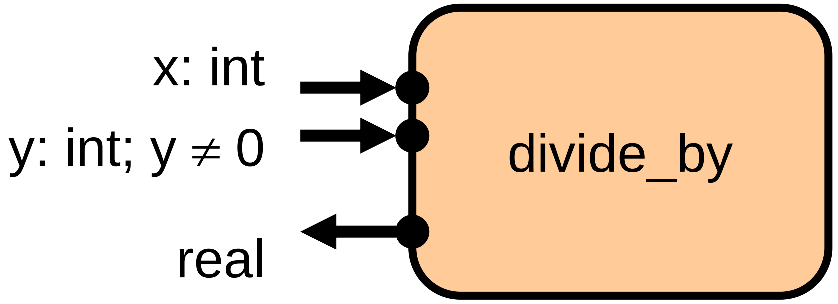
Interface-based Design 3

Tom Henzinger
EPFL and UC Berkeley

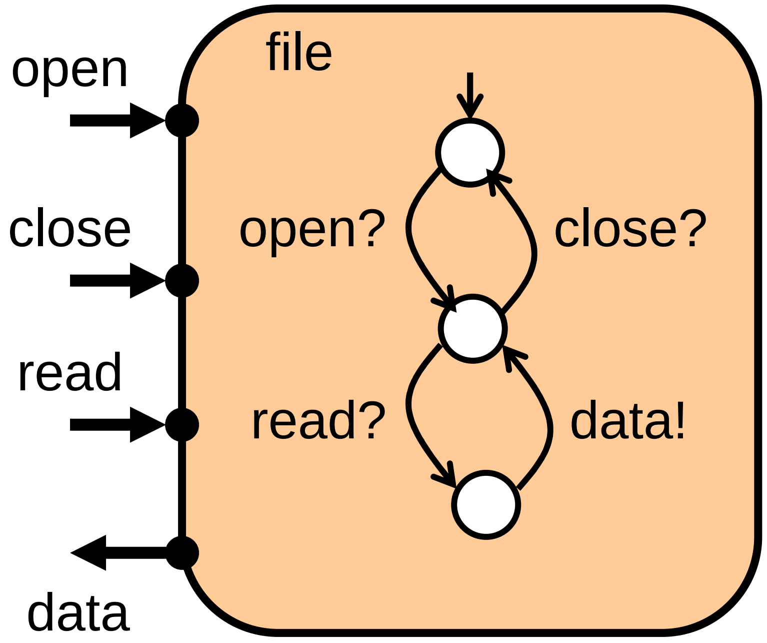
An Assertional Interface

This interface constrains the client's data.

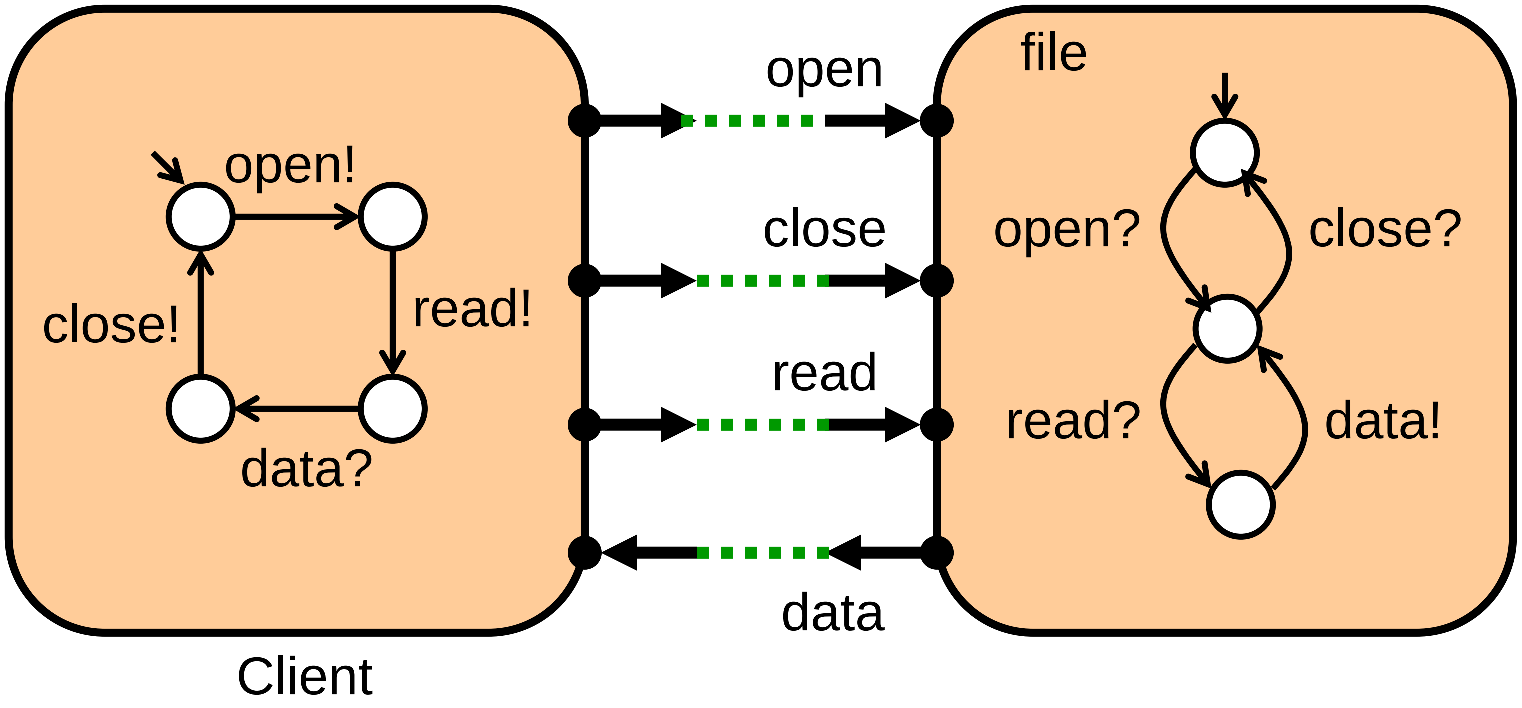


An Automaton Interface

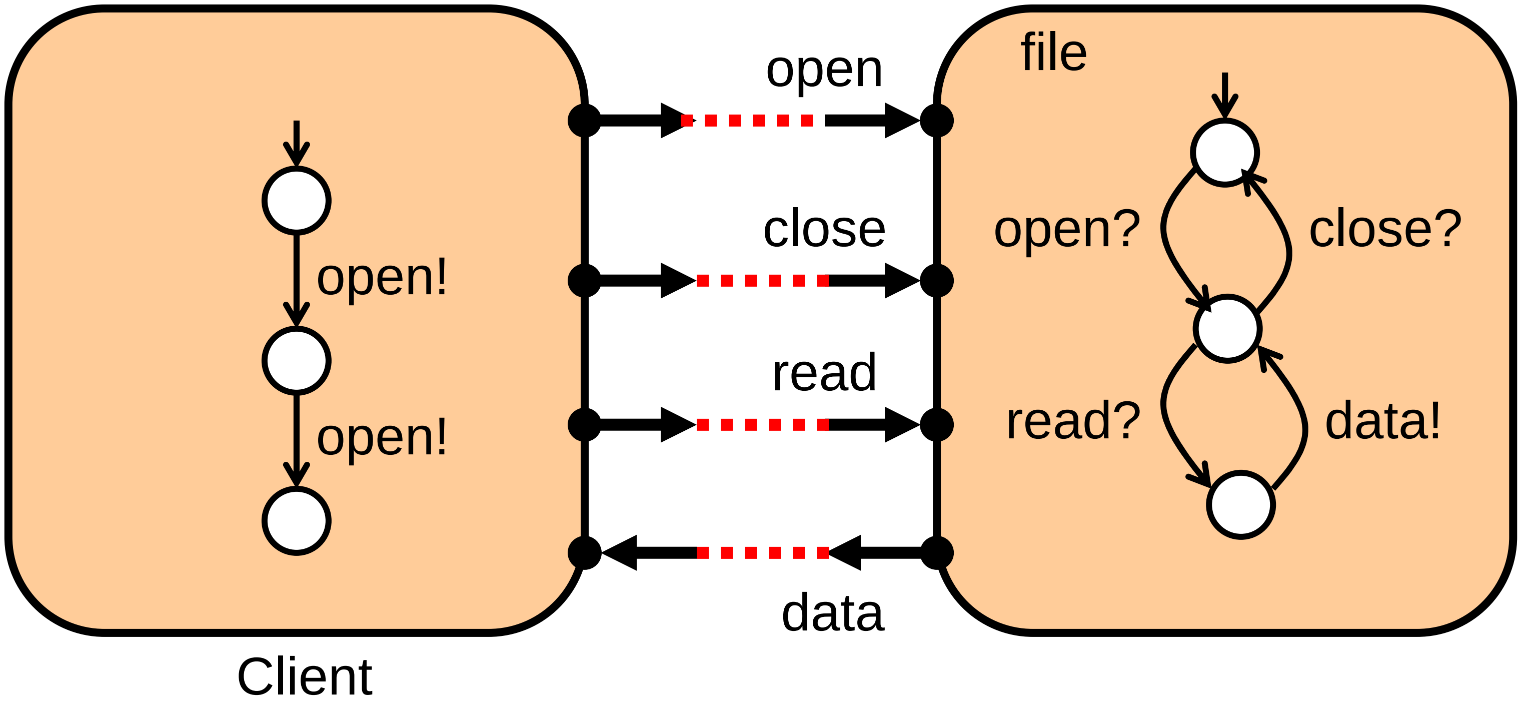
This interface constrains the client's control.



Two Compatible Automaton Interfaces



Two Incompatible Automaton Interfaces



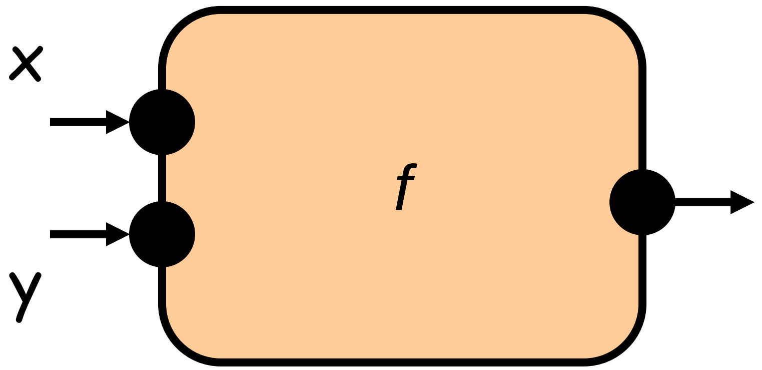
Today's Lecture:

How do we check the compatibility of automaton interfaces ?

What is the composition of two compatible automaton interfaces ?

Free Inputs:

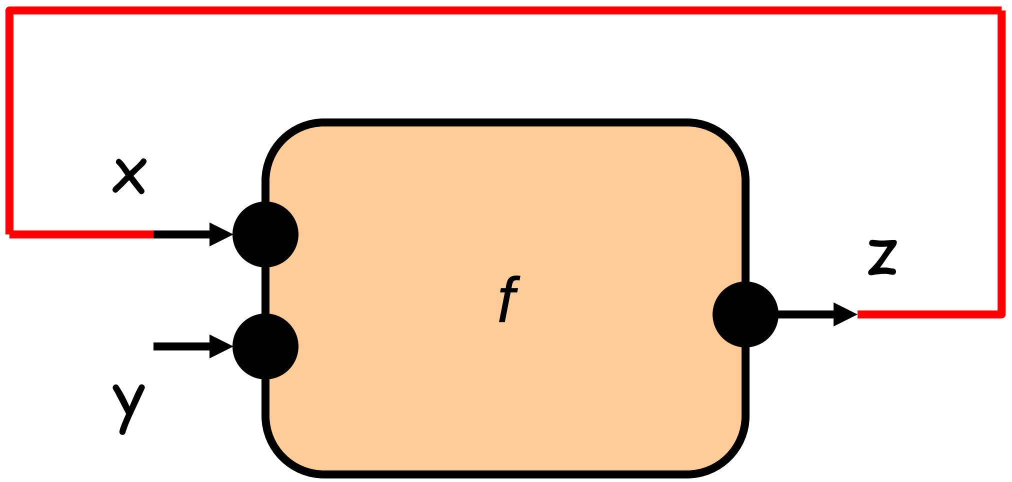
Interface Composition propagates Environment Constraints.



$x=0 \Rightarrow y=0$ true

Free Inputs:

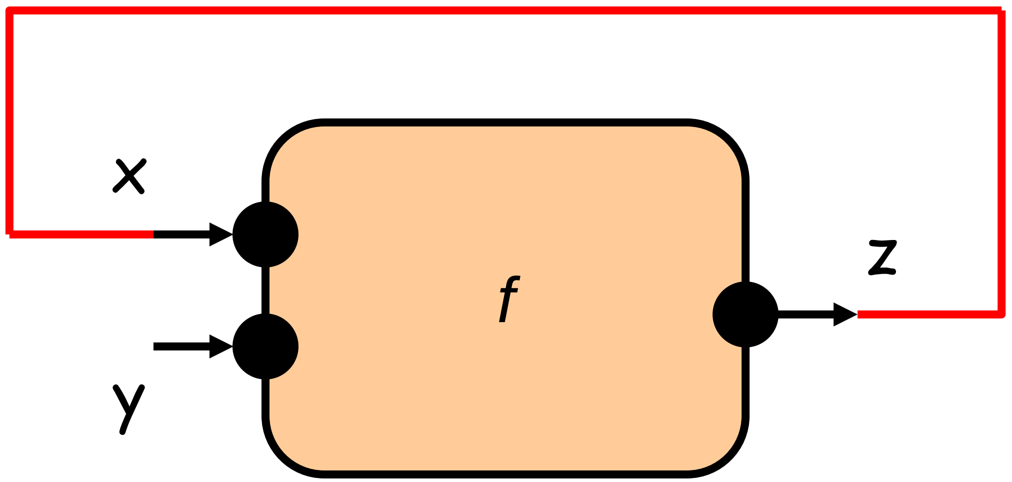
Interface Composition propagates Environment Constraints.



$x=0 \Rightarrow y=0$ true

Free Inputs:

Interface Composition propagates Environment Constraints.

