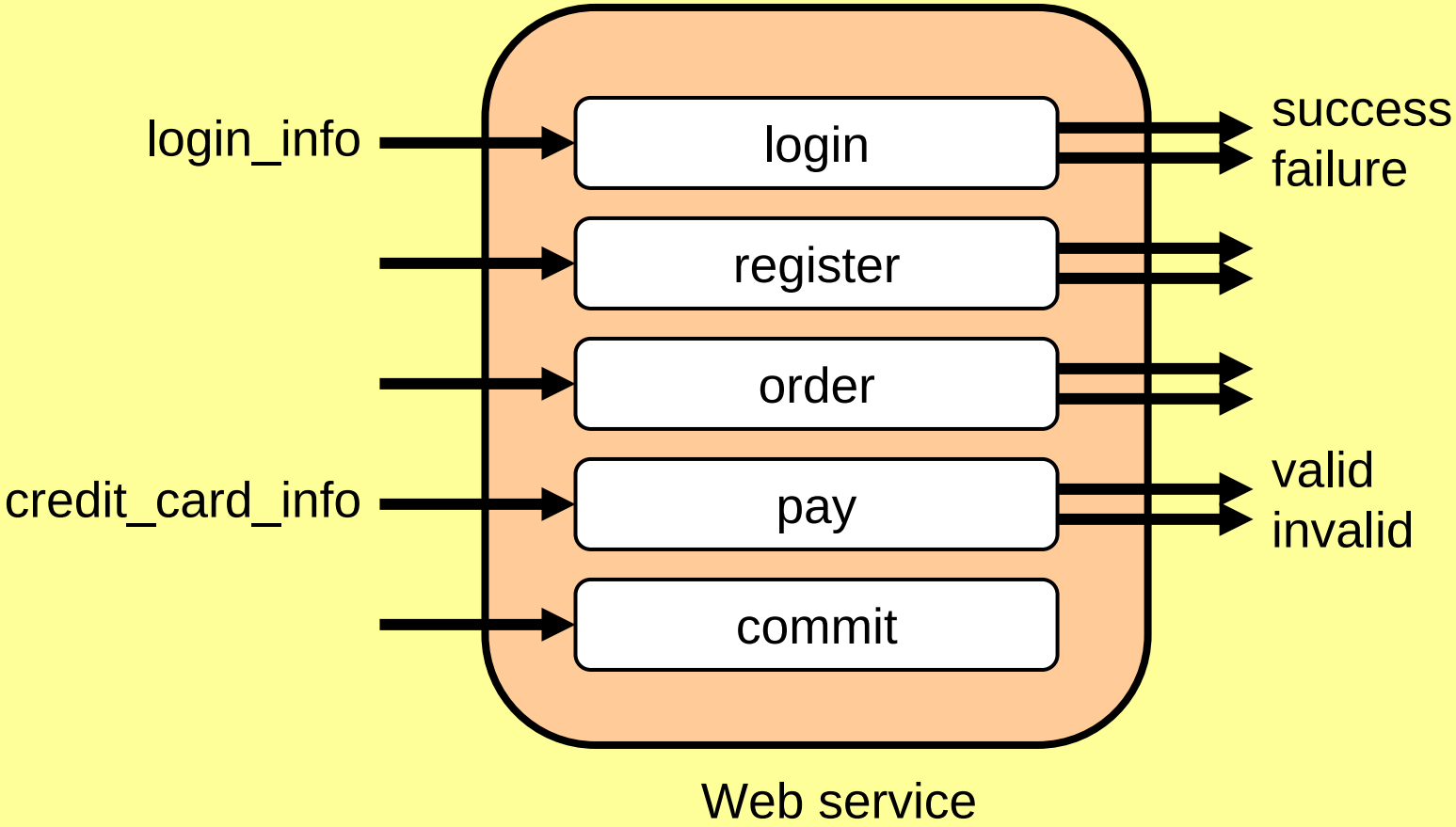
Automaton Interface Incompatibility



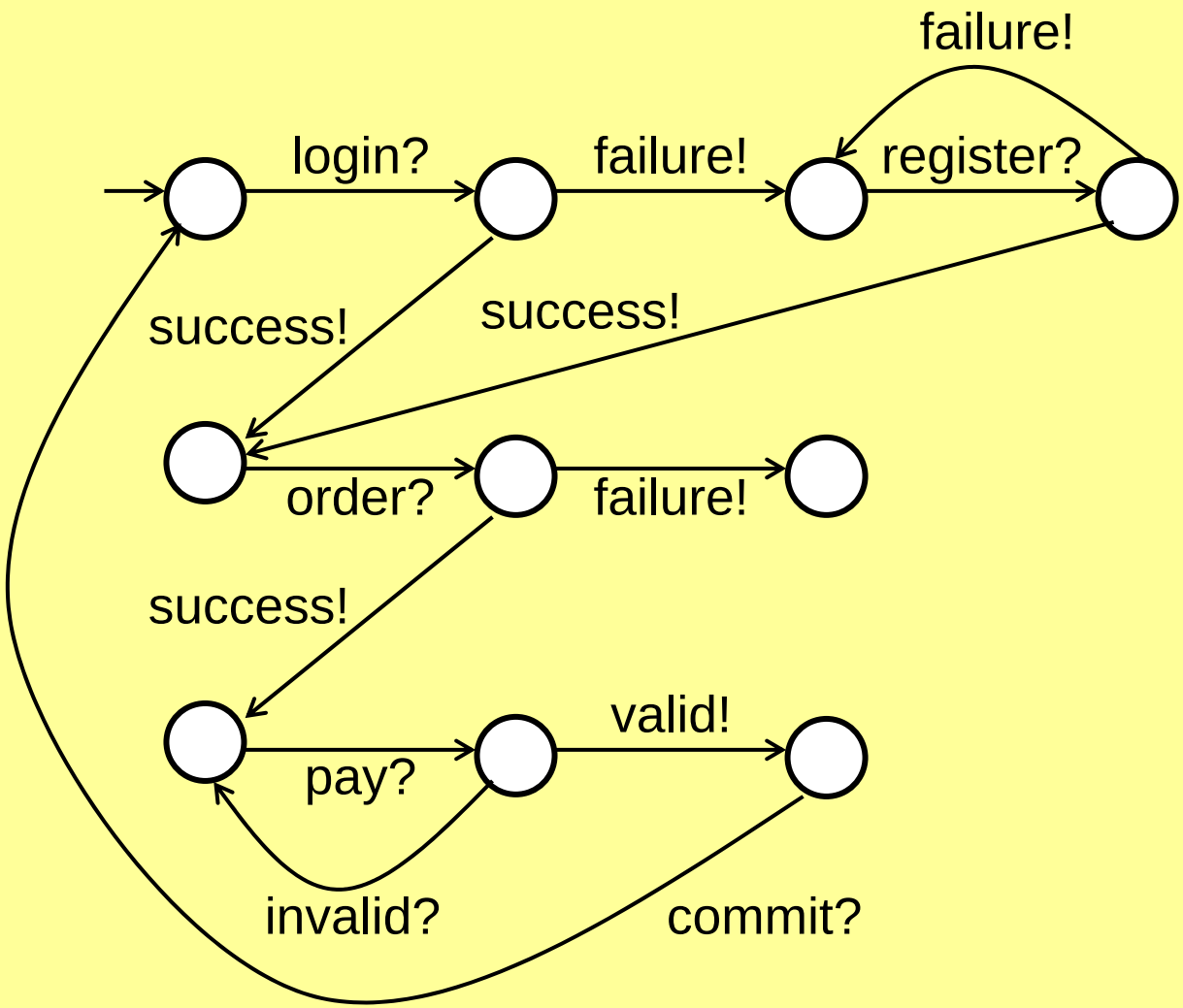
Automaton Interface Incompatibility



Another Stateless Interface Fragment



