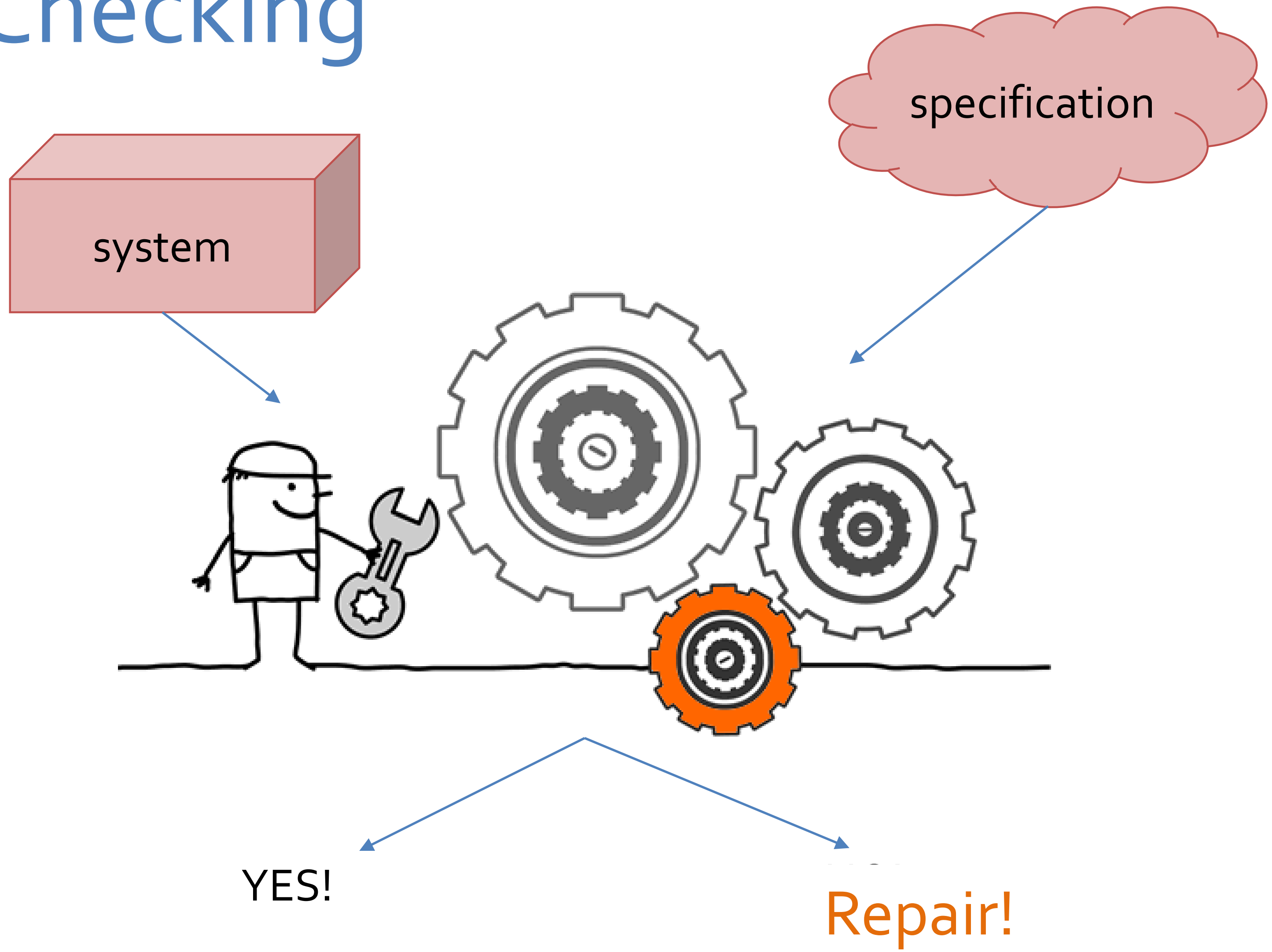
ASSUME GUARANTEE OR REPAIR
COMPOSITIONAL VERIFICATION AND REPAIR OF C-LIKE PROGRAMS

Hadar Frenkel
the Technion, Israel

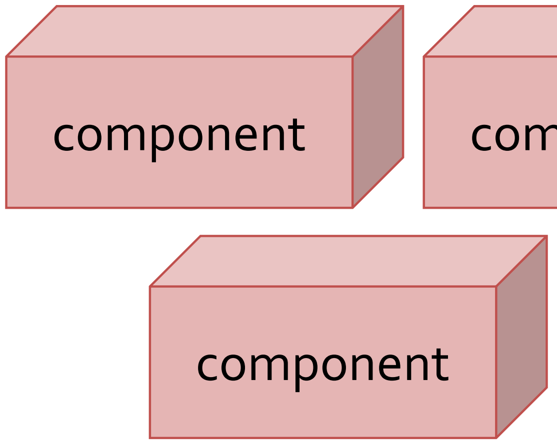
Joint work with Orna Grumberg, Corina Pasareanu, and Sarai Sheinvald

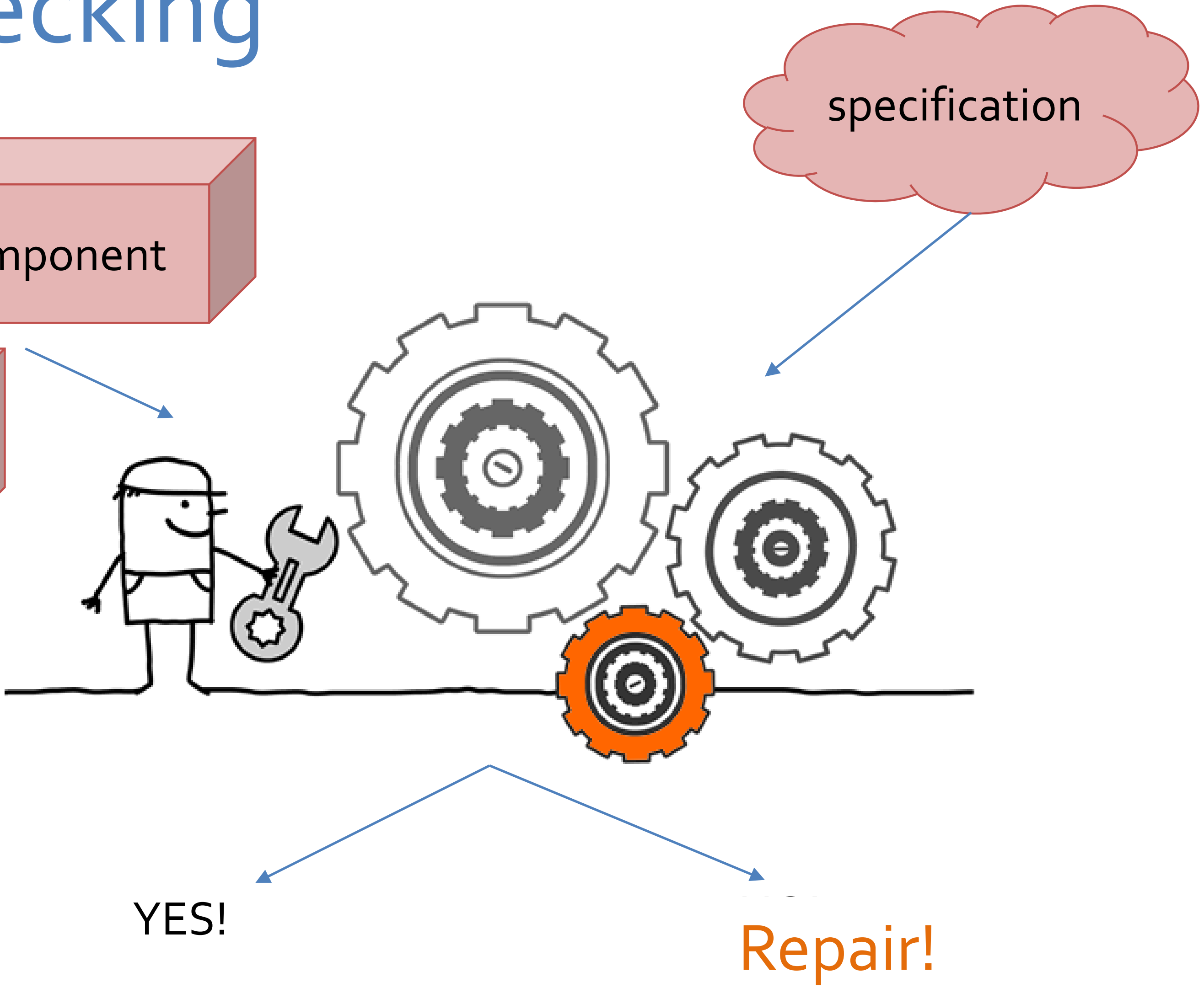
@TACAS 2020

Model Checking



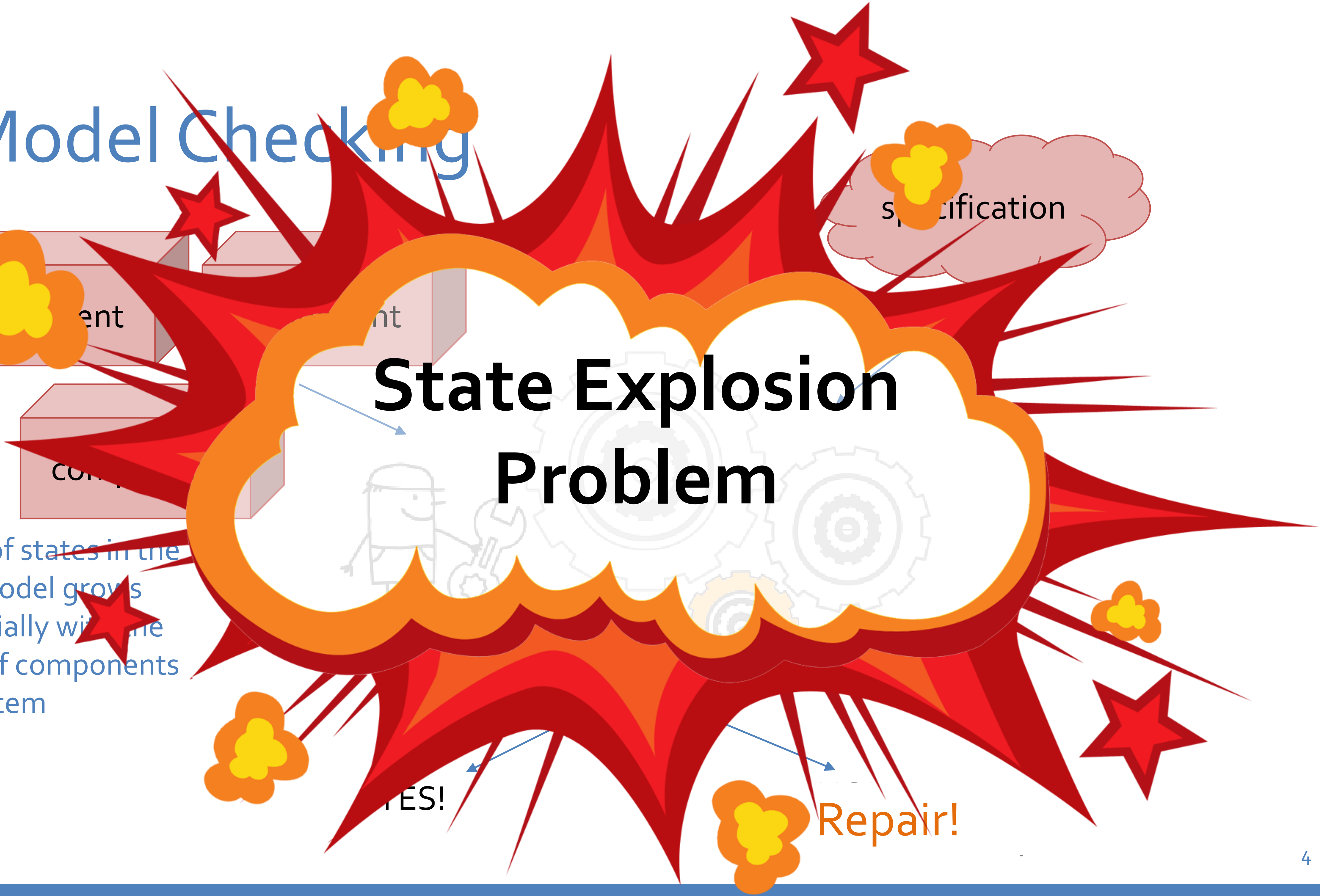
Model Checking





Number of states in the system model grows exponentially with the number of components in the system

