Learning Assumptions for Compositional Verification

J. M. Cobleigh, D. Giannakopoulou and C. S. Pasareanu

TACAS 2003

Verification course, May 15, 2017

Main Limitation of Model Checking:

The state explosion problem

- The number of states in the system model grows exponentially with
  - the number of variables
  - the number of components in the system
- A solution to state explosion problem: **Compositional Verification**

Compositional Verification

- **Inputs:**
  - composite system $M_1 \| M_2$
  - property P
- **Goal**: check if $M_1 \| M_2 \models P$

- First attempt: "**divide and conquer**" approach
- Problem: usually impossible to verify each component separately.
  - a component is typically designed to satisfy its requirements in *specific* **environments** (contexts)

Compositional Verification

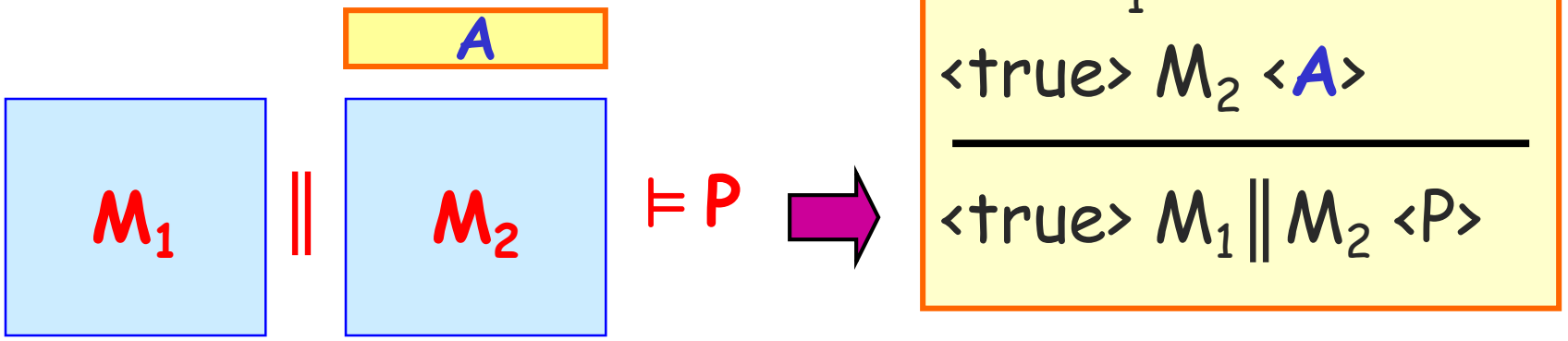
➔ **Assume-Guarantee (AG)** paradigm:
introduces **assumptions** representing a component's environment

Instead of: Does **component** satisfy **property**?
Ask: Under **assumption** A on its environment, does the component **guarantee** the property?

<A> M <P>: whenever **M** is part of a system satisfying the **assumption A**, then the system must also **guarantee P**

Useful AG Rule for Safety Properties

1. check if a component M_1 guarantees P when it is a part of a system satisfying assumption A.
2. discharge assumption: show that the remaining component M_2 (the environment) satisfies A.



Assume-Guarantee

$$\frac{\langle A \rangle\ M_1\ \langle P \rangle \qquad \langle true \rangle\ M_2\ \langle A \rangle}{\langle true \rangle\ M_1 \| M_2\ \langle P \rangle}$$

- Crucial element: **assumption** A.
- Has to be **strong enough** to eliminate violations of P, but also **general enough** to reflect the environment M_2 appropriately.
- requires non-trivial human input in defining assumptions.

How to automatically construct assumptions ?

Outline

- √ Motivation
- Setting
- Automatic Generation of Assumptions for the AG Rule
- Learning algorithm
- Assume-Guarantee with Learning
- Example

Labeled Transition Systems (LTS)

- Act – set of **observable actions**
  - LTSs communicate using observable actions
- τ – **local** \ internal action

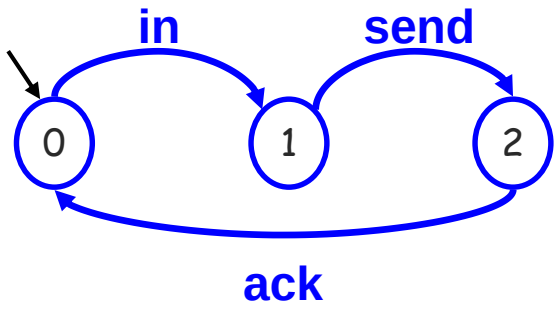
LTS $M = (Q, q^0, \alpha M, \delta)$

- **Q** : finite non-empty set of **states**
- **$q^0 \in Q$** : **initial state**
- **$\alpha M \subseteq$** Act : observable actions
- **$\delta \subseteq Q \times (\alpha M \cup \{\tau\}) \times Q$** : **transition relation**

- A computation π of M is a sequence of states and actions:

  $(q_0, a_0, q_1, a_1, \ldots, a_k, q_k)$ such that for every $i \geq 0$, $(q_i, a_i, q_{i+1}) \in \delta$

Labeled Transition Systems (LTS)



Traces:
<in>, <in, send>,...

- **Trace** of an LTS **M** : finite sequence of observable actions occurring on a computation, starting at the initial state.