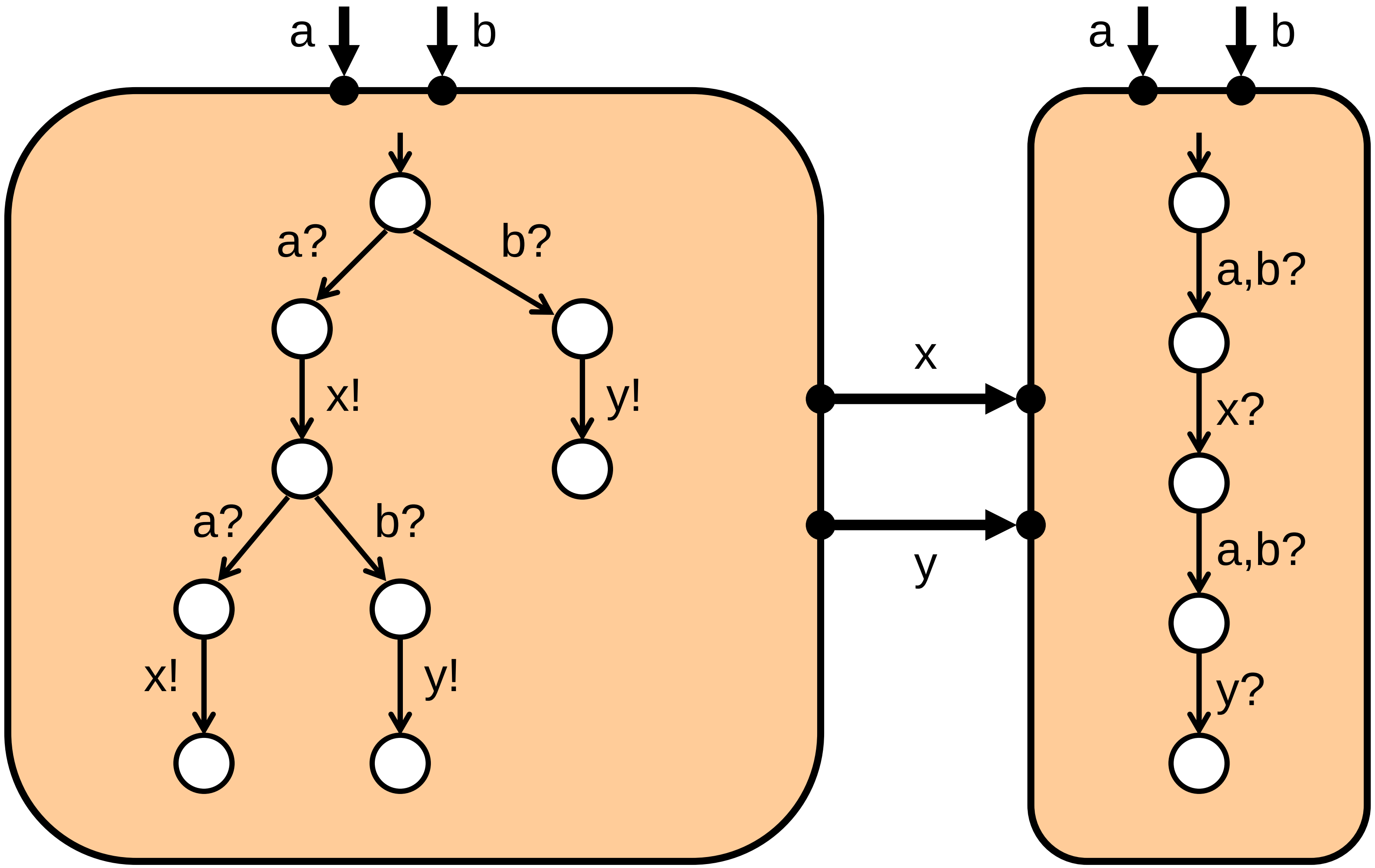


y=0

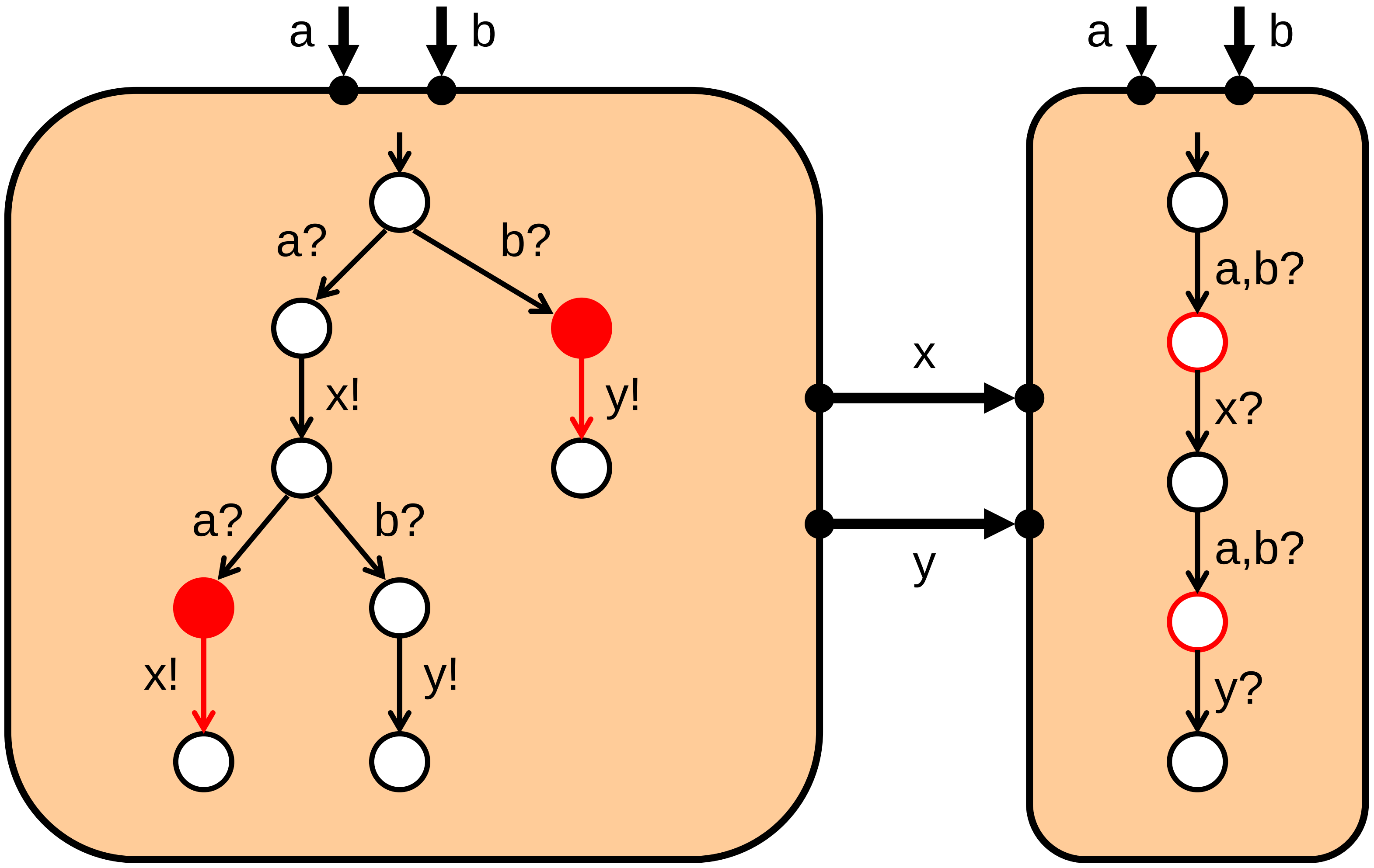
true

The environment is helpful !

