Learning Assumptions for Compositional Verification

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

TACAS 2003

Verification course, May 22, 2017

Lecture 2

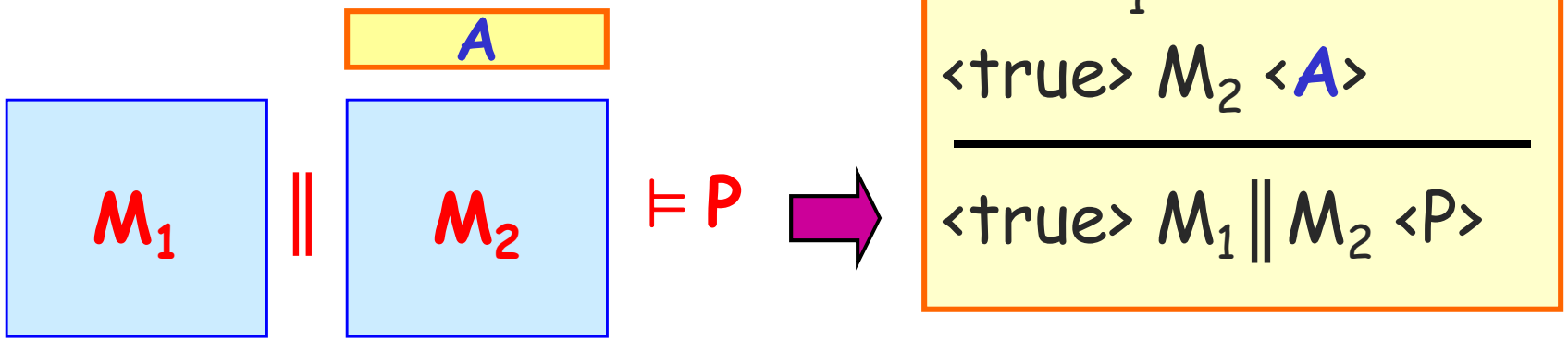
Compositional Verification

Basic building block:

<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.

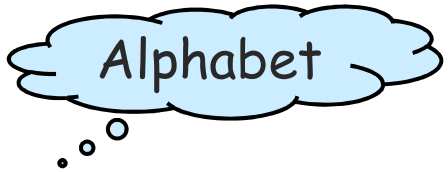
How to automatically construct assumptions ?

Labeled Transition Systems (LTS)

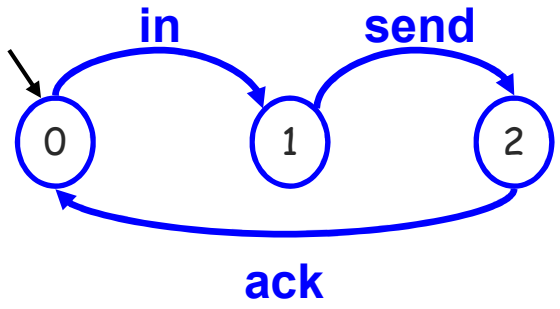
- Act – set of **observable actions**
  - LTSs communicate using observable actions
- τ – **local** \ internal action

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**



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.

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 $q_1 \in Q_1$

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 $M \models P$, $\alpha P \subseteq \alpha M$

Model Checking $M \models P$

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

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

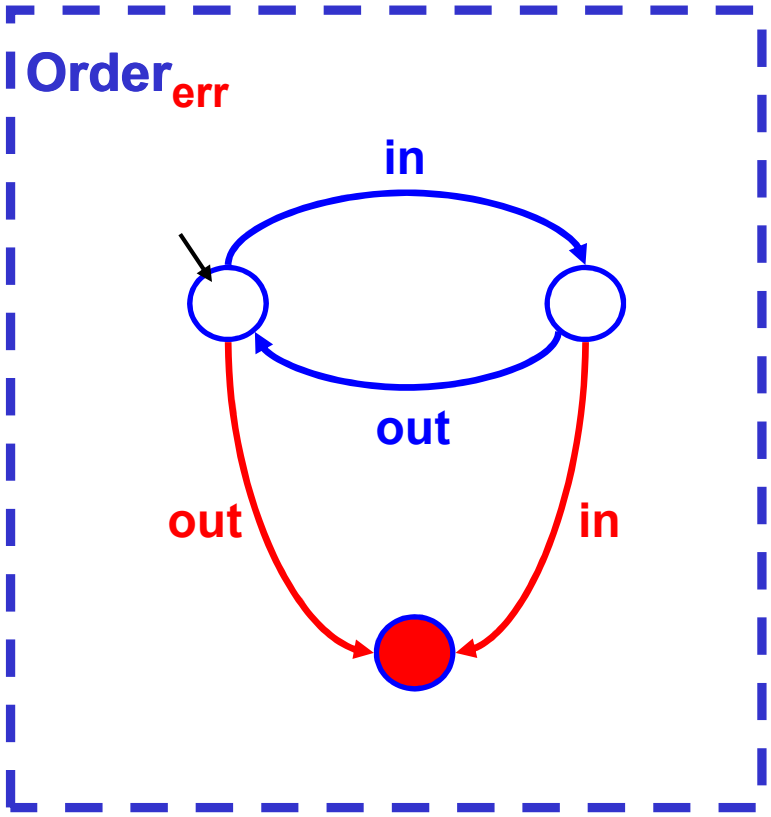
$P = (Q, q^0, \alpha P, \delta)$

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

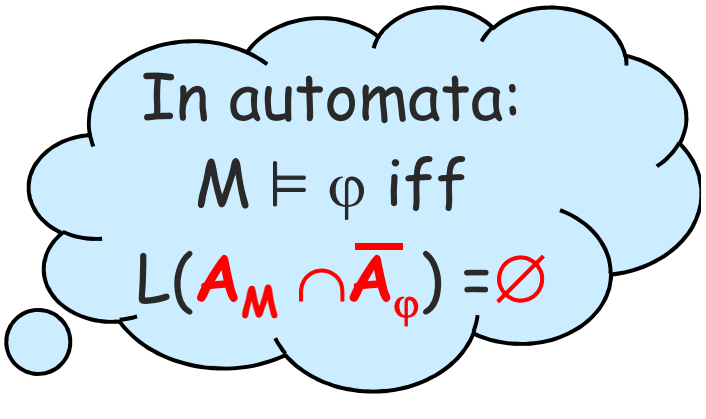
$$\delta' = \delta \cup \{(q,a,\pi) \mid a \in \alpha P \text{ and } \nexists q' \in Q: (q,a,q') \in \delta\}$$

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

Example



Model Checking $M \models P$



Theorem:

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

Recall that,

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

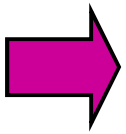
- $M \parallel P_{err}$ synchronizes on αP

End summary of lecture 1

Assume Guarantee Reasoning

- **Assumptions**: also expressed as **safety LTSs**.
- $\langle A \rangle\ M\ \langle P \rangle$ is true iff $A \parallel M \models P$
  i.e. π is unreachable in $A \parallel M \parallel P_{err}$

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



$$\frac{A \parallel M_1 \models P \qquad M_2 \models A}{M_1 \parallel M_2 \models P}$$

M_1, M_2 : LTSs

P, A : safety LTSs

Outline

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

Important concept: Weakest assumption for Σ

Definition: Let

M_1 - LTS component

P – safety property

Σ - interface alphabet of M_1 and environment

The weakest assumption A_w is a deterministic LTS such that:

- $\alpha A_w = \Sigma \cup (\alpha P - \alpha M_1)$
- For every M'_2 such that $\alpha A_w \subseteq \alpha M'_2$
  $\langle true \rangle\ M_1 || M'_2\ \langle P \rangle$ iff $\langle true \rangle\ M'_2\ \langle A_w \rangle$

Note that, $\langle true \rangle\ M_1 || A_w \langle P \rangle$ holds

Weakest assumption for M_1, M_2, P

- Interface alphabet for M_1 and M_2 and P:

  $\Sigma_I = (\alpha M_1 \cup \alpha P) \cap \alpha M_2$

- The weakest assumption for M_1, M_2 and P is defined as the weakest assumption for Σ_I

Observation 1

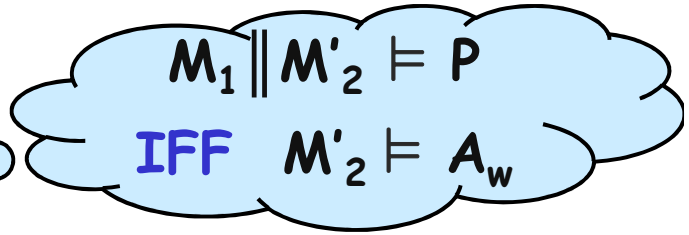
$$\frac{A \parallel M_1 \models P \qquad M_2 \models A}{M_1 \parallel M_2 \models P}$$

- AG Rule will return **conclusive** results with the **Weakest assumption A_w** under which M_1 satisfies P:
- Under assumption A_w on M_1's environment, M_1 satisfies P , i.e. $A_w \parallel M_1 \models P$
- **Weakest**: forall env. M'_2: $M_1 \parallel M'_2 \models P$ **IFF** $M'_2 \models A_w$

➔ Given A_w, to check if $M_1 \parallel M_2 \models P$, check if M_2 meets the assumption A_w.

sufficient and **necessary** assumption on M_1's env.

How to Obtain A_w ?



- Given **module M_1, property P** and M_1's **interface** with its environment $\Sigma = (\alpha M_1 \cup \alpha P) \cap \alpha M_2$:

A_w describes exactly **all traces** σ over Σ such that in the context of σ, M_1 satisfies P

- Expressible as a safety LTS
- Can be computed algorithmically

Drawback of using A_w : if the computation runs out of memory, no assumption is obtained.