Model Checking



Number of states in the system model grows exponentially with the number of components in the system

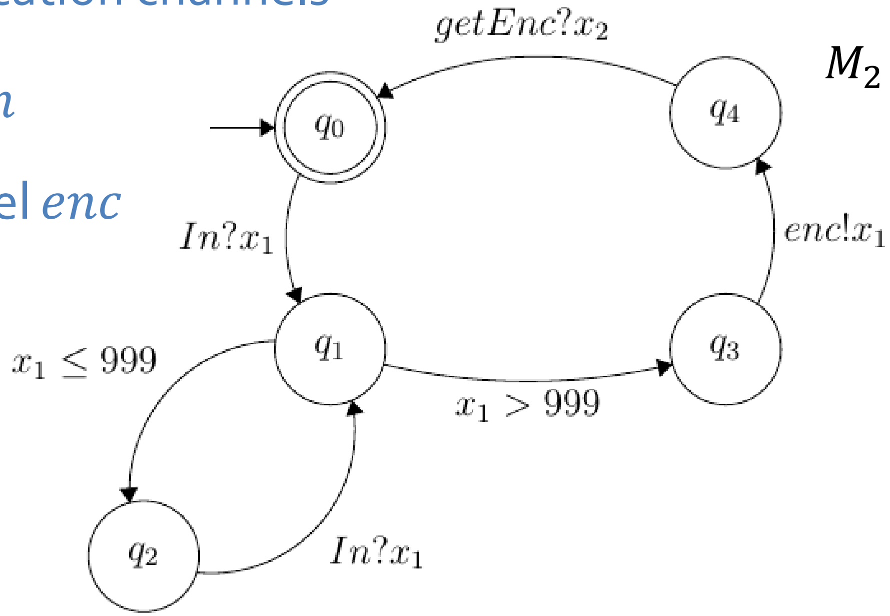
COMPOSITIONAL VERIFICATION AND REPAIR OF C-LIKE PROGRAMS

- *Model checking* and *repair* algorithm for communicating systems
- Exploit the partition of the system into components

Communicating Systems

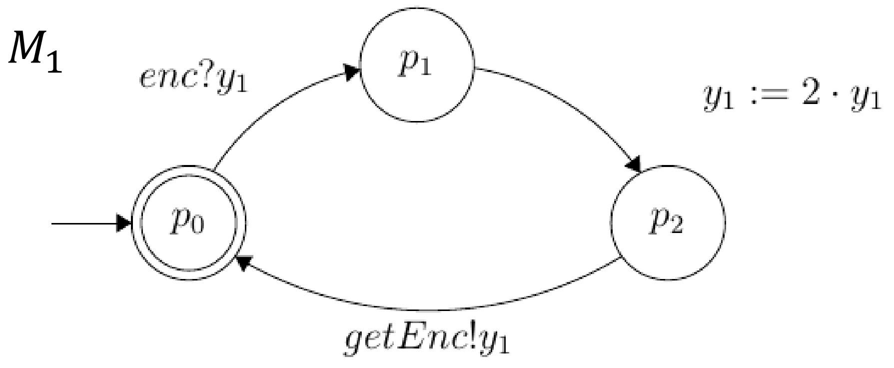
- C-like programs
- Each component is described as a control-flow graph (automaton)
  - Alphabet: program statements & communication channels
- $In?\,x_1$ – reads a value to x_1 through channel In
- $enc!\,x_1$ – sends the value of x_1 through channel enc

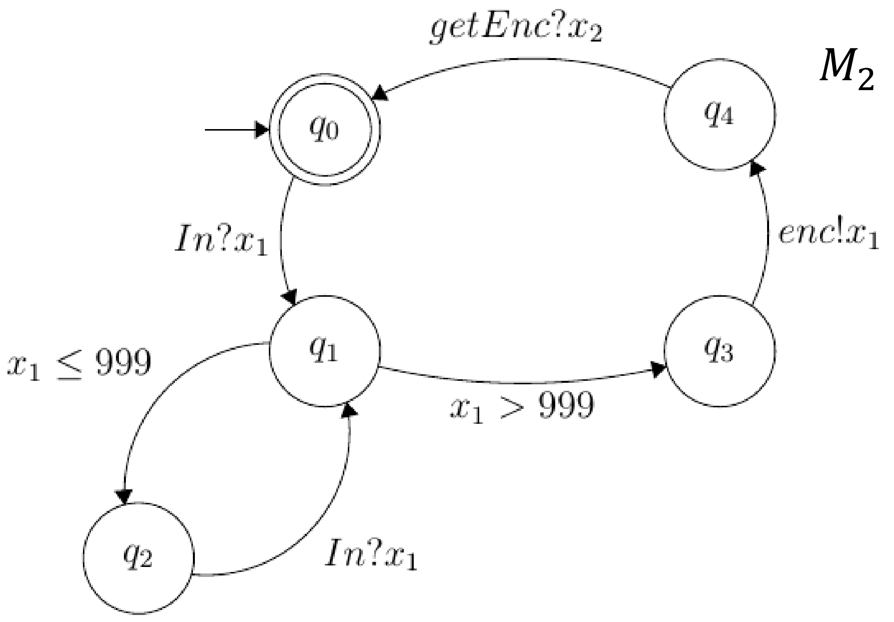
```
while (true)
    pass = readInput;
    while (pass ≤ 999)
        pass = readInput;
    pass2 = encrypt(pass);
```



Example

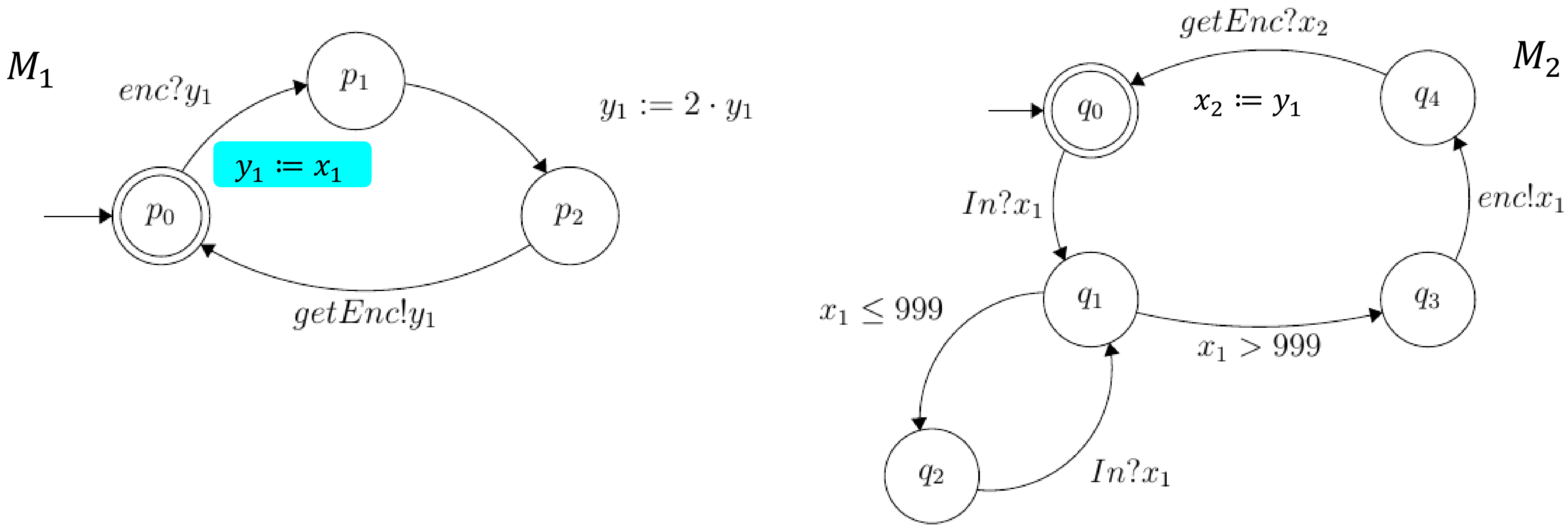
Synchronization using read-write channels, Interleaving on all other alphabet