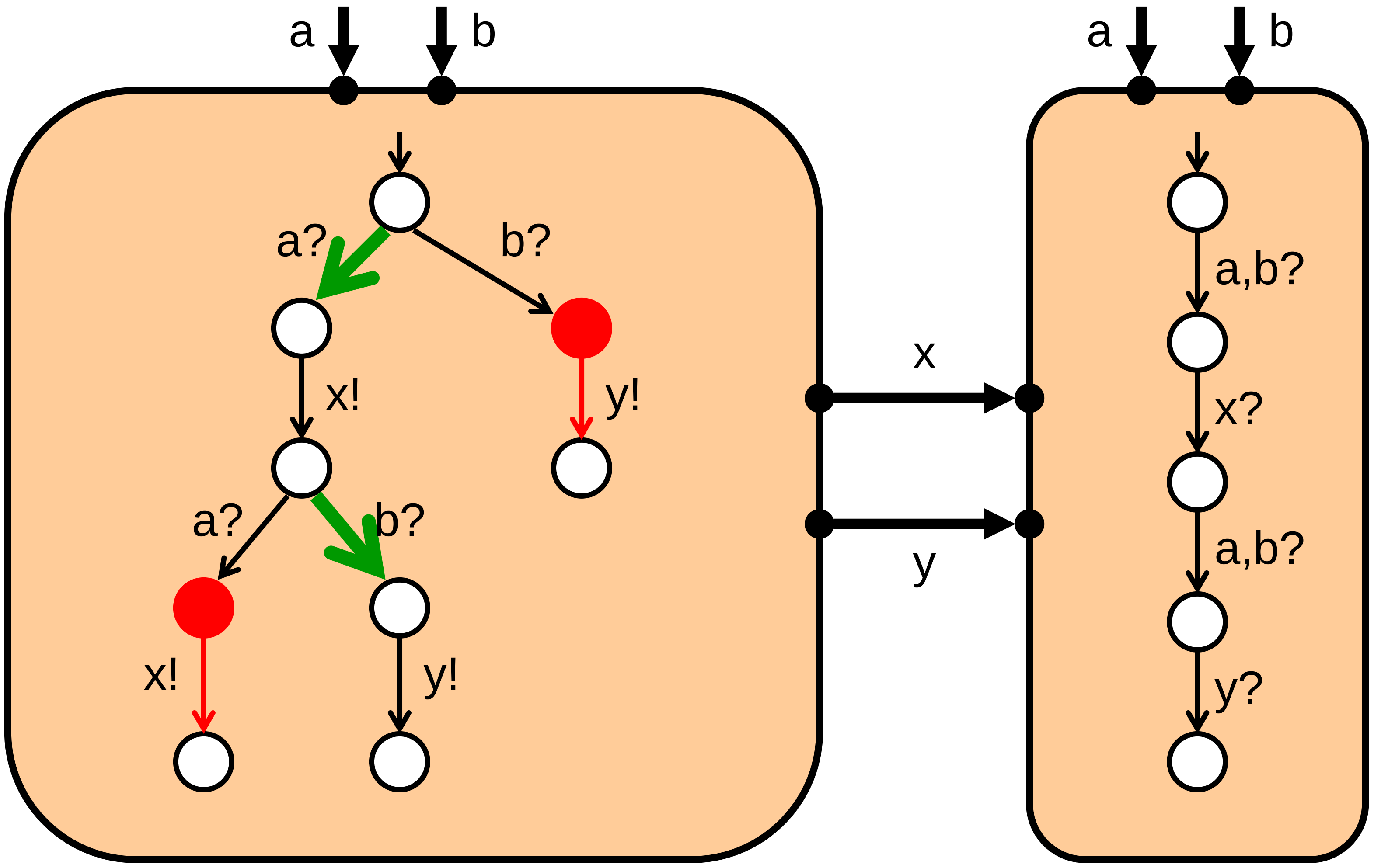
Automaton Interface Composition



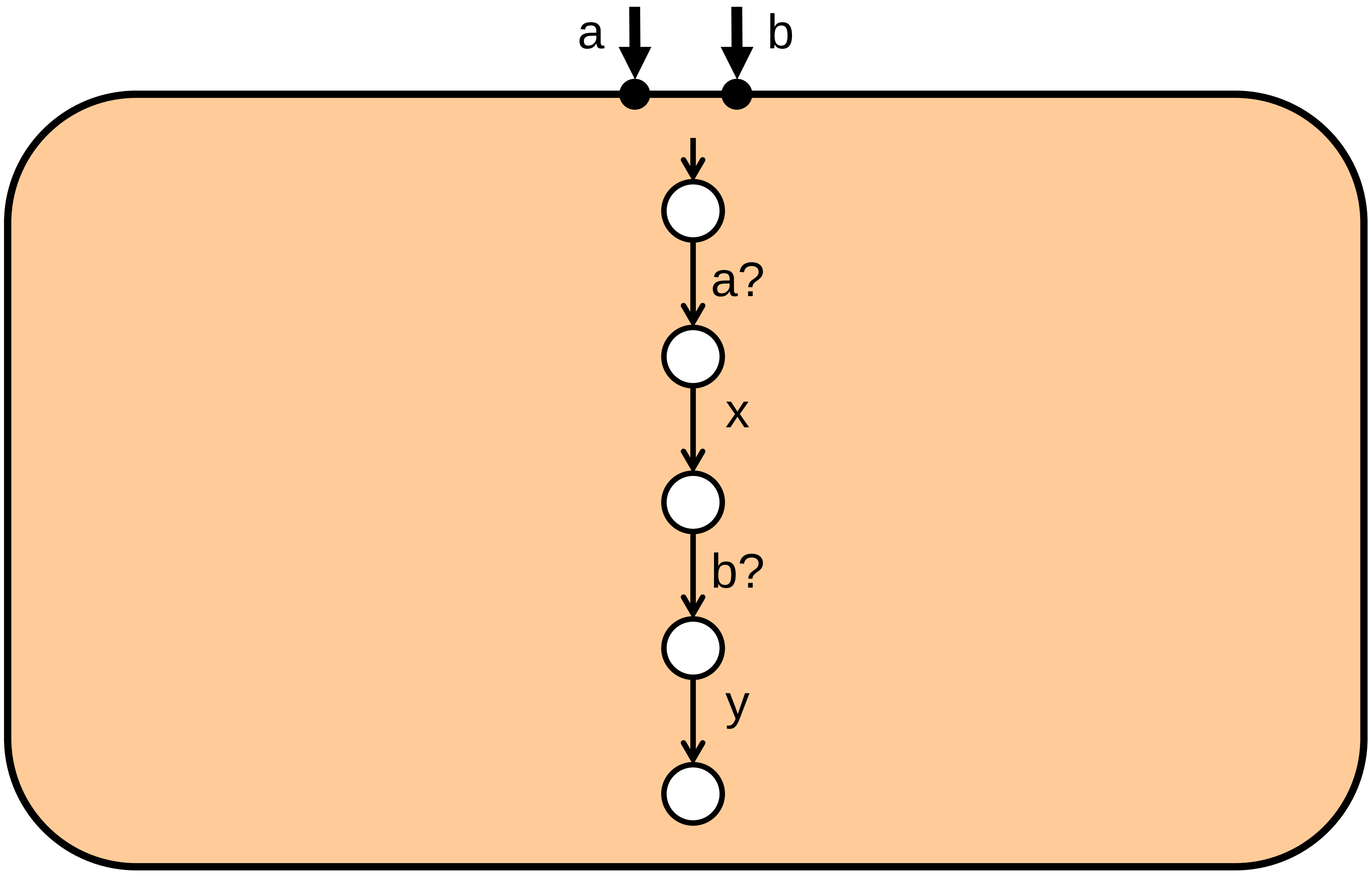
Automaton Interface Composition



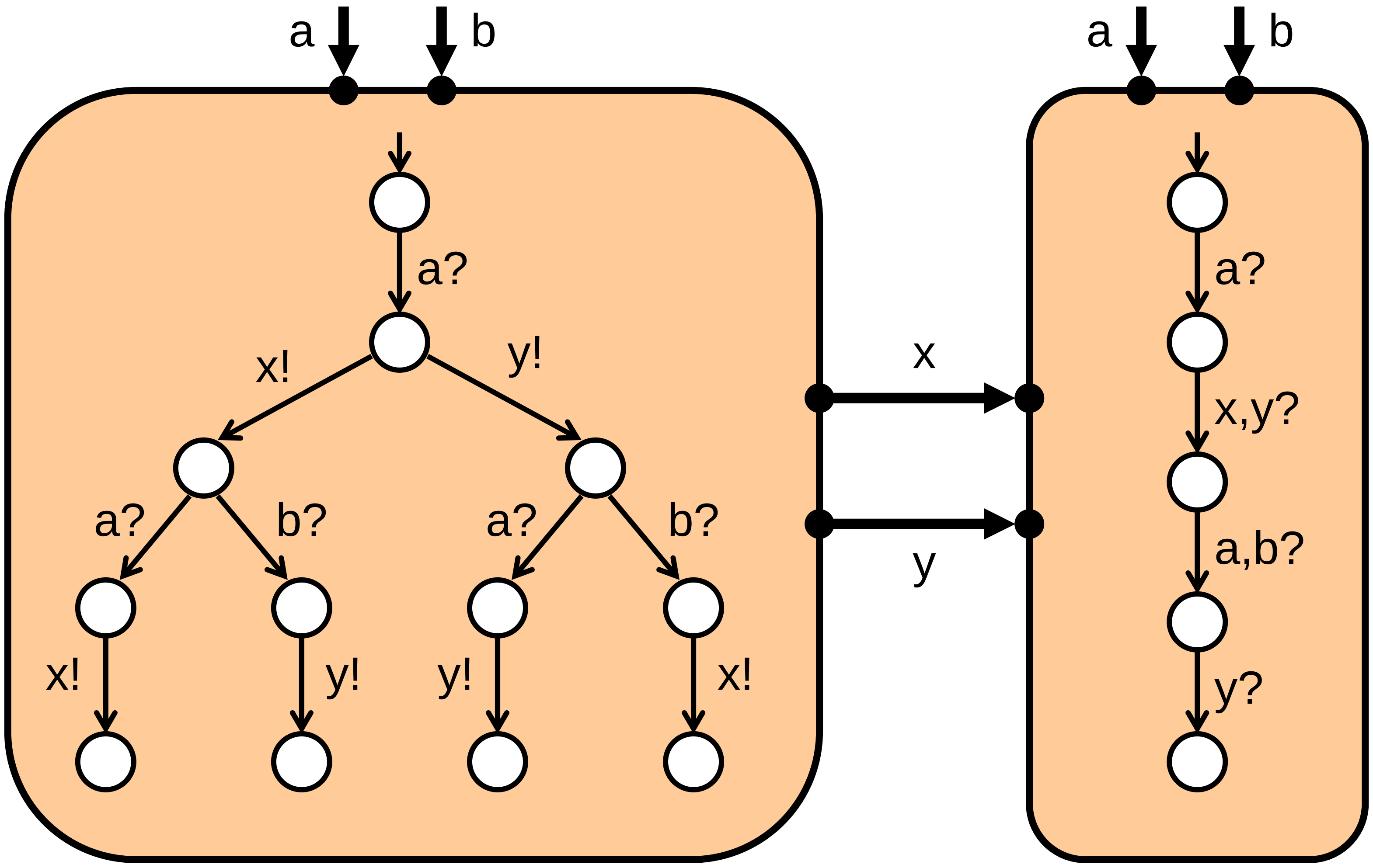
Automaton Interface Composition



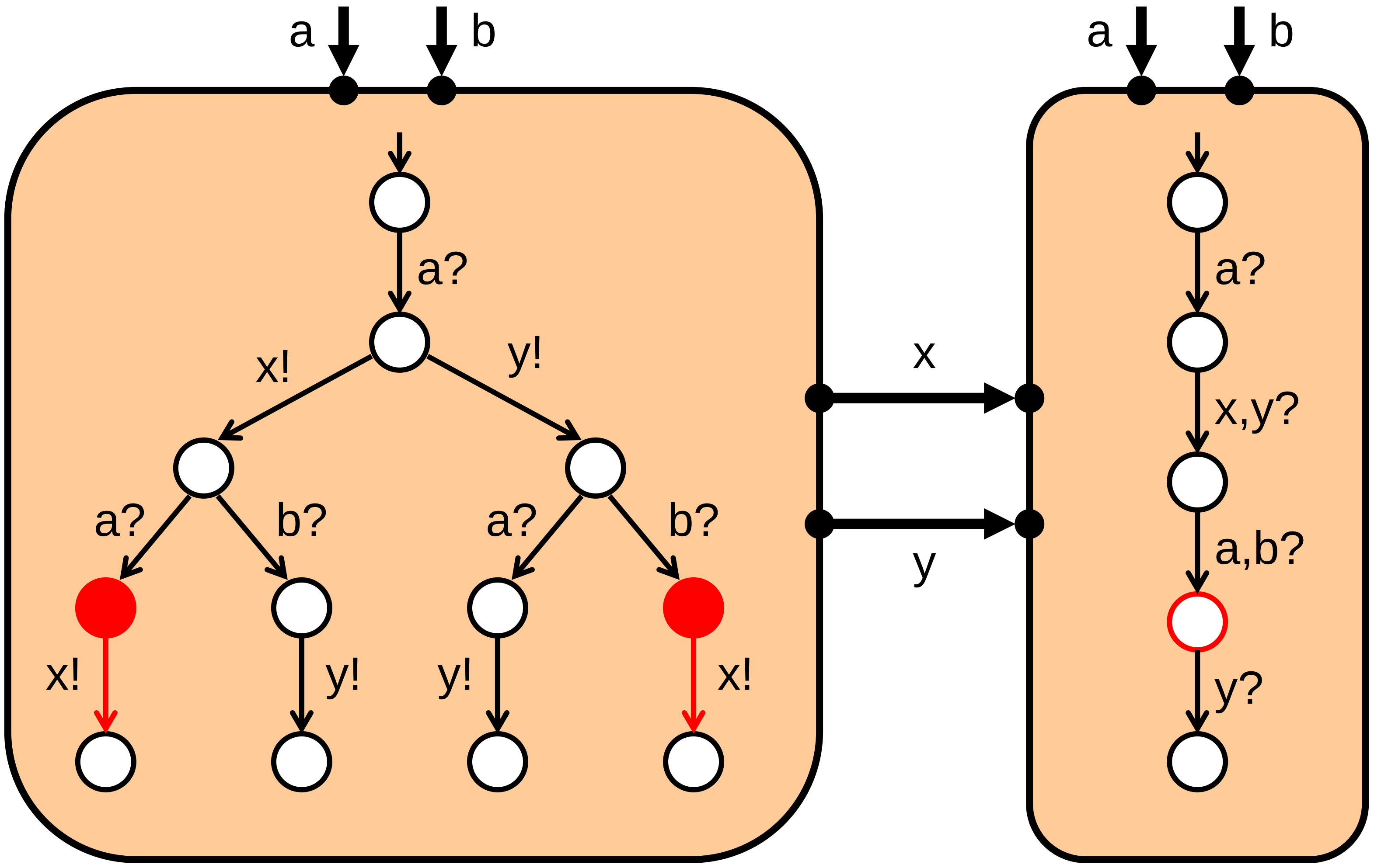
The Composite Interface



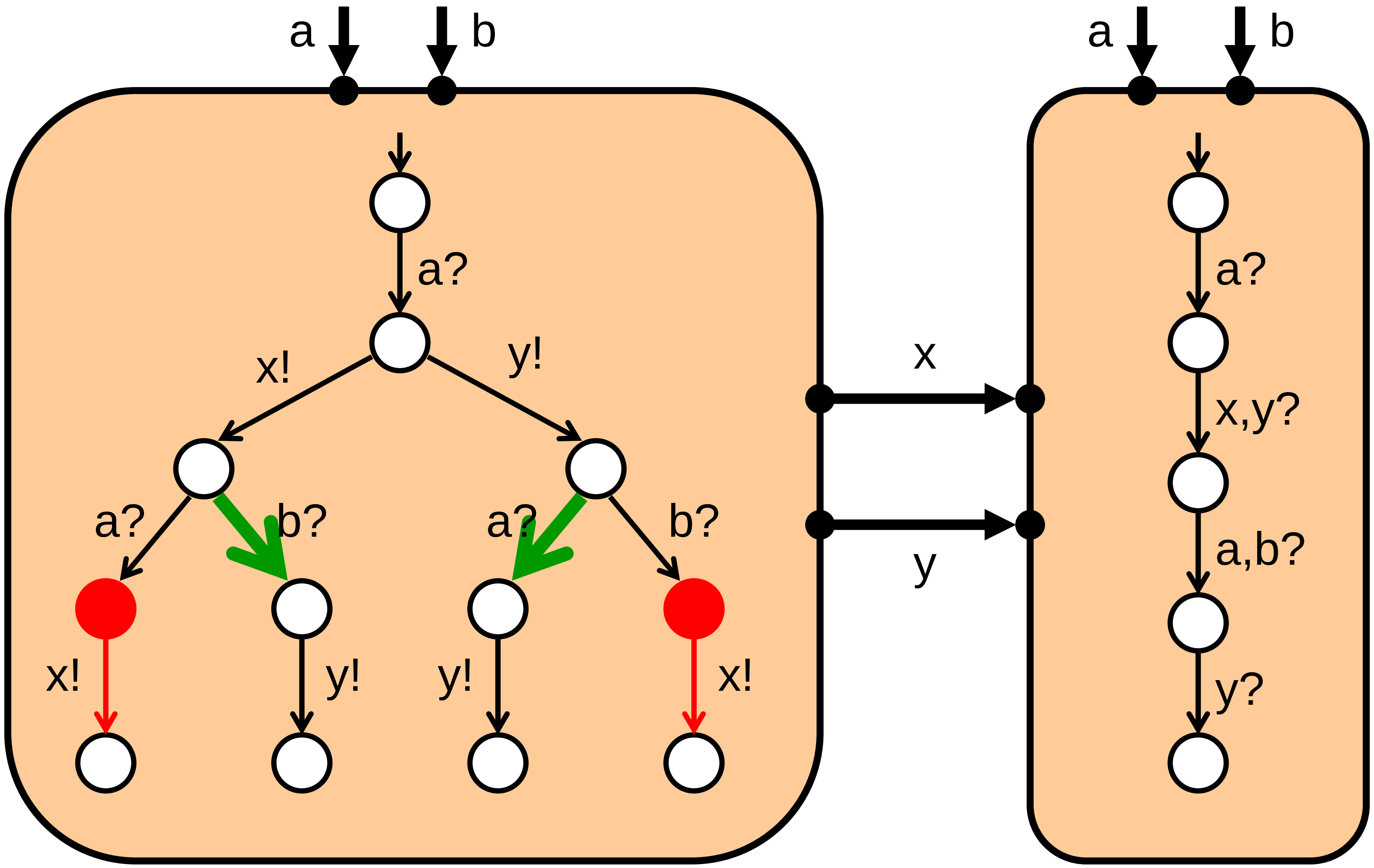
Automaton Interface Composition



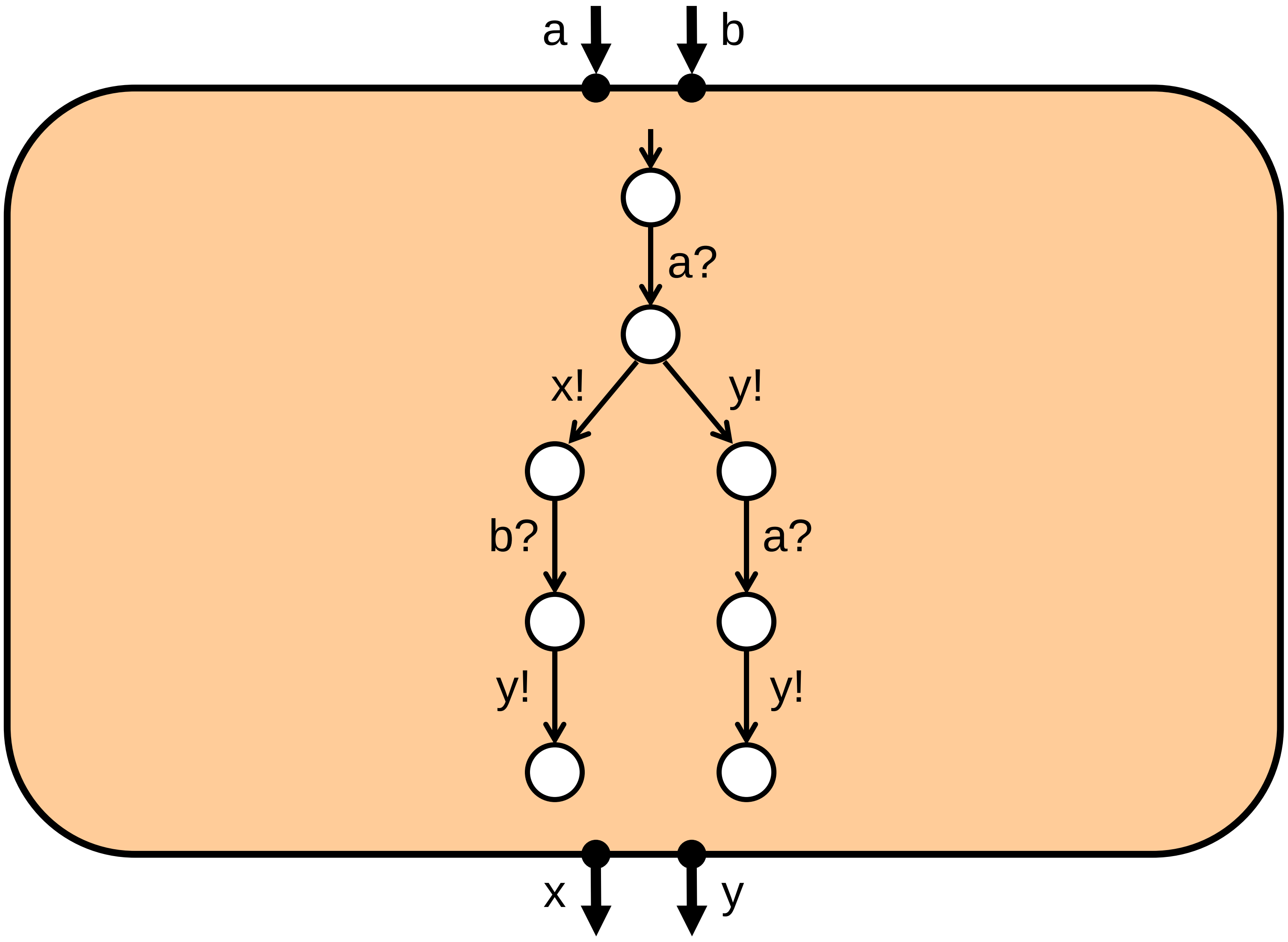
Automaton Interface Composition



Automaton Interface Composition



The Composite Interface



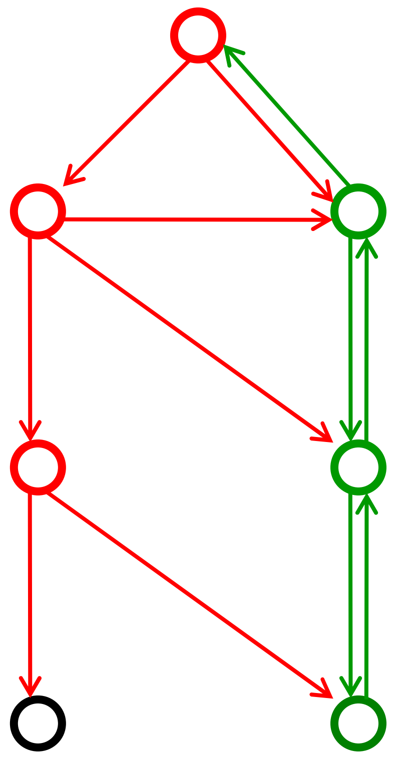
Lesson 3:

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.

Graph Games

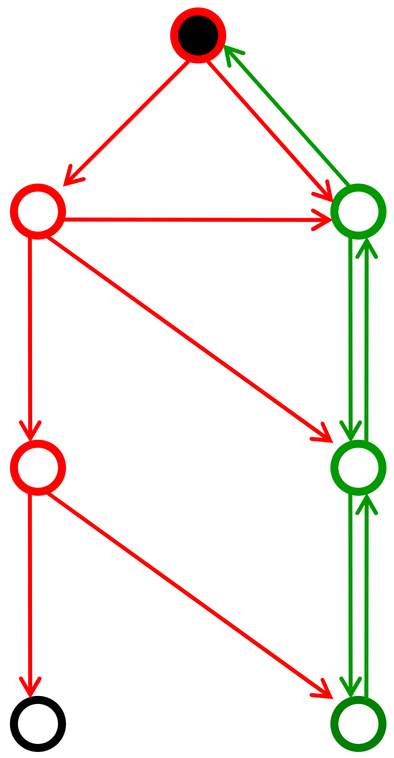


AND-OR Graph:

OR Nodes

AND Nodes

Graph Games

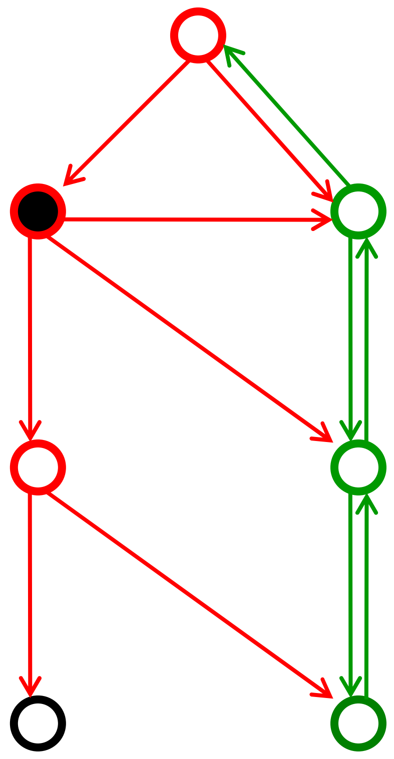


AND-OR Graph:

OR Player

AND Player

Graph Games

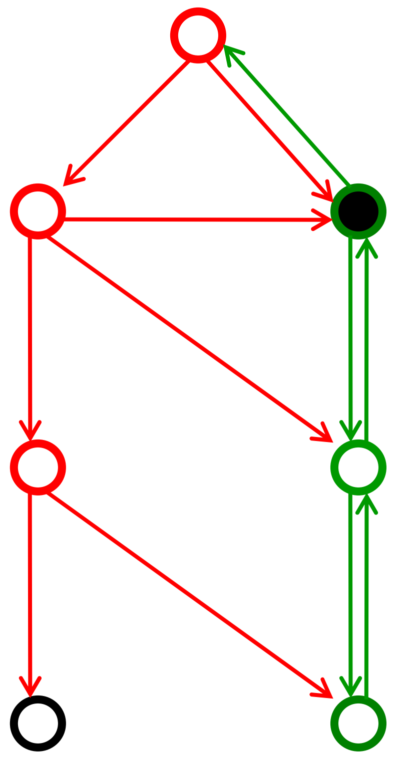


AND-OR Graph:

OR Player

AND Player

Graph Games

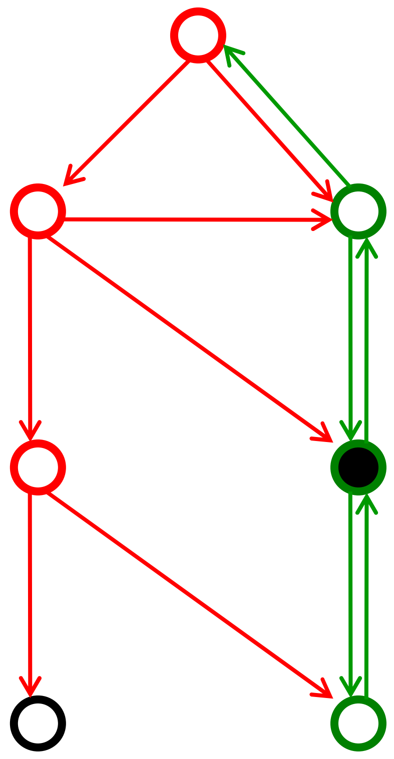


AND-OR Graph:

OR Player

AND Player

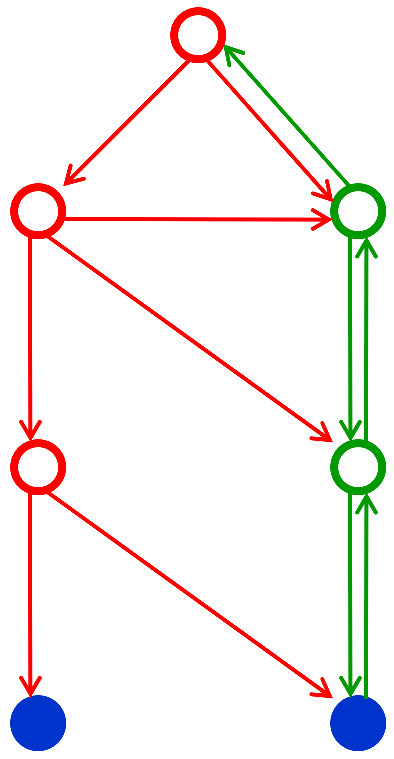
Graph Games



AND-OR Graph:

OR Player
AND Player

Safety Games

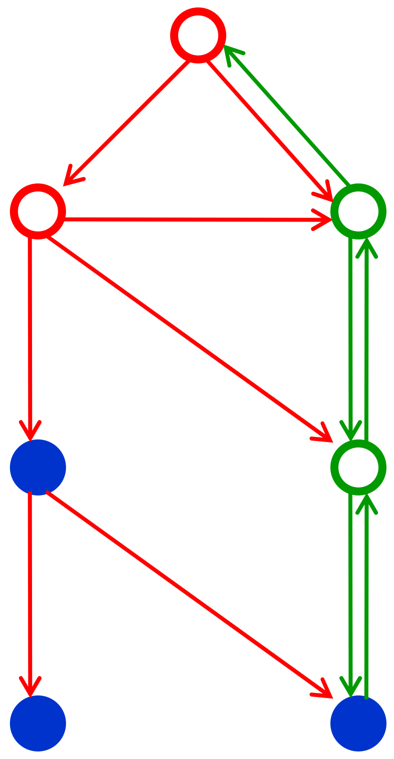


AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

Safety Games

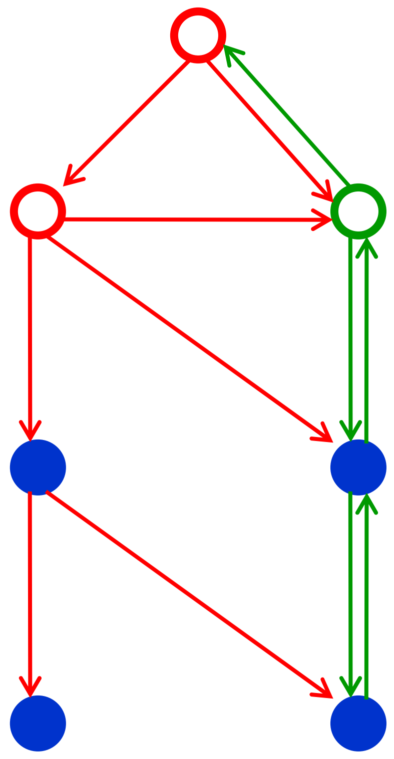


AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

Safety Games

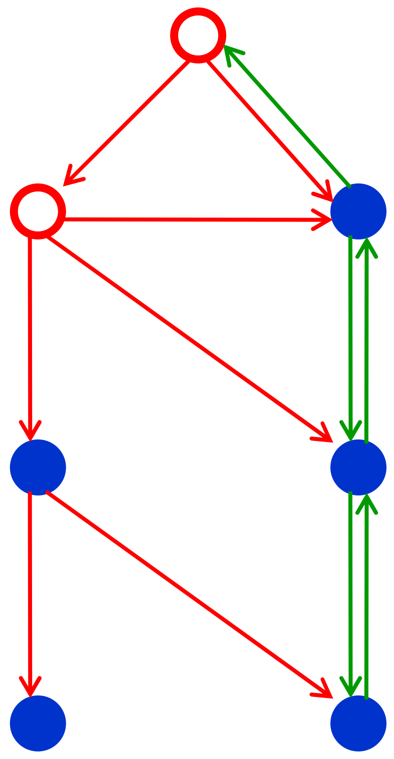


AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

Safety Games

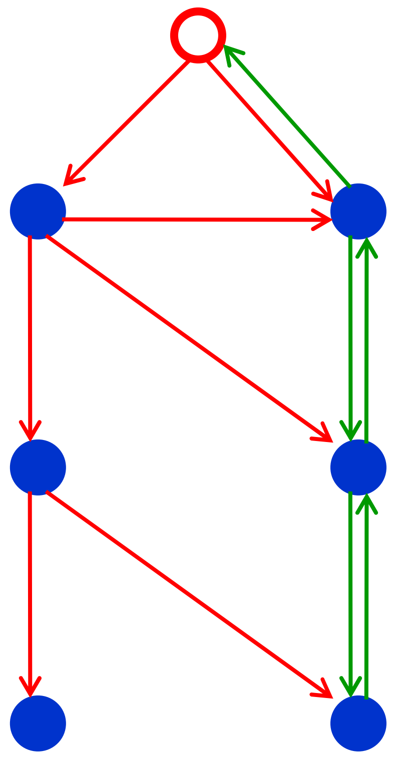


AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

Safety Games

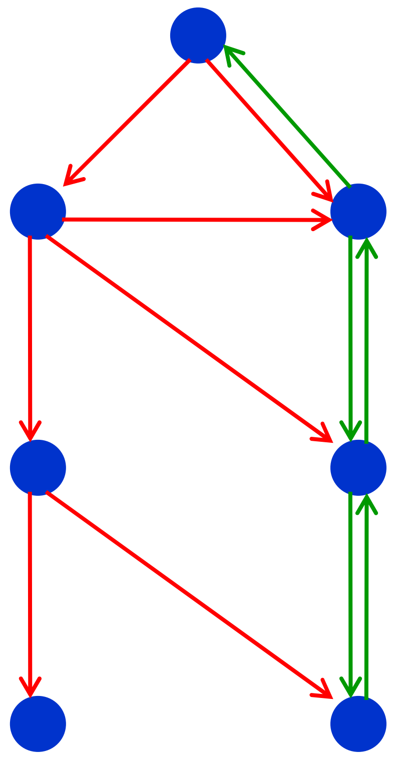


AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

Safety Games

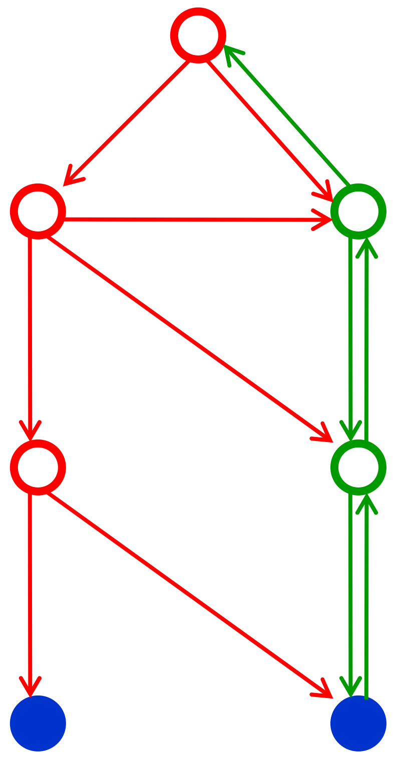


AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

Safety Games



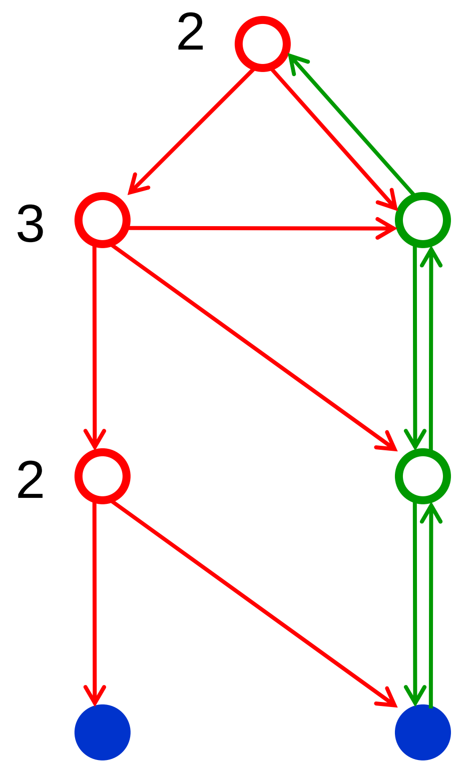
AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

Complexity?

Safety Games

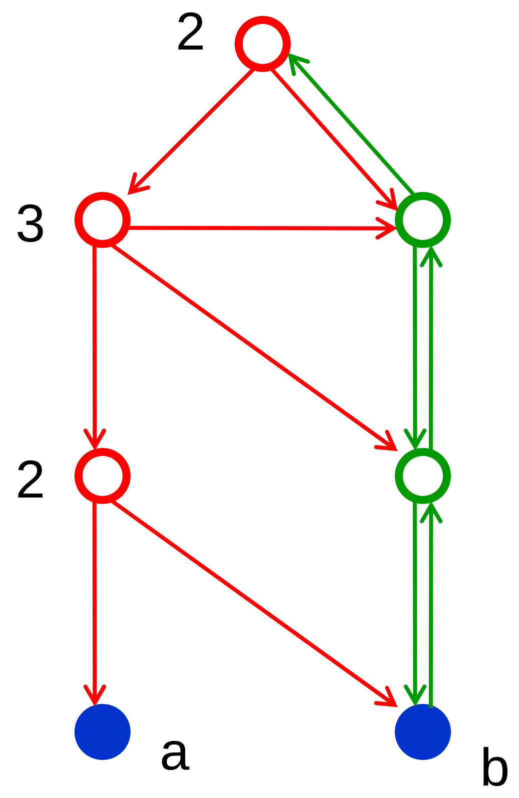


AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

Safety Games



AND-OR Graph:

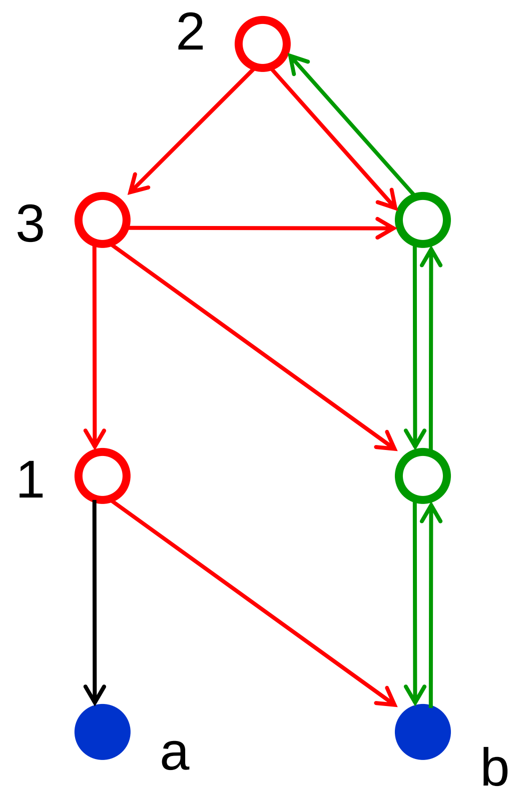
OR Player

AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

a b

Safety Games



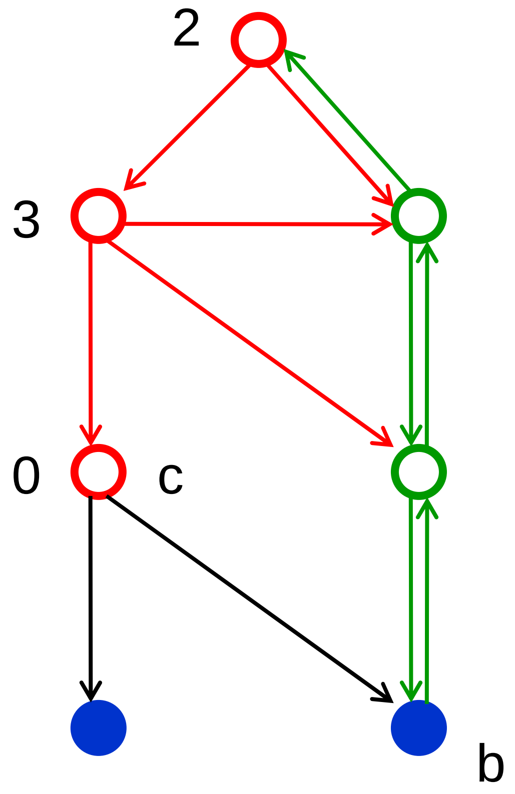
AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

b

Safety Games



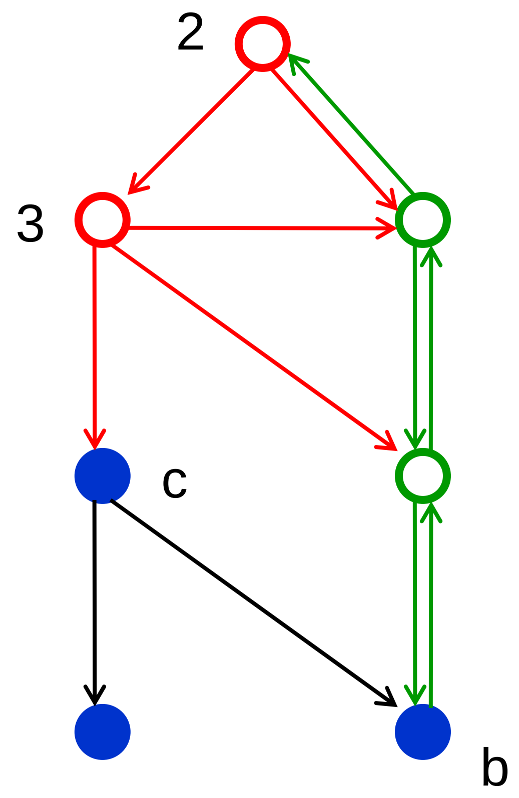
AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

b c

Safety Games



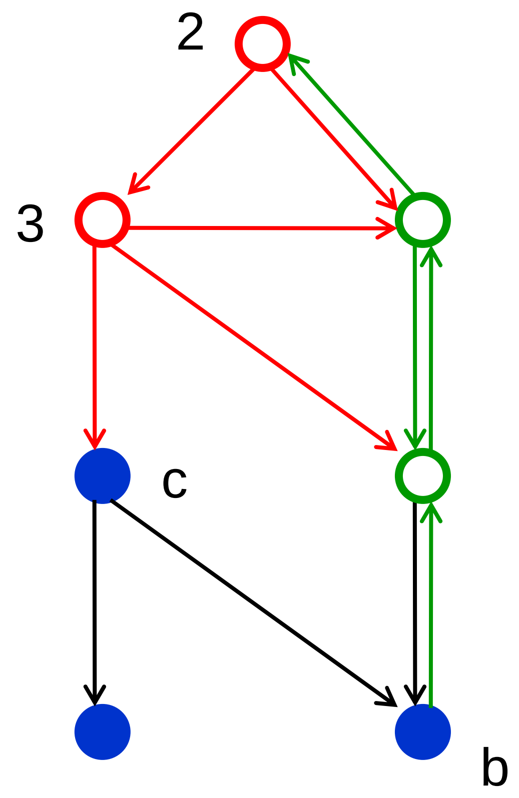
AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

b c

Safety Games



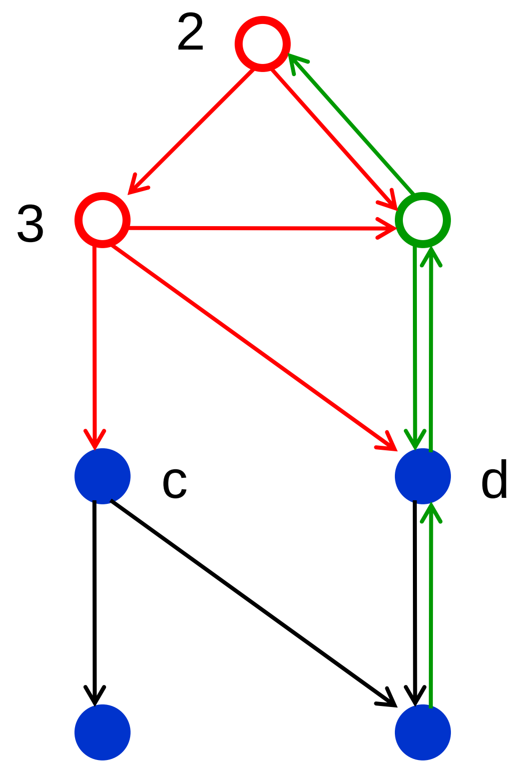
AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