Observation 2

$$\frac{A \parallel M_1 \models P \qquad M_2 \models A}{M_1 \parallel M_2 \models P}$$

- No need to use the **weakest** env. assumption A_w.
- AG rule might be applicable with **stronger** (less general) assumption.

➔ Instead of finding A_w:

- Use **learning** algorithm to **learn** A_w .
- Use candidates A_i produced by learning algorithm as **candidate assumptions**: try to apply AG rule with A_i.

Given a Candidate Assumption A_i

$$\frac{A \parallel M_1 \models P \qquad M_2 \models A}{M_1 \parallel M_2 \models P}$$

If $A_i \parallel M_1 \models P$ does **not** hold:

Assumption A_i is not tight enough

➔ need to **strengthen** A_i

Given a Candidate Assumption A_i

$$\frac{A \parallel M_1 \models P \qquad M_2 \models A}{M_1 \parallel M_2 \models P}$$

Suppose $A_i \parallel M_1 \models P$ holds:

If $M_2 \models A_i$ also holds ➔ $M_1 \parallel M_2 \models P$ **verified!**

Otherwise: $\sigma \in L(M_2)$ but $(\sigma{\uparrow}\alpha\Sigma) \notin L(A_i)$ -- **cex**

P violated by M_1 in the context of **cex=**$(\sigma{\uparrow}\alpha\Sigma)$?

Yes: real violation !

➔ $M_1 \parallel M_2 \models P$ **falsified** !

No: spurious violation.

➔ need to find better approximation of A_w.

"cex $\parallel M_1$" $\not\models P$?

Given a Candidate Assumption A_i

$$\frac{A \parallel M_1 \models P \qquad M_2 \models A}{M_1 \parallel M_2 \models P}$$

Note:

If $A_i \parallel M_1 \models P$ does **not** hold:

➔ traces are removed from A_i

If $M_2 \models A_i$ does **not** hold, and cex is spurious:

➔ traces are added to A_i

Changes to A_i's are non-monotonic but we get "closer" to A_w

"cex $\|$ M_1" $\models$ P ?

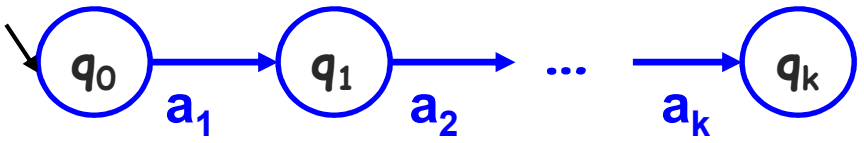
"cex $\in L(A_w)$" ?

Compose M_1 with LTS A_{cex} over $\alpha\Sigma$

cex=$a_1,\ldots a_k$ ➔ A_{cex} :

$Q=\{q_0,\ldots,q_k\}$, $q^0 = q_0$

$=\{ (q_i,a_{i+1},q_{i+1}) \mid 0 \leq i < k \}$



Model check $A_{cex} \| M_1 \models P$

Is π reachable in $A_{cex} \| M_1 \| P_{err}$?

yes → Real violation

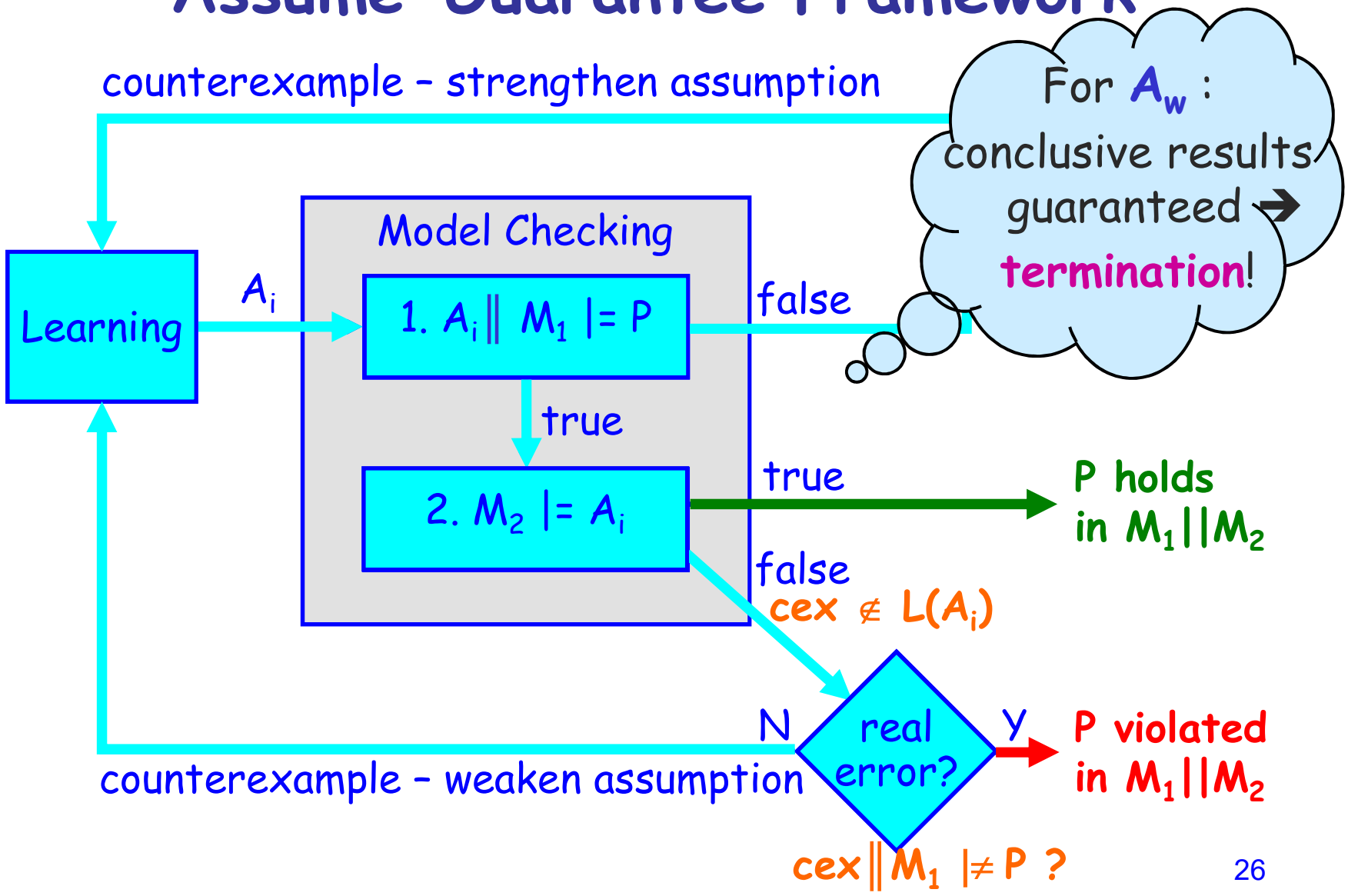
no → Spurious cex

Alternatively: simulate **cex** on $M_1 \| P_{err}$

cex $\| M_1 \models P$ **iff** when **cex** is simulated on $M_1 \| P_{err}$

it **cannot lead** to π (**error**) state.

Assume-Guarantee Framework



Outline

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

Learning Algorithm for DFA – L*

- **L*:** by Angluin, improved by Rivest & Schapire
- learns an unknown regular language **U**
- produces a **D**eterministic **F**inite-state **A**utomaton (**DFA**) **C** such that **L(C) = U**

DFA M = (Q, q^0, aM, δ, F) :

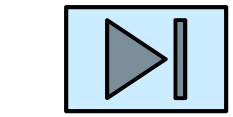
- **Q, q^0, aM, δ** : as in deterministic LTS
- **F $\subseteq$ Q** : accepting states
- **L(M) = {σ | $\delta(q^0, \sigma) \in F$}**

Learning Algorithm for DFA – L*

- L* interacts with a **Teacher** to answer two types of questions:
  - **Membership queries**: is string **σ** in **U** ?
  - **Conjectures**: for a candidate DFA C_i, is $L(C_i) = U$?
    - answers are **(true)** or **(false + counterexample)**

Equivalence Queries

Conjectures C_1, C_2, … converge to C



Outline

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

Assume-Guarantee with Learning

Reminder:

- Use **learning** algorithm to **learn** A_w .
- Use candidates produced by learning as **candidate assumptions** A_i for AG rule.

In order to **use L*** to produce assumptions A_i:

- Show that **L(A_w)** is **regular**
- Translate **DFA** to **safety LTS** (assumption)
- Implement the **teacher**

$L(A_w)$ is Regular

- A_w is expressible as a **safety LTS**
- Translation of **safety LTS** *A* into **DFA** *C*:

  $A = (Q, q^0, \alpha M, \delta)$ ➔

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

➔ There **exists** a **DFA** *C* s.t. $L(C) = L(A_w)$

➔ L* can be used to learn a DFA that accepts $L(A_w)$

DFA to Safety LTS

Weakest assumption is prefix-closed

- **DFAs** C_i returned by L* are **complete**, **minimal** and in this setting also **prefix-closed**:
  - if $\sigma \in L(C_i)$, then every **prefix** of σ is **also** in $L(C_i)$.

➔ C_i contains a single **non-accepting** state q_{nf} and **no accepting** state is reachable from q_{nf}.

➔ To get a **safety LTS** simply remove non-accepting state and all its ingoing transitions :

$$C_i = (Q \cup \{q_{nf}\},\ q^0,\ \alpha M,\ \delta,\ Q) \rightarrow$$

$$A_i = (Q,\ q^0,\ \alpha M,\ \delta \cap (Q \times \alpha M \times Q))$$

Implementing the Teacher

A_w is **unknown**…

$\sigma \in L(A_w)$ iff in context of σ, M_1 satisfies **P**

- **Membership** query: $\sigma \in L(A_w)$?