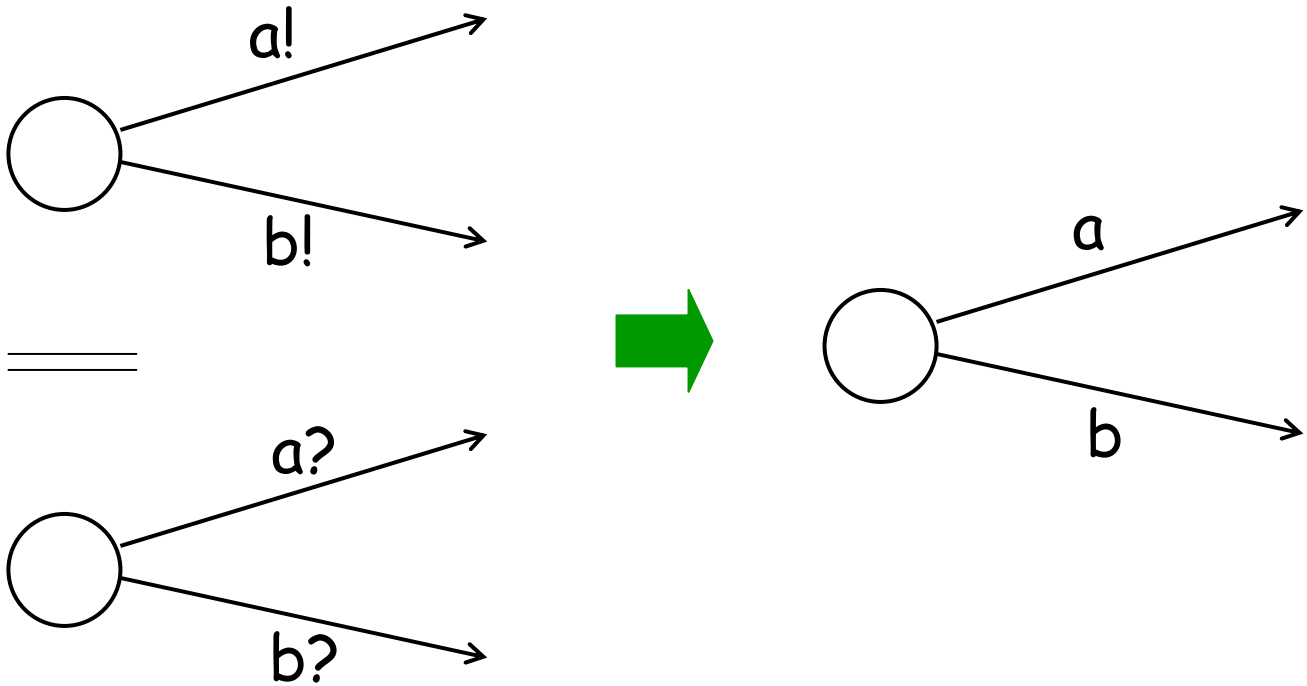
Another Stateful Interface Fragment

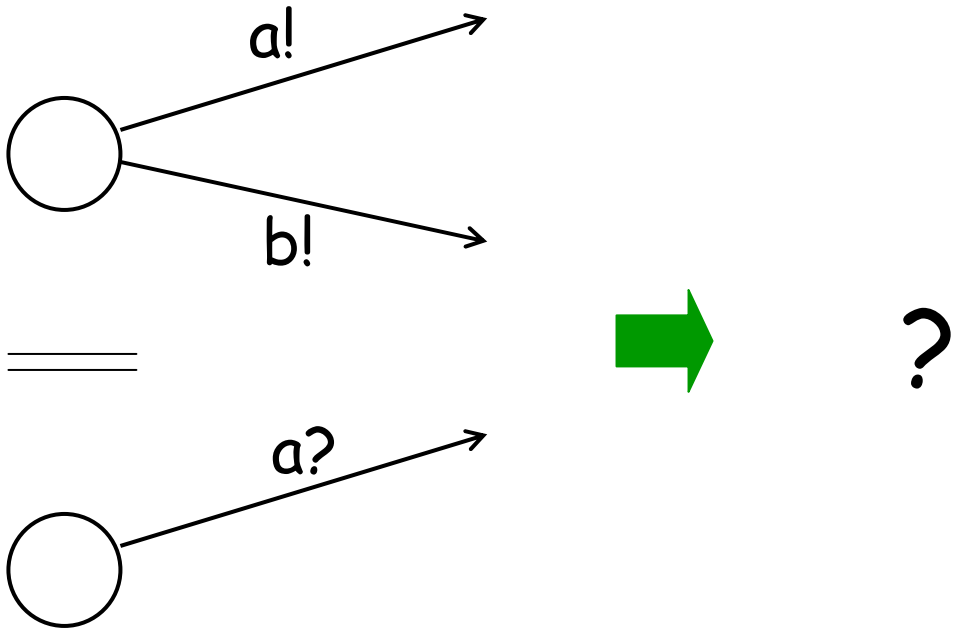


Our objective is to demonstrate:

Automaton Interface Compatibility Checking is game solving.