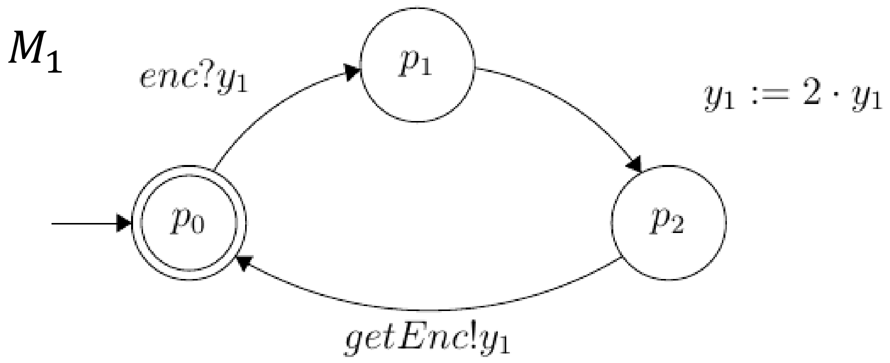
Example

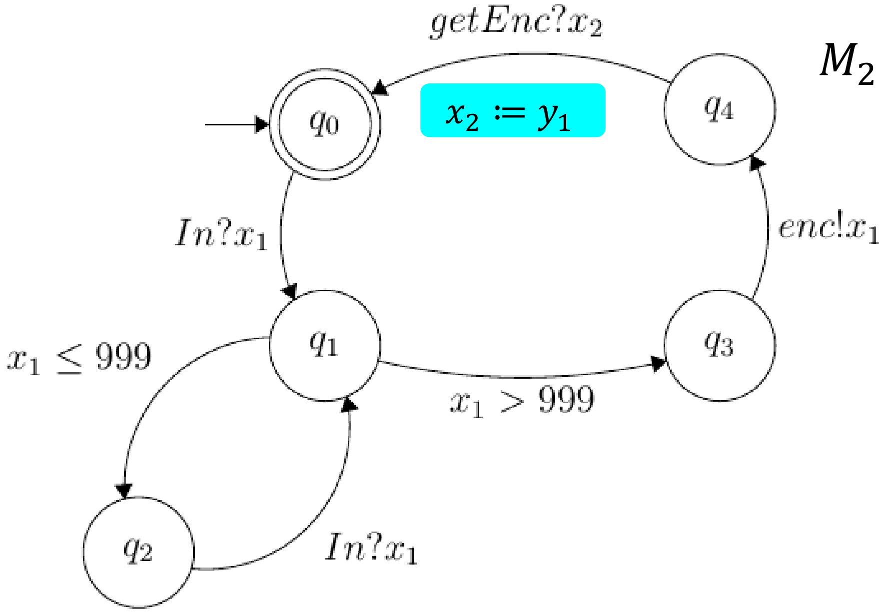
Synchronization using read-write channels, Interleaving on all other alphabet



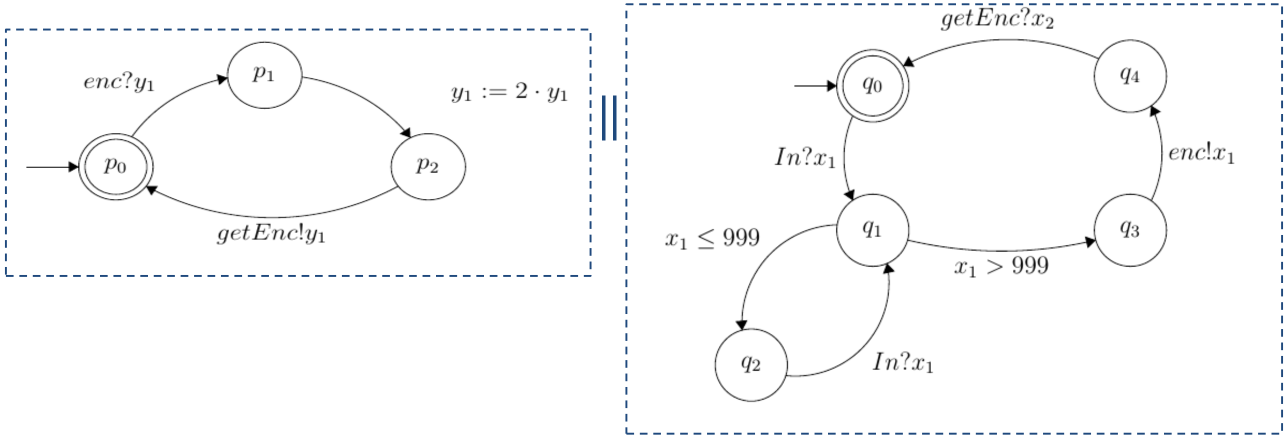
Example

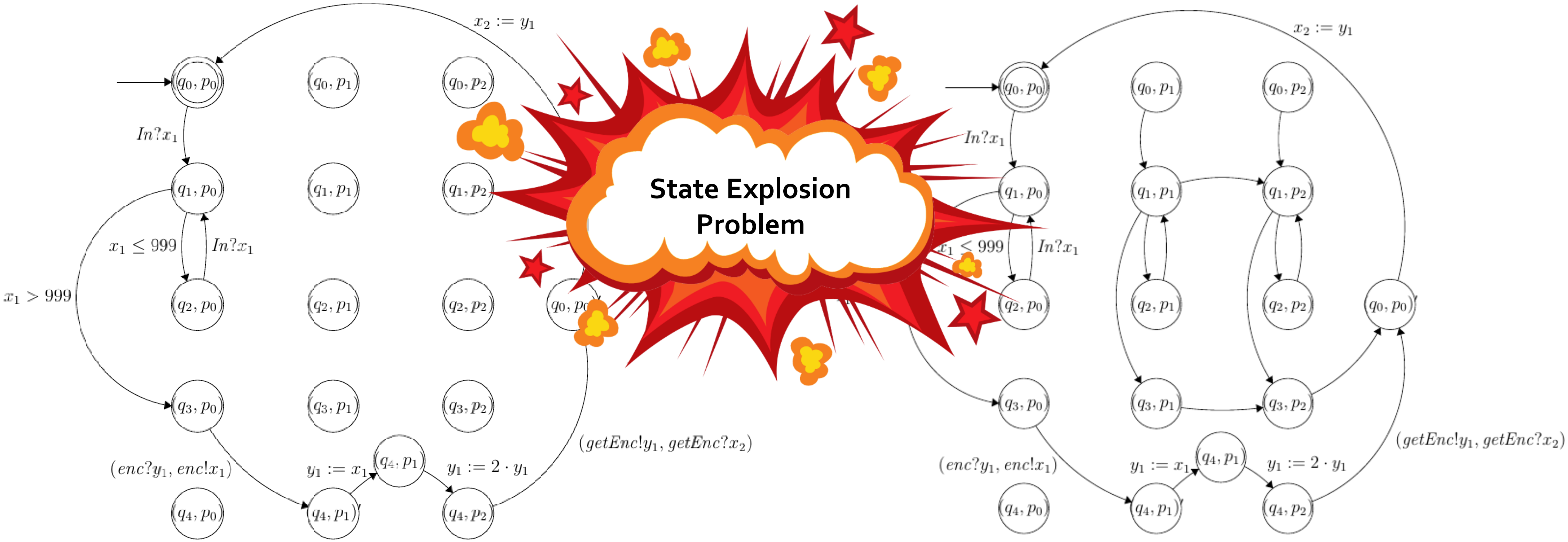
Synchronization using read-write channels, Interleaving on all other alphabet





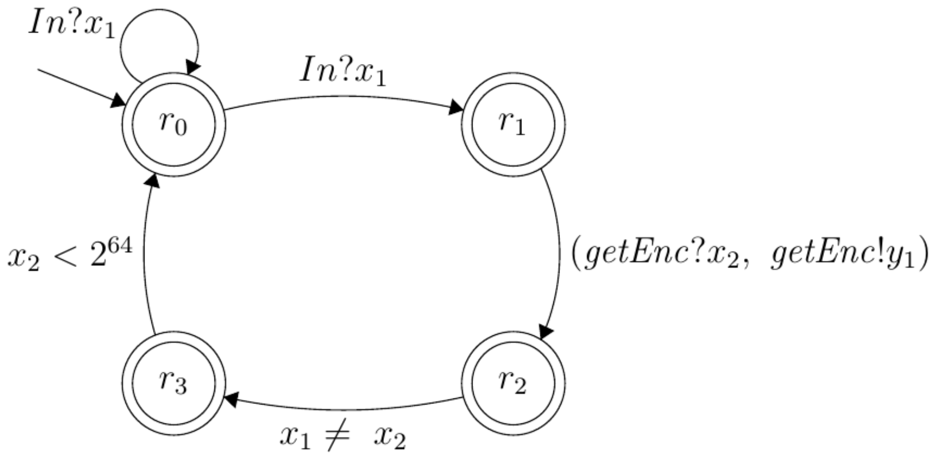
Example


enc?y1
p1
y1 := 2 · y1
p0
p2
getEnc!y1
||
getEnc?x2
q0
q4
In?x1
enc!x1
x1 ≤ 999
q1
q3
x1 > 999
q2
In?x1


x2 := y1
(q0, p0)
(q0, p1)
(q0, p2)
In?x1
(q1, p0)
(q1, p1)
(q1, p2)
x1 ≤ 999
In?x1
x1 > 999
(q2, p0)
(q2, p1)
(q2, p2)
(q3, p0)
(q3, p1)
(q3, p2)
(enc?y1, enc!x1)
y1 := x1
(q4, p1)
y1 := 2 · y1
(getEnc!y1, getEnc?x2)
(q4, p0)
(q4, p2)
State Explosion Problem

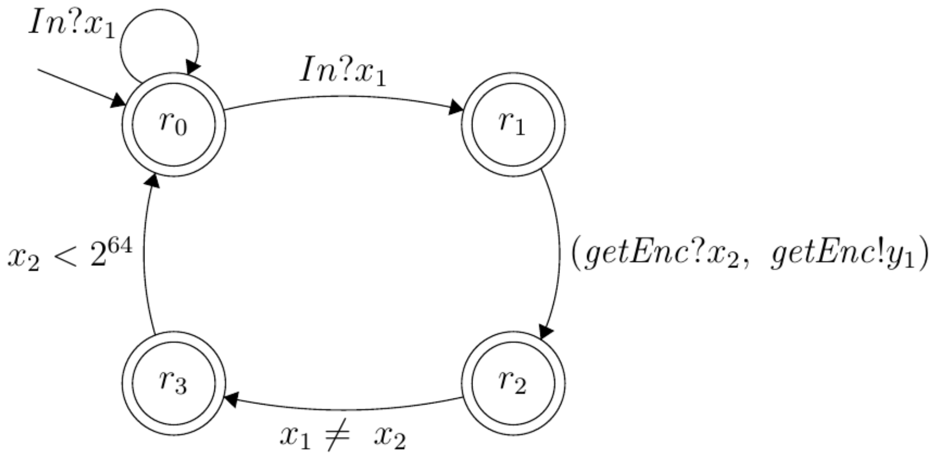
Specifications

- Safety properties
- Alphabet:
- (Common) communication channels
- Syntactic requirements:
  program behavior through time



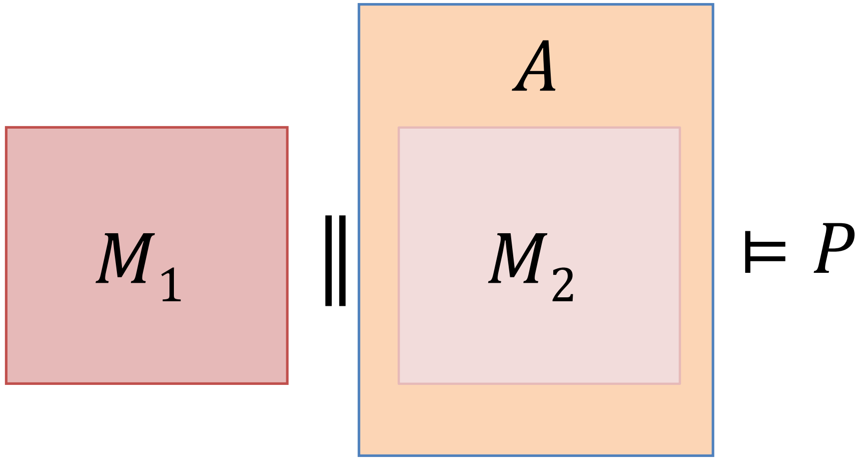
Specifications

- Safety properties
- Alphabet:
- (Common) communication channels
- Syntactic requirements:
  program behavior through time
- Constraints over local variables
- Semantic requirements:
  - “the entered password is different from the encrypted password”
  - “there is no overflow”



Compositional Verification

- **Assume-Guarantee** (AG) paradigm [Pnueli, 1985]:
  - assumptions represent component's environment
- Under assumption *A* on its environment, does the component guarantee the property?