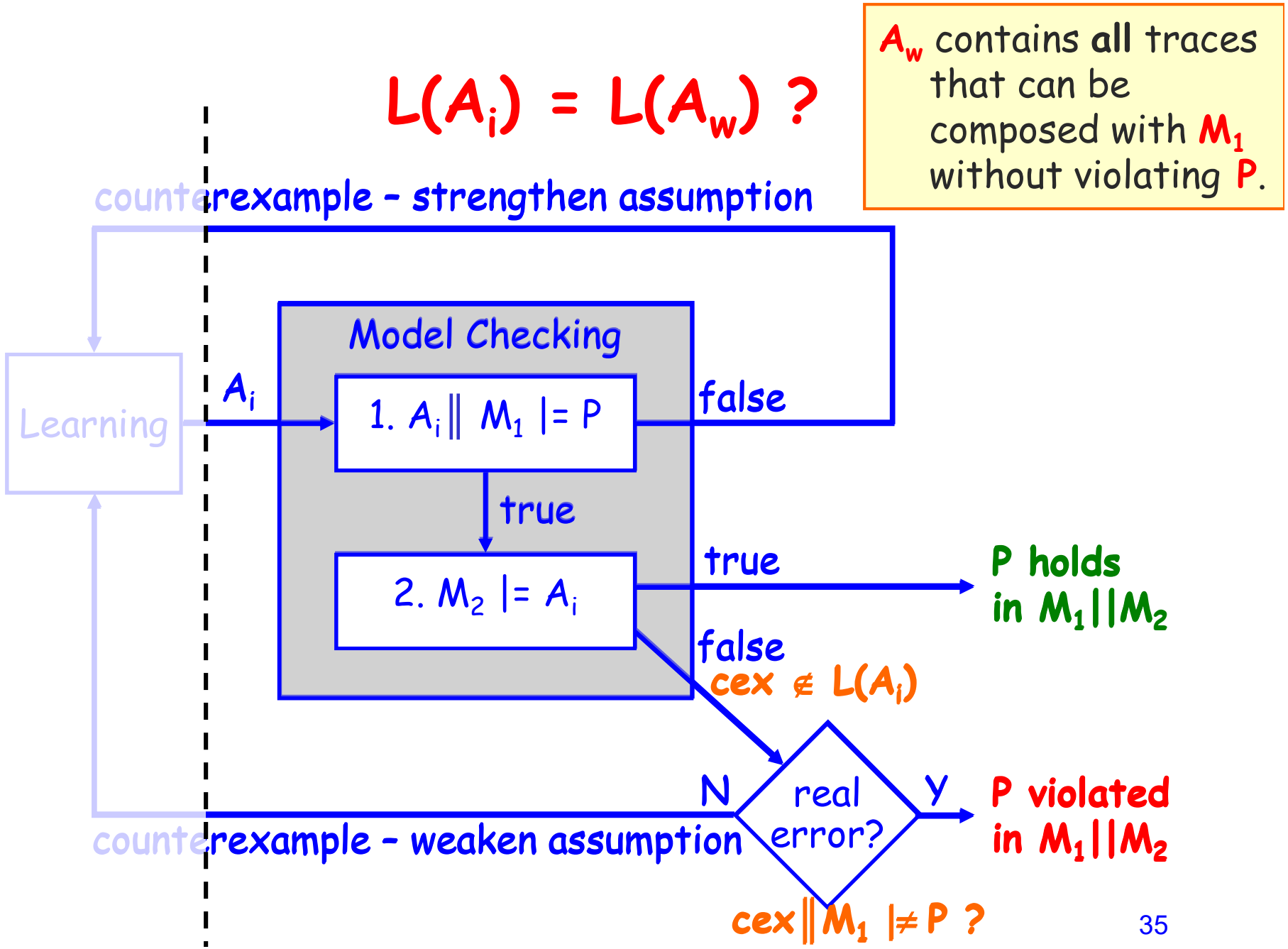
  ➔ Check if "$\sigma \| M_1$" $\models$ **P** :

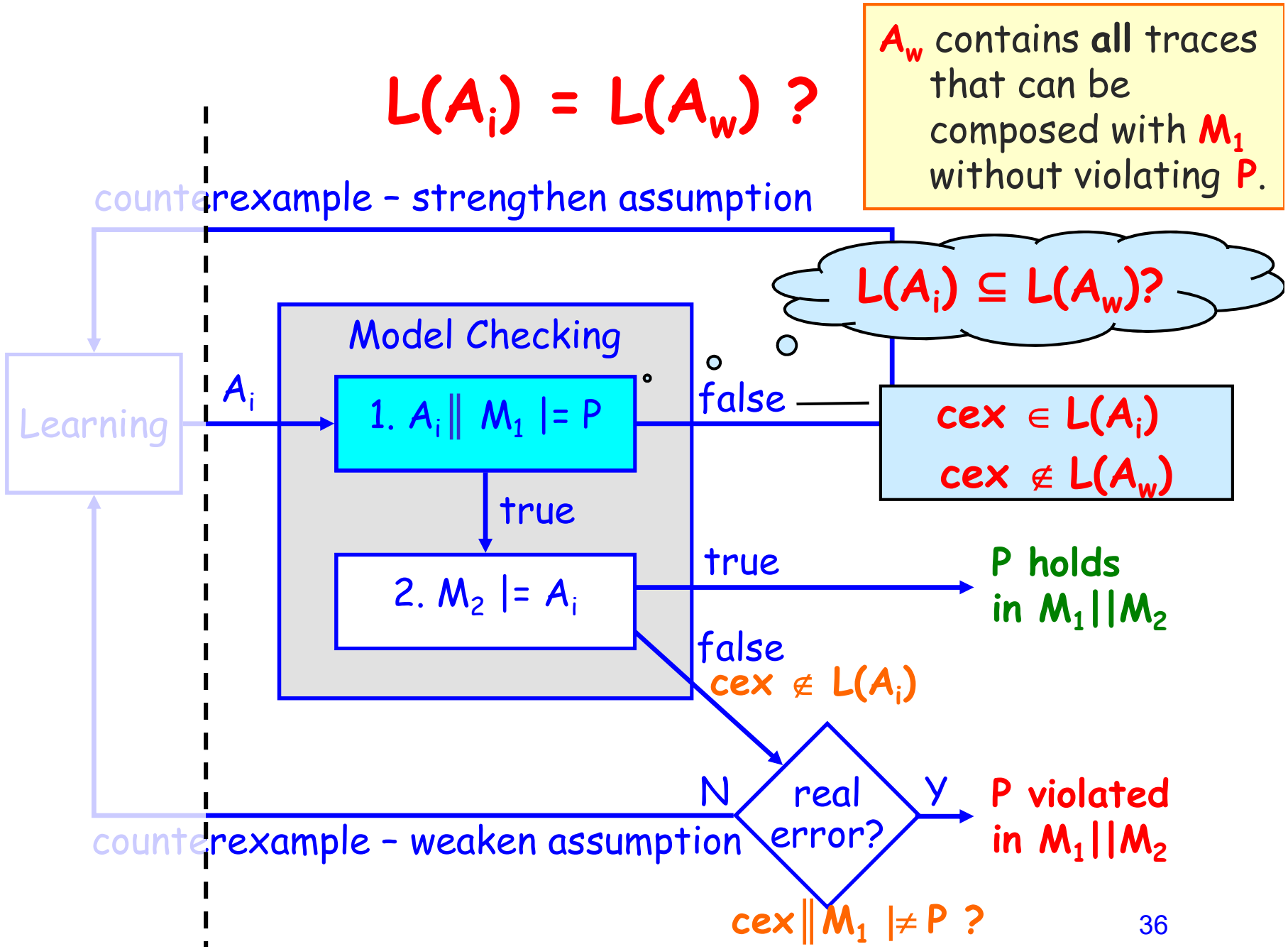
  - **Model checking**: is π **reachable** in $A_\sigma \| M_1 \| P_{err}$?