- **L(M) =** the **Language** of **M** : the set of all traces of M.
  - **L(M) is prefix closed**

Parallel Composition $M_1 \| M_2$

- Components **synchronize** on **common observable** actions (communication).
- The remaining actions are **interleaved**.

$M_1 = (Q_1, q^0_1, \alpha M_1, \delta_1)$, $M_2 = (Q_2, q^0_2, \alpha M_2, \delta_2)$

➔ $M_1 \| M_2 = (Q, q^0, \alpha M, \delta)$

- $Q = Q_1 \times Q_2$
- $q^0 = (q^0_1, q^0_2)$
- $\alpha M = \alpha M_1 \cup \alpha M_2$

Transition Relation

Synchronization on $a \in \alpha M_1 \cap \alpha M_2$:

$(q_1, a, q_1') \in \delta_1$ and $(q_2, a, q_2') \in \delta_2$:

$\rightarrow ((q_1, q_2),\ a,\ (q_1', q_2')) \in \delta$

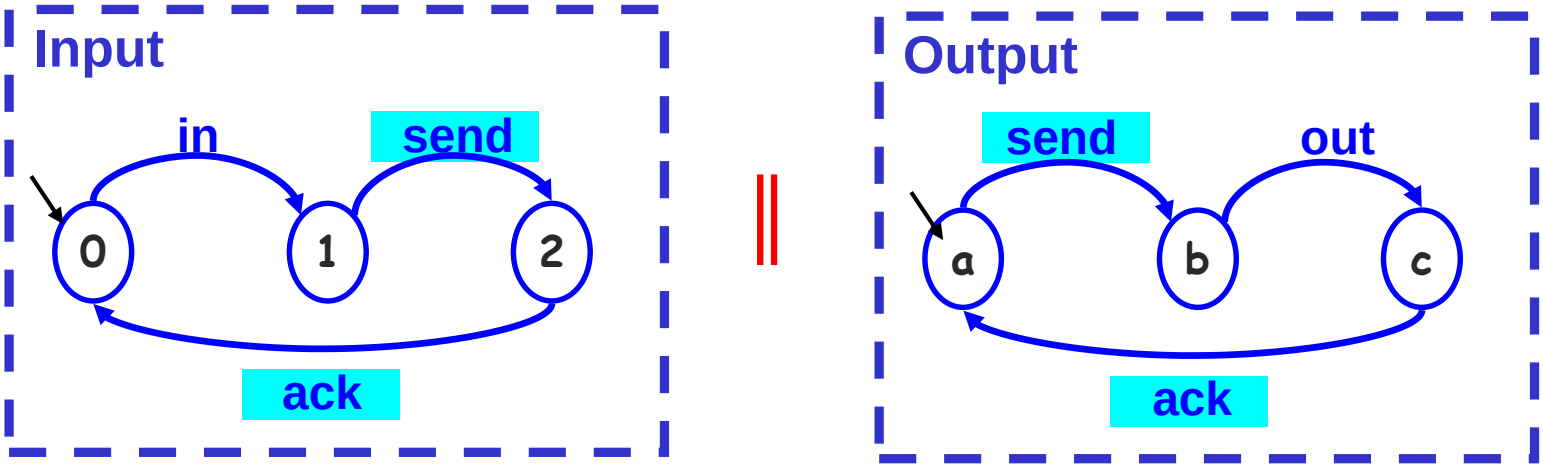
Interleaving on $a \in (\alpha M \setminus (\alpha M_1 \cap \alpha M_2)) \cup \{\tau\}$:

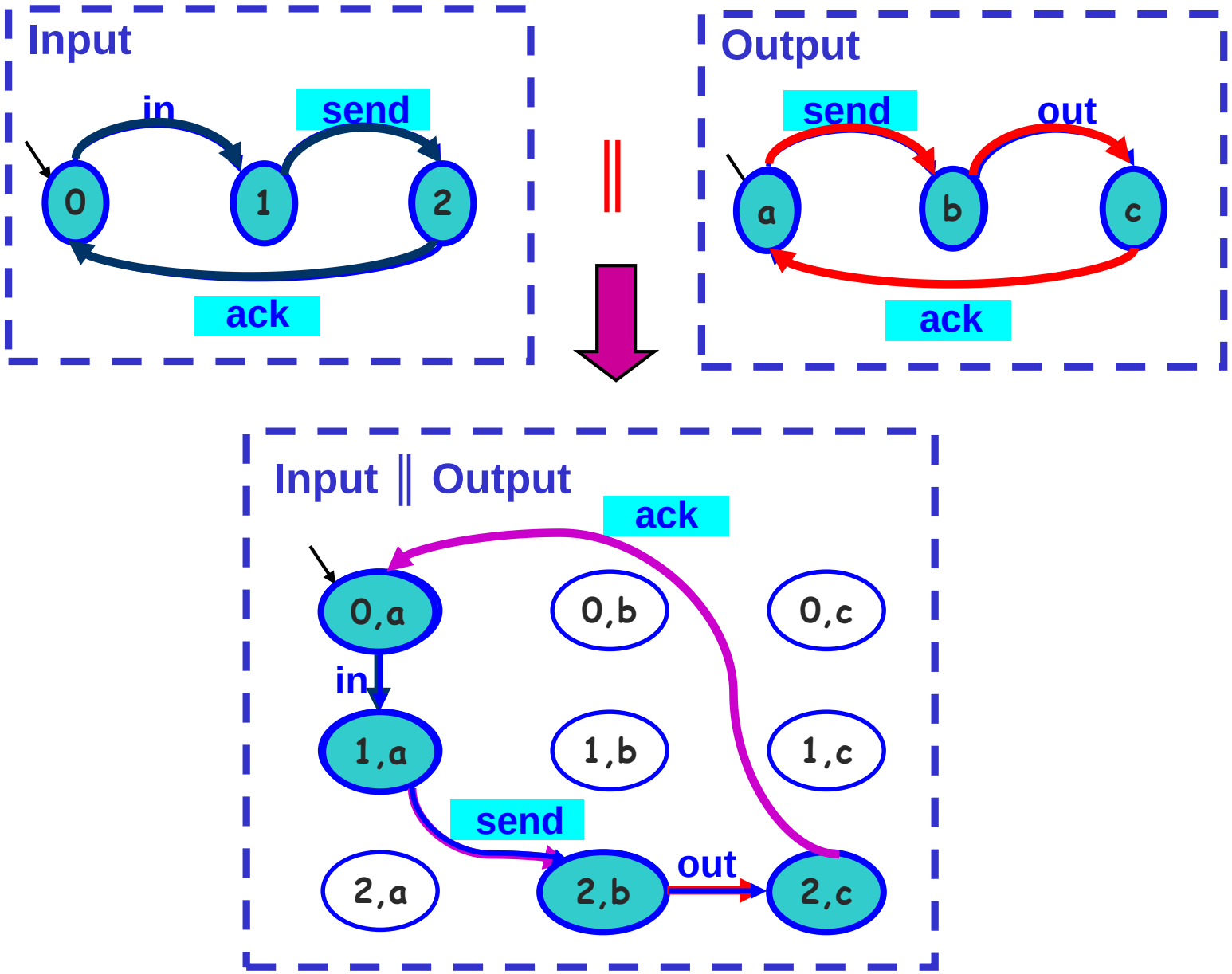
- $(q_1, a, q_1') \in \delta_1$ and $a \notin \alpha M_2$:

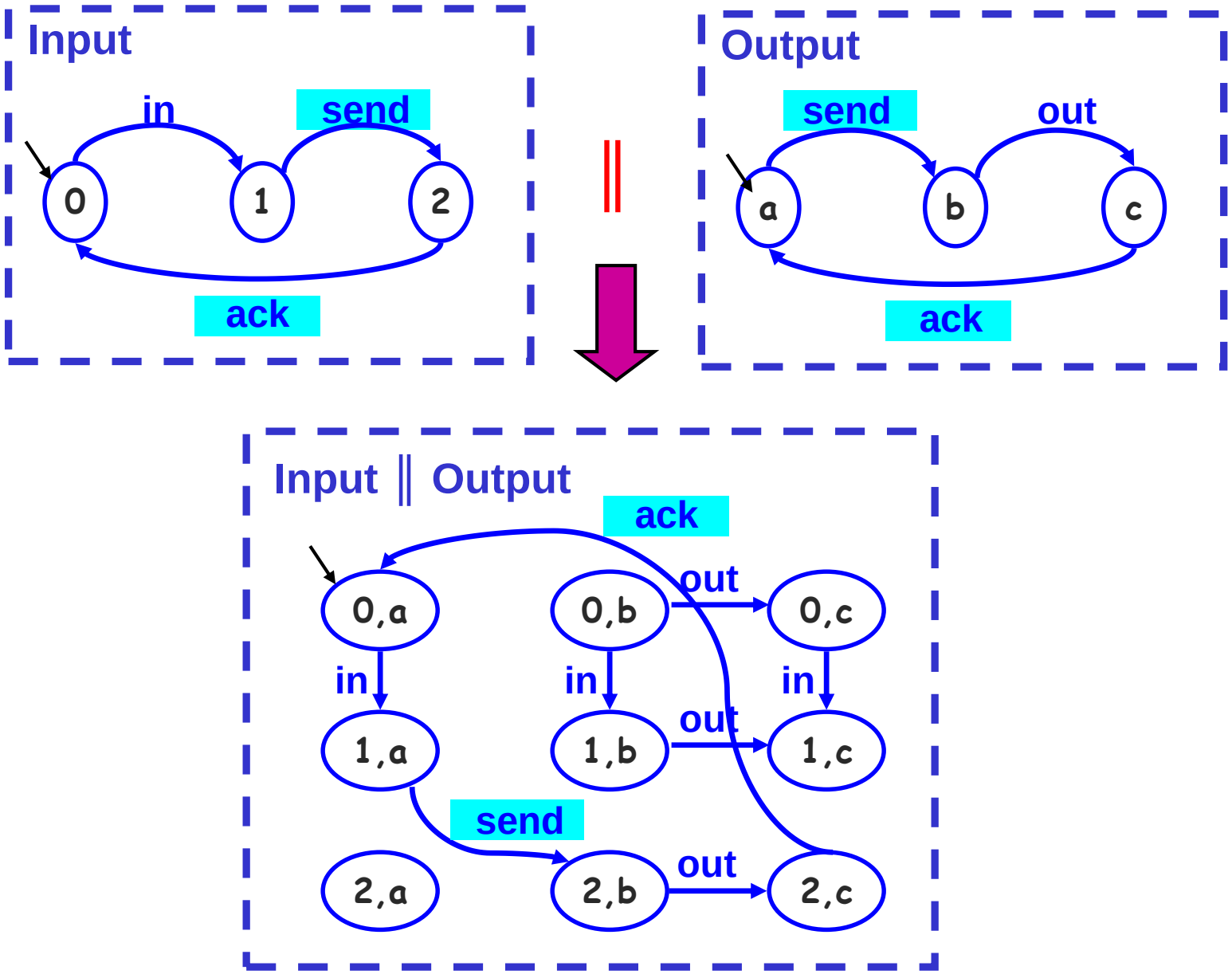
  $\rightarrow ((q_1, q_2),\ a,\ (q_1', q_2)) \in \delta$ for any $q_2 \in Q_2$
- $(q_2, a, q_2') \in \delta_2$ and $a \notin \alpha M_1$:

  $\rightarrow ((q_1, q_2),\ a,\ (q_1, q_2')) \in \delta$ for any $q_1 \in Q_1$

Example




Input
in
send
0
1
2
ack
Output
send
out
a
b
c
ack
Input ‖ Output
ack
0,a
0,b
0,c
in
1,a
1,b
1,c
send
2,a
2,b
out
2,c


Input
in
send
0
1
2
ack
||
Output
send
out
a
b
c
ack
Input || Output
ack
0,a
0,b
0,c
out
in
in
in
1,a
1,b
1,c
out
send
2,a
2,b
2,c
out

- **Lemma:**

  A trace $\sigma = a_0,\ldots,a_k$ is a trace of M iff

  - Its restriction $\sigma\uparrow\alpha M_1$ is a trace of M_1, and
  - Its restriction $\sigma\uparrow\alpha M_2$ is a trace of M_2

For $\Sigma \subseteq Act$, $\sigma\uparrow\Sigma$ is the trace obtained from σ by removing all occurrences of actions $a \notin \Sigma$.

Example: ⟨in, send, ack⟩↑{in, ack} = ⟨in, ack⟩

Safety Properties

Also expressed as LTSs, but of a special kind:

Safety LTS :

- **Deterministic**:
  - Does **not** contain τ-transitions, and
  - every state has **at most one** outgoing transition for each action:
    $(q,a,q'), (q,a,q'') \in \delta \rightarrow q' = q''$

Safety Properties

For a **safety LTS, P**:

- **L(P)** describes the set of **legal** (acceptable) behaviors over αP.

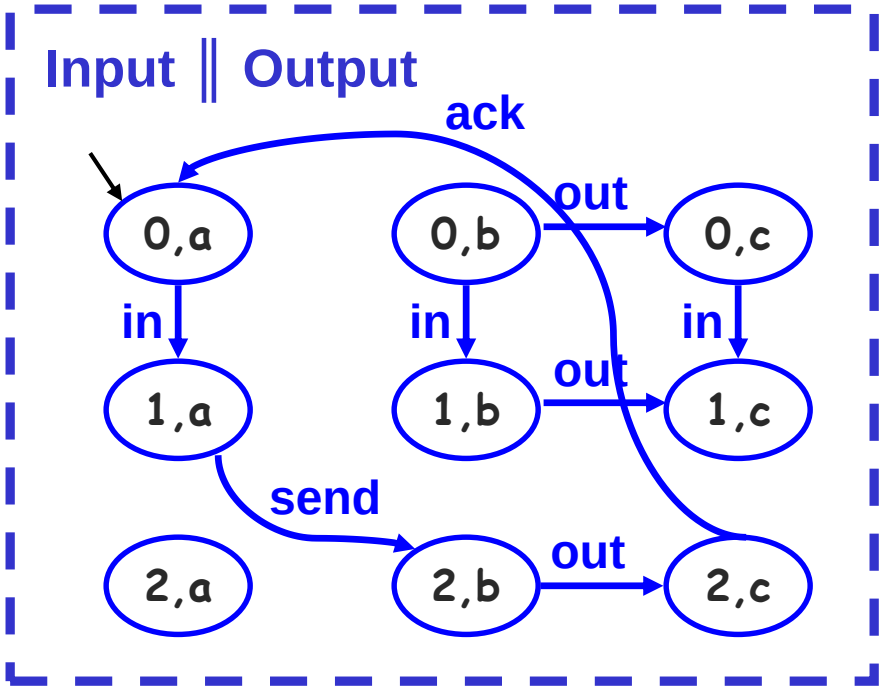
Language Containment
$L(M) \uparrow \alpha P \subseteq L(P)$

$M \models P$ iff $\forall \sigma \in L(M) : (\sigma \uparrow \alpha P) \in L(P)$