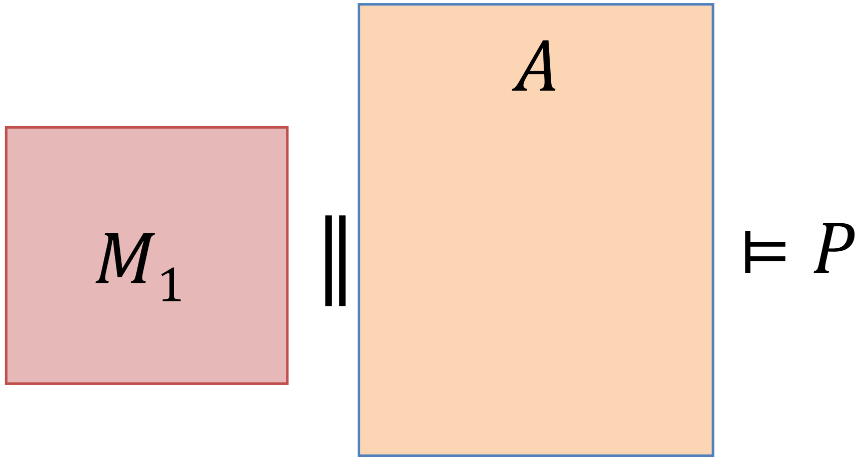


AG Rule for Safety Properties

Find an **assumption** A such that

1. Component M_1 **guarantees** P when it is a part of a system satisfying A

$M_1 || A \vDash P$



AG Rule for Safety Properties

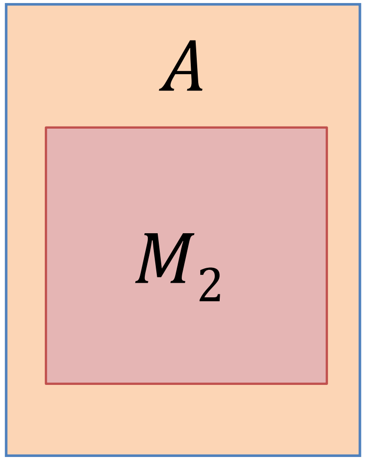
Find an **assumption** A such that

1. Component M_1 **guarantees** P when it is a part of a system satisfying A

$M_1 || A \models P$

2. M_2 satisfies A

$M_2 \models A$



AG Rule for Safety Properties

Find an **assumption** A such that

1. Component M_1 **guarantees** P when it is a part of a system satisfying A

$M_1 || A \vDash P$

2. M_2 satisfies A

$M_2 \vDash A$

Conclude that $M_1 || M_2 \vDash P$

$M_1 \quad || \quad M_2 \quad \vDash P$

AG Rule for Safety Properties

Find an **assumption** A such that

1. Component M_1 **guarantees** P when it is a part of a system satisfying A

$M_1 || A \vDash P$

2. M_2 satisfies A

$M_2 \vDash A$

Can we automatically construct A?

Conclude that $M_1 || M_2 \vDash P$

$M_1 \;||\; M_2 \;\vDash P$

L^* Algorithm for Learning Regular Languages [D.Angluin 1987]

- Learning assumptions for compositional verification
  [J. M. Cobleigh, D. Giannakopoulou and C. S. Pasareanu TACAS 2003]
- Given a regular language L, we learn a DFA A such that $\mathcal{L}(A) = L$

Teacher

Learner

L^* Algorithm for Learning Regular Languages [D.Angluin 1987]

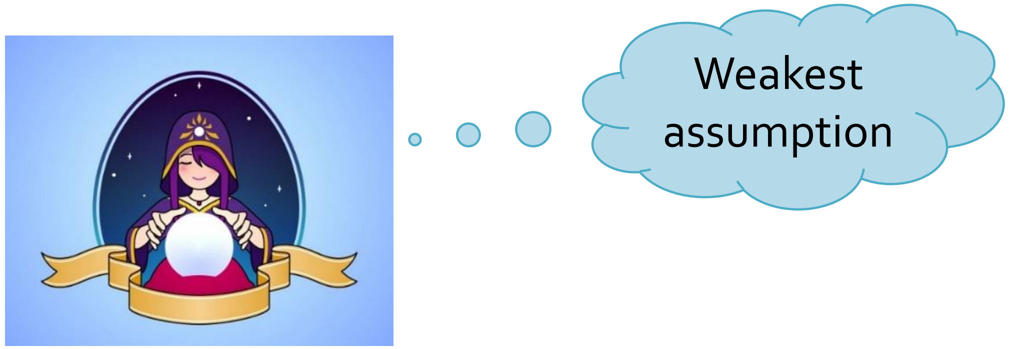
- Learning assumptions for compositional verification
  [J. M. Cobleigh, D. Giannakopoulou and C. S. Pasareanu TACAS 2003]
- Given a regular language L, we learn a DFA A such that $\mathcal{L}(A) = L$
- Membership + equivalence queries

Teacher

Learner

L^* Algorithm for Learning Regular Languages [D.Angluin 1987]

- Learning assumptions for compositional verification
  [J. M. Cobleigh, D. Giannakopoulou and C. S. Pasareanu TACAS 2003]
- Given a regular language L, we learn a DFA A such that $\mathcal{L}(A) = L$
- Try to use intermediate candidates A_i as assumptions for AG rule
- But, the weakest assumption is not regular in our case



$$M_1 || A_i \vDash P$$
$$M_2 \vDash A_i$$

$$M_1 || M_2 \vDash P$$

$$M_1 || \boldsymbol{M_2} \vDash P$$
$$M_2 \vDash \boldsymbol{M_2}$$

$$M_1 || M_2 \vDash P$$

A New Goal for Learning

- The teacher answers queries according to the *syntactic language* of M_2
- Regular since it is given as an automaton

$$M_1 || \boldsymbol{M_2} \vDash P$$
$$M_2 \vDash \boldsymbol{M_2}$$

$$M_1 || M_2 \vDash P$$

A New Goal for Learning

- The teacher answers queries according to the *syntactic language* of M_2
- Regular since it is given as an automaton

But I already know M_2 ...

Teacher

You might find a much smaller assumption!

Learner

AG rule with learning

Membership queries

Is $w \in L$?

AG rule with learning

Membership queries

Is $w \in L$?

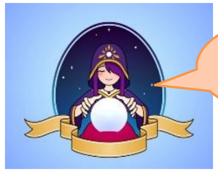
Yes \ no

AG rule with learning

Real error!

Membership queries

Is $w \in L$?