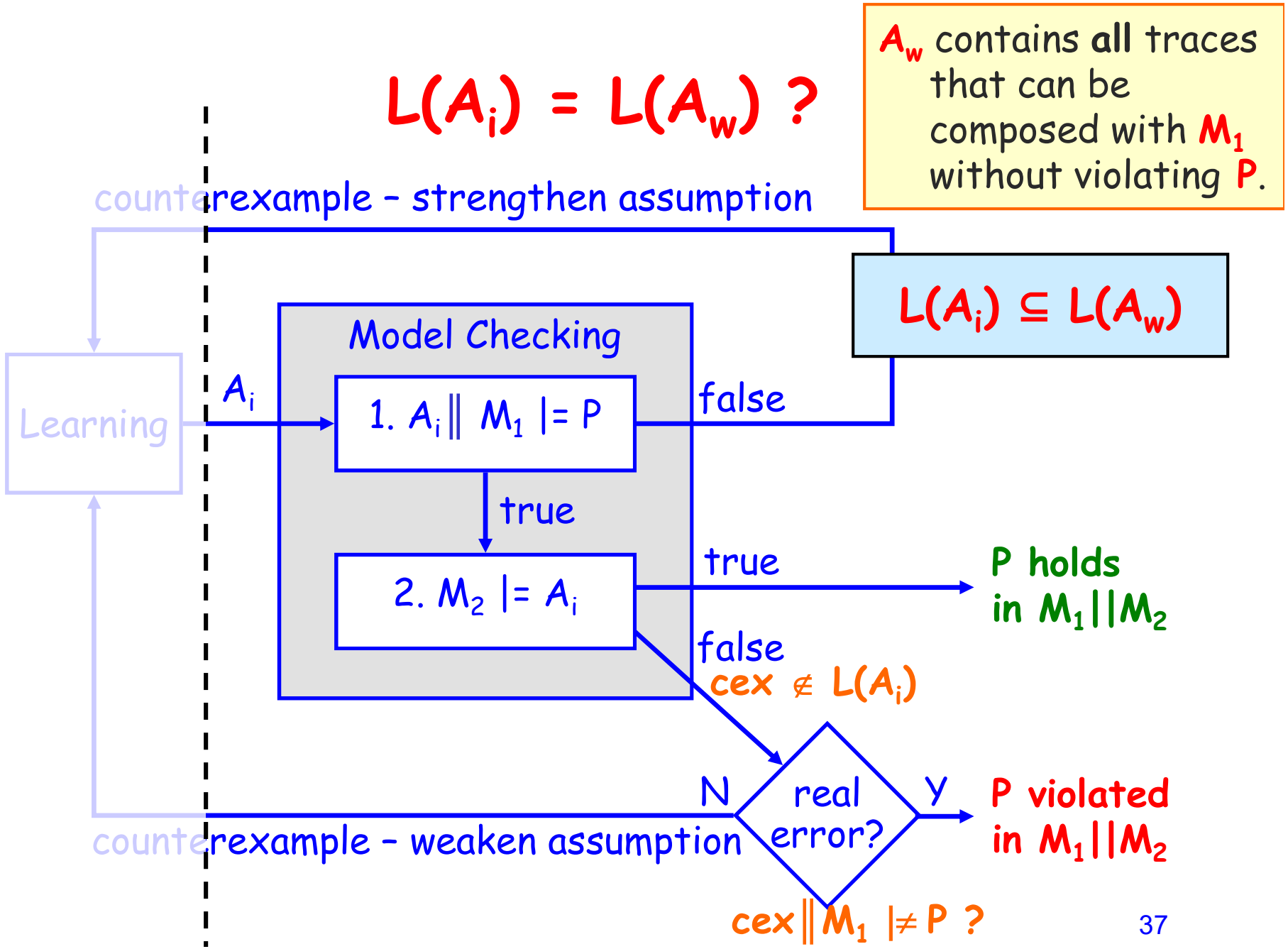
  or - **Simulation**: is π (**error**) state **reachable** when simulating σ on $M_1 \| P_{err}$?

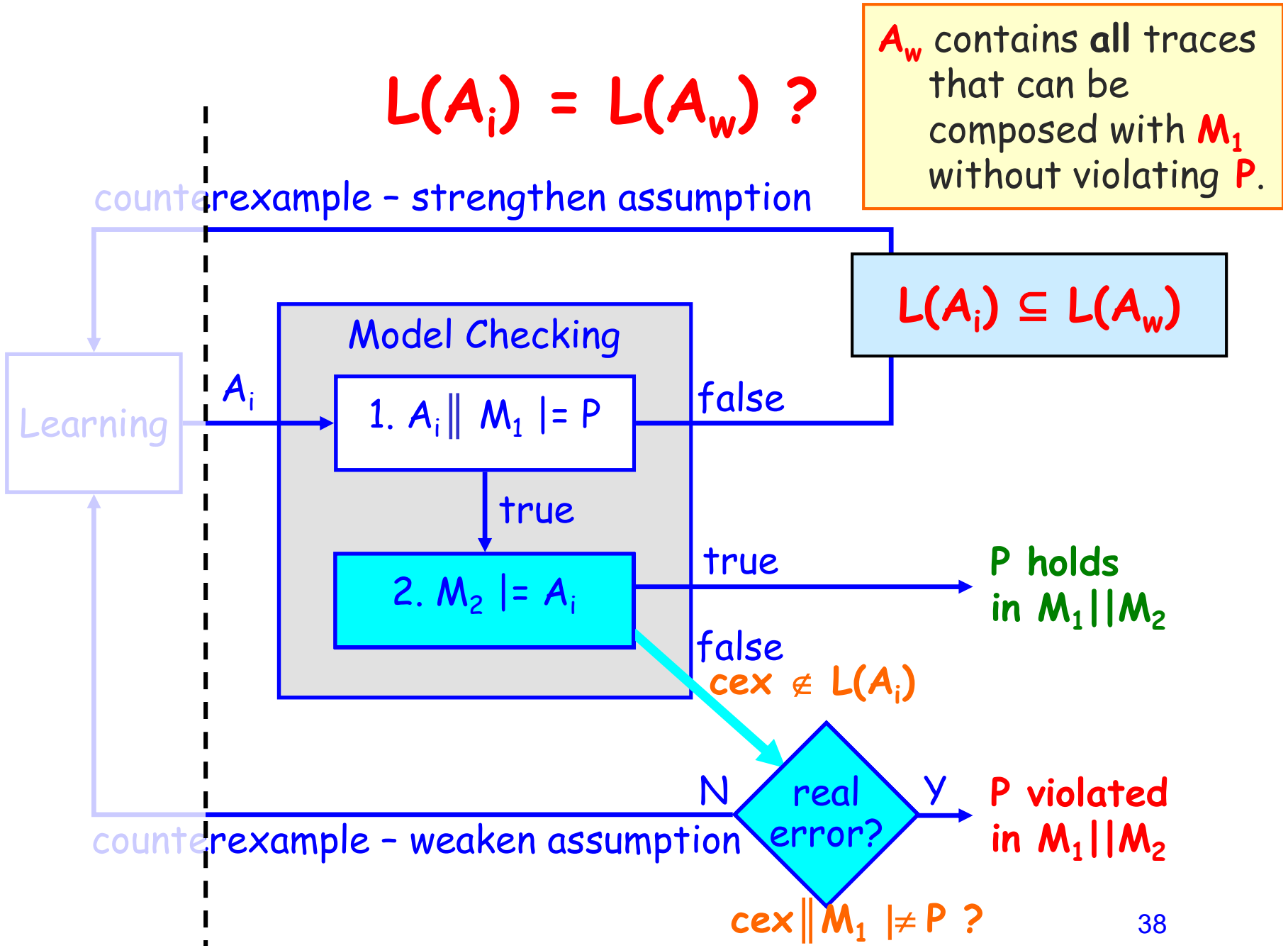
- **Equivalence** query: $L(C_i) = L(A_w)$?

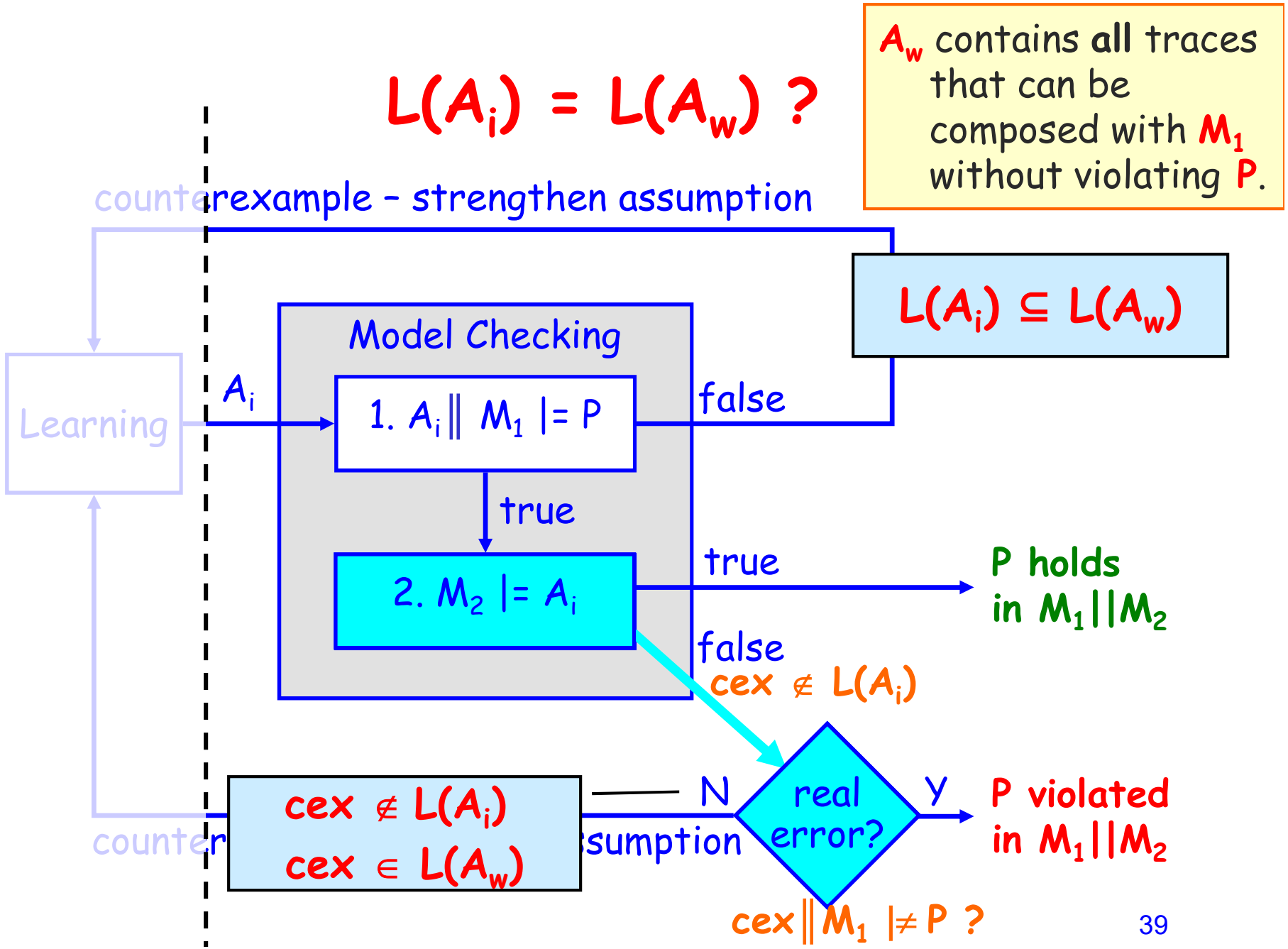
  - Translate C_i into a safety LTS A_i
  - Use it as candidate assumption for **AG rule**


$L(A_i) = L(A_w)$?
A_w contains all traces that can be composed with M_1 without violating P.
counterexample – strengthen assumption
Learning
Model Checking
A_i
1. $A_i \| M_1 \models P$
false
true
2. $M_2 \models A_i$
true
P holds in $M_1 \| M_2$
false
cex $\notin L(A_i)$
N
real error?
Y
P violated in $M_1 \| M_2$
counterexample – weaken assumption
cex $\| M_1 \not\models P$?