c

Safety Games



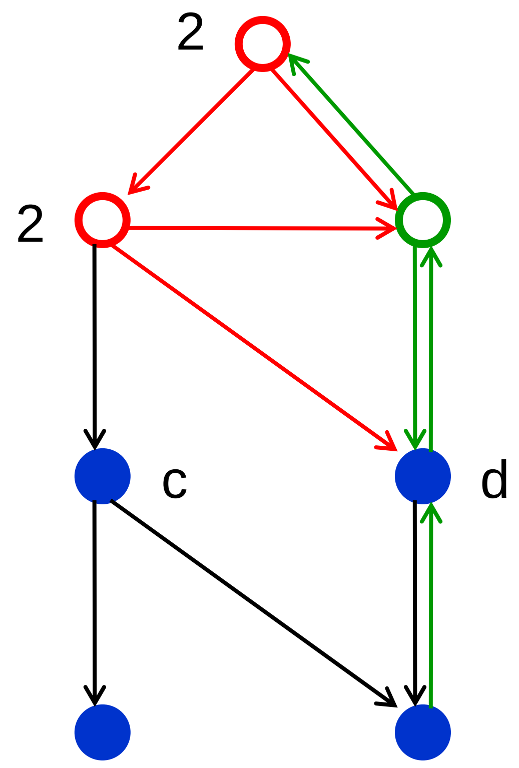
AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

c d

Safety Games



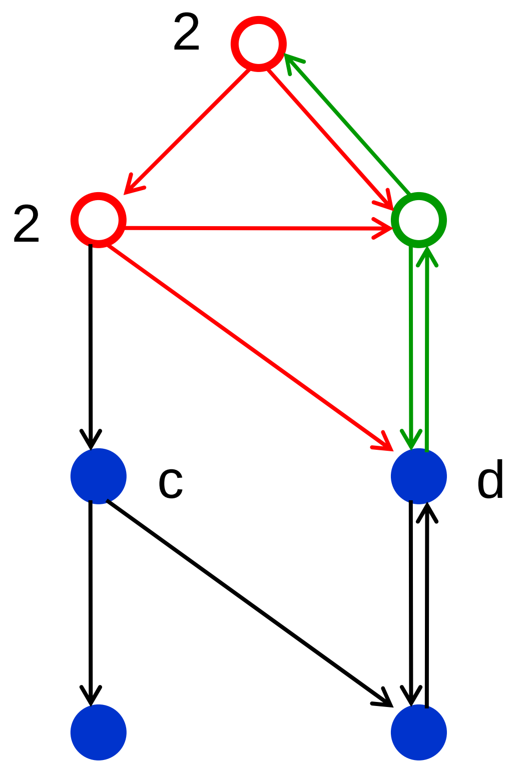
AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

d

Safety Games



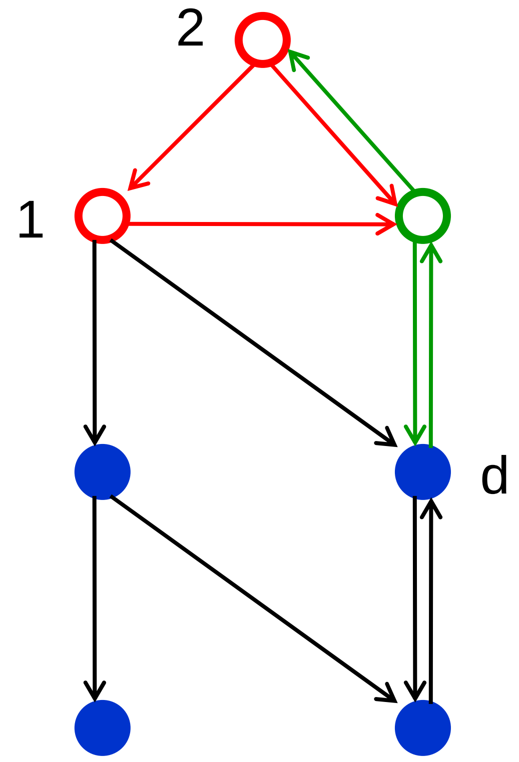
AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

d

Safety Games



AND-OR Graph:

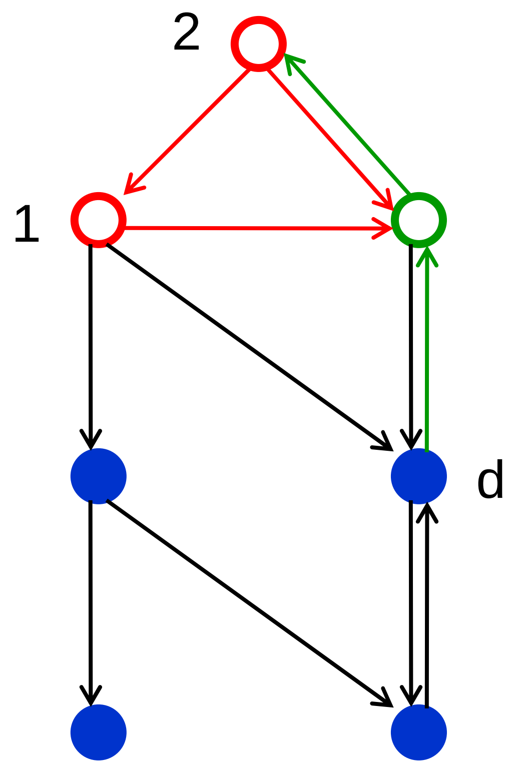
OR Player

AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

d

Safety Games

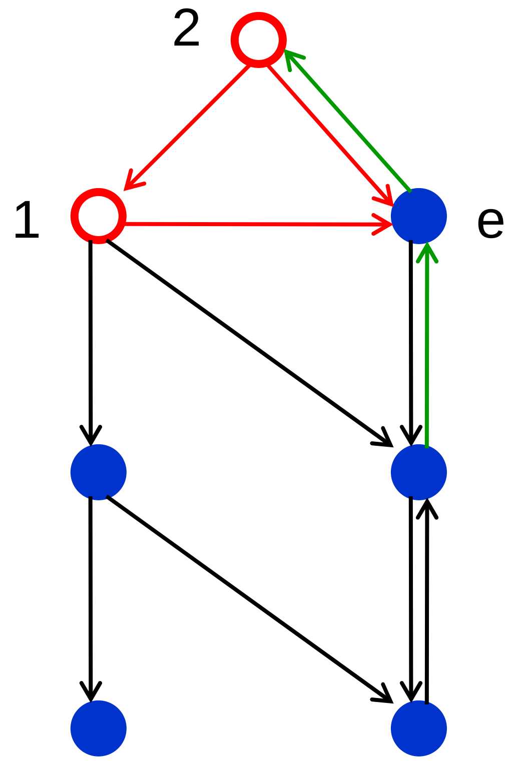


AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

Safety Games



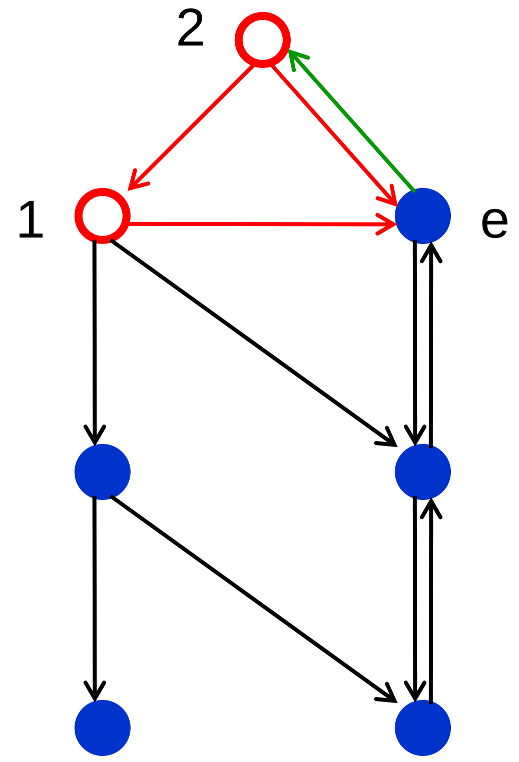
AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

e

Safety Games



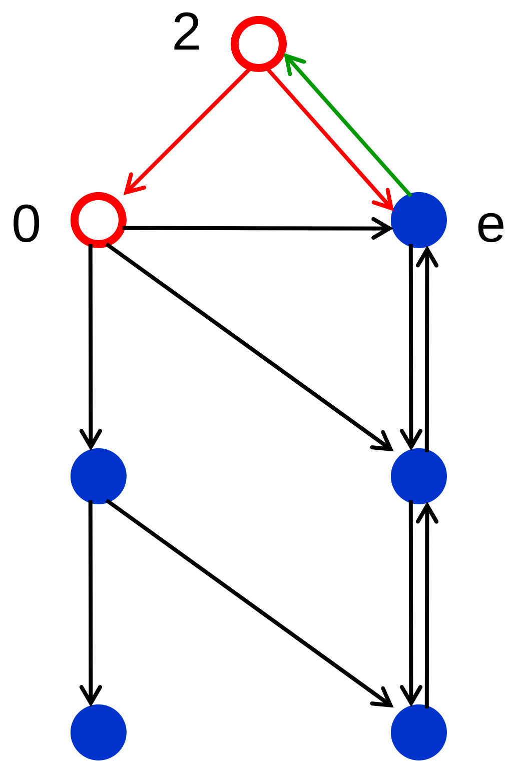
AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

e

Safety Games



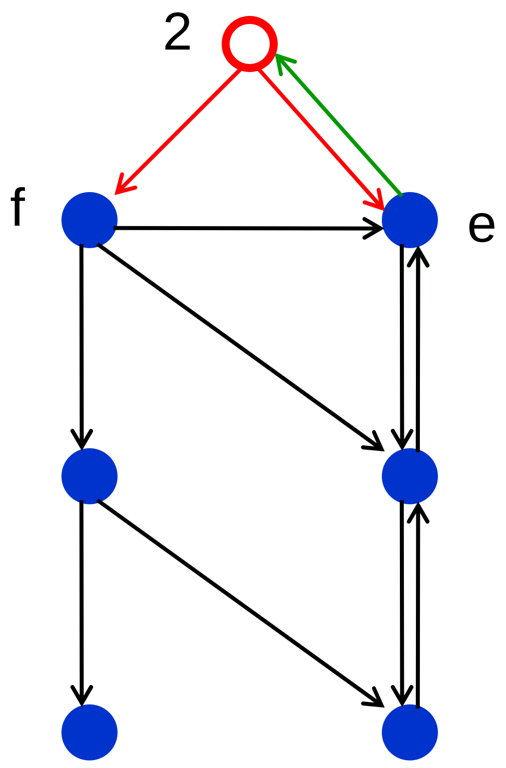
AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

e

Safety Games



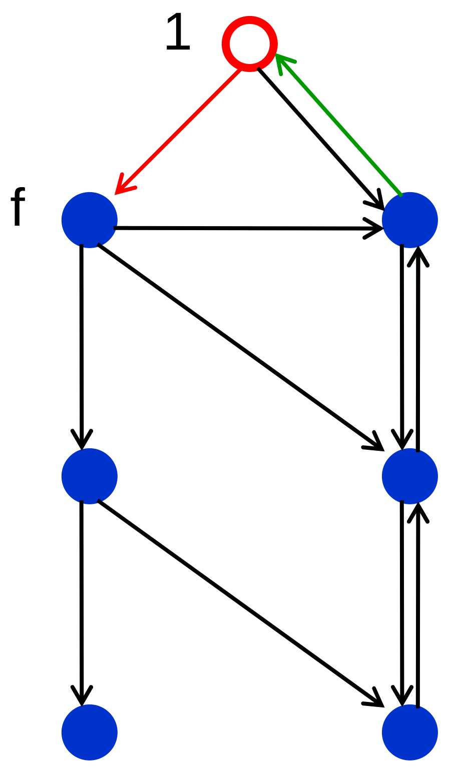
AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

e f

Safety Games



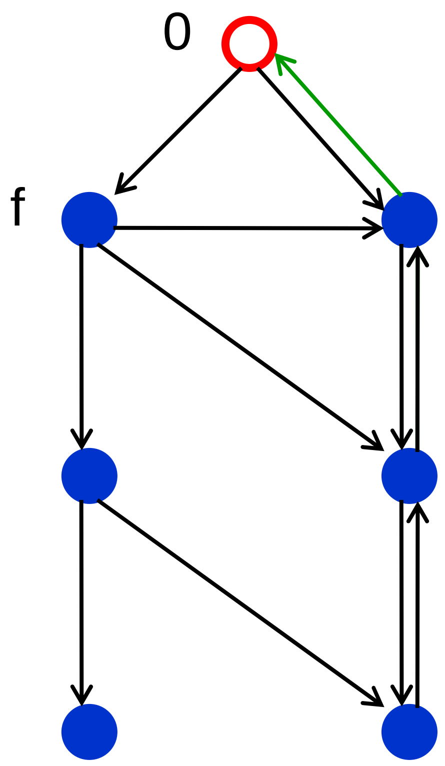
AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

f

Safety Games

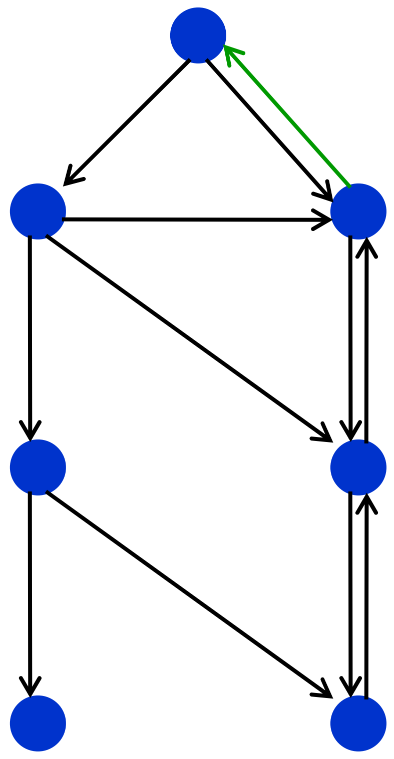


AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

Safety Games



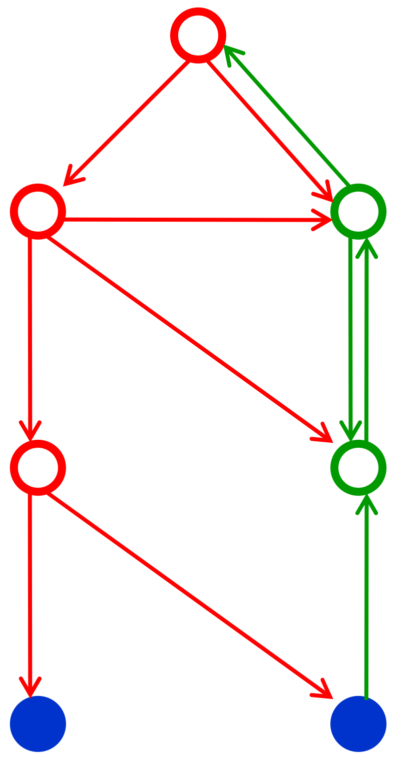
AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

Linear Time (P-Complete)

Safety Games

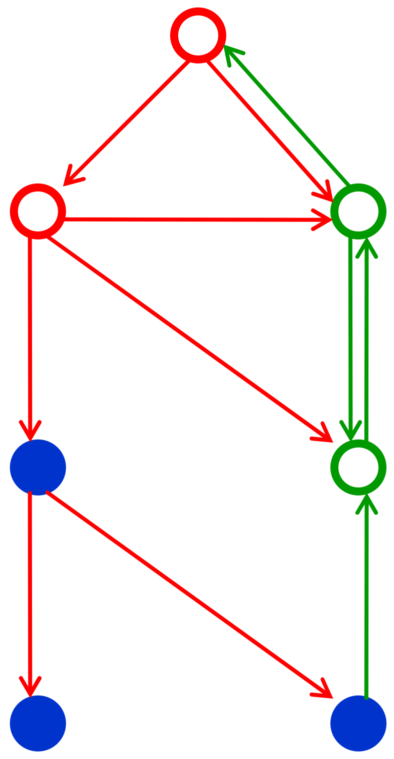


AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

Safety Games

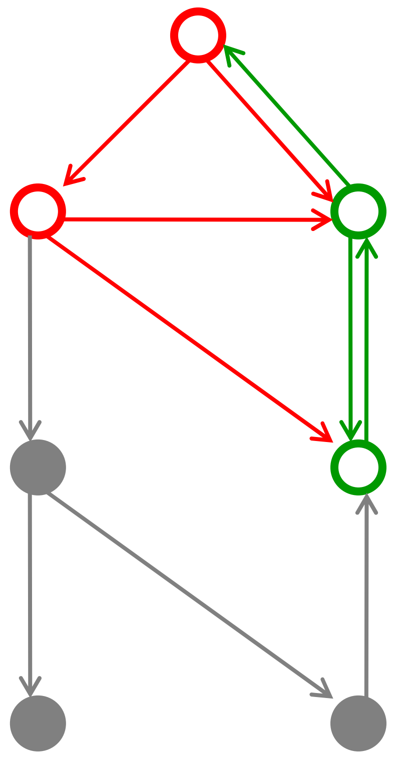


AND-OR Graph:

OR Player
AND Player

From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

Safety Games



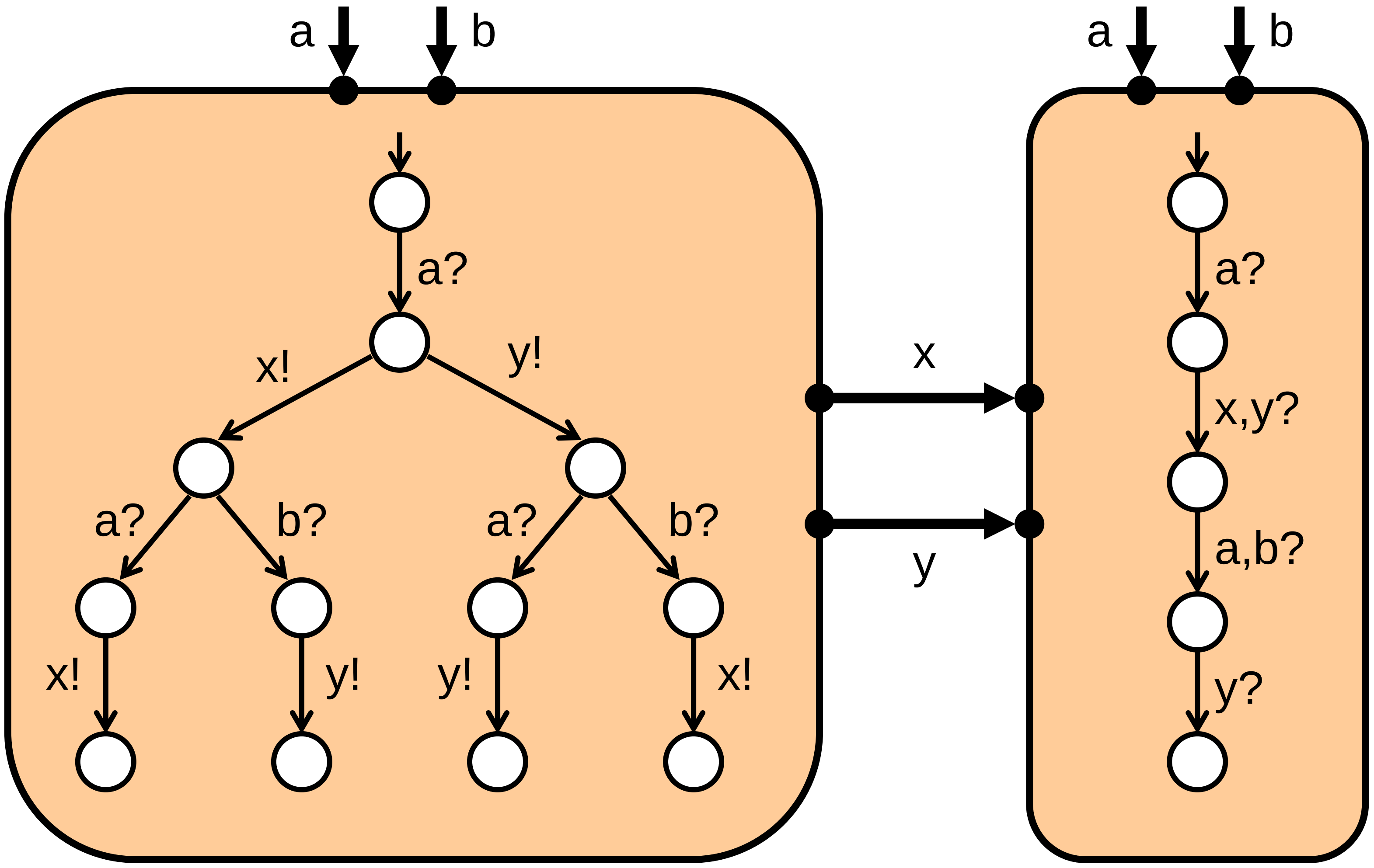
AND-OR Graph:

OR Player
AND Player

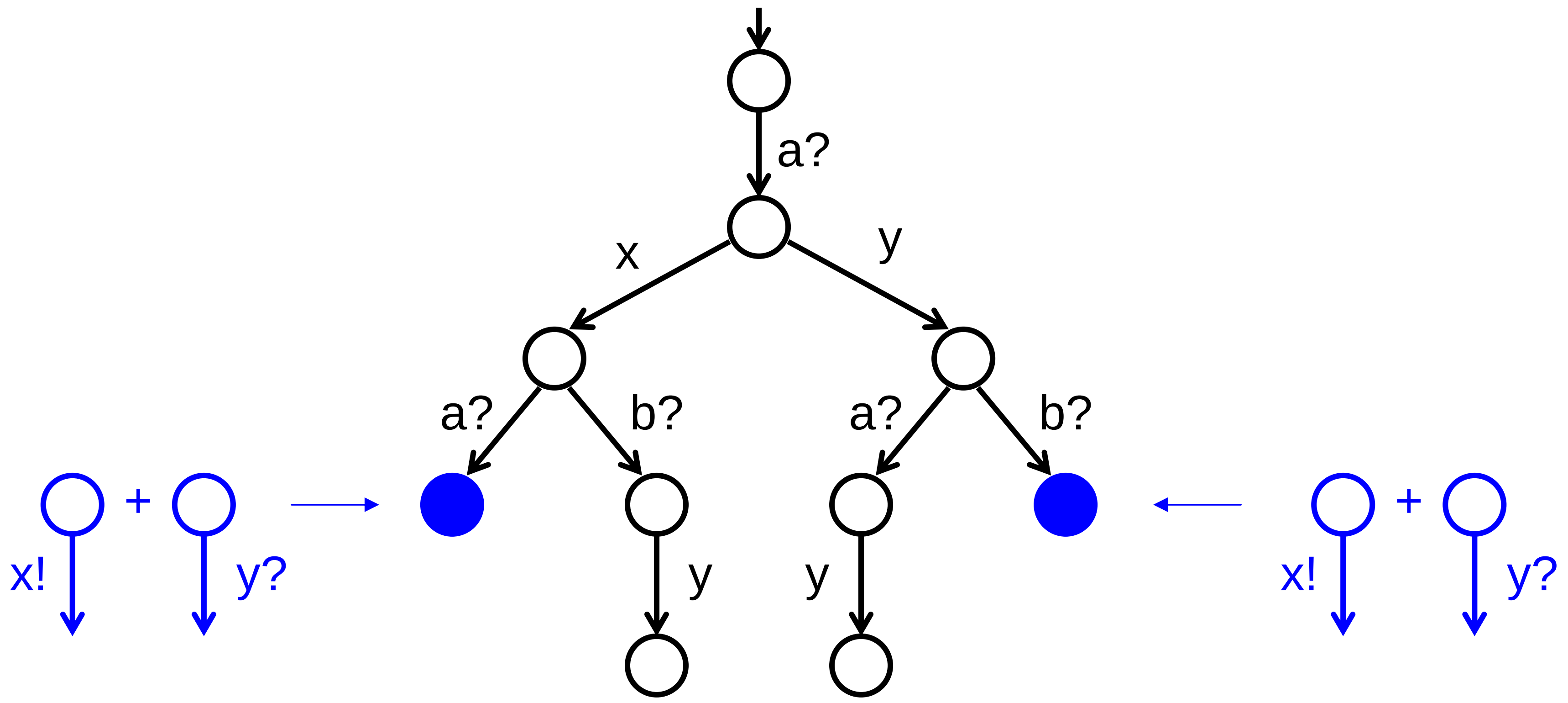
From which nodes does the OR Player have a strategy to avoid ERROR Nodes ?

Most general memoryless pure strategy exists.

From Interfaces to Games

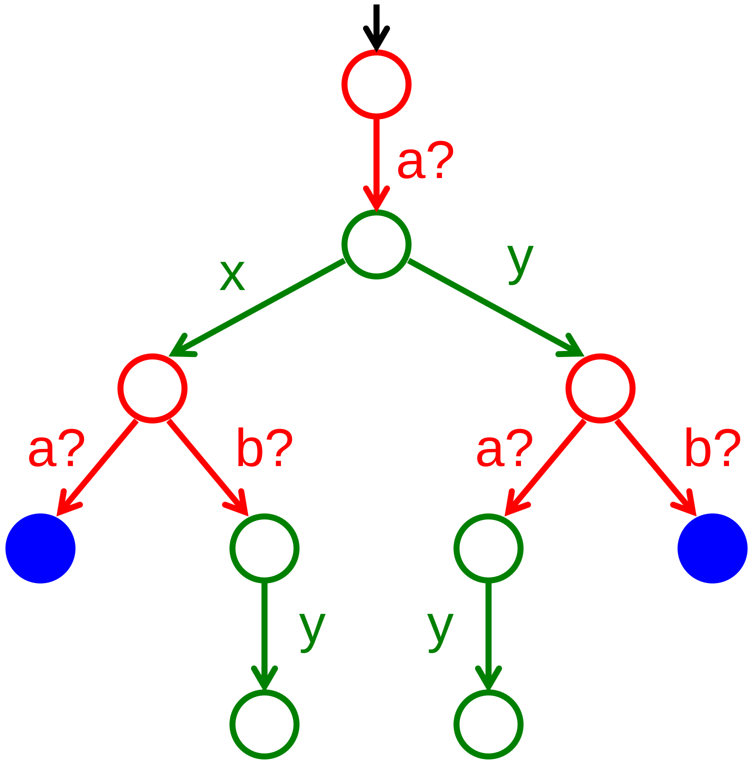


From Interfaces to Games



ERROR states of product automaton.

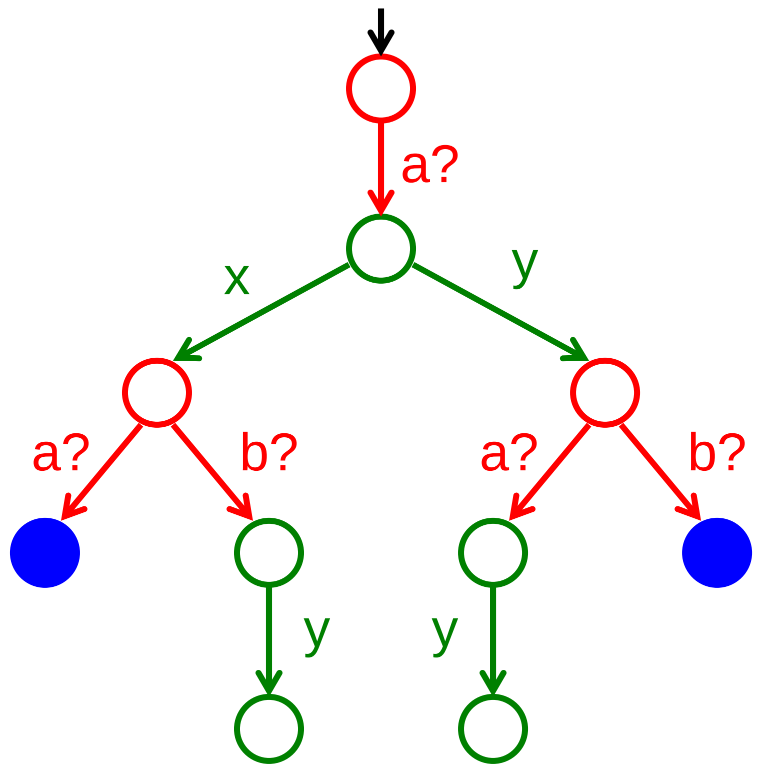
From Interfaces to Games



OR Player ... External choices made by the environment

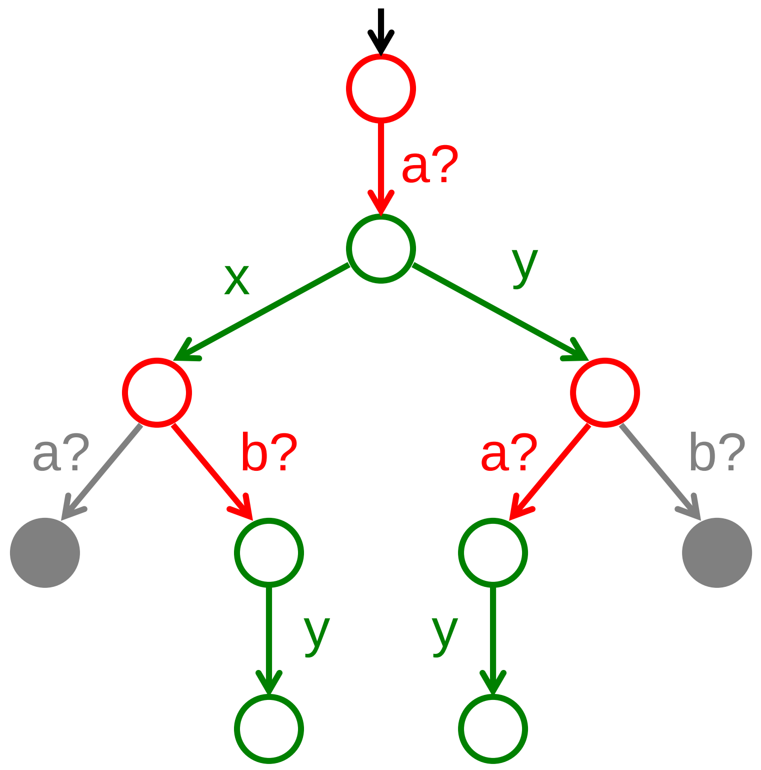
AND Player ... Internal choices made by the interface product

From Interfaces to Games



Does the Environment have a strategy to avoid the ERROR states?

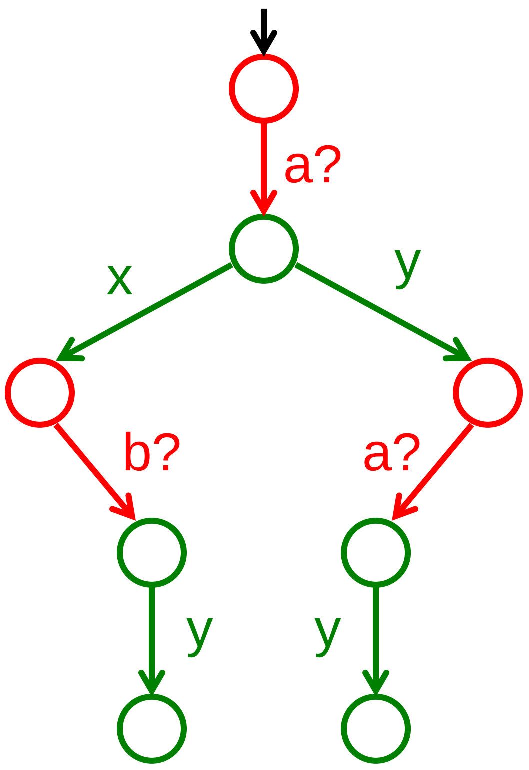
From Interfaces to Games



Yes

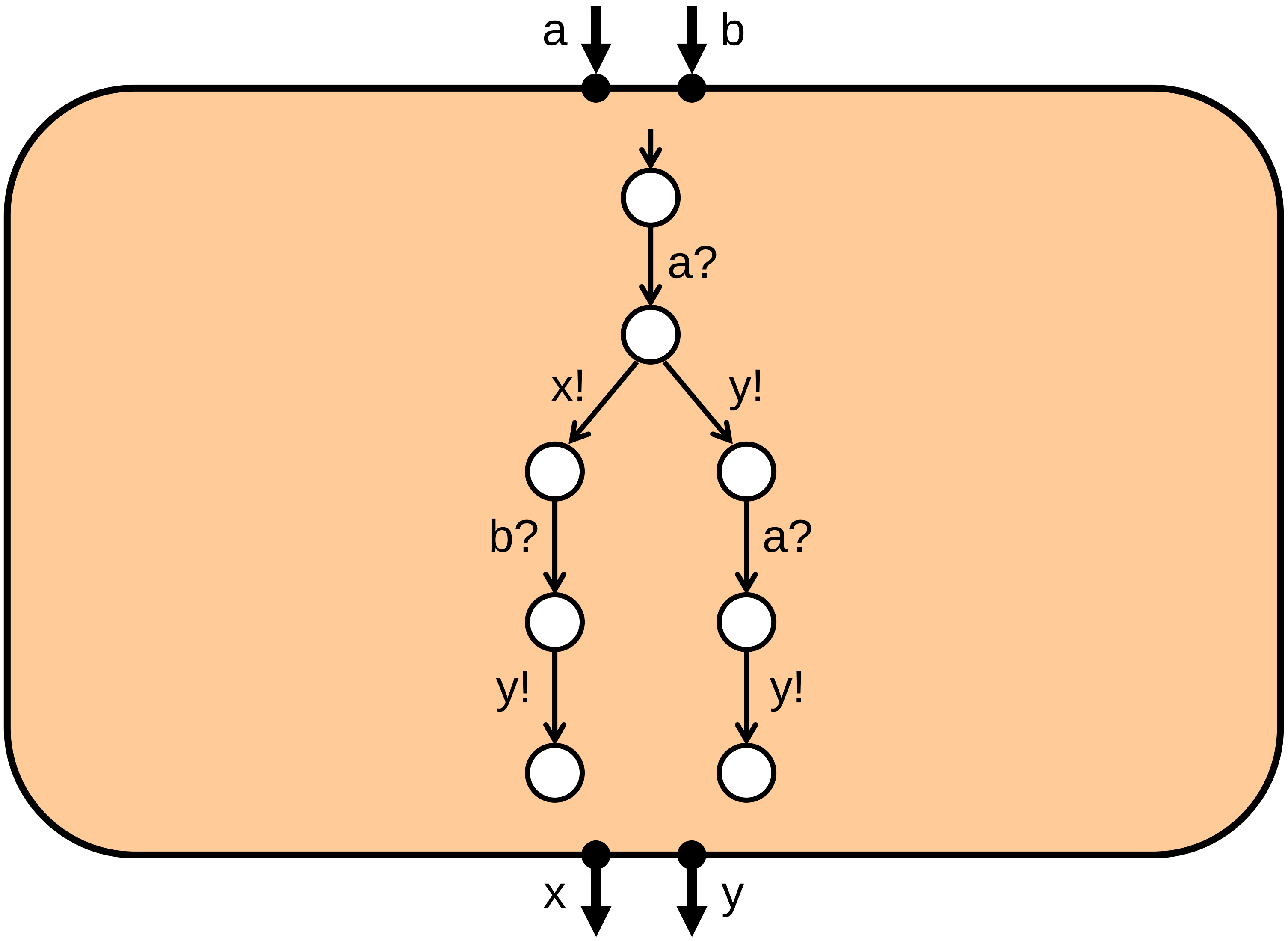
Does the Environment have a strategy to avoid the ERROR states?