Yes \ no

P is violated in $M_1 \parallel M_2$

AG rule with learning

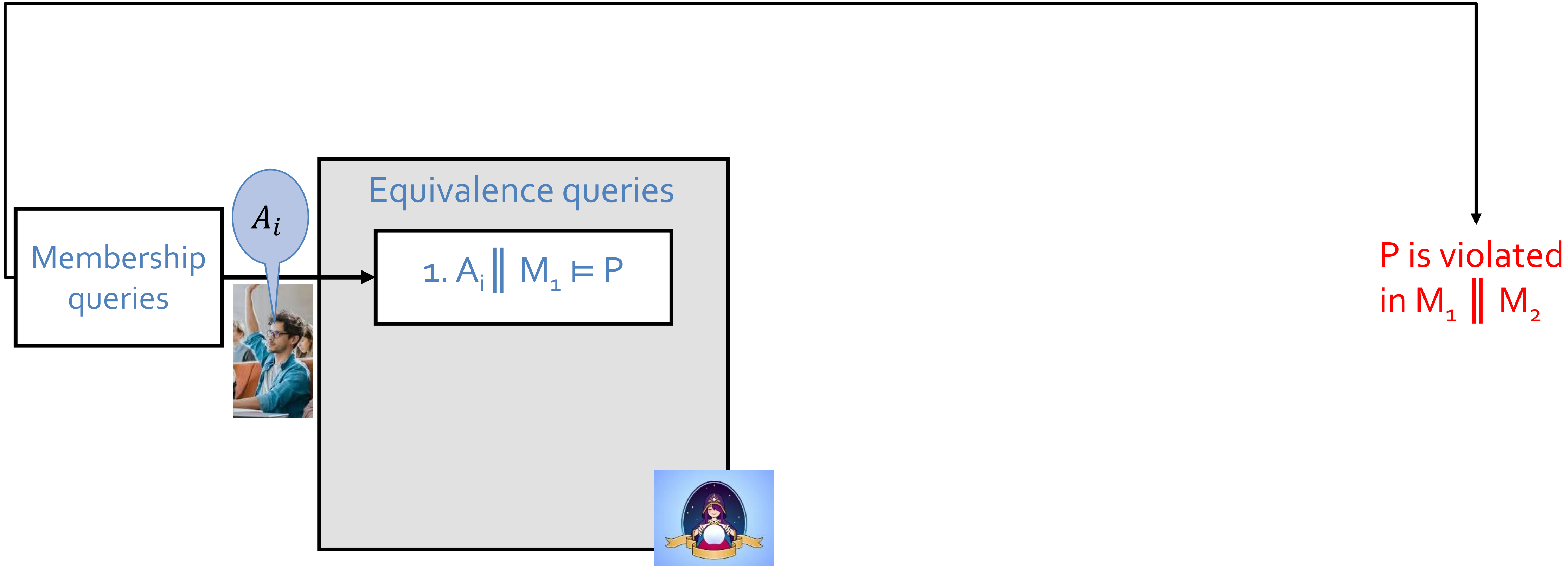
Membership queries

A_i

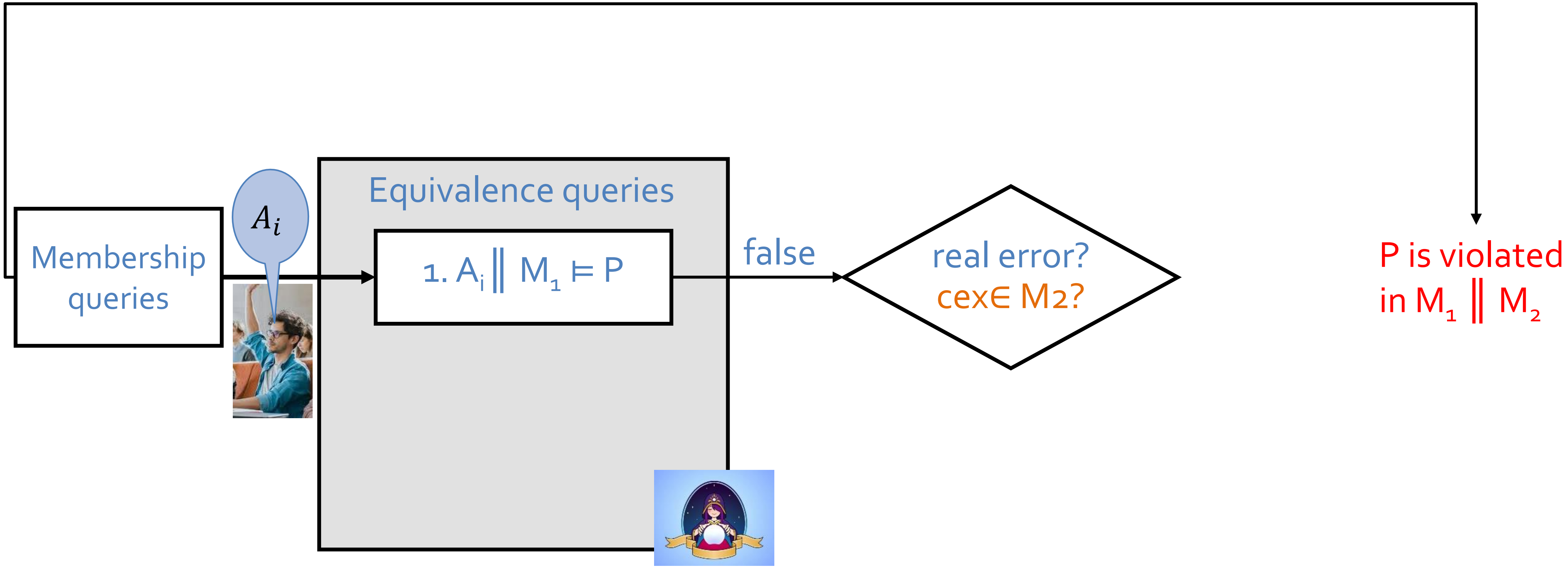
Equivalence queries

P is violated in $M_1 \parallel M_2$

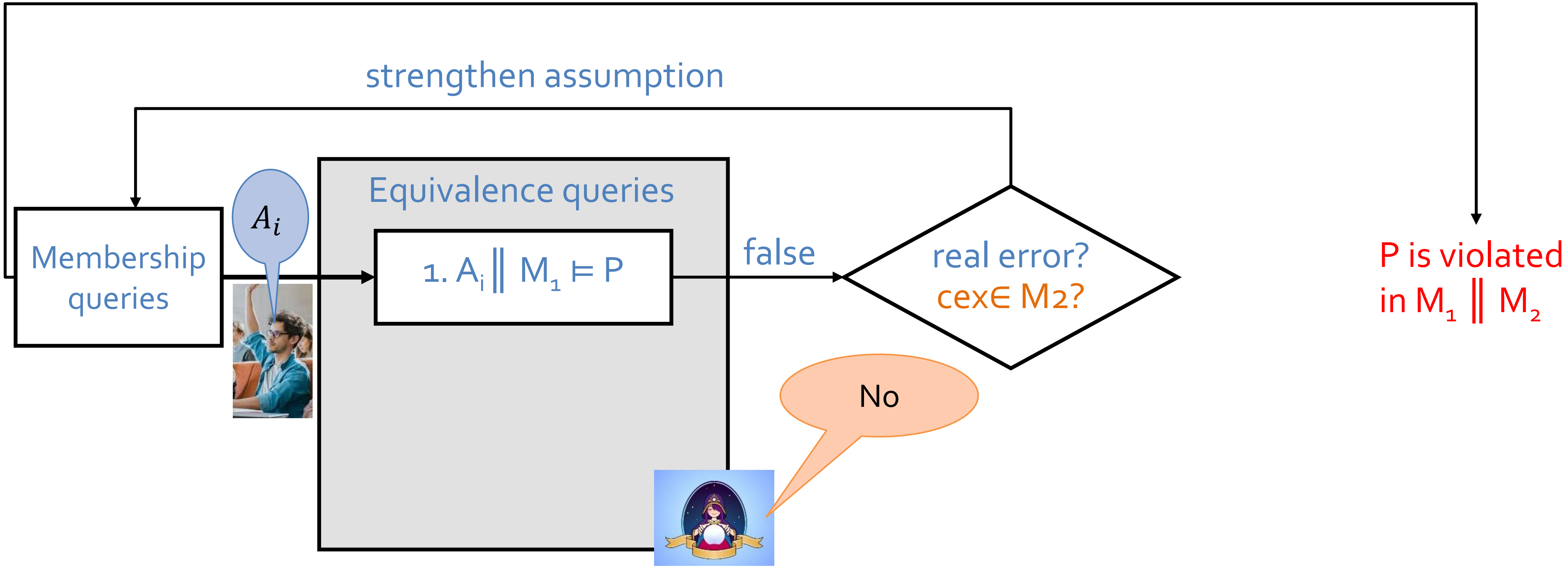
AG rule with learning



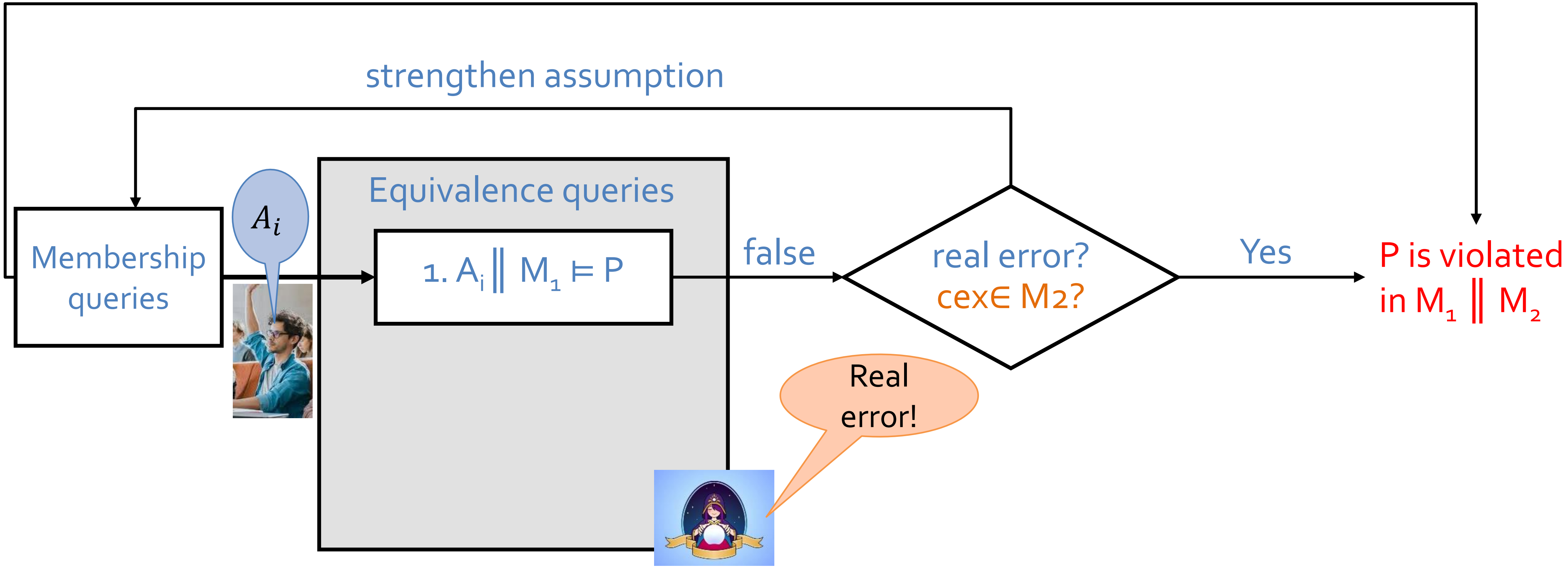
AG rule with learning



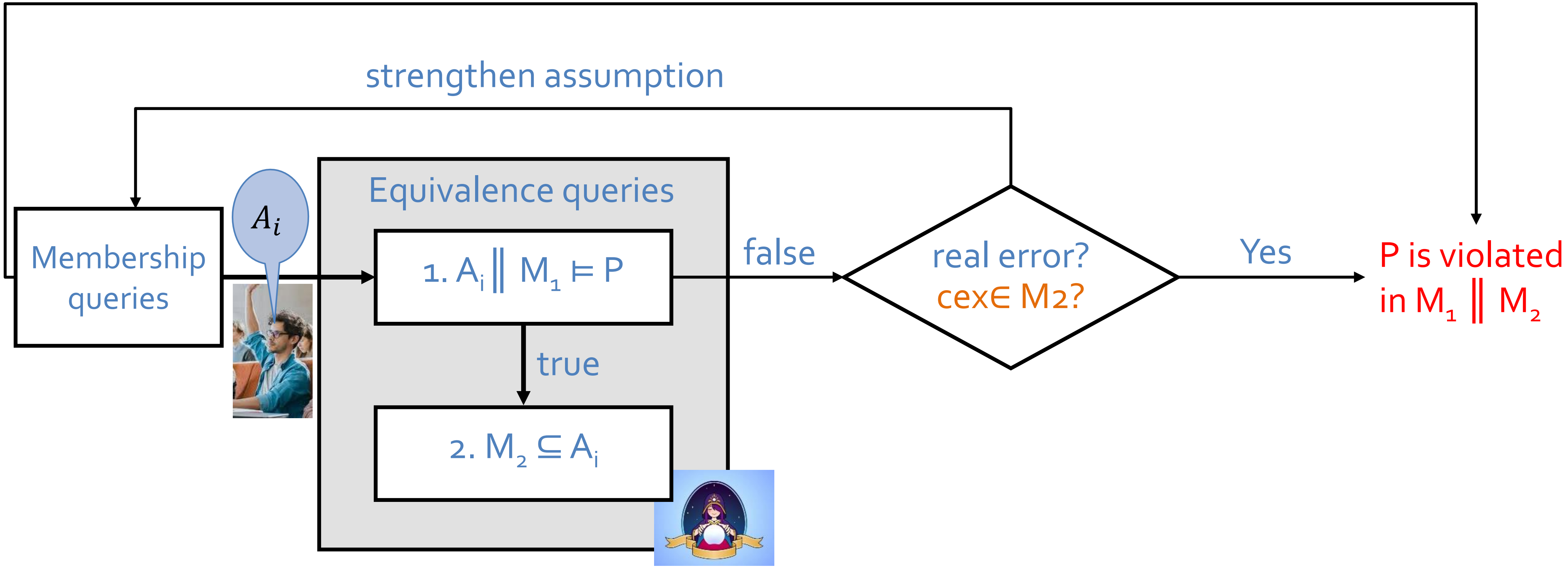
AG rule with learning



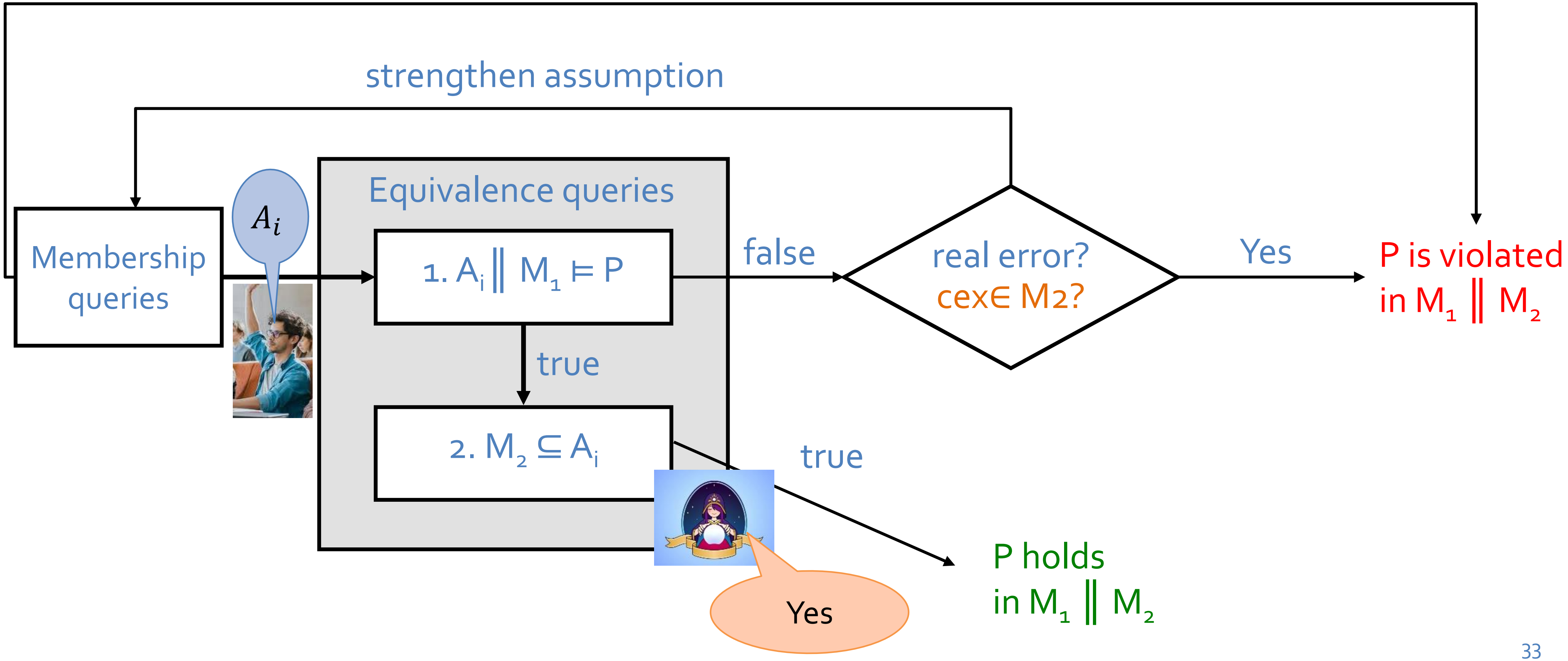
AG rule with learning



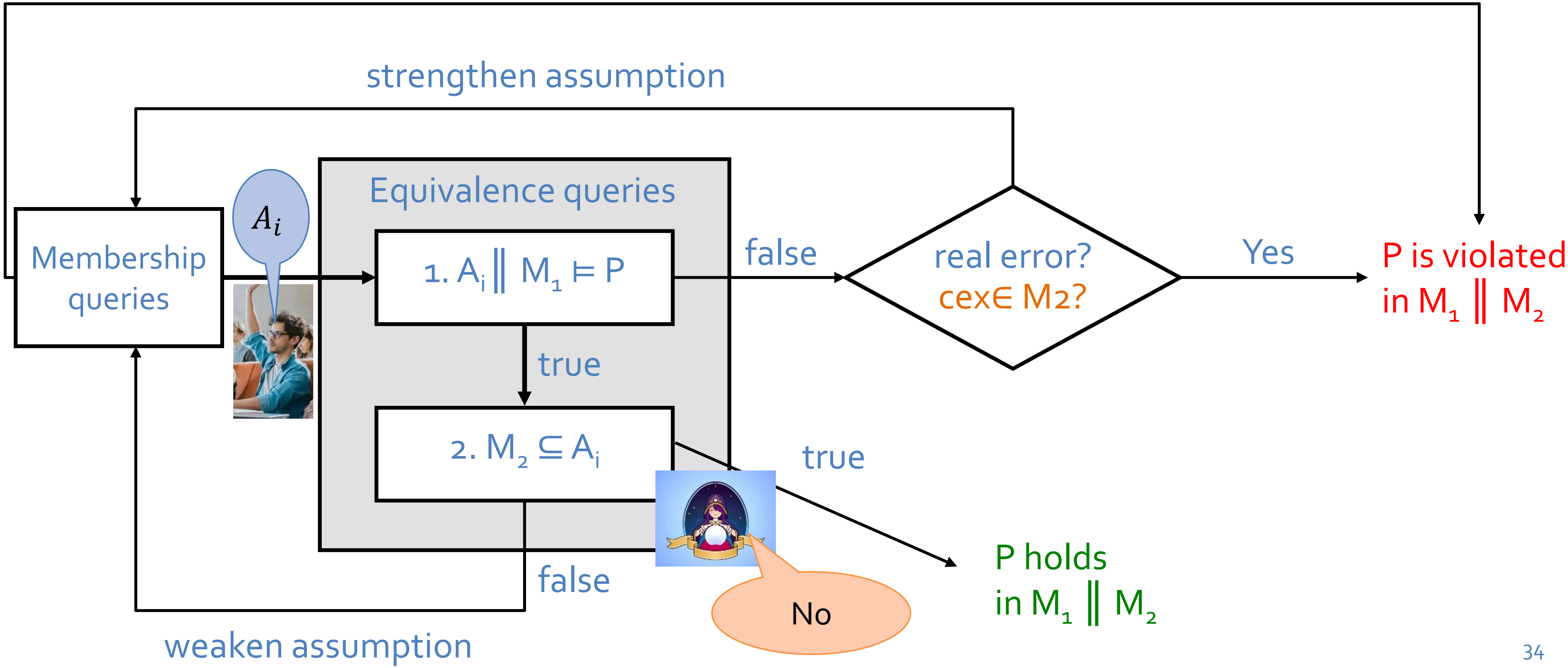
AG rule with learning



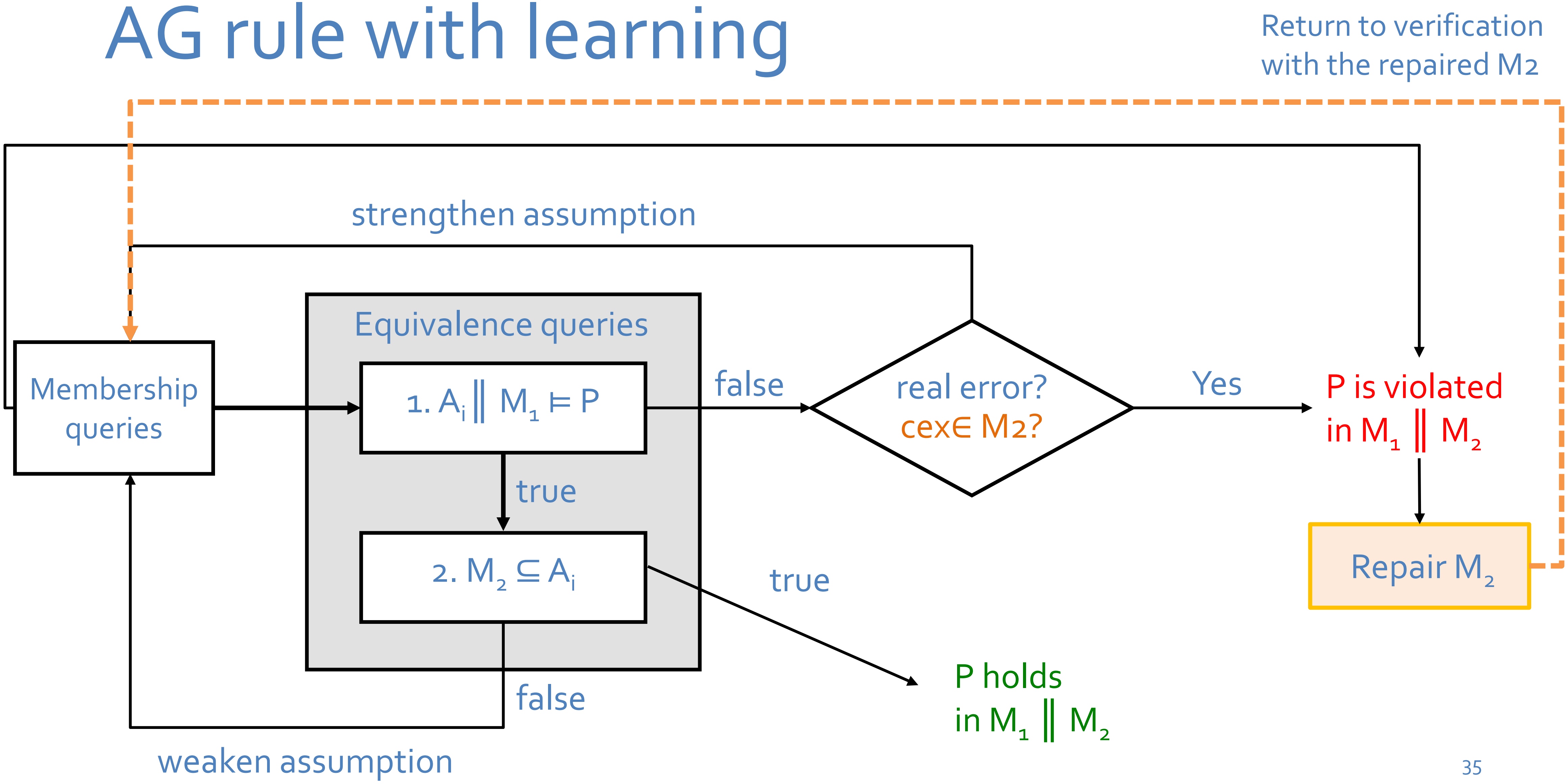
AG rule with learning



AG rule with learning