$L(A_i) = L(A_w)$?
A_w contains all traces that can be composed with M_1 without violating P.
counterexample – strengthen assumption
$L(A_i) \subseteq L(A_w)$?
Learning
Model Checking
A_i
1. $A_i \| M_1 \models P$
false
$cex \in L(A_i)$
$cex \notin L(A_w)$
true
2. $M_2 \models A_i$
true
P holds in $M_1 || M_2$
false
$cex \notin L(A_i)$
N
real error?
Y
P violated in $M_1 || M_2$
counterexample – weaken assumption
$cex \| M_1 \not\models P$?

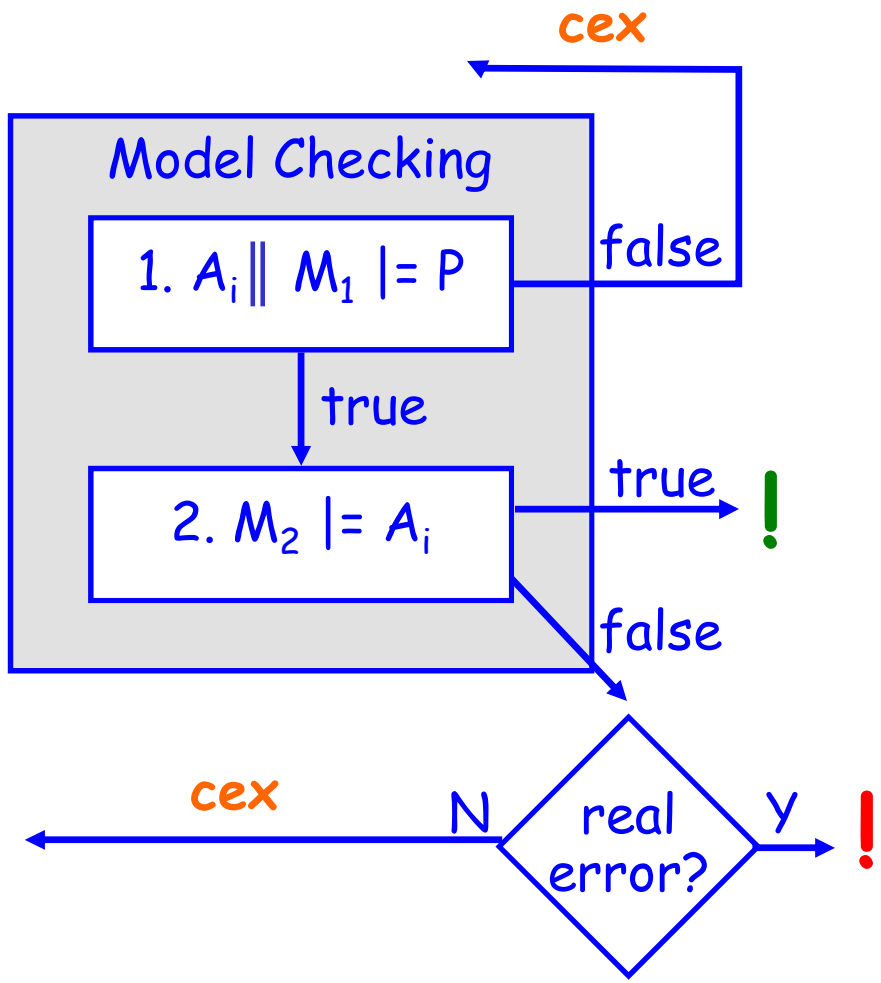

$L(A_i) = L(A_w)$?
A_w contains all traces that can be composed with M_1 without violating P.
counterexample – strengthen assumption
$L(A_i) \subseteq L(A_w)$
Model Checking
Learning
A_i
1. $A_i \| M_1 \models P$
false
true
2. $M_2 \models A_i$
true
P holds in $M_1 || M_2$
false
cex $\notin L(A_i)$
N
real error?
Y
P violated in $M_1 || M_2$
counterexample – weaken assumption
cex $\| M_1 \not\models P$?


$L(A_i) = L(A_w)$?
A_w contains all traces that can be composed with M_1 without violating P.
counterexample – strengthen assumption
$L(A_i) \subseteq L(A_w)$
Model Checking
Learning
A_i
1. $A_i \| M_1 \models P$
false
true
2. $M_2 \models A_i$
true
P holds in $M_1 || M_2$
false
cex $\notin L(A_i)$
N
real error?
Y
P violated in $M_1 || M_2$
counterexample – weaken assumption
cex $\| M_1 \not\models P$?


A_w contains all traces that can be composed with M_1 without violating P.
$L(A_i) = L(A_w)$?
counterexample – strengthen assumption
$L(A_i) \subseteq L(A_w)$
Model Checking
Learning
A_i
1. $A_i \| M_1 \models P$
false
true
2. $M_2 \models A_i$
true
P holds in $M_1 \| M_2$
false
cex $\notin L(A_i)$
real error?
N
Y
P violated in $M_1 \| M_2$
cex $\notin L(A_i)$
cex $\in L(A_w)$
counter
ssumption
cex $\| M_1 \not\models P$?

Equivalence Query - Summary

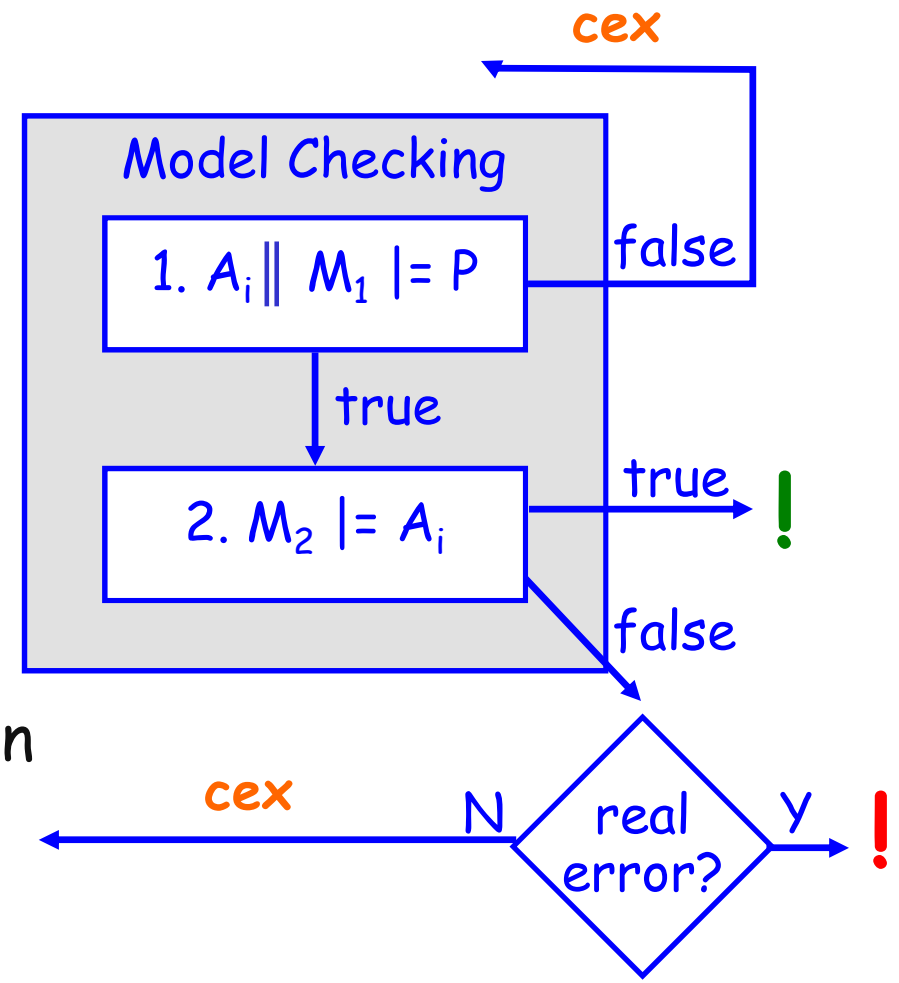
2 applications of
model checking
+ 1 **model checking**
or **simulation**