From Interfaces to Games



The most general environment strategy.

The Composite Interface



So far, we have used a simple lock-step model of concurrency.

An interface formalism can be built around any model of concurrency.

So far, we have used a simple lock-step model of concurrency.

An interface formalism can be built around any model of concurrency.

Example:

$$\begin{tikzcd}
\boxed{\text{I/O Automata [Lynch]}} \arrow[d] \\
\boxed{\text{Interface Automata [deAlfaro,H]}}
\end{tikzcd}$$

So far, we have used a simple lock-step model of concurrency.

An interface formalism can be built around any model of concurrency.

Example:

$$\begin{tikzcd}\boxed{\text{I/O Automata [Lynch]}} \arrow[d, "\textit{same syntax}"', "\textit{different semantics}"] \\ \boxed{\text{Interface Automata [deAlfaro,H]}}\end{tikzcd}$$

-Total composition
-A process model

-Partial composition
-Compatibility check
-An interface model

An Interface Automaton

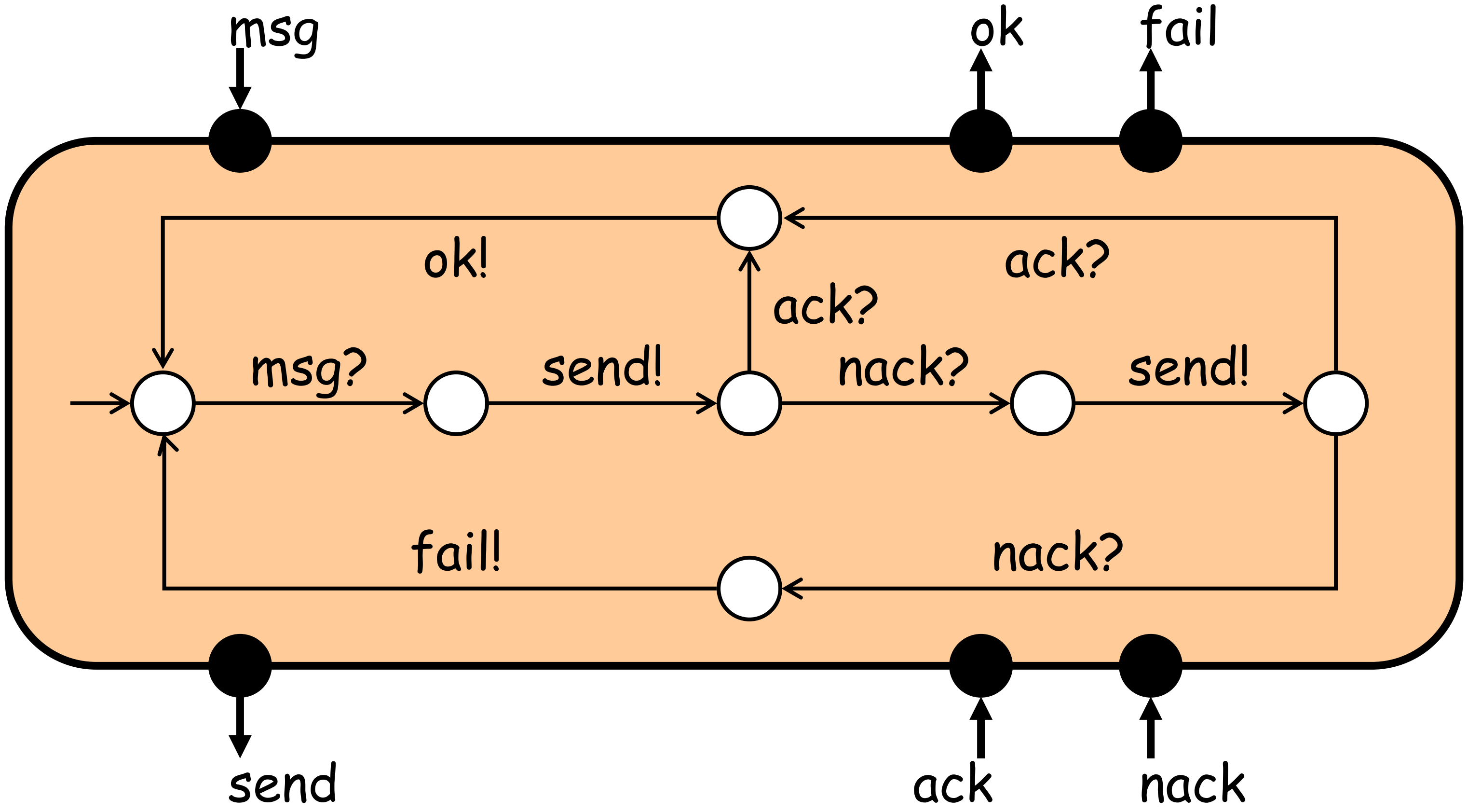
$F = (Q_F, Q^0_F, A^I_F, A^O_F, A^H_F, T_F)$

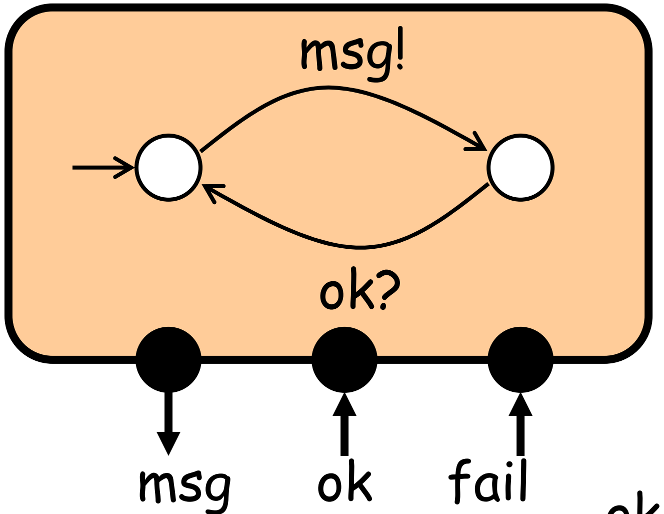
Q_F ... set of *states*

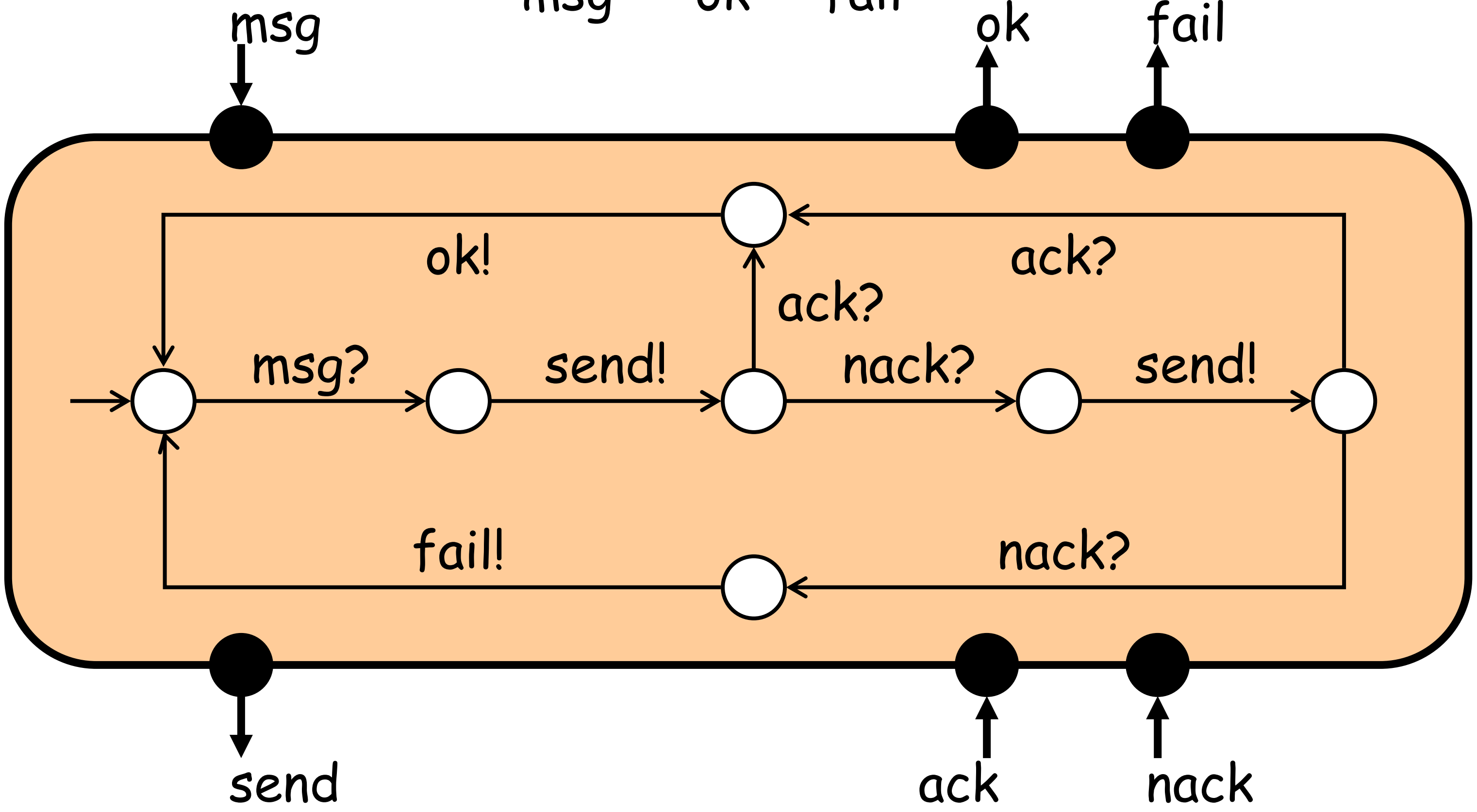
Q^0_F µ Q_F ... set of *initial states*

$$\left.\begin{array}{l} A^I_\mathrm{F} \ \text{ ... } \ \textit{input actions} \\ A^O_\mathrm{F} \ \text{ ... } \ \textit{output actions} \\ A^H_\mathrm{F} \ \text{ ... } \ \textit{internal}\text{ (hidden) }\textit{actions} \end{array}\right\} \begin{array}{l} \text{mutually disjoint} \\ A_F = A^I_F \text{ [} A^O_F \text{ [} A^H_F \end{array}$$

T_F µ Q_F £ A_F £ Q_F ... set of *transitions*


msg
ok
fail
ok!
ack?
ack?
msg?
send!
nack?
send!
fail!
nack?
send
ack
nack


msg!
ok?
msg
ok
fail


msg
ok
fail
ok!
ack?
ack?
msg?
send!
nack?
send!
fail!
nack?
send
ack
nack

The Product of Interface Automata

F and *G* are *composable* if

A^{H}_{F} Å A_{G} = ; A^{H}_{G} Å A_{F} = ; A^{I}_{F} Å A^{I}_{G} = ; A^{O}_{F} Å A^{O}_{G} = ;

If *F* and *G* are composable, then

$$\left.\begin{array}{l}
Q_{F\,\text{£}\,G} = Q_{F}\ \text{£}\ Q_{G} \\
Q^{0}_{F\,\text{£}\,G} = Q^{0}_{F}\ \text{£}\ Q^{0}_{G} \\
A^{I}_{F\,\text{£}\,G} = (A^{I}_{F}\ [\ A^{I}_{G})\ \backslash\ shared(F,G) \\
A^{O}_{F\,\text{£}\,G} = (A^{O}_{F}\ [\ A^{O}_{G})\ \backslash\ shared(F,G) \\
A^{H}_{F\,\text{£}\,G} = A^{H}_{F}\ [\ A^{H}_{G}\ [\ shared(F,G)
\end{array}\right\}\quad shared(F,G) = A_{F}\ \text{Å}\ A_{G}$$

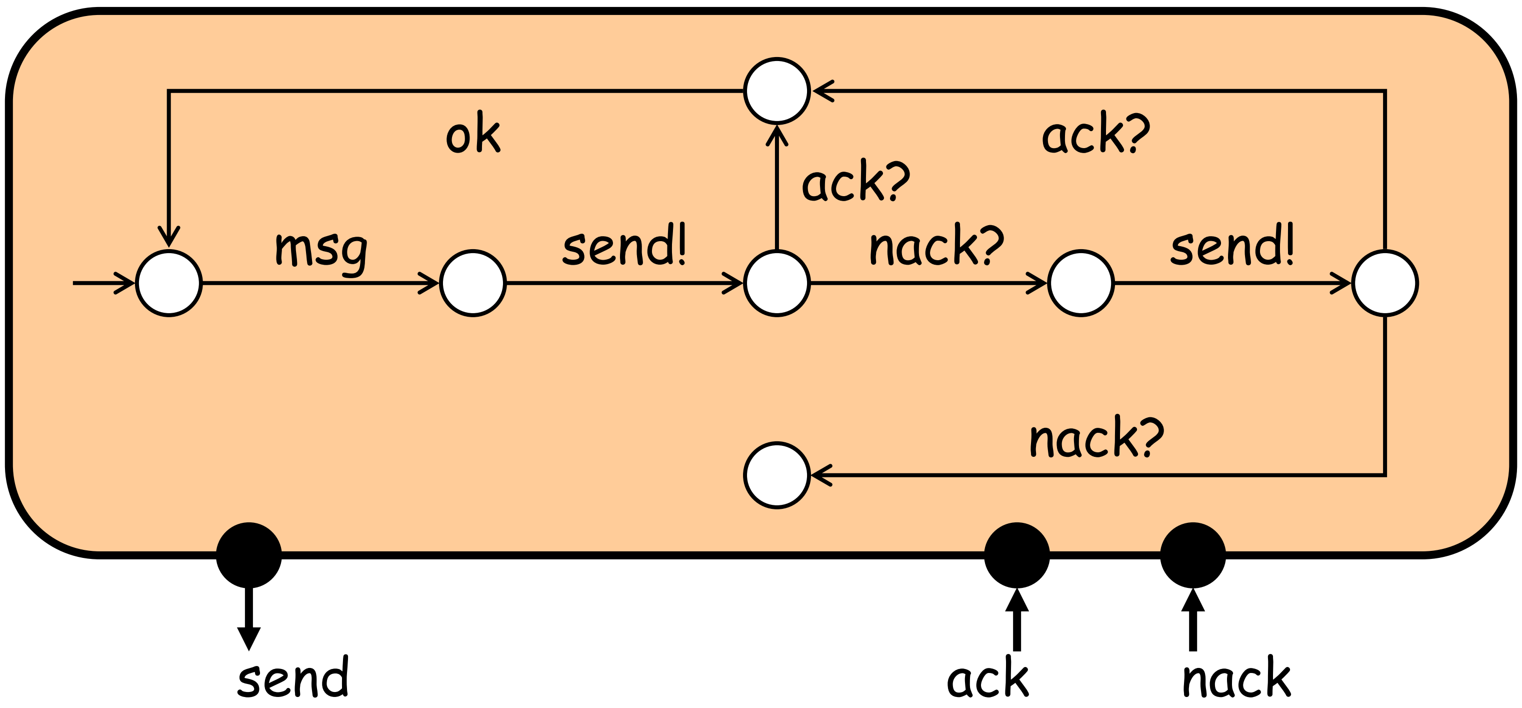
The Product of Interface Automata, continued

$T_{F \text{ £ } G}$ =

{ $((f,g), a, (f',g))$: (f,a,f') 2 T_F Æ $a \notin shared(F,G)$ } [

{ $((f,g), b, (f,g'))$: (g,b,g') 2 T_G Æ $b \notin shared(F,G)$ } [

{ $((f,g), c, (f',g'))$: (f,c,f') 2 T_F Æ (g,c,g') 2 T_G Æ c 2 $shared(F,G)$ }



The product automaton.