AG rule with learning

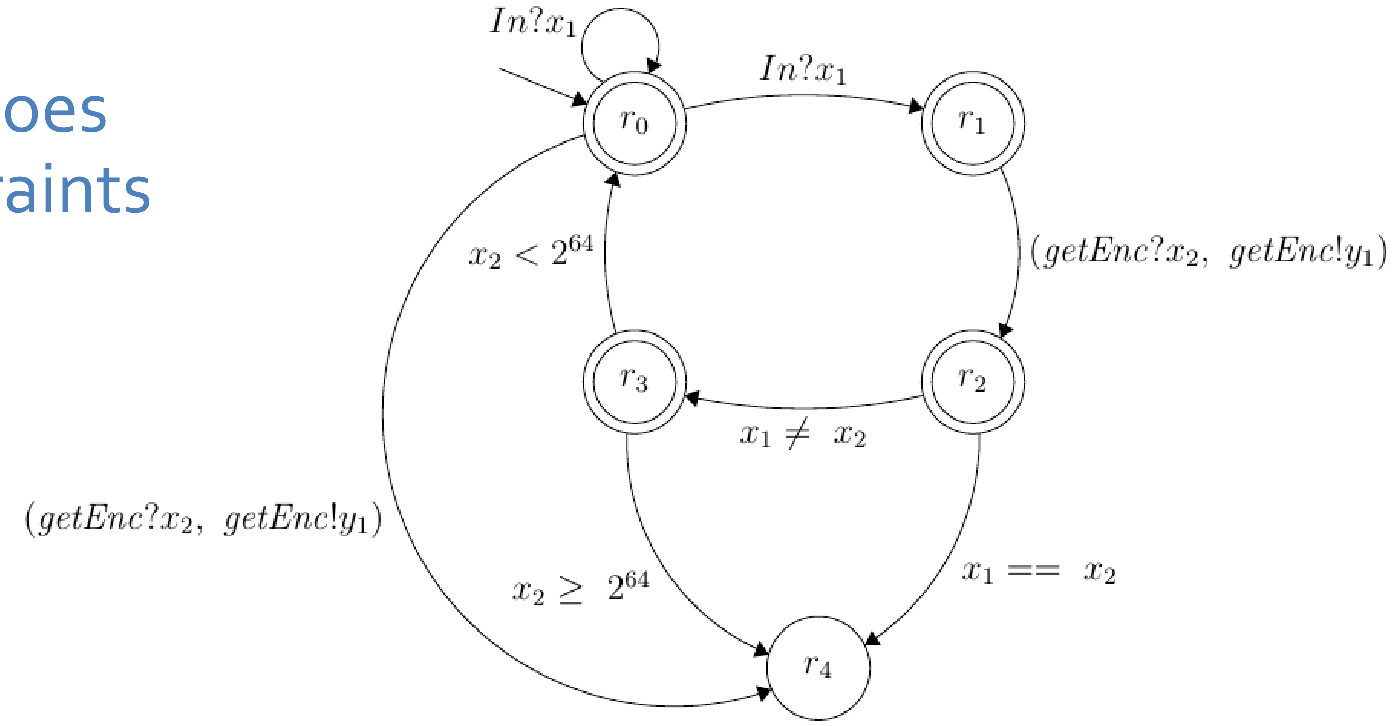


Assume Guarantee or Repair

- Repair by elimination of error traces
- Two types of repair
  - Syntactic repair
  - Semantic repair

Assume Guarantee or **Repair**

Syntactic repair – counterexample does not contain constraints

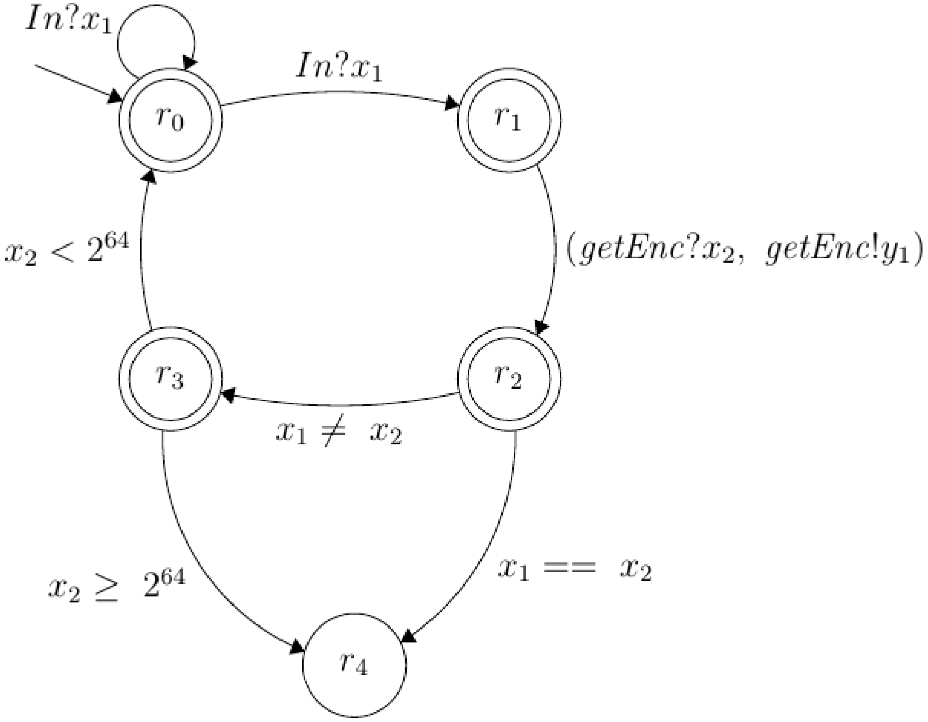


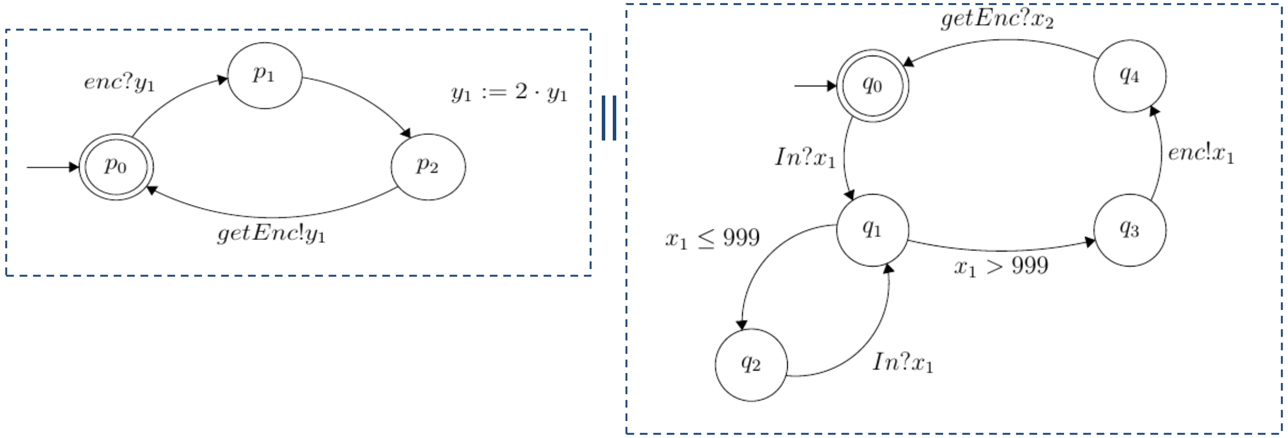
Syntactic Repair

- Implemented 3 methods to removing the trace $\boldsymbol{t}$:
  - **Exact**
    remove exactly $\boldsymbol{t}$ from M_2
  - **Approximate**
    add an intermediate state and use it to direct some traces off the accepting state, including $\boldsymbol{t}$
  - **Aggressive**