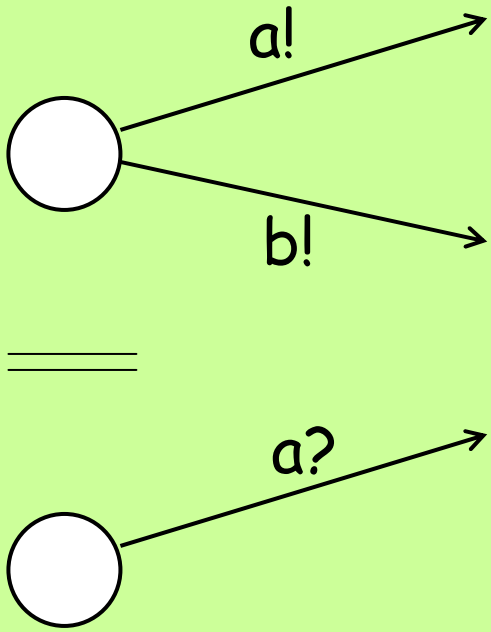
Automaton Interface Refinement (“Subinterfacing”) is alternating simulation.


a!
b!
a?
b?
a
b


a!
b!
a?
?

Alternative 1: Total Composition

“Be Prepared for Every Input”

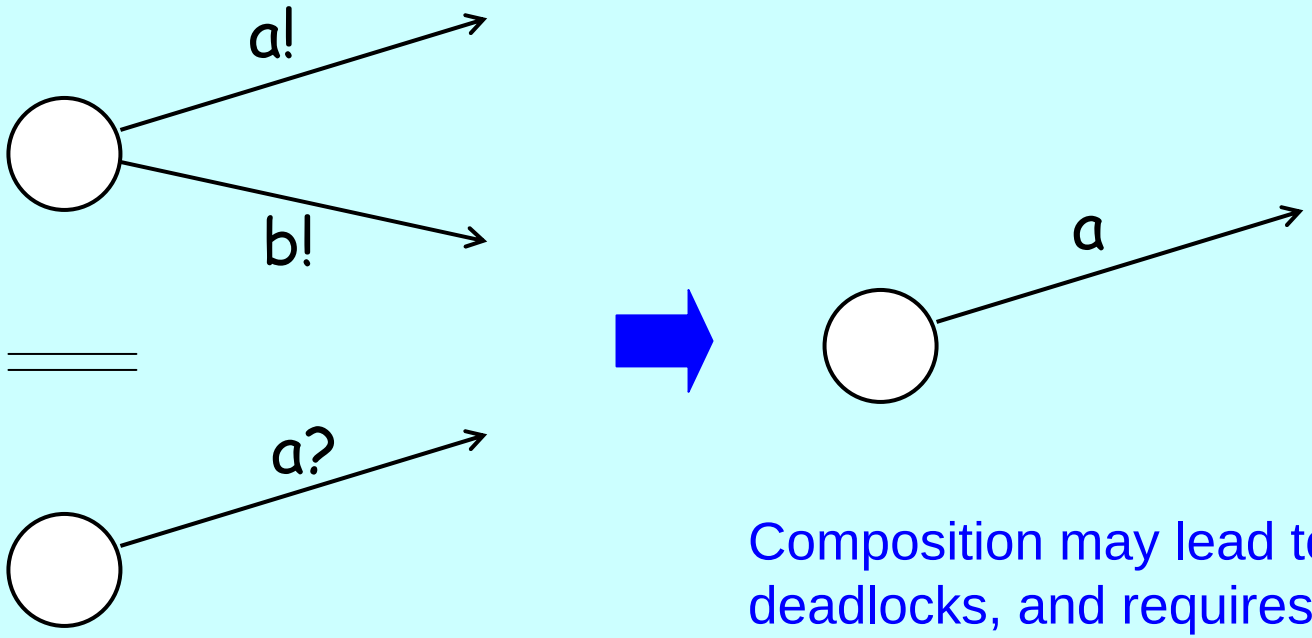


This is an illegal component description, because it is not prepared to accept input b.

I/O Automata [Lynch], TLA [Lamport]

Alternative 2: Process Algebra

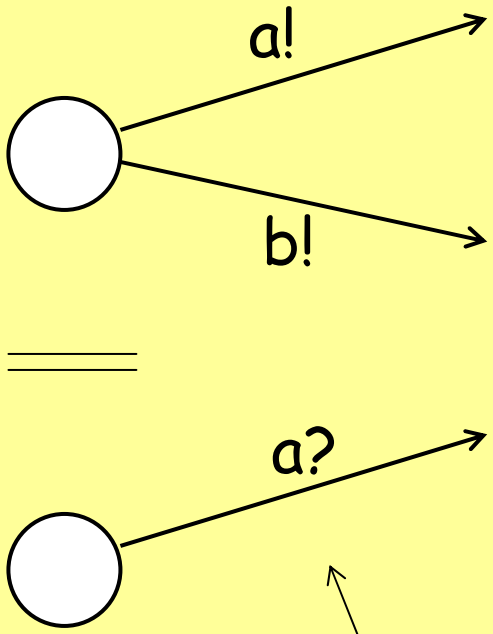
"Compose, then Check"



Composition may lead to deadlocks, and requires verification if this is undesirable.

Alternative 3: Interface Algebra

"Check, then Compose"

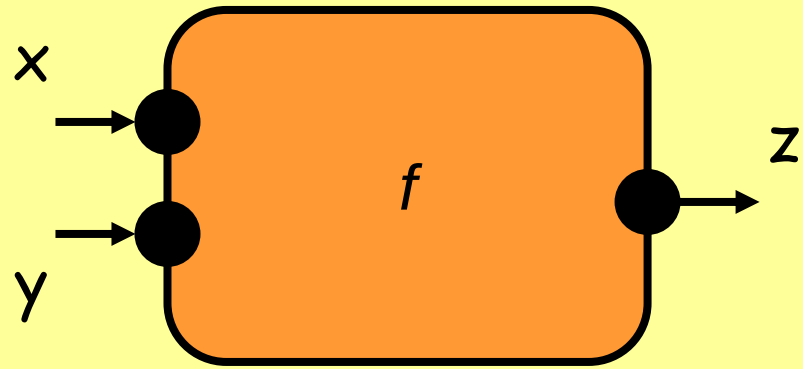


These interfaces are incompatible, because the receiver expects the environment to provide input b.

This is a constraint on the environment (an "input assumption"), not a description of the component's behavior.