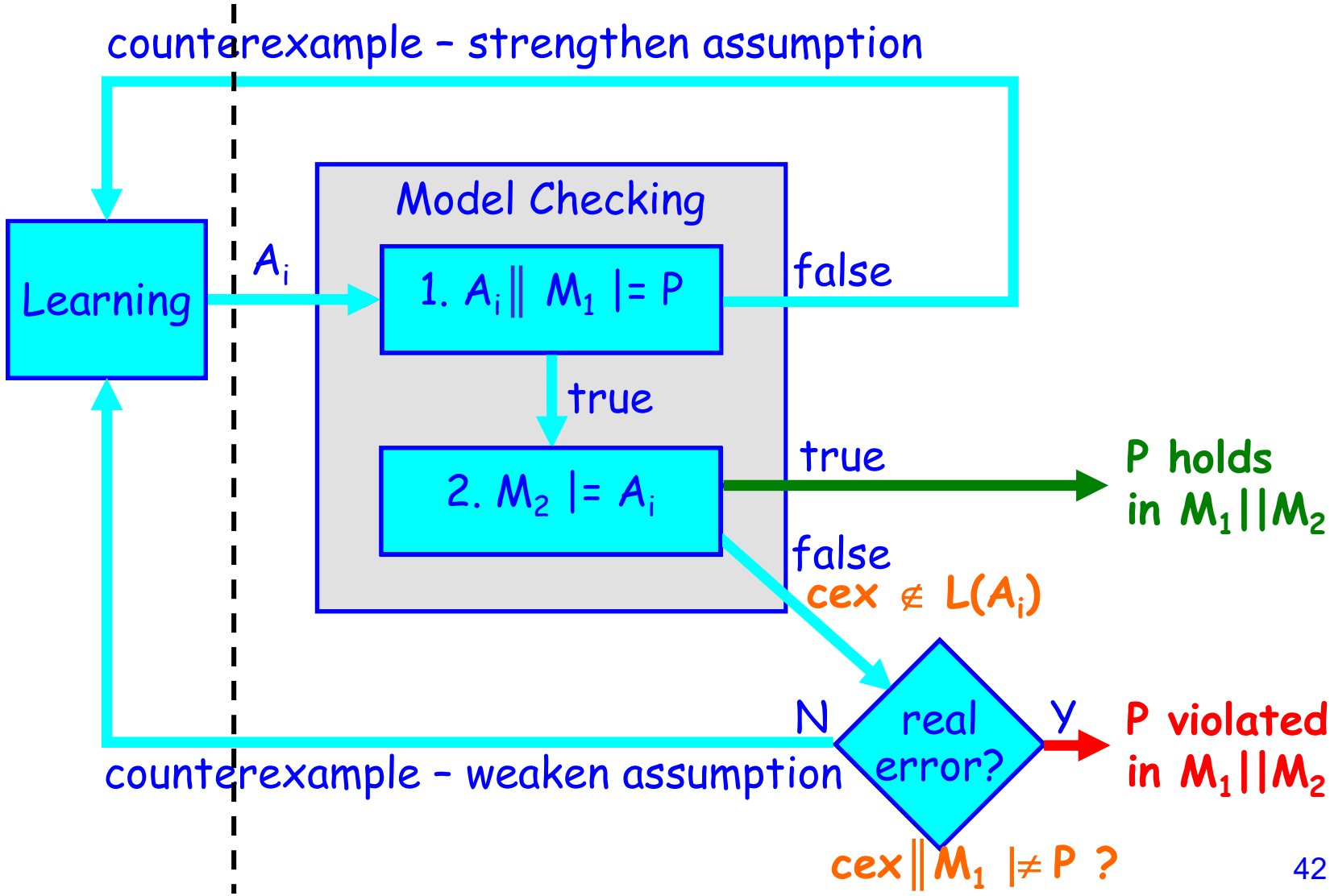
Equivalence Query - Summary

If $L(A_i) = L(A_w)$:
AG rule returns
conclusive results ➔
No need to continue with L*

If $L(A_i) \neq L(A_w)$:
Can terminate and abort L*.
Otherwise, cex is returned in
1 if $L(A_i) \not\subseteq L(A_w)$, or
2 if $L(A_i) \subsetneq L(A_w)$



Assume-Guarantee Framework



Characteristics of Framework

- **AG** uses conjectures produced by **L*** as candidate assumptions A_i
- **L*** uses **AG** as teacher

- L* terminates

  ➔ Framework guaranteed to **terminate**:

  - **At latest** terminates when A_w is produced.
  - **Possibly** terminates **before** A_w is produced!

Characteristics of Framework

- L* makes at most $n = |A_w|$ equivalence queries
- # membership queries: $O(kn^2 + n \log m)$
  - m: size of largest counterexample
  - k: size of Σ

or simulation

- membership query ➔ Model checking X 1
- equivalence query ➔ Model checking X 3

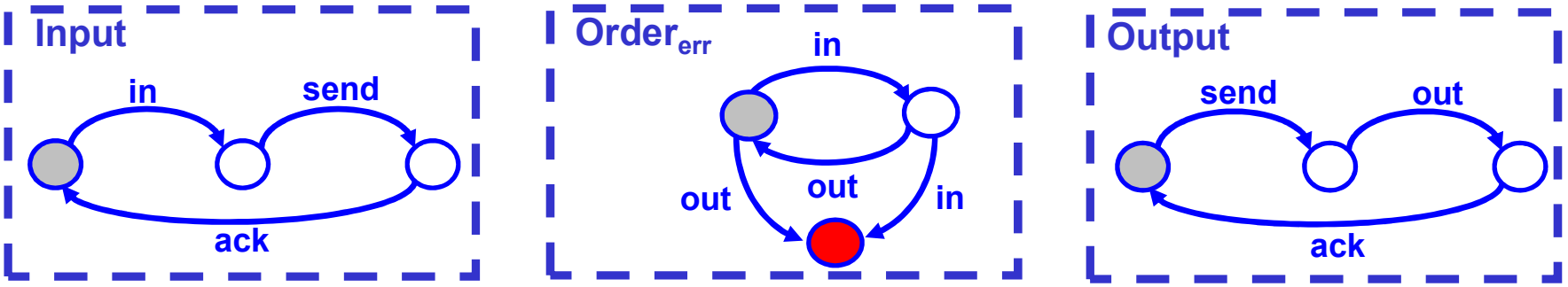
➔ $O(kn^2 + n \log m)$ simulations
+ $O(n)$ model checking applications.

Each MC involves either M_1 or M_2 - not both!

Outline

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

Example



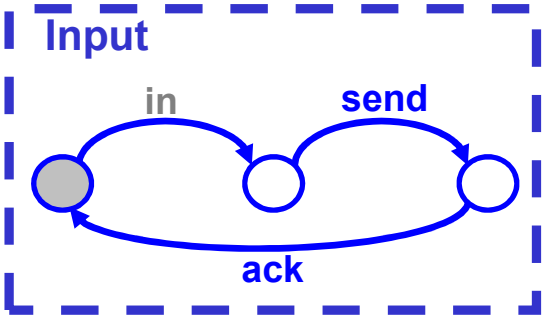
Check: **Input** ‖ **Output** ⊨ **Order** ?

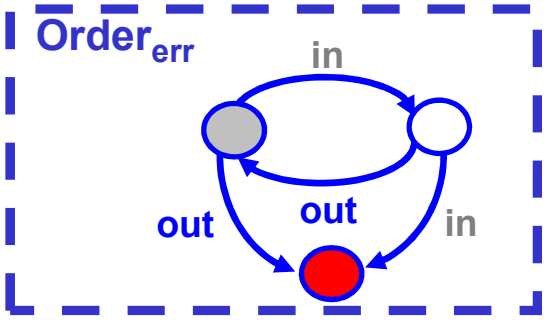
M_1 **M_2** **P**

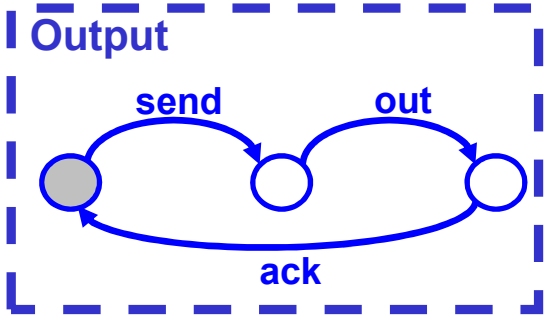
Σ (assumption's alphabet) : **{send, out, ack}**

$\Sigma = (\alpha M_1 \cup \alpha P) \cap \alpha M_2$

Conjectures





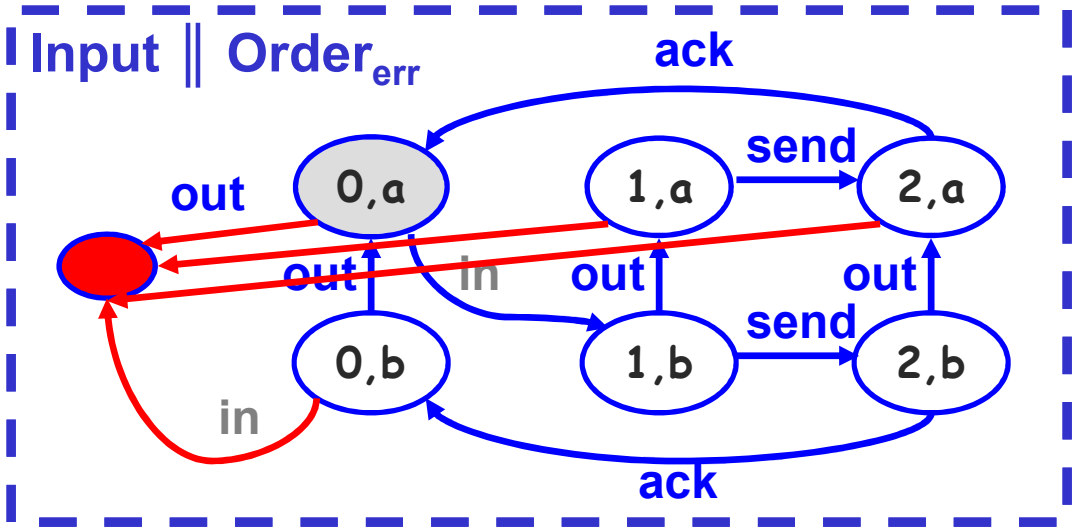


A_1:

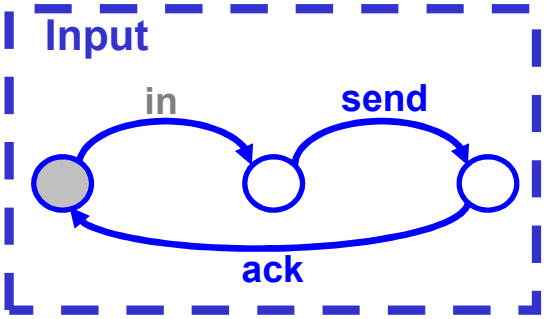
Step 1:

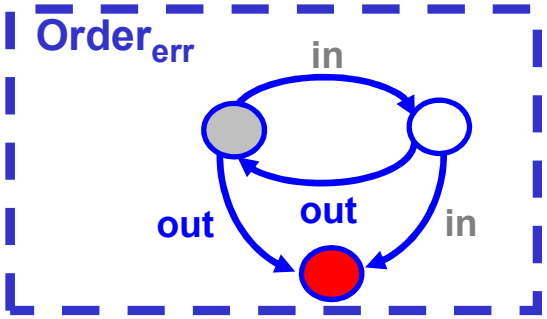
$A_1 \| \text{Input} \models \text{Order}$

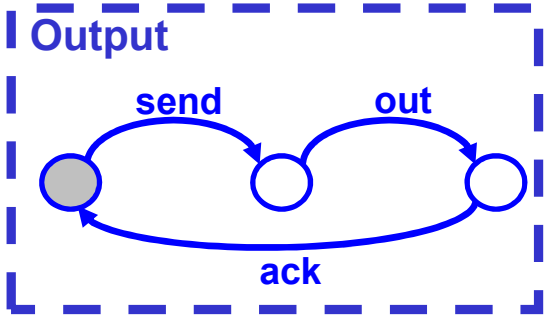
$\Sigma = \{\text{send, out, ack}\}$



Conjectures







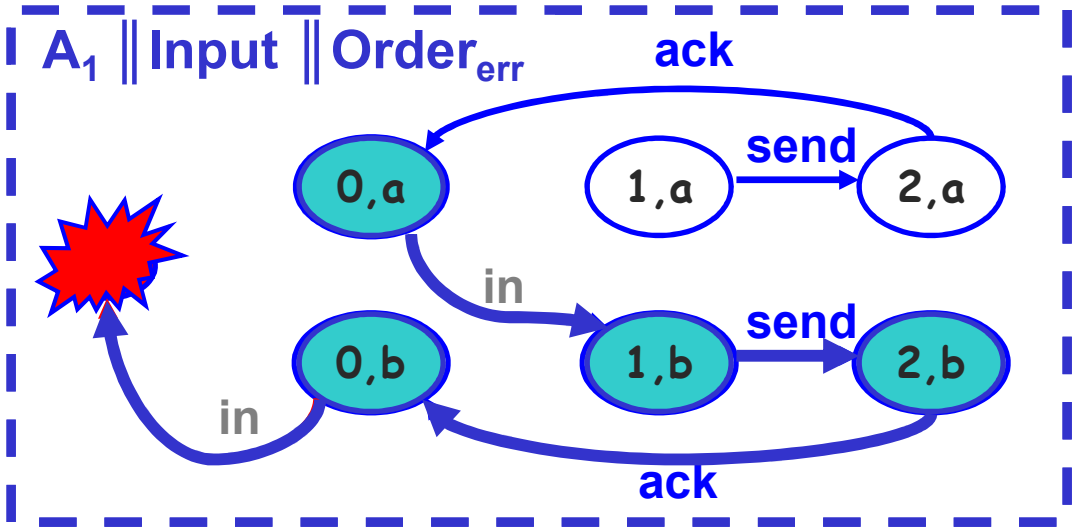
A_1:

Step 1:

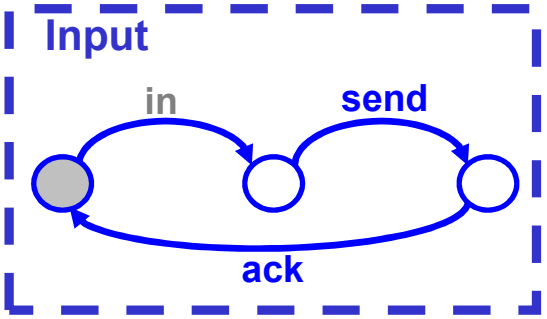
$A_1 \| Input \models Order$

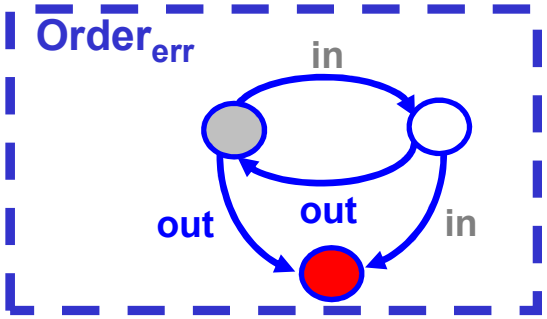
$A_1 \| Input \equiv Input$ with Σ={in,send,ack,out}

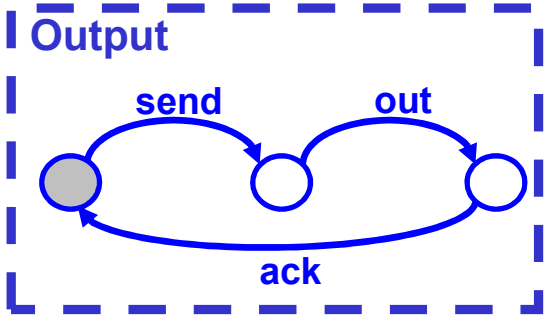
Σ = {send, out, ack}



Conjectures







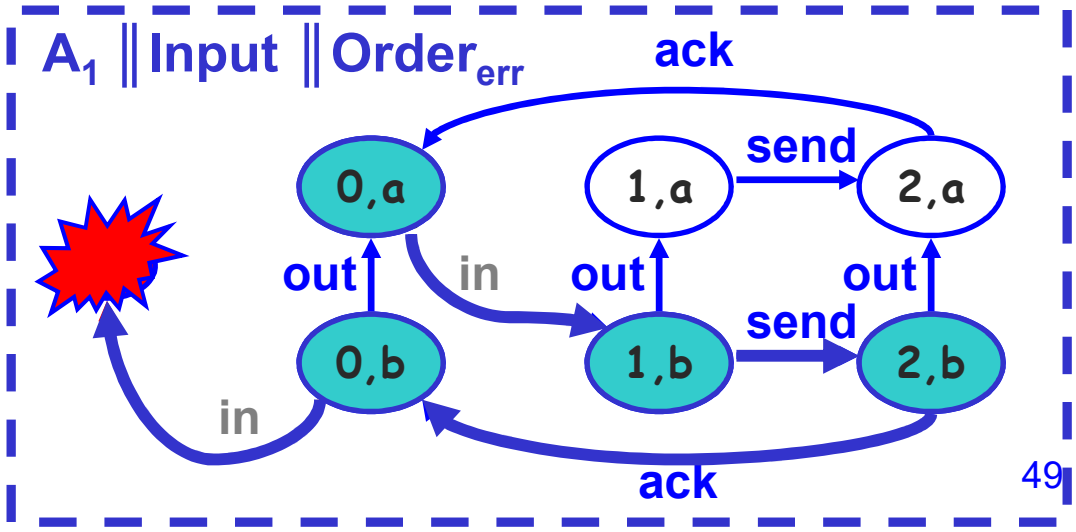
A_1:

Step 1: $A_1 \| \text{Input} \| \text{Order} \models$

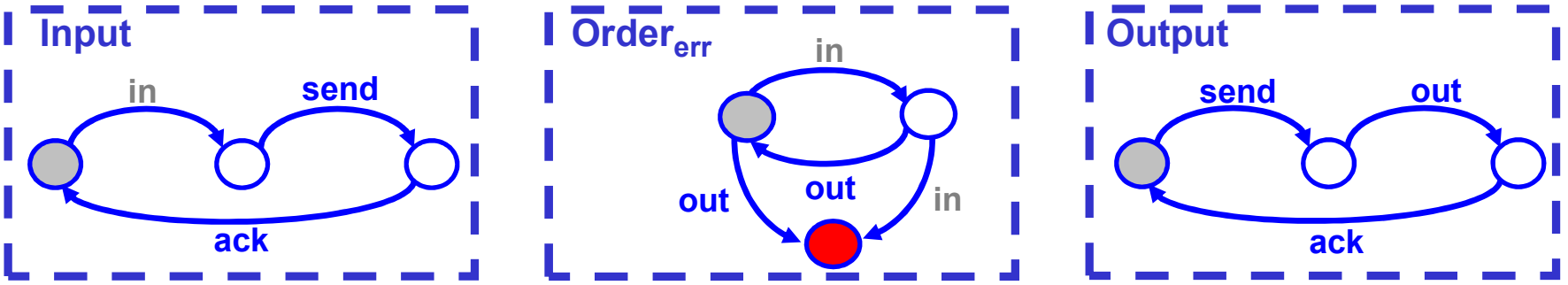
Counterexample: $c = \langle \text{in,send,ack,in} \rangle$