    make the accepting state that $\boldsymbol{t}$ reaches not-accepting

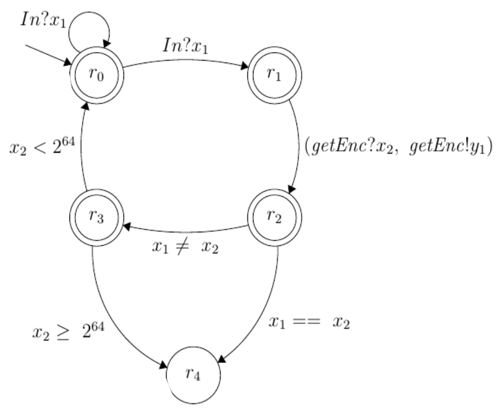
Assume Guarantee or **Repair**

Semantic repair – counterexample contains violated constraints of the specification





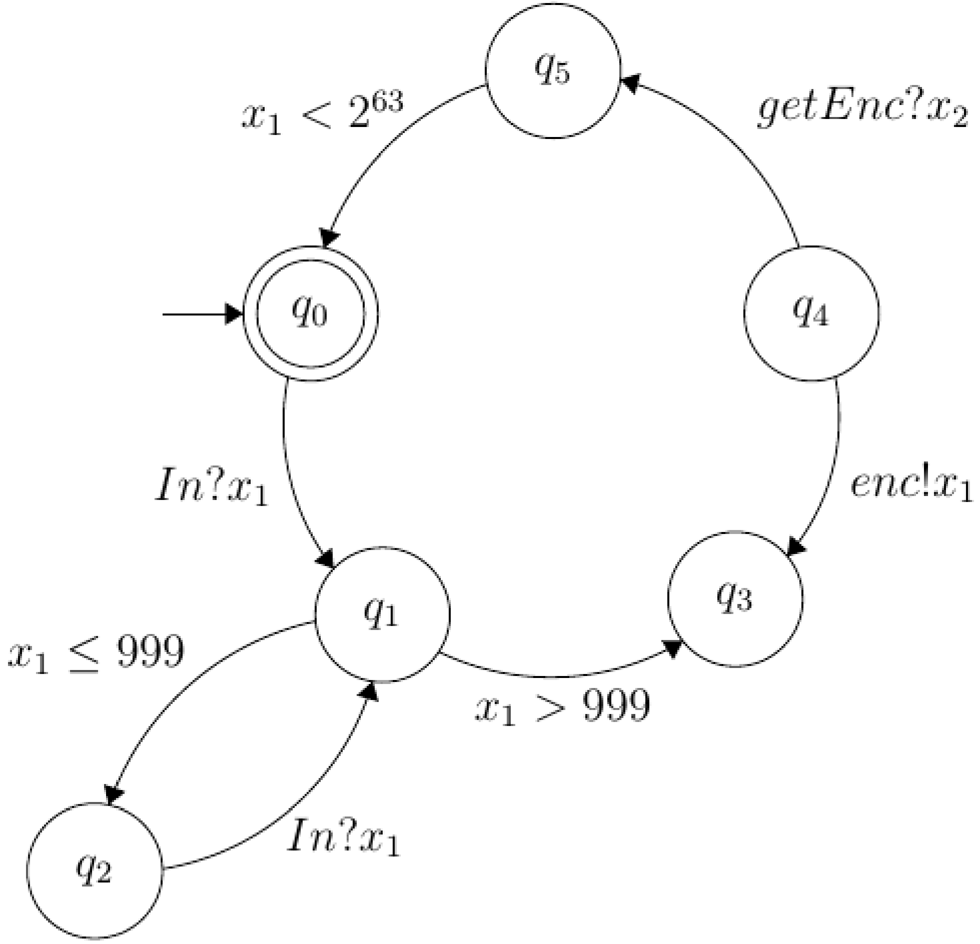
Semantic Repair



- AGR returns a counterexample $\boldsymbol{t}$, for input $x_1 = 2^{63}$
- Goal: make $\boldsymbol{t}$ infeasible by adding a new constraint $\boldsymbol{\mathcal{C}}$ such that
  - $(\varphi_t \wedge \boldsymbol{\mathcal{C}} \rightarrow \mathit{false})$
- Applying abduction, quantifier elimination and simplification results in $\boldsymbol{\mathcal{C}} = (x_1 < 2^{63})$