Recall:

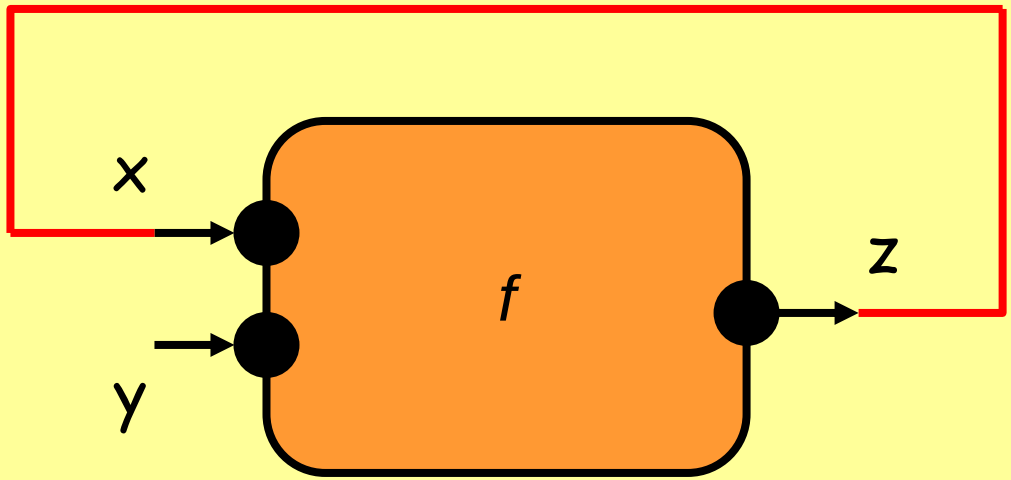
Interface Composition propagates Environment Constraints.



$x=0 \Rightarrow y=0$ true

Recall:

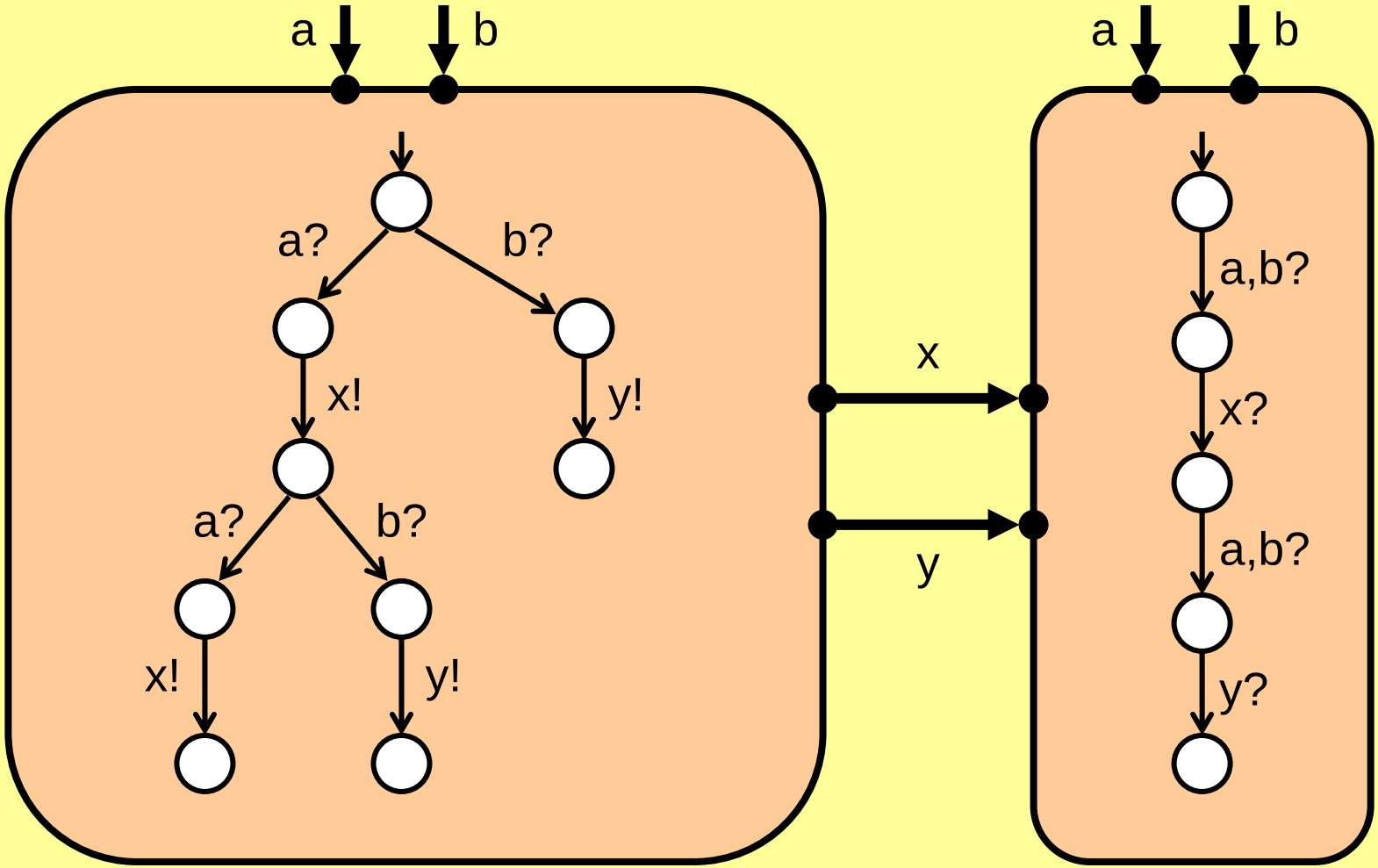
Interface Composition propagates Environment Constraints.