The Error States of the Product Automaton

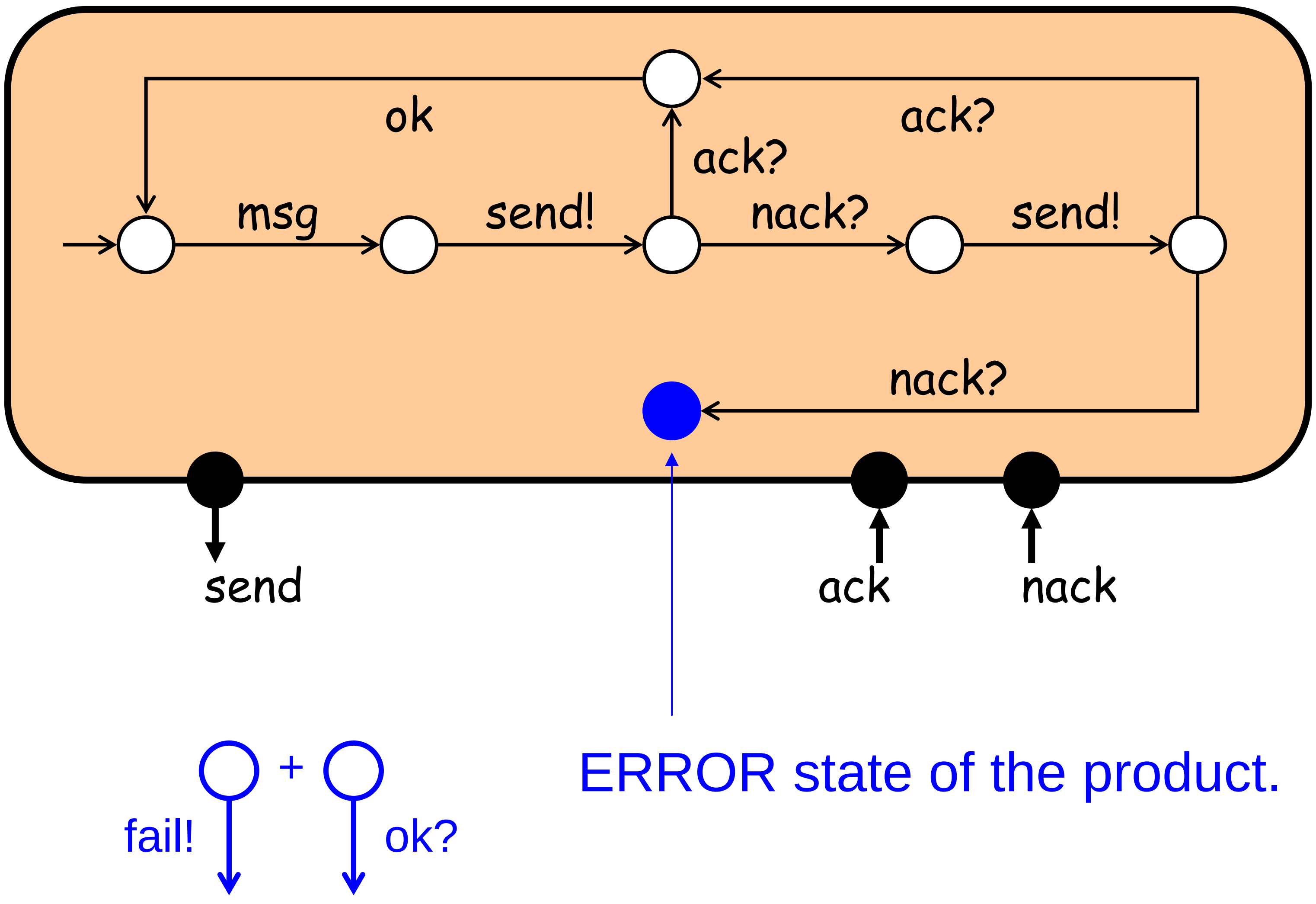
Error(F,G) = { *(f,g)* :   *(9 a* 2 *shared(F,G))*

(a 2 *$A^O{}_F$(f)* Æ *a* ∉ *$A^I{}_G$(g))* Ç

(a 2 *$A^O{}_G$(g)* Æ *a* ∉ *$A^I{}_F$(f))* }

where *A(q)* = { *a* 2 *A* : *(9 q')* *(q,a,q')* 2 *T* }

Note: I/O automata are input enabling (A^I(q) = A^I for all q) and therefore have no error states.



ERROR state of the product.

The Compatibility of Interface Automata

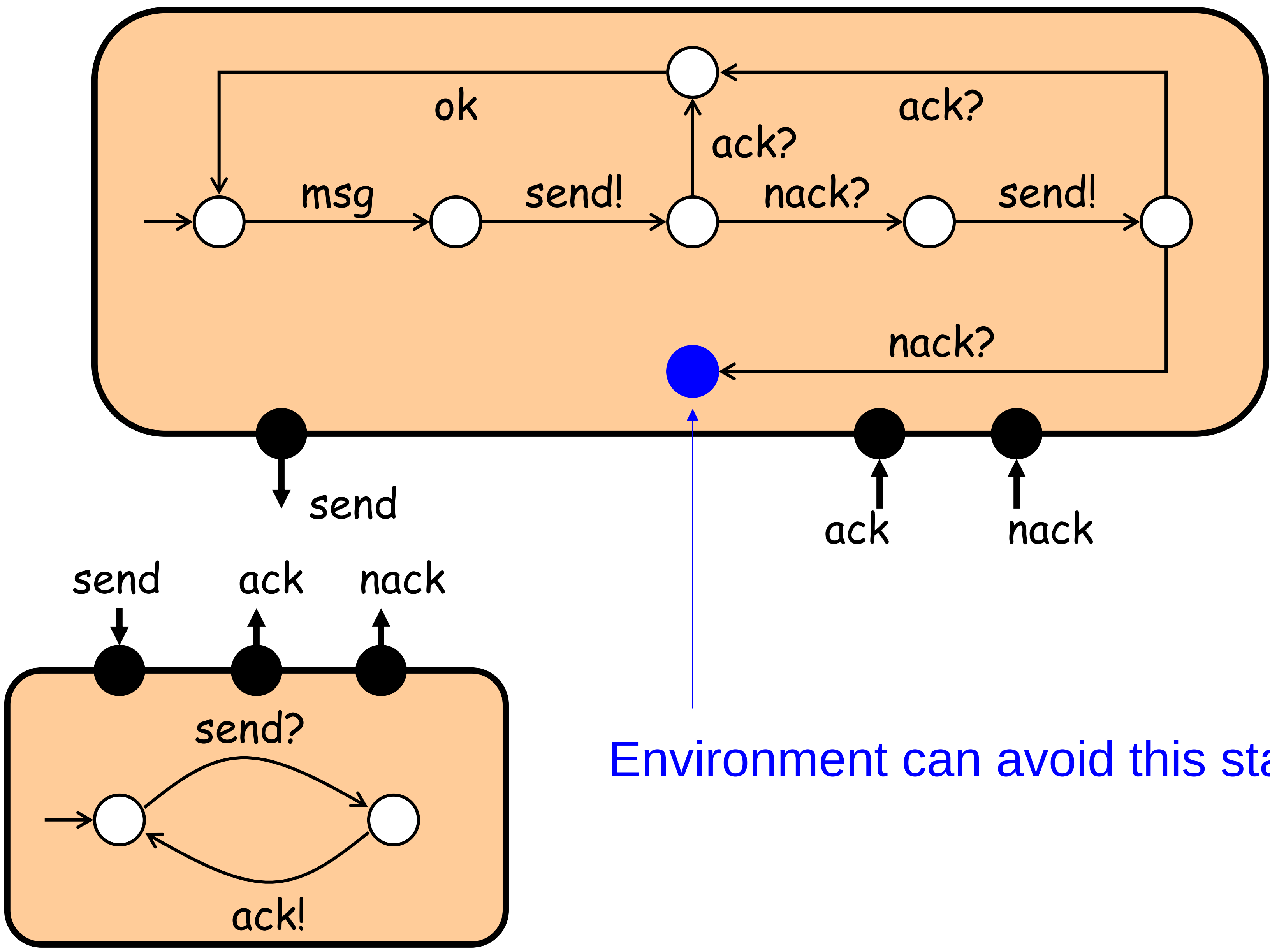
An *environment* for an interface automaton *F* is an interface automaton *E* such that

1. *F* is composable with *E*
2. $A^I_F = A^O_E$
3. *Error(F,E)* = ;

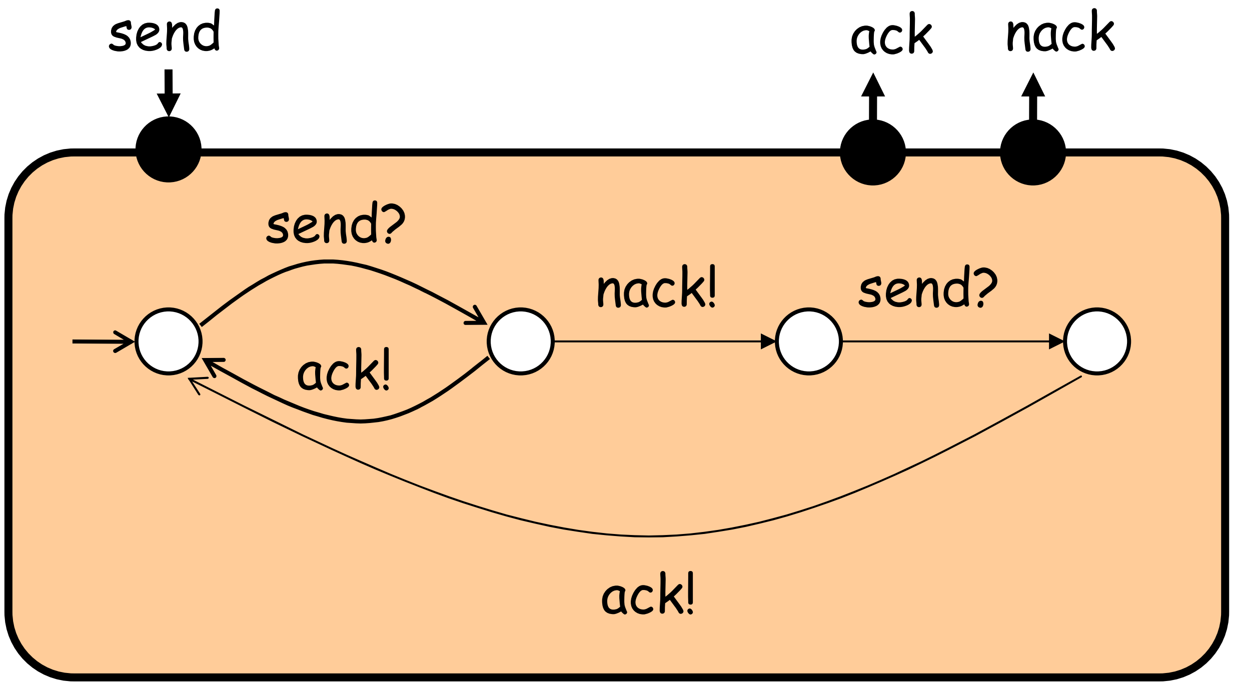
A *helpful* environment for two composable interface automata *F* and *G* is an environment *E* for the product *F* £ *G* such that

4. no state in *Error(F,G)* £ Q_E is reachable in (*F* £ *G*) £ *E*

Two interface automata *F* and *G* are *compatible* if they are composable and there exists a helpful environment.



Environment can avoid this state.



The most general helpful environment.

Computing the Composite Interface Automaton

1. Construct product automaton.

Computing the Composite Interface Automaton

1. Construct product automaton.
2. Mark error states as incompatible.

Computing the Composite Interface Automaton

1. Construct product automaton.
2. Mark error states as incompatible.
3. Until no more incompatible states can be added: mark state q as incompatible if the environment cannot prevent an incompatible state to be entered from q.

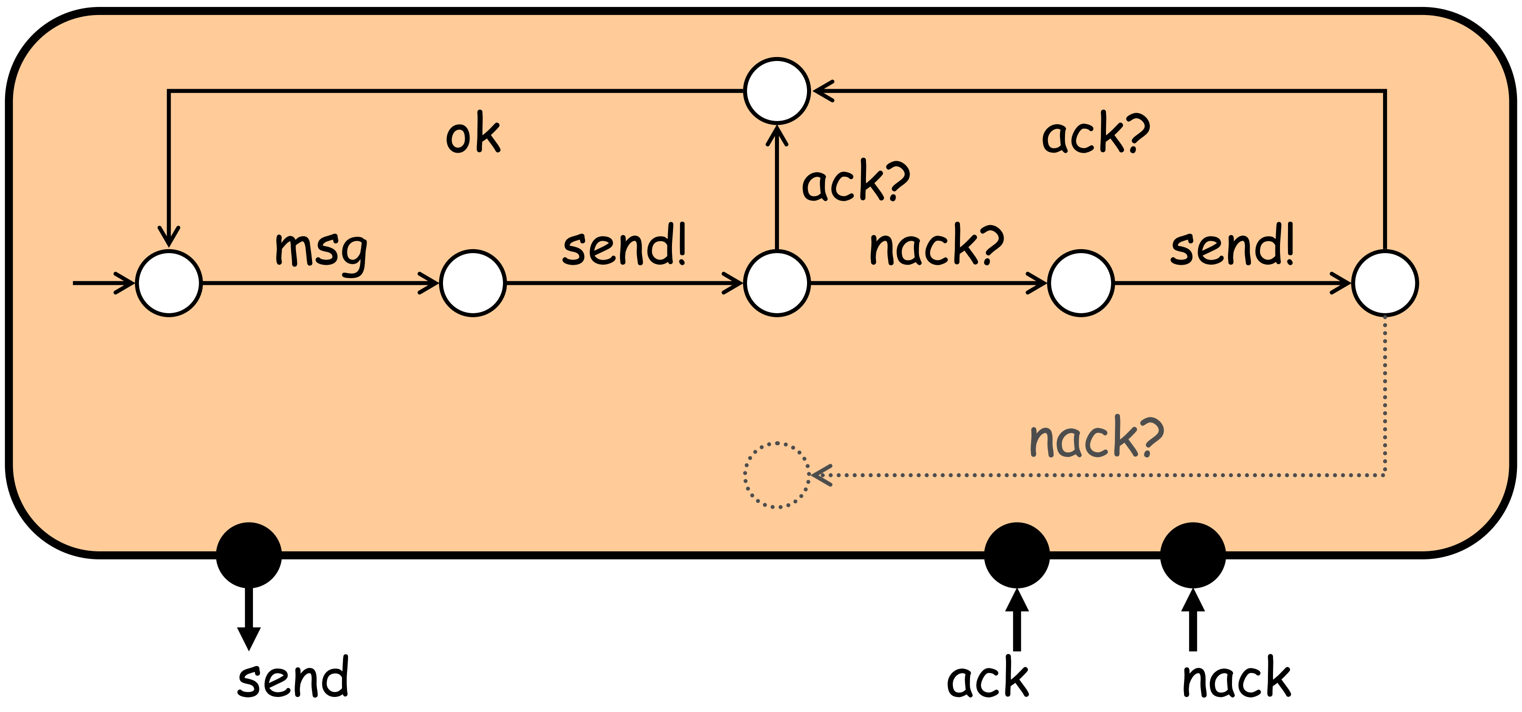
Computing the Composite Interface Automaton

1. Construct product automaton.
2. Mark error states as incompatible.
3. Until no more incompatible states can be added: mark state q as incompatible if the environment cannot prevent an incompatible state to be entered from q.
4. If the initial state is incompatible, then the two interfaces are incompatible. Otherwise, the composite interface is the product automaton without the incompatible states.

Computing the Composite Interface Automaton

1. Construct product automaton.
2. Mark error states as incompatible.
3. Until no more incompatible states can be added: mark state q as incompatible if the environment cannot prevent an incompatible state to be entered from q.
4. If the initial state is incompatible, then the two interfaces are incompatible. Otherwise, the composite interface is the product automaton without the incompatible states.

This procedure computes the most general helpful environment as the most general strategy of the environment to avoid error states.



The composite interface automaton.