AUTOMATA OVER INFINITE DATA DOMAINS: LEARNABILITY AND APPLICATIONS IN PROGRAM VERIFICATION AND REPAIR

Hadar Frenkel

Advisors: Orna Grumberg & Sarai Sheinvald

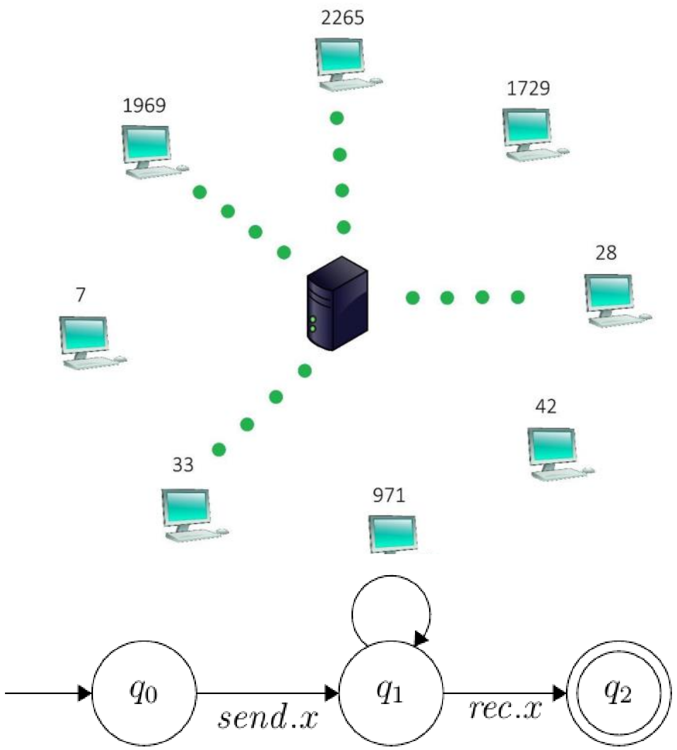
AUTOMATA OVER INFINITE DATA DOMAINS: LEARNABILITY AND APPLICATIONS IN PROGRAM VERIFICATION AND REPAIR

Hadar Frenkel

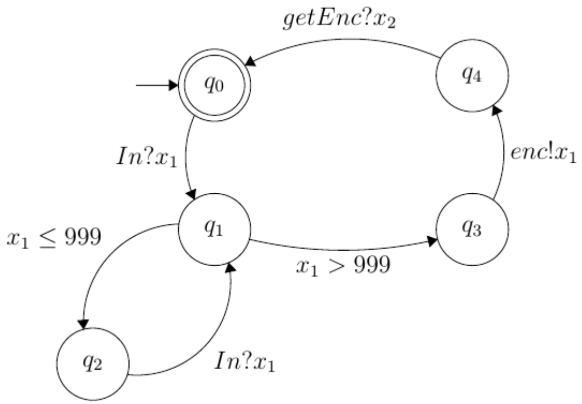
Advisors: Orna Grumberg & Sarai Sheinvald

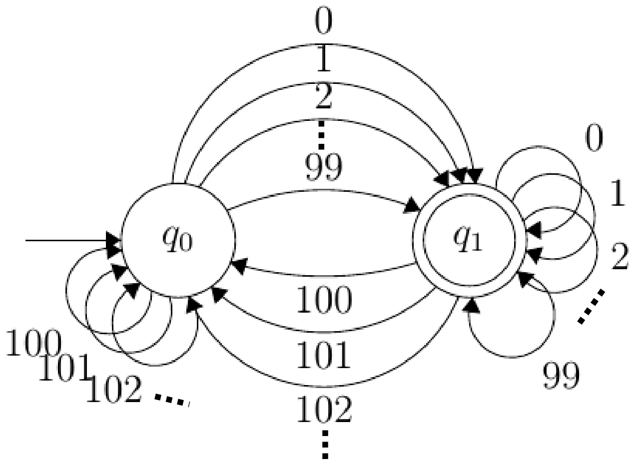
Automata over Infinite Data Domains

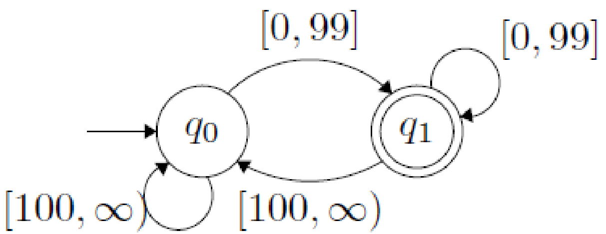
- Model infinite-state system using a finite model