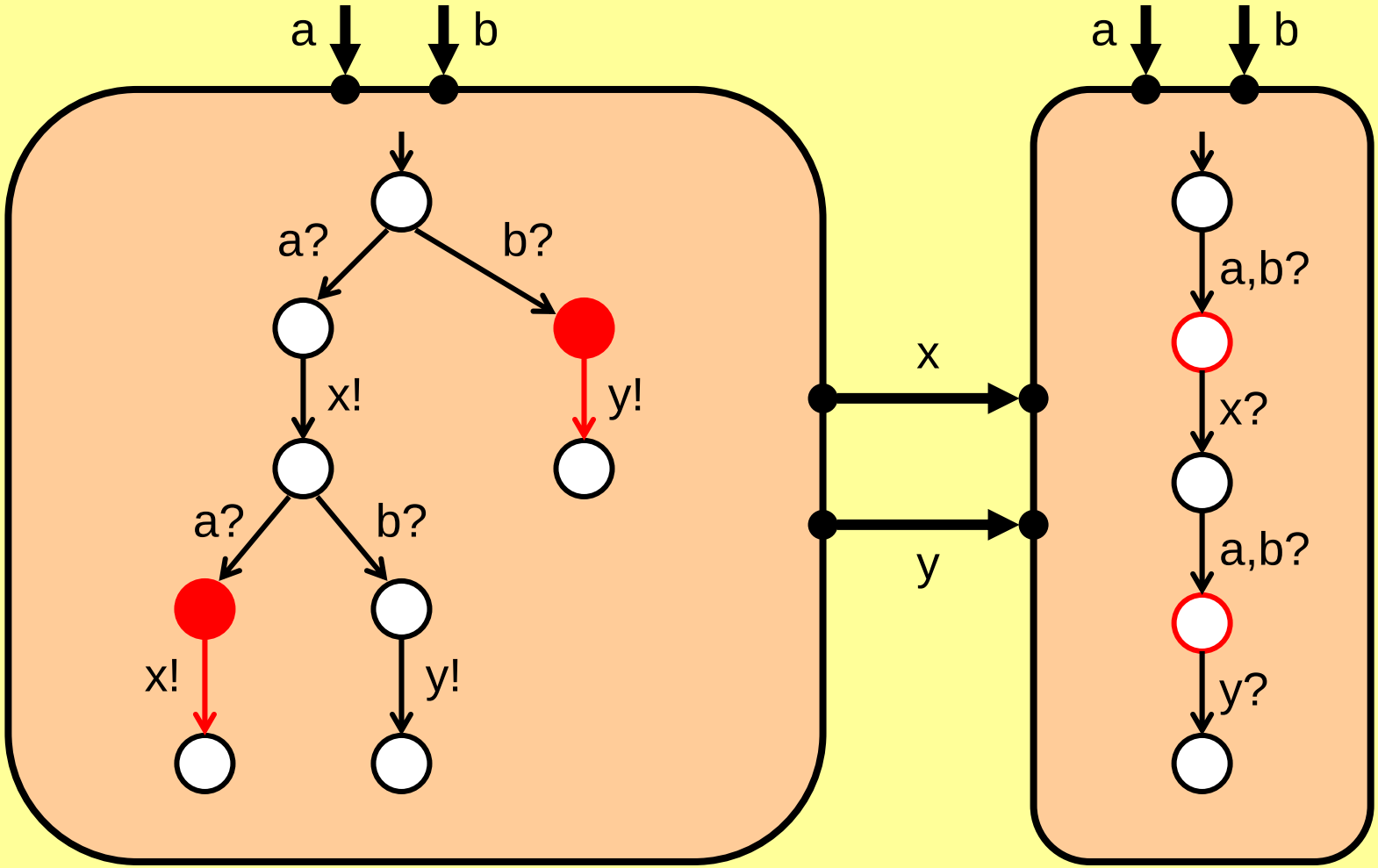
y=0

true

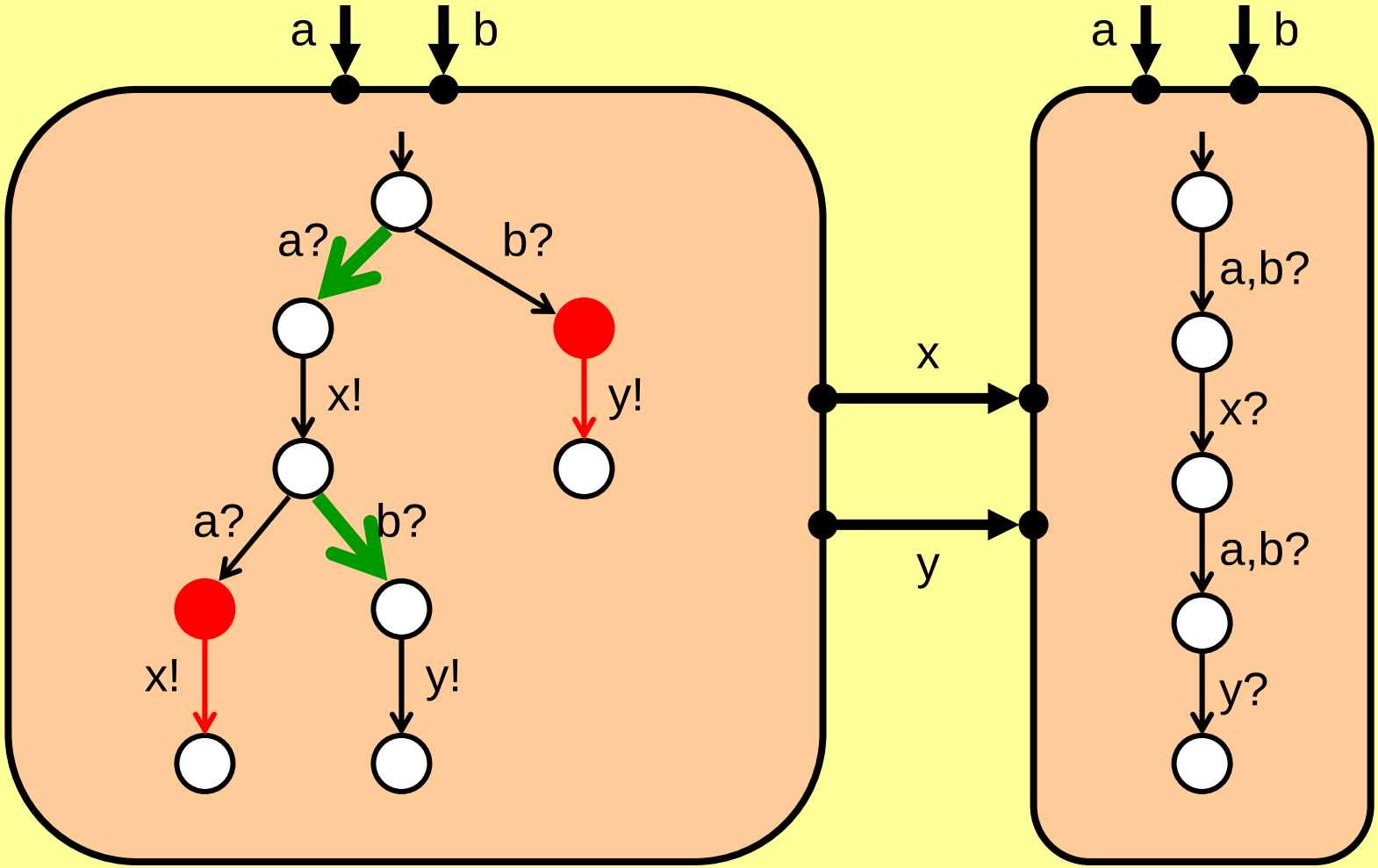
Interface Automaton Composition



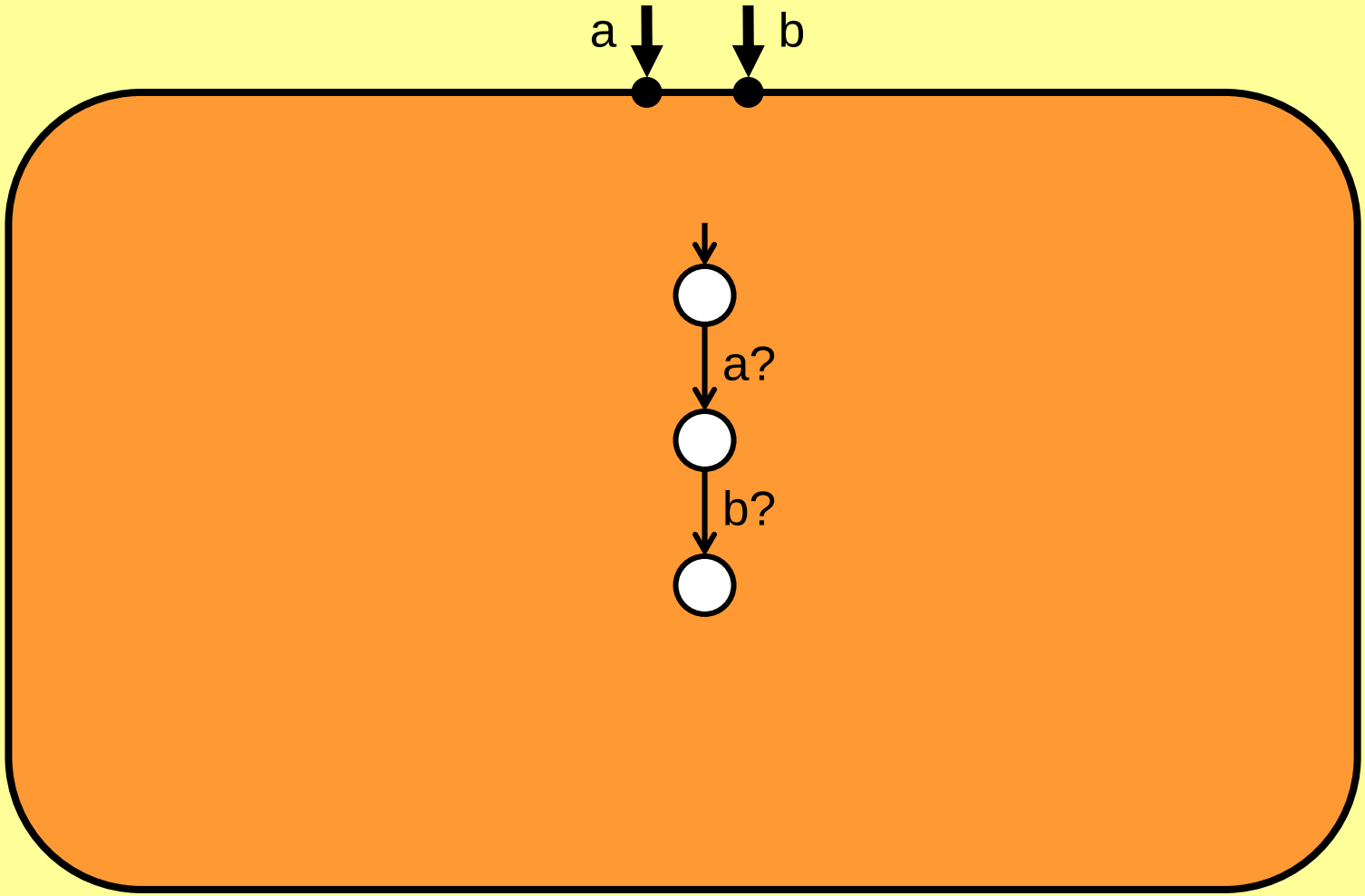
Interface Automaton Composition