Result

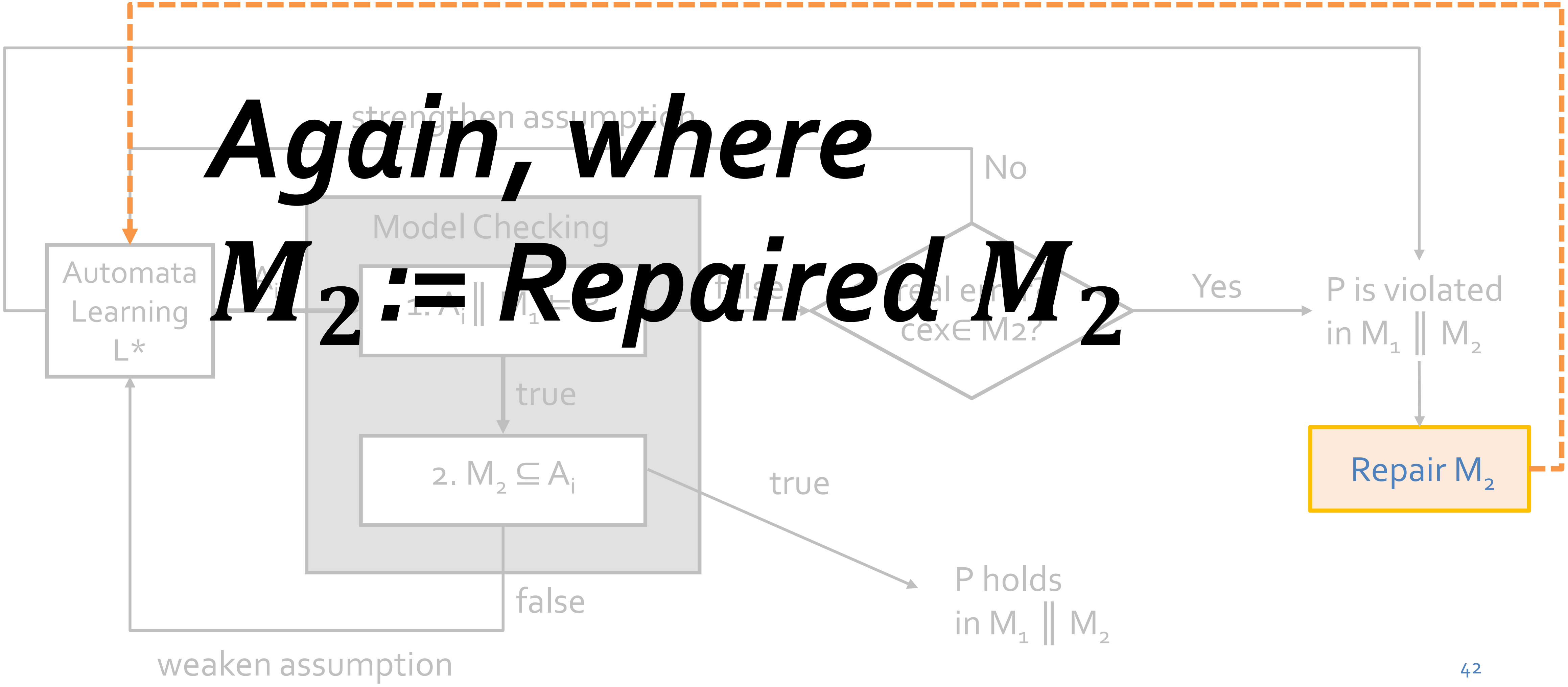
```
while (true)
    pass = readInput;
    while (pass ≤ 999)
        pass = readInput;
    pass2 = encrypt(pass);
    assume pass<2^63;
```



AG rule with learning

Return to verification with the repaired M2

Again, where $M_2 := \text{Repaired } M_2$



Termination

- In case $M_1 || M_2 \vDash P$
- M_2 is a correct assumption for the AG rule
- M_2 is regular, therefore L^* terminates

→ In the case of *verification*, termination is guaranteed

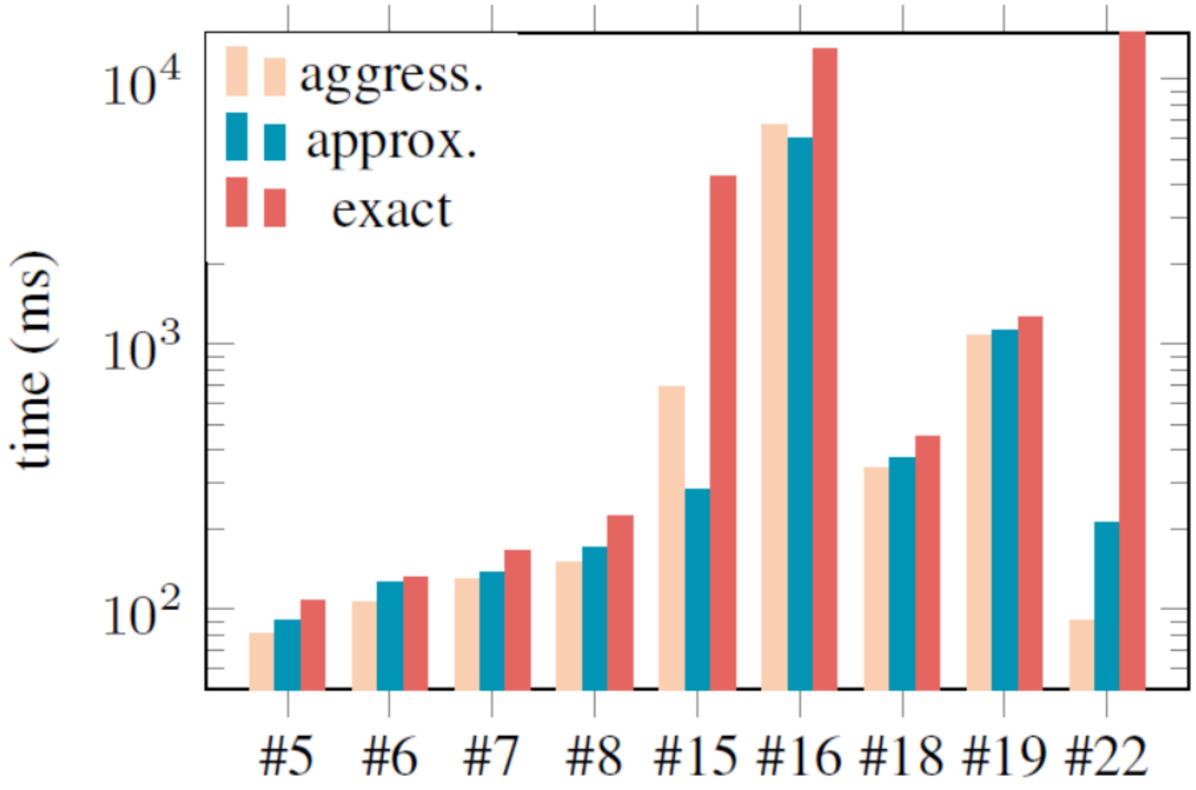
$$M_1 || \boldsymbol{M_2} \vDash P$$
$$M_2 \vDash \boldsymbol{M_2}$$

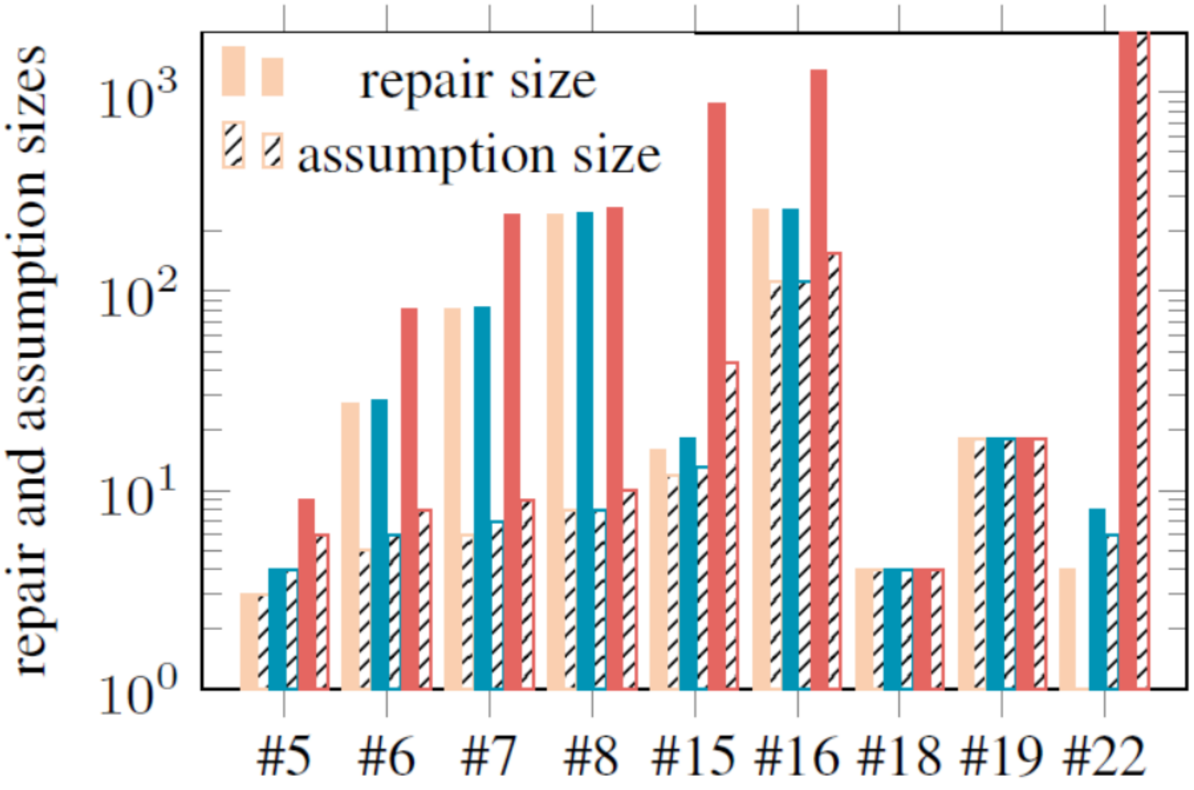
$$M_1 || M_2 \vDash P$$

- In case $M_1 || M_2 \nvDash P$
- Every iteration with an erroneous M_2 will result in a cex

→ In the case of an error, *progress* is guaranteed

Comparing Repair Methods (logarithmic scale)





#15, #16, #18, #19 apply also abduction

AGR Summary

- Modular verification for communicating systems
- Adjusting automata learning to systems with data
- Iterative and incremental verification and repair to prove correctness of repaired system

Thank you! Questions?

- Modular verification for communicating systems
- Adjusting automata learning to systems with data
- Iterative and incremental verification and repair to prove correctness of repaired system