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





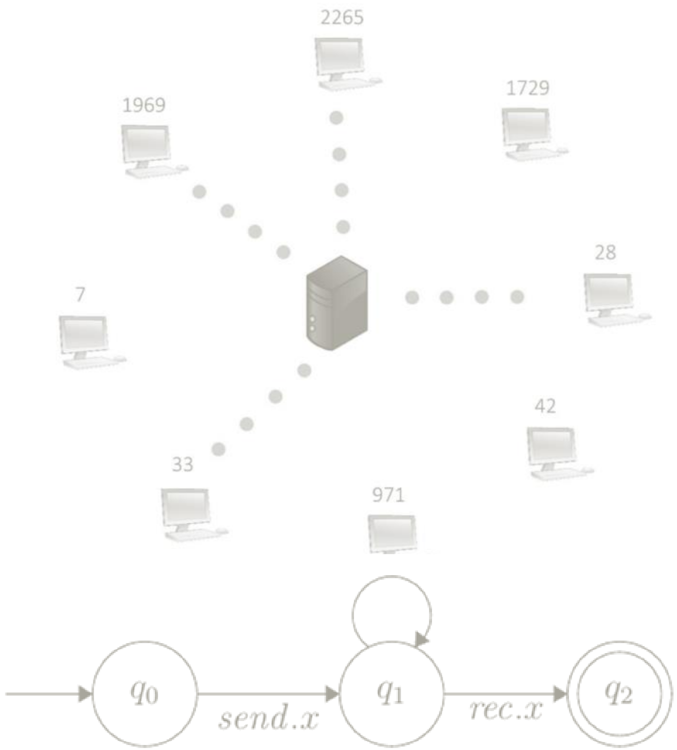


AUTOMATA OVER INFINITE DATA DOMAINS: LEARNABILITY AND APPLICATIONS IN PROGRAM VERIFICATION AND REPAIR

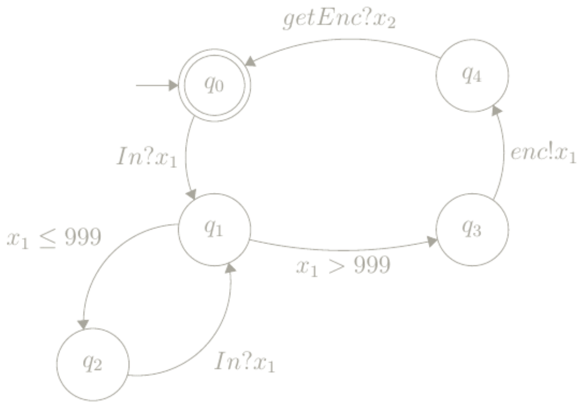
Hadar Frenkel

Advisors: Orna Grumberg & Sarai Sheinvald

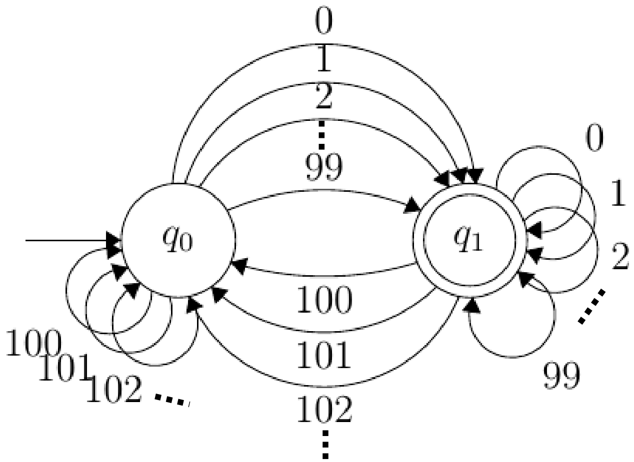
Learnability

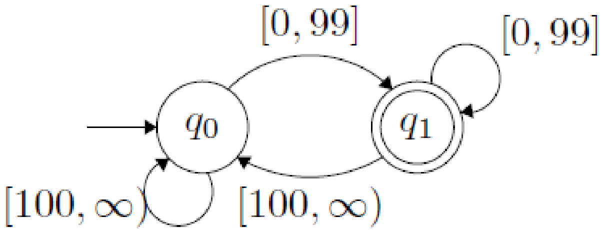



```
while (true)
      pass = readInput;
      while (pass ≤ 999)
        pass = readInput;
      pass2 = encrypt(pass);
```



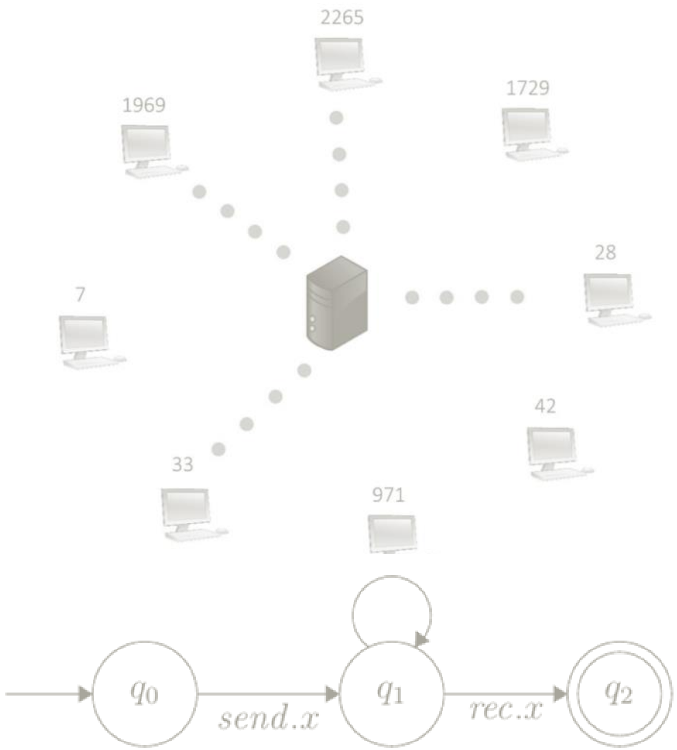
Learning symbolic automata (conditions for learning: L* and identification in the limit)





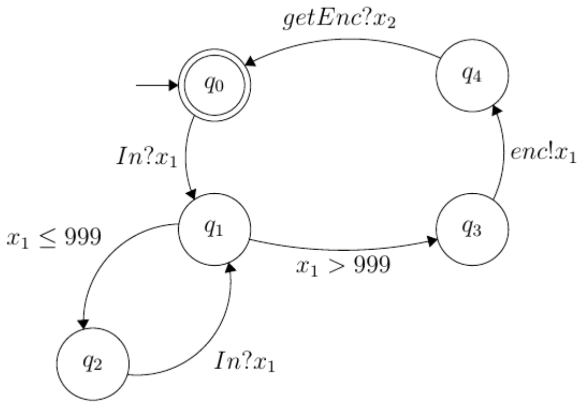
[Fisman, Frenkel, Zilles]

Learnability



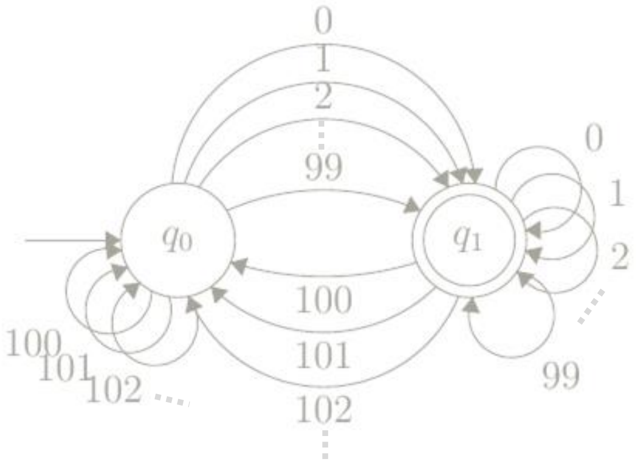
Adapting L* algorithm for communicating programs

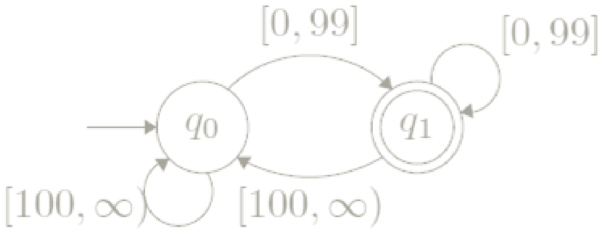
```
while (true)
      pass = readInput;
      while (pass ≤ 999)
        pass = readInput;
      pass2 = encrypt(pass);
```



[Frenkel, Grumberg, Pasareanu, Sheinvald 20]

Learning symbolic automata (conditions for learning: L* and identification in the limit)





[Fisman, Frenkel, Zilles]

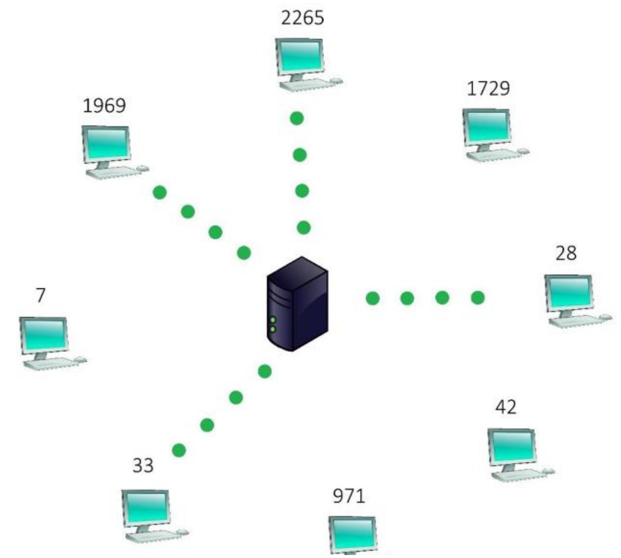
AUTOMATA OVER INFINITE DATA DOMAINS: LEARNABILITY AND APPLICATIONS IN PROGRAM VERIFICATION AND REPAIR

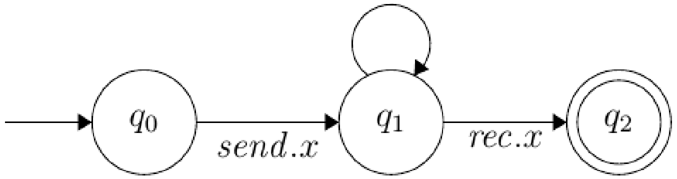
Hadar Frenkel

Advisors: Orna Grumberg & Sarai Sheinvald

Applications in Program Verification and Repair

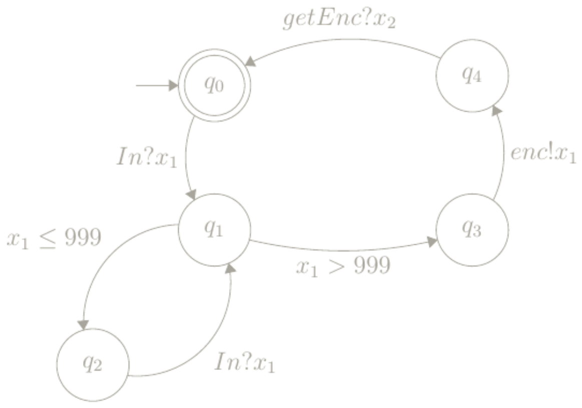
Bounded model-checking algorithm



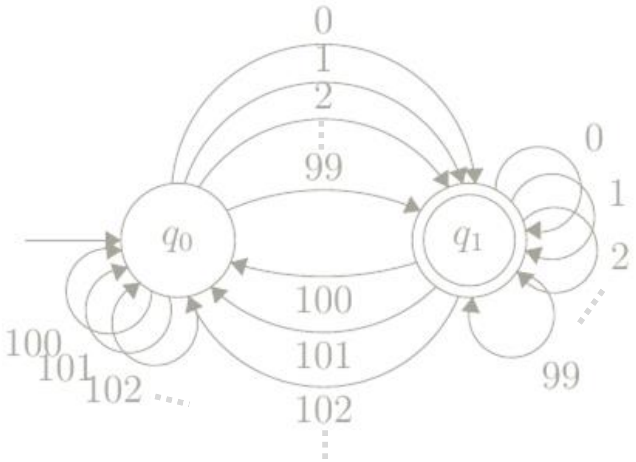


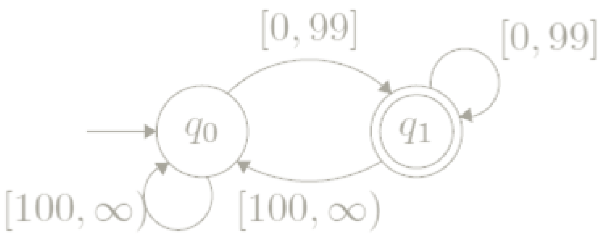
[Frenkel, Grumberg, Sheinvald 17, 19]

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



[Frenkel, Grumberg, Pasareanu, Sheinvald 20]

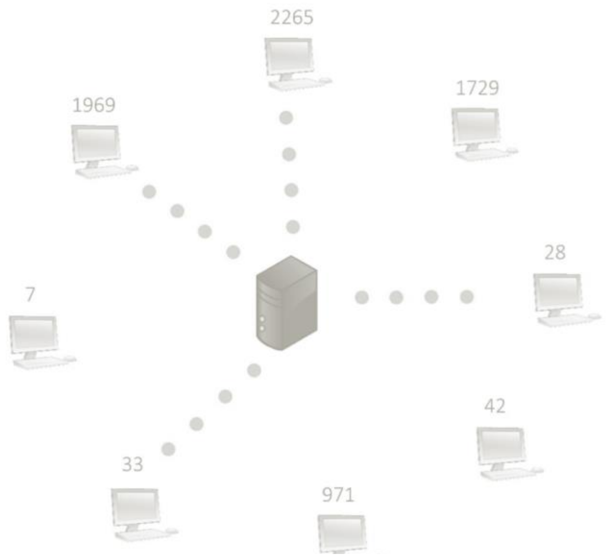


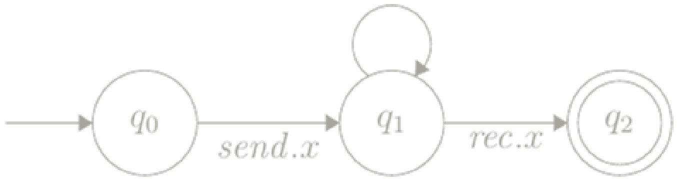


[Fisman, Frenkel, Zilles]

Applications in Program Verification and Repair

Bounded model-checking algorithm

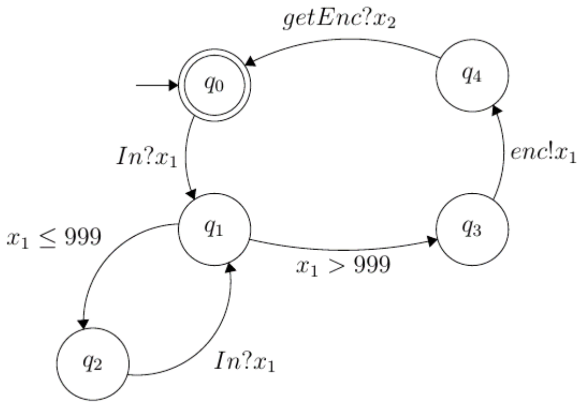




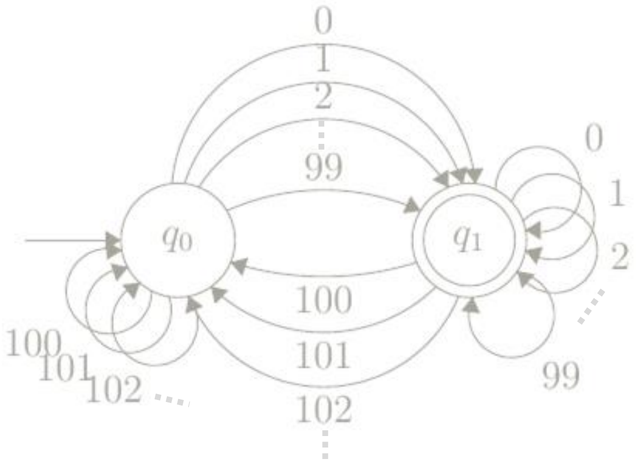
[Frenkel, Grumberg, Sheinvald 17, 19]

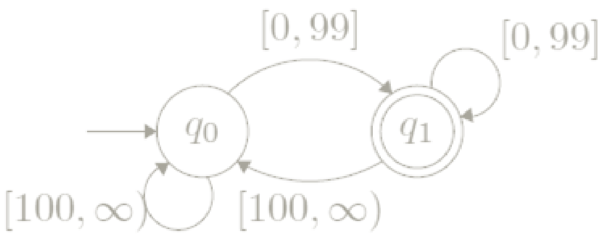
Compositional verification and repair algorithm

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



[Frenkel, Grumberg, Pasareanu, Sheinvald 20]





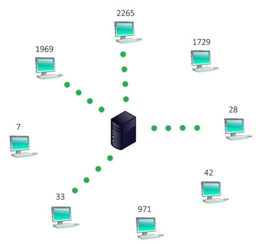
[Fisman, Frenkel, Zilles]

MODEL CHECKING SYSTEMS OVER INFINITE DATA

Joint work with Orna Grumberg and Sarai Sheinvald

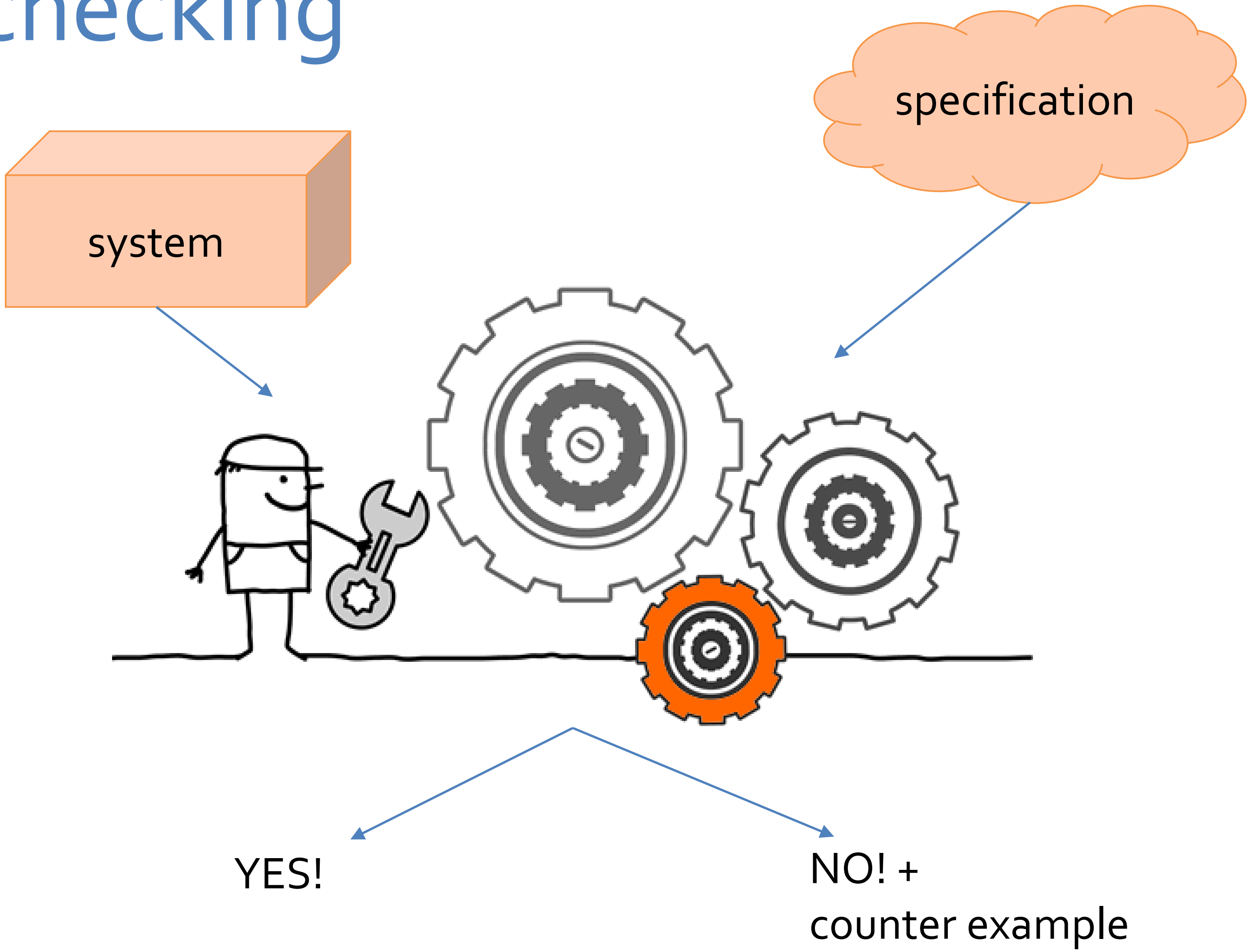
@NFM 2017, @Journal of automated reasoning 2019

Goal

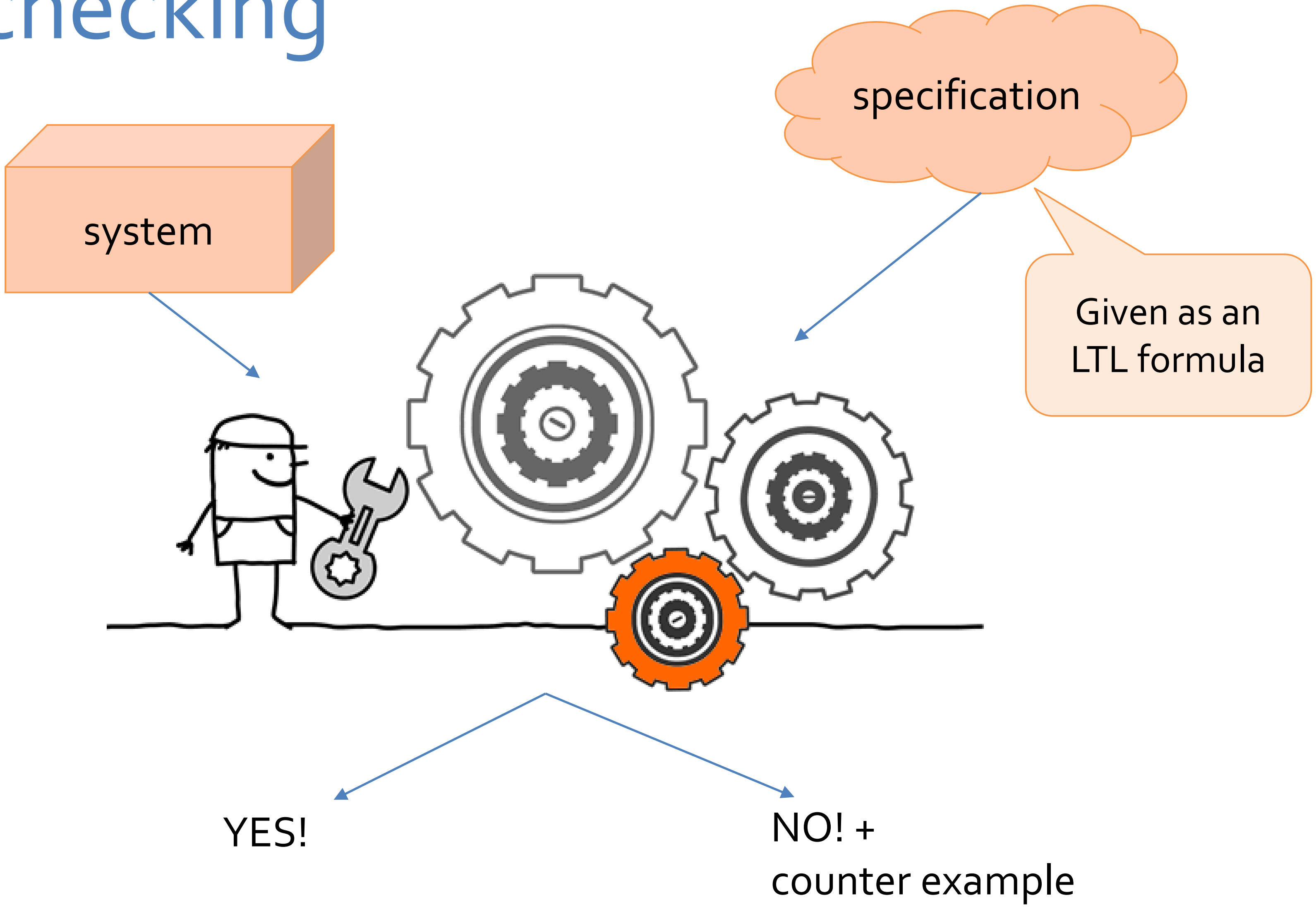


- Develop a Model checking process for systems over infinite data domains
- Using the automata-theoretic approach

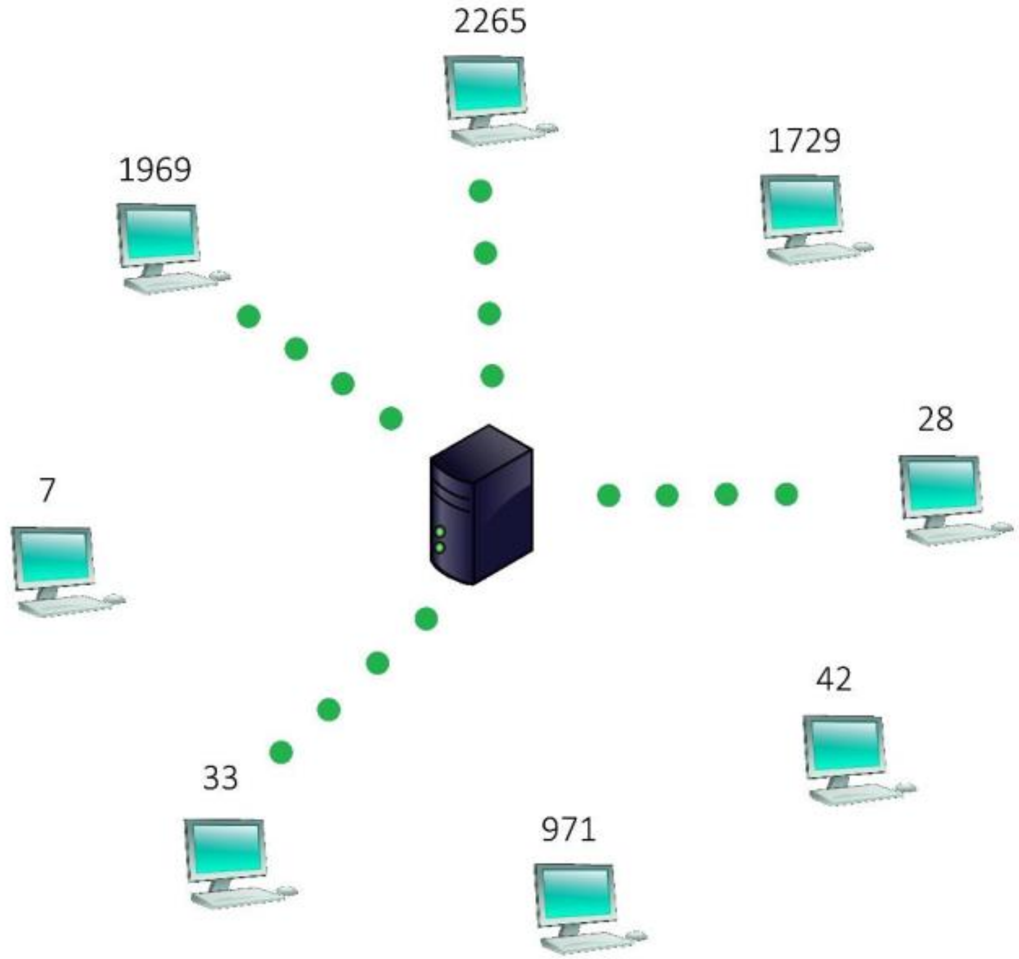
Model checking



Model checking



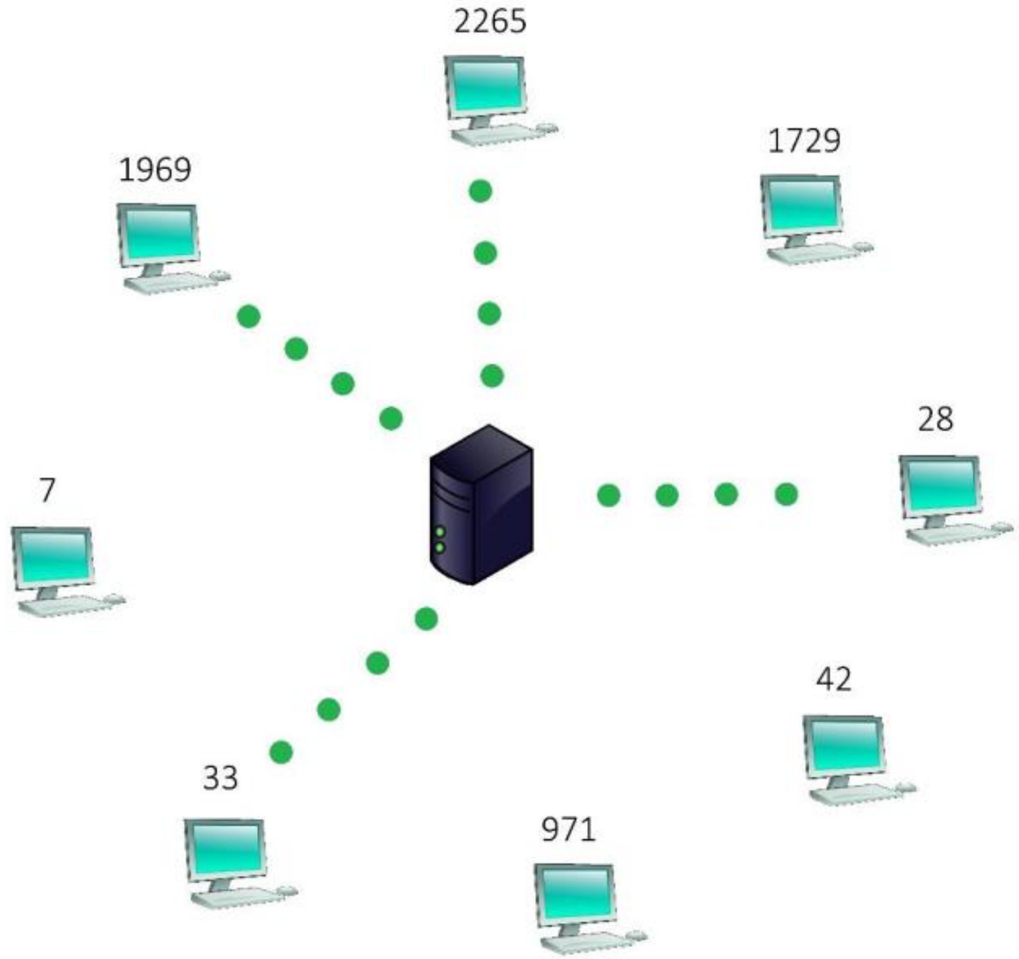
Verification of Systems over Infinite Data Domains



Verification of Systems over Infinite Data Domains

- LTL cannot express the property

"every client is eventually active"