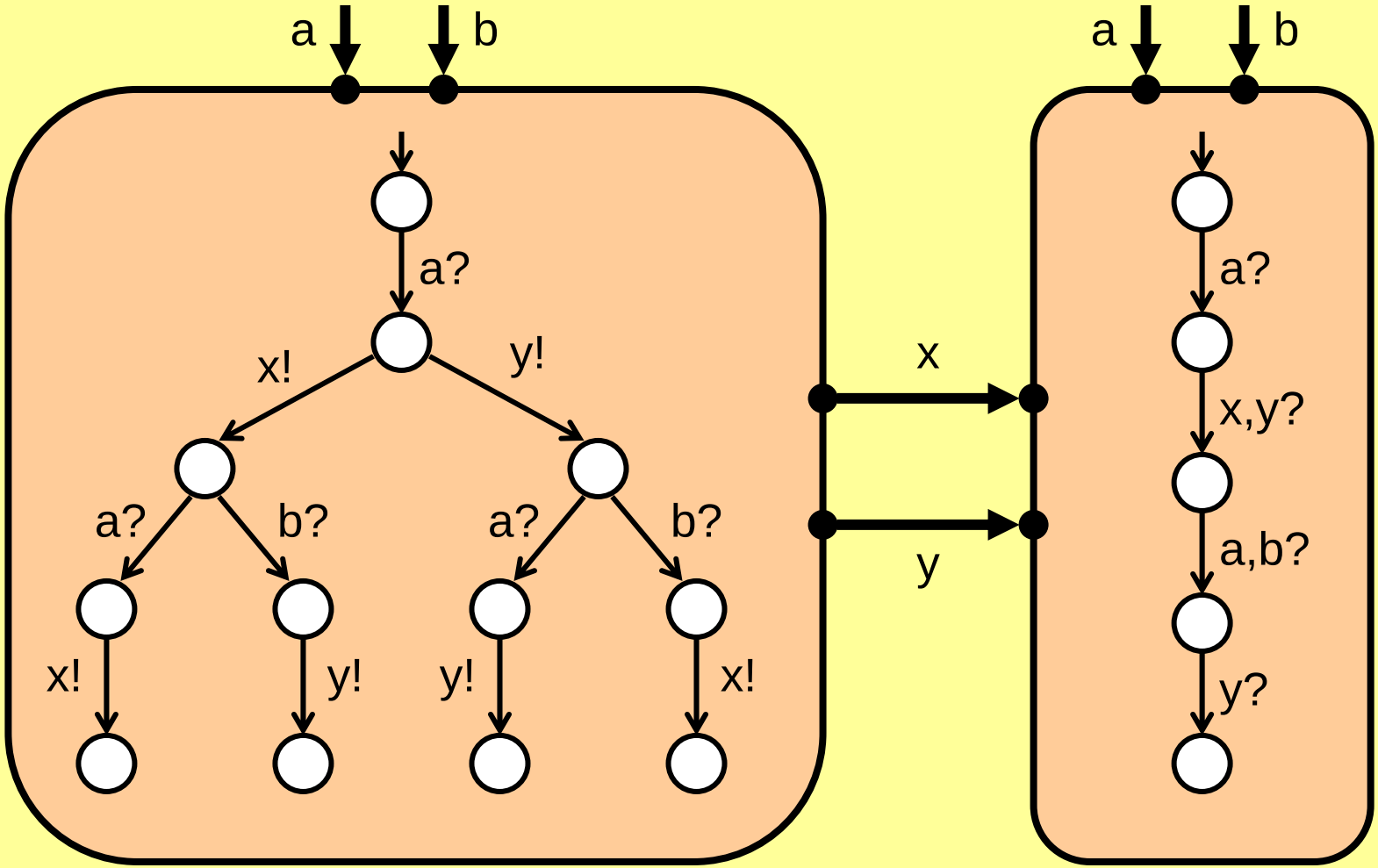
Interface Automaton Composition



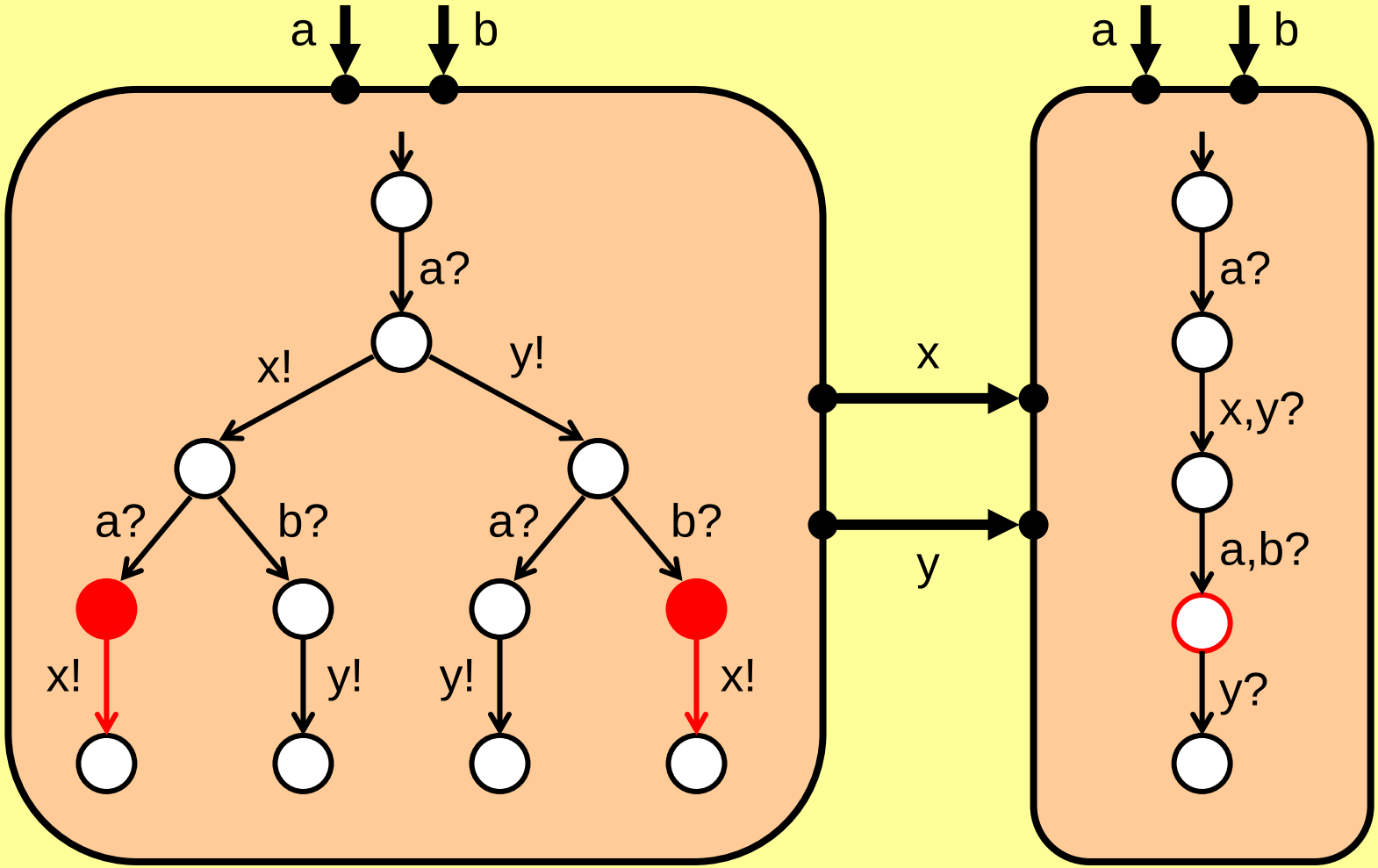
The Composite Interface



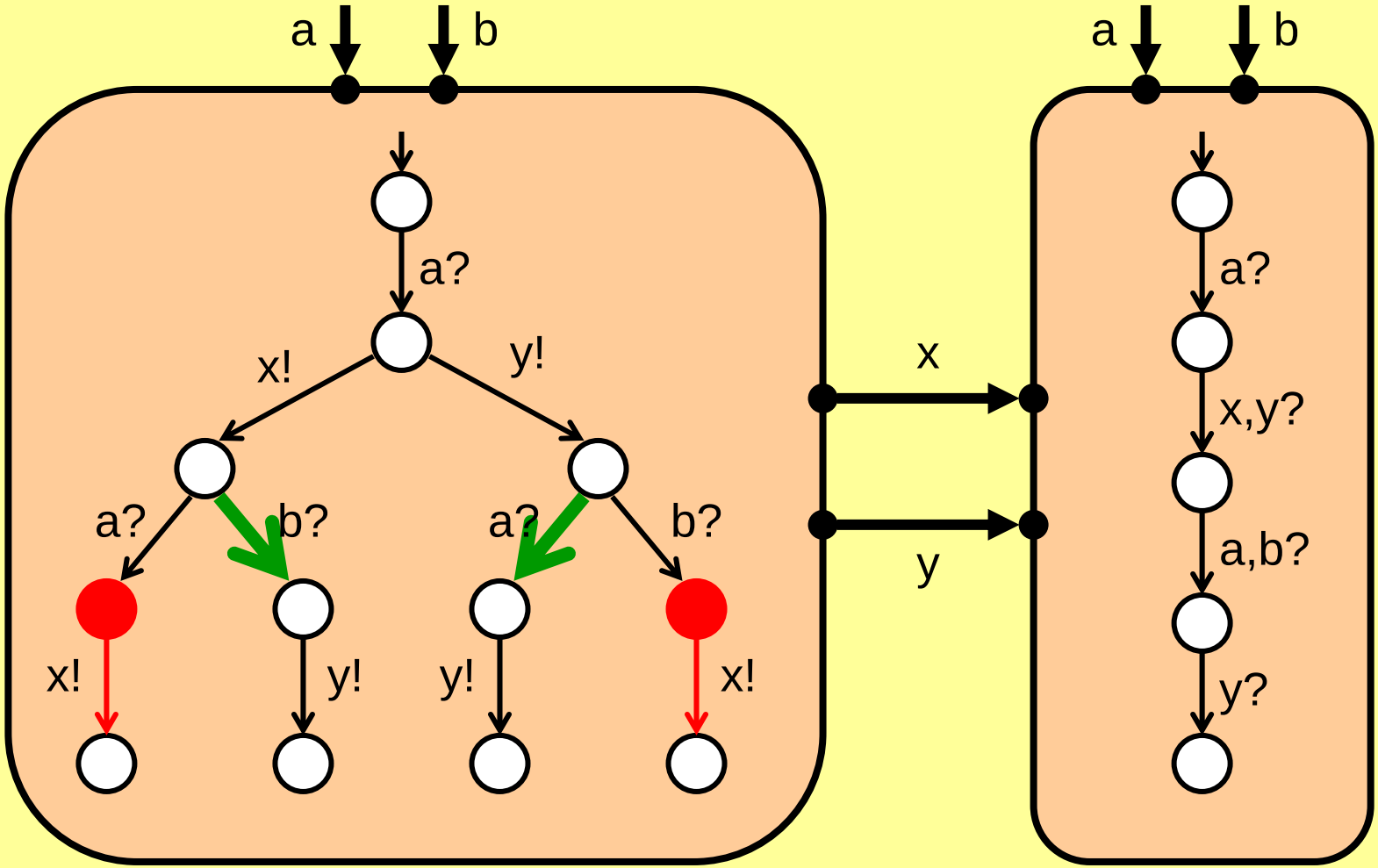