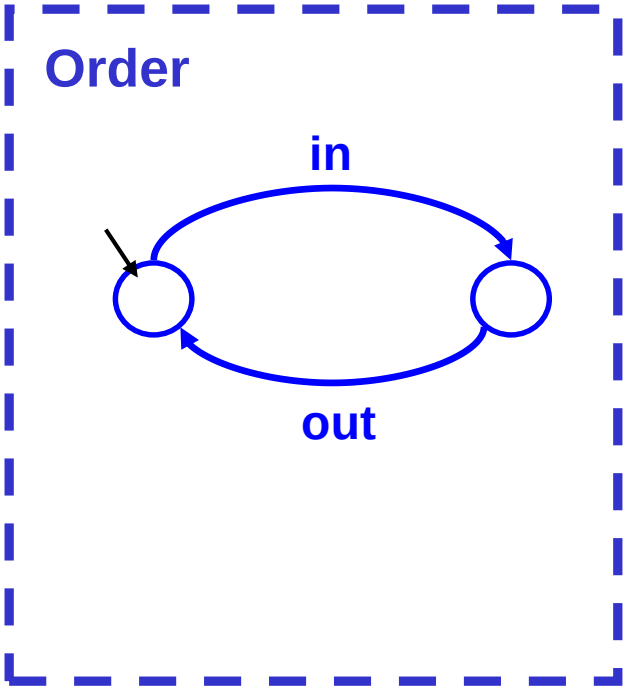
Note that, since we check $M \models P$, $\alpha P \subseteq \alpha M$

Example

P



$\models ?$



Pref(<in,send,out,ack>*)

↑{in,out}

$\subseteq ?$

√

Pref(<in,out>*)

Model Checking $M \models P$

Safety LTS P ➔ an **Error LTS**, P_{err} :

- "traps" violations with special **error state** π.

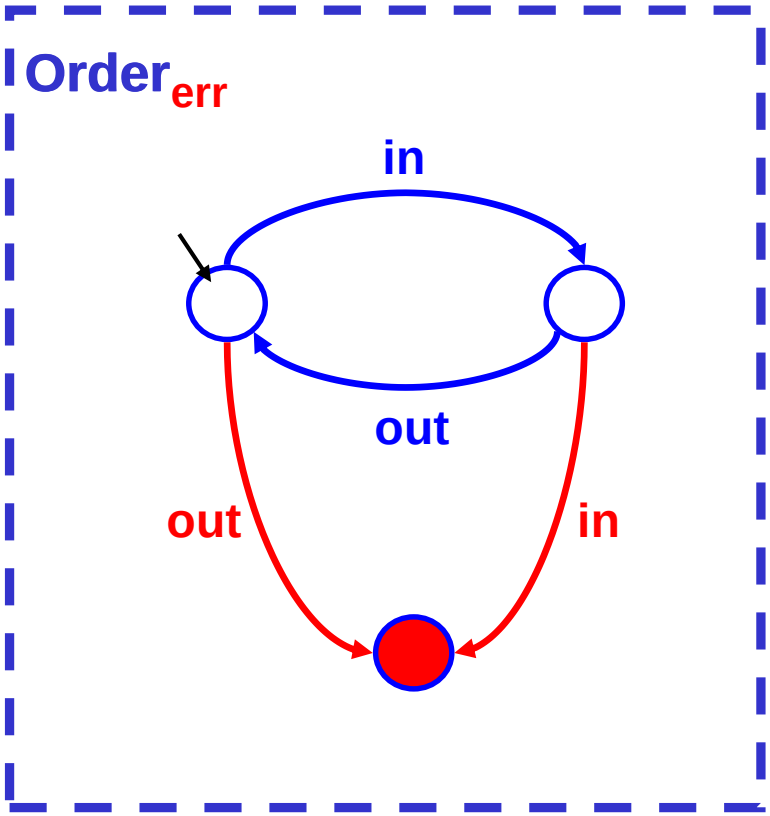
$P = (Q, q^0, \alpha P, \delta)$

➔ $P_{err} = (Q \cup \{\pi\}, q^0, \alpha P, \delta')$, where:

$$\delta' = \delta \cup \{(q,a,\pi) \mid a \in \alpha P \text{ and } \nexists q' \in Q: (q,a,q') \in \delta\}$$

- Error LTS is **complete**.
- π is a deadend state: has no outgoing transitions.

Example



Model Checking $M \models P$

In automata:
$M \models \varphi$ iff
$L(A_M \cap \overline{A}_\varphi) = \emptyset$

Theorem:

- $M \models P$ iff π is **unreachable** in $M \parallel P_{err}$

Remark: composition $M \parallel P_{err}$ is defined as before with a small exception:

If the **target state** of a transition in P_{err} is π, so is the target state of the corresponding transitions in the **composed** system $M \parallel P_{err}$

Transition Relation

δ_1: M, δ_2: P_{err}

δ: $M \| P_{err}$

Synchronization on $a \in \alpha M_1 \cap \alpha M_2$:

$(q_1, a, q_1') \in \delta_1$ and $(q_2, a, q_2') \in \delta_2$:

$\rightarrow ((q_1, q_2),\ a,\ (q_1', q_2')) \in \delta$

Interleaving on $a \in (\alpha M \setminus (\alpha M_1 \cap \alpha M_2)) \cup \{\tau\}$:

- $(q_1, a, q_1') \in \delta_1$ and $a \notin \alpha M_2$:

  $\rightarrow ((q_1, q_2),\ a,\ (q_1', q_2)) \in \delta$ for any $q_2 \in Q_2$
- $(q_2, a, q_2') \in \delta_2$ and $a \notin \alpha M_1$:

  $\rightarrow ((q_1, q_2),\ a,\ (q_1, q_2')) \in \delta$ for any $q_1 \in Q_1$

Theorem:

$M \models P$ iff π is **unreachable** in $M \parallel P_{err}$

Recall that,

- $M \models P$ iff $\forall \sigma \in L(M) : (\sigma \uparrow \alpha P) \in L(P)$

- $M \parallel P_{err}$ synchronizes on αP

Theorem:

$M \models P$ iff π is **unreachable** in $M \parallel P_{err}$

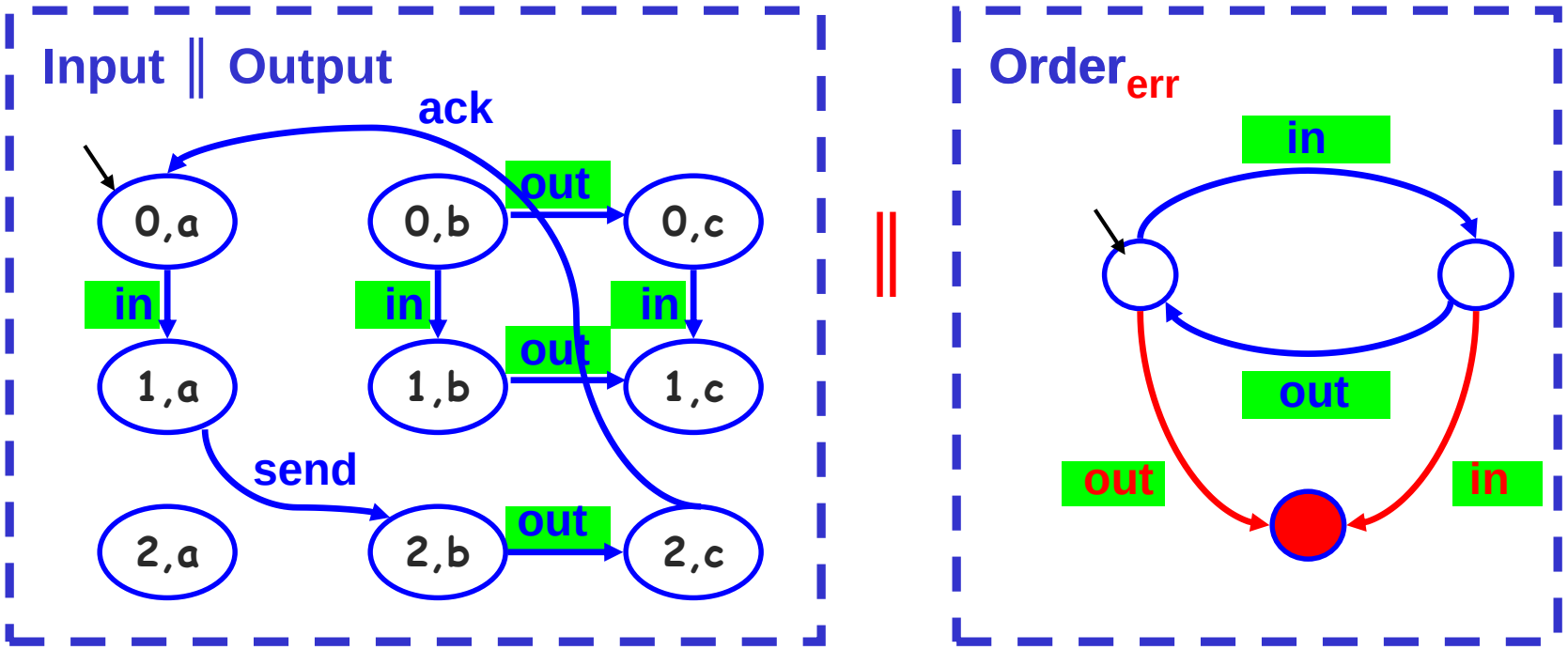
Proof sketch:

$M \parallel P_{err}$ has a transition on action $a \in \alpha P$ to π iff

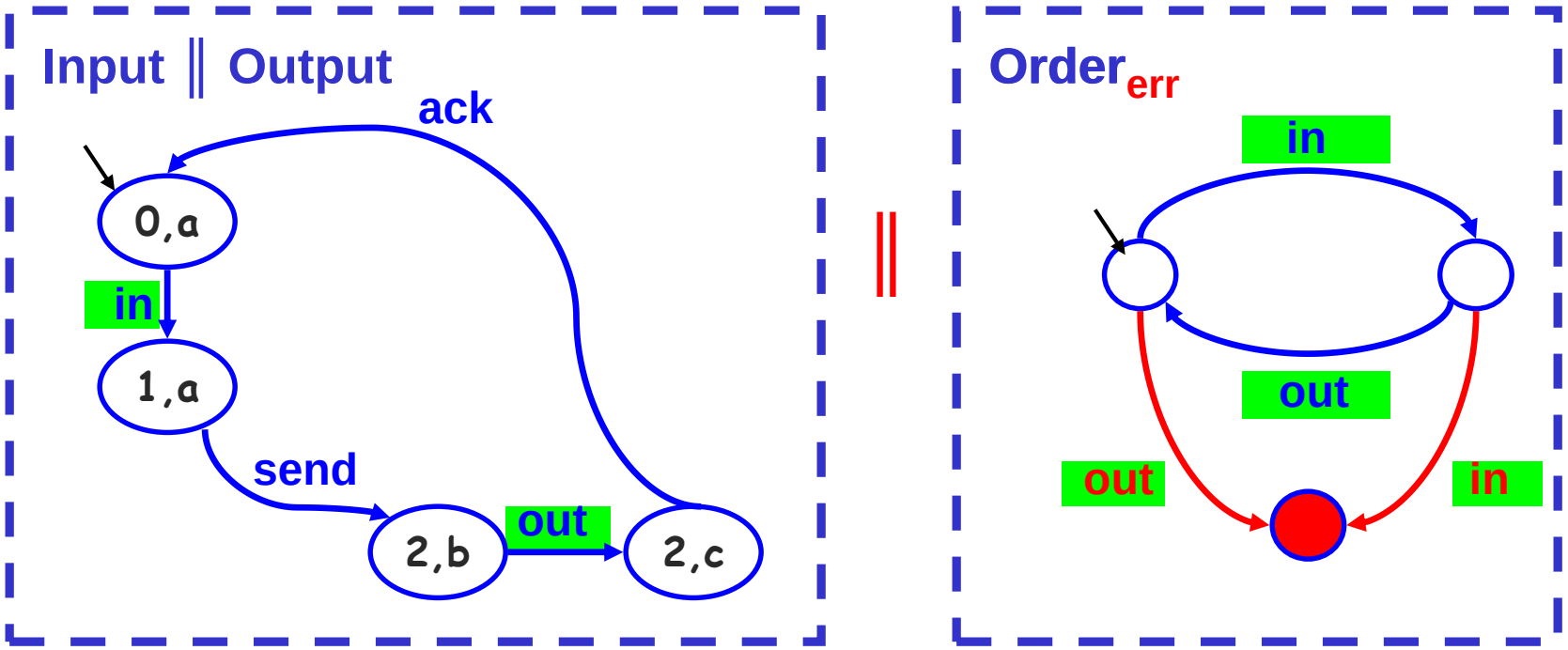
There is a reachable state in $M \parallel P_{err}$ from which

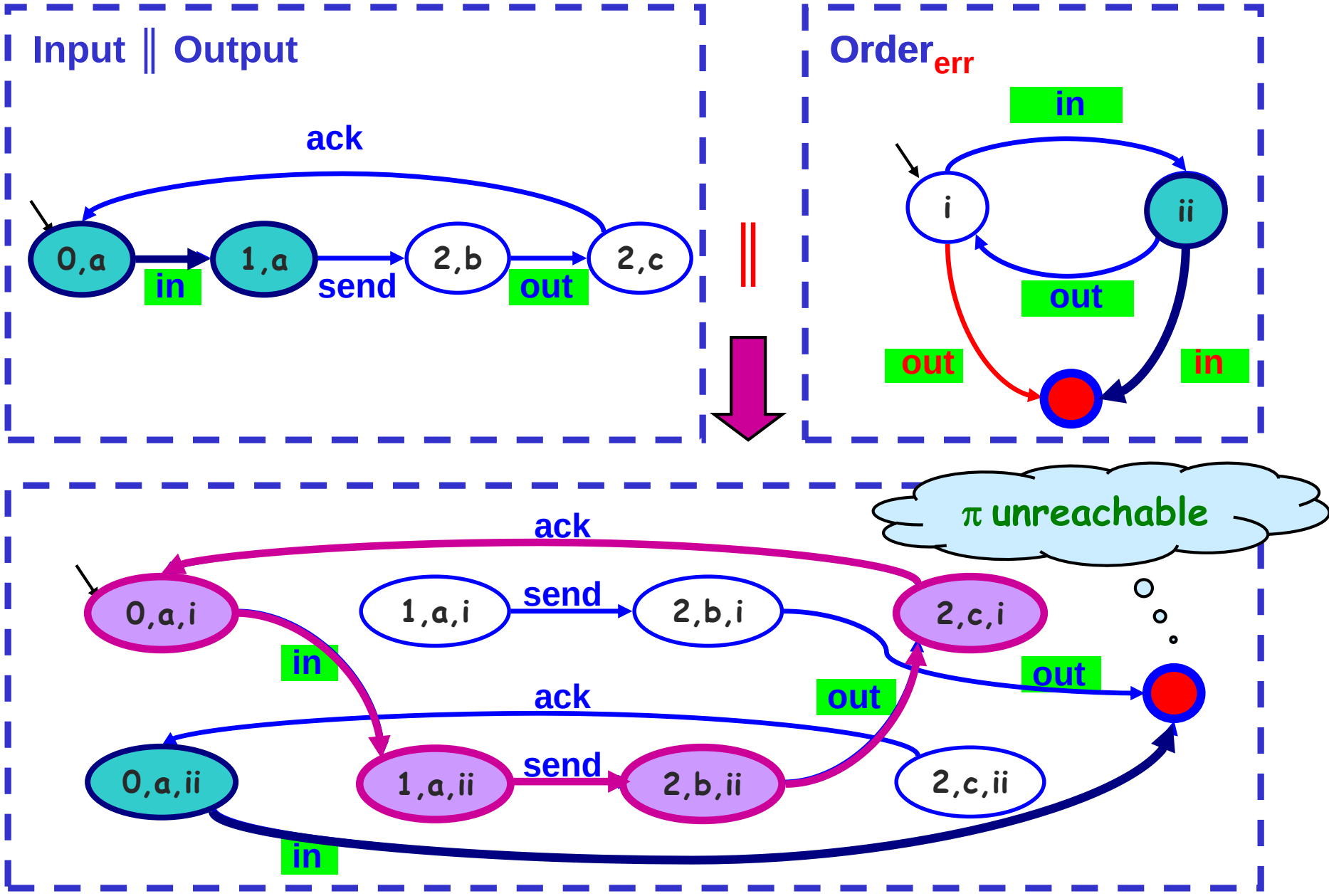
- M can take action a, but
- P cannot take action a
  - therefore a on P_{err} goes to π