Return to L*: $c{\uparrow}\Sigma = \langle \text{send,ack} \rangle \in L(A_1) \setminus L(A_w)$

$\Sigma = \{\text{send, out, ack}\}$



Conjectures



A_1: ack, send

→ **Step 1:** $A_1 \| \text{Input} \models \text{Order}$

→ **Counterexample:** $c = \langle \text{in,send,ack,in} \rangle$

→ **Return to L*:** $c{\uparrow}\Sigma = \langle \text{send,ack} \rangle \in L(A_1) \setminus L(A_w)$

Queries ⇒ A_2: send, ack, out, send

$c{\uparrow}\Sigma \notin L(A_2)$

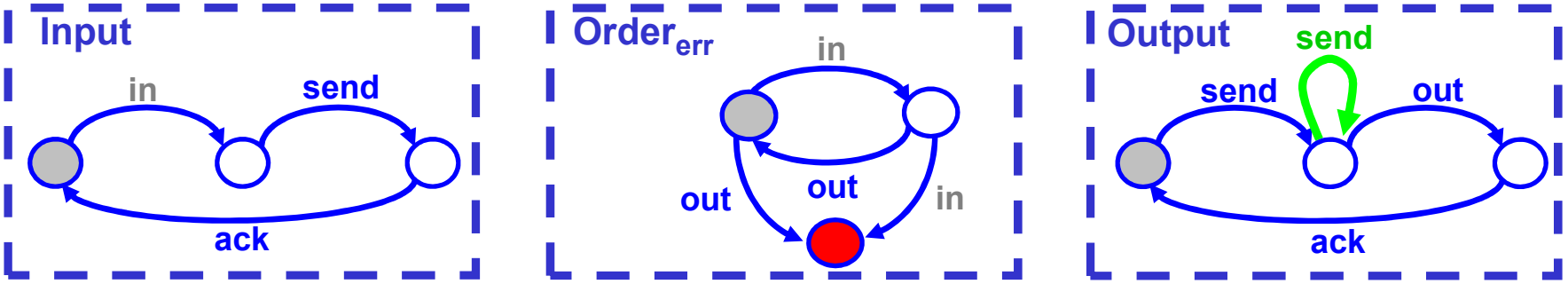
→ **Step 1:** $A_2 \| \text{Input} \models \text{Order}$ **True**

→ **Step 2:** $\text{Output} \models A_2$ **True**

→ property Order holds on Input ‖ Output

$\Sigma = \{\text{send, out, ack}\}$

Another Example



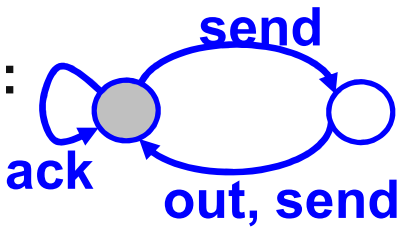
Step 2:
Output' ⊨ A_2

Counterexample:
c=⟨send,send,out⟩

real error?

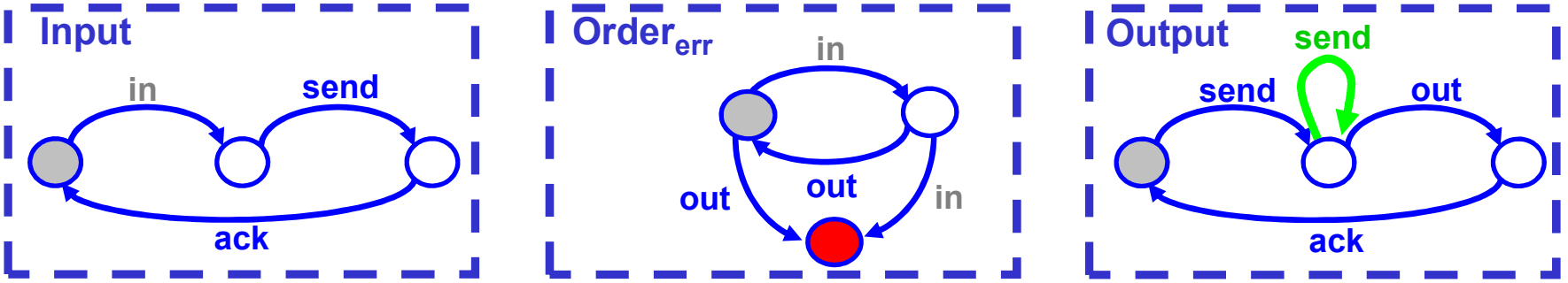
Simulate c↑Σ = ⟨send,send,out⟩ on Input ‖ Order$_{err}$

A_2:



Σ = {send, out, ack}

Another Example



Step 2: Output' $\models A_2$

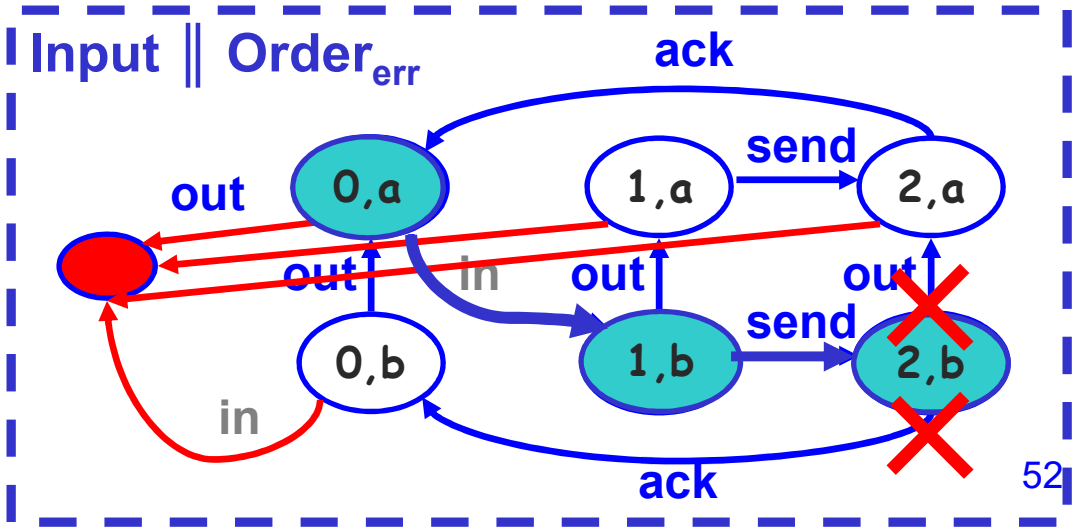
Counterexample: c=⟨send,send,out⟩

real error?

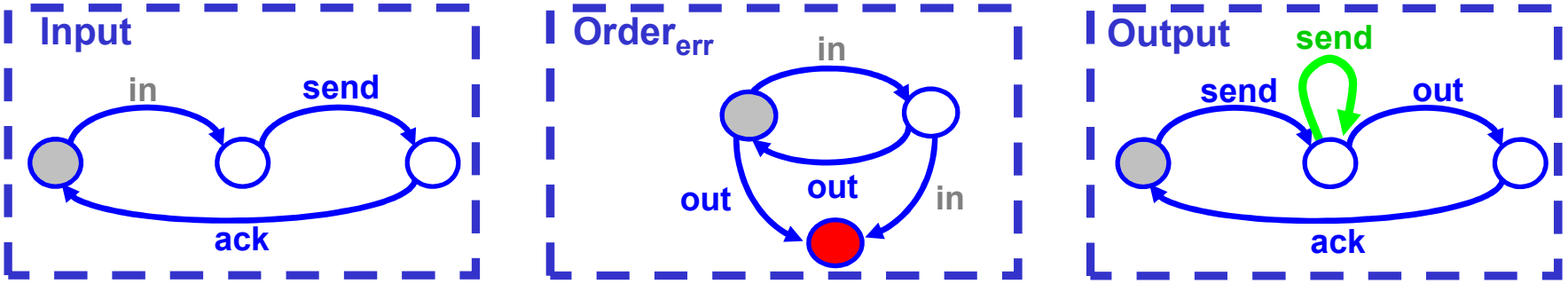
Simulate $c\uparrow\Sigma$ = ⟨send,send,out⟩ on Input ‖ Order$_{err}$

π unreachable
not a real error!

Σ = {send, out, ack}



Another Example



Step 2: Output' |=A_2

Counterexample: c=⟨send,send,out⟩

real error?

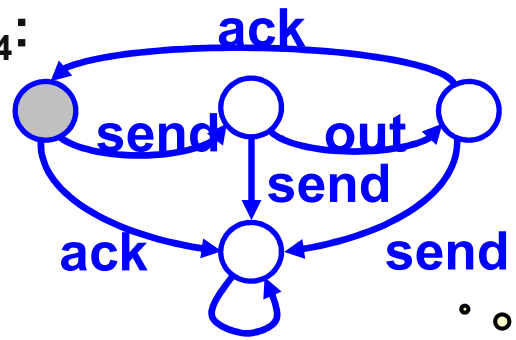
Simulate c↑Σ = ⟨send,send,out⟩ on Input ‖ Order$_{err}$

π unreachable
not a real error!

Return c↑Σ to L*

∈ $L(A_w) \setminus L(A_2)$

A_4:



property Order holds on Input ‖ Output'

A_w

Σ = {send, out, ack}

Conclusion

- Generate assumptions for assume-guarantee reasoning in an **incremental** and fully **automatic** fashion, using **learning**.
- Each iteration of **assume-guarantee** may conclude that the required property is **satisfied** or **violated** in the system.
- Assumption generation converges to an assumption that is **necessary** and **sufficient** for the property to hold in the specific system.

The End