Interface Automaton Composition



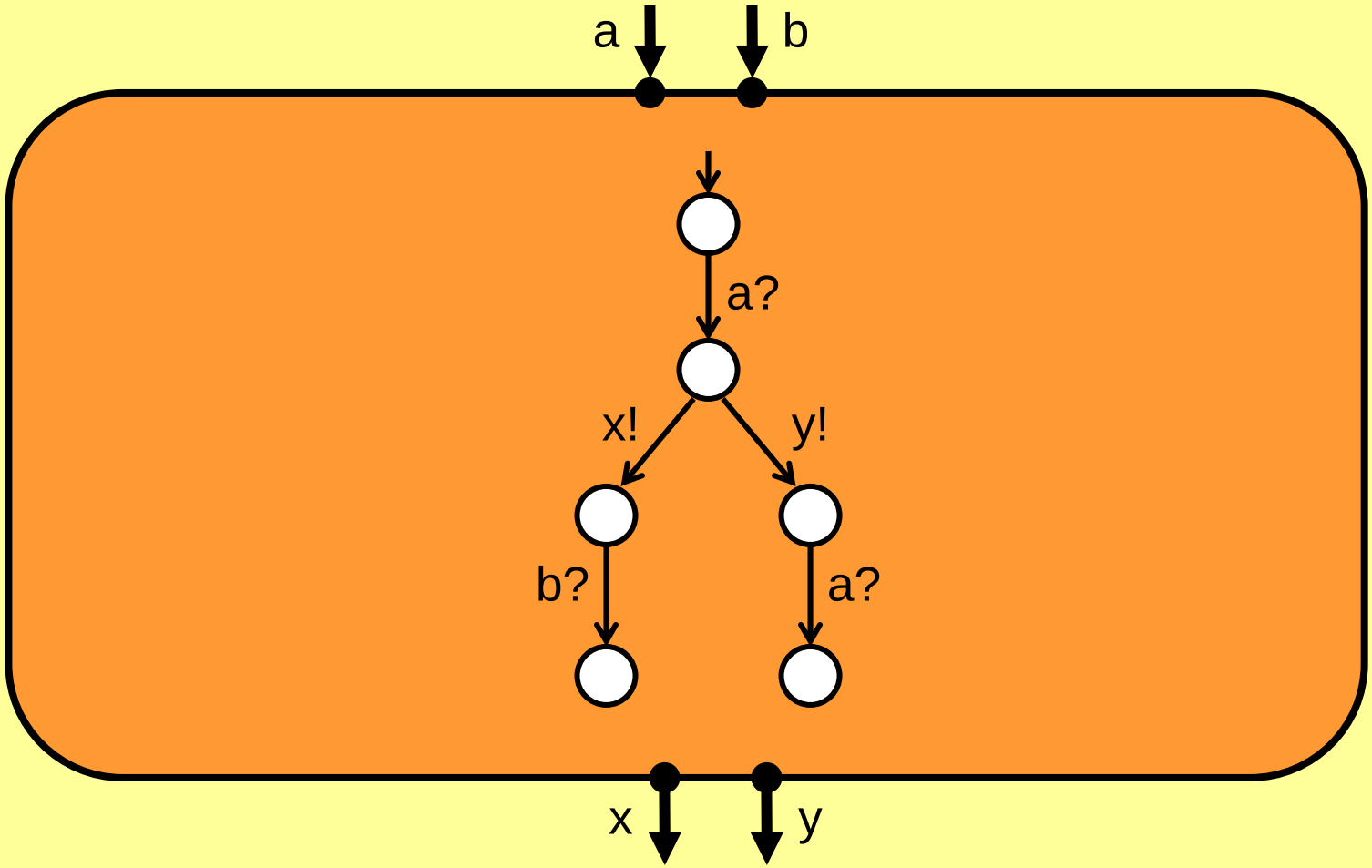
Interface Automaton Composition



Interface Automaton Composition



The Composite Interface



Lesson 3:

Stateful Interfaces are Games!

-Player Input vs. player Internal.

-The composite interface is the product restricted to those states from which player Input has a strategy to avoid incompatibilities.