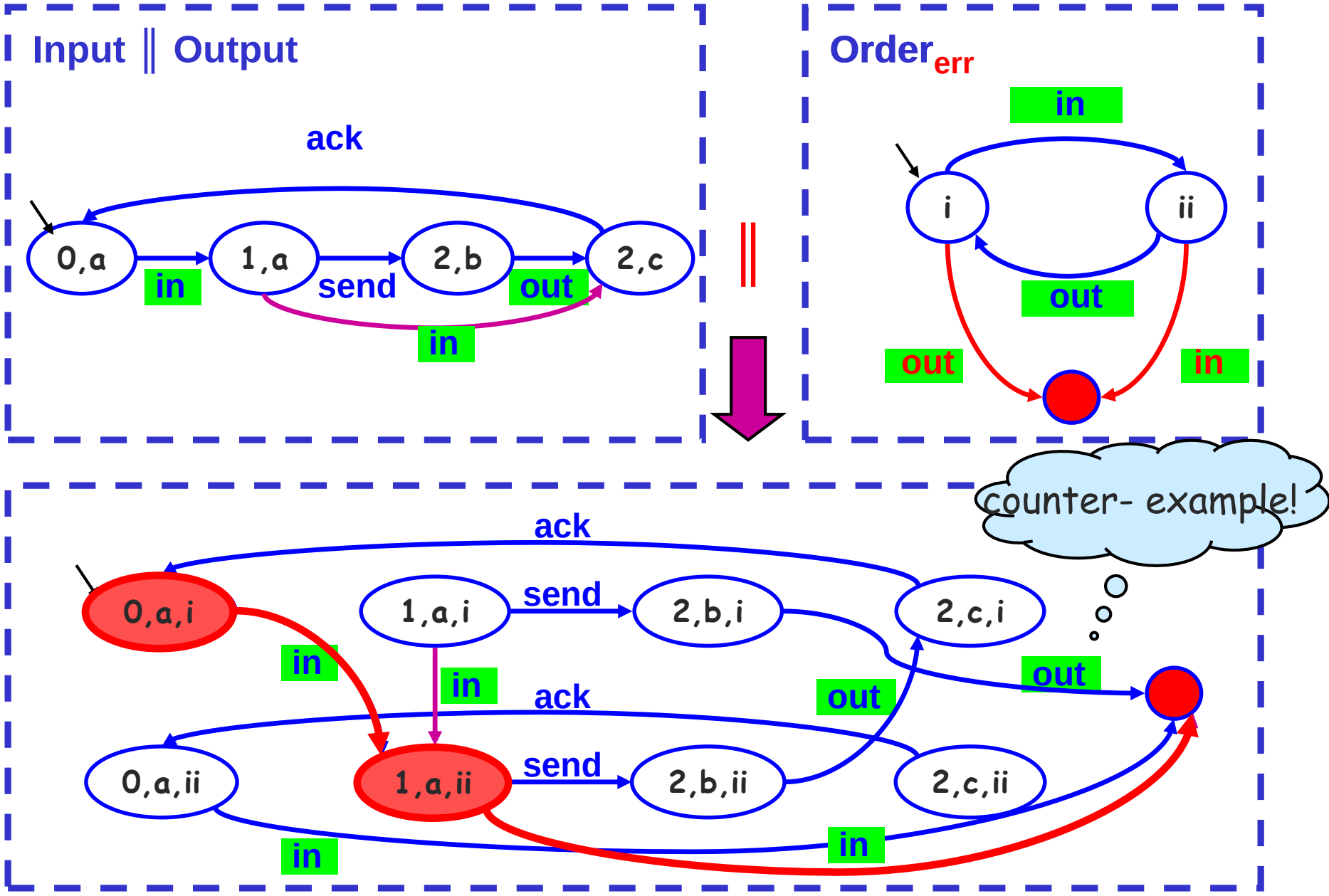
Example


Input ‖ Output
ack
0,a
0,b
0,c
out
in
in
in
1,a
1,b
1,c
out
send
2,a
2,b
2,c
out
‖
Order err
in
out
out
in

Example




Input ‖ Output
ack
0,a
in
1,a
send
2,b
out
2,c
‖
Order err
in
i
ii
out
out
in
π unreachable
ack
0,a,i
in
1,a,i
send
2,b,i
2,c,i
out
ack
out
0,a,ii
1,a,ii
send
2,b,ii
2,c,ii
in


Input ‖ Output
ack
0,a
in
1,a
send
2,b
out
2,c
in
‖
Order err
in
i
ii
out
out
in
counter- example!
ack
0,a,i
in
1,a,i
send
2,b,i
2,c,i
in
out
ack
out
0,a,ii
1,a,ii
send
2,b,ii
2,c,ii
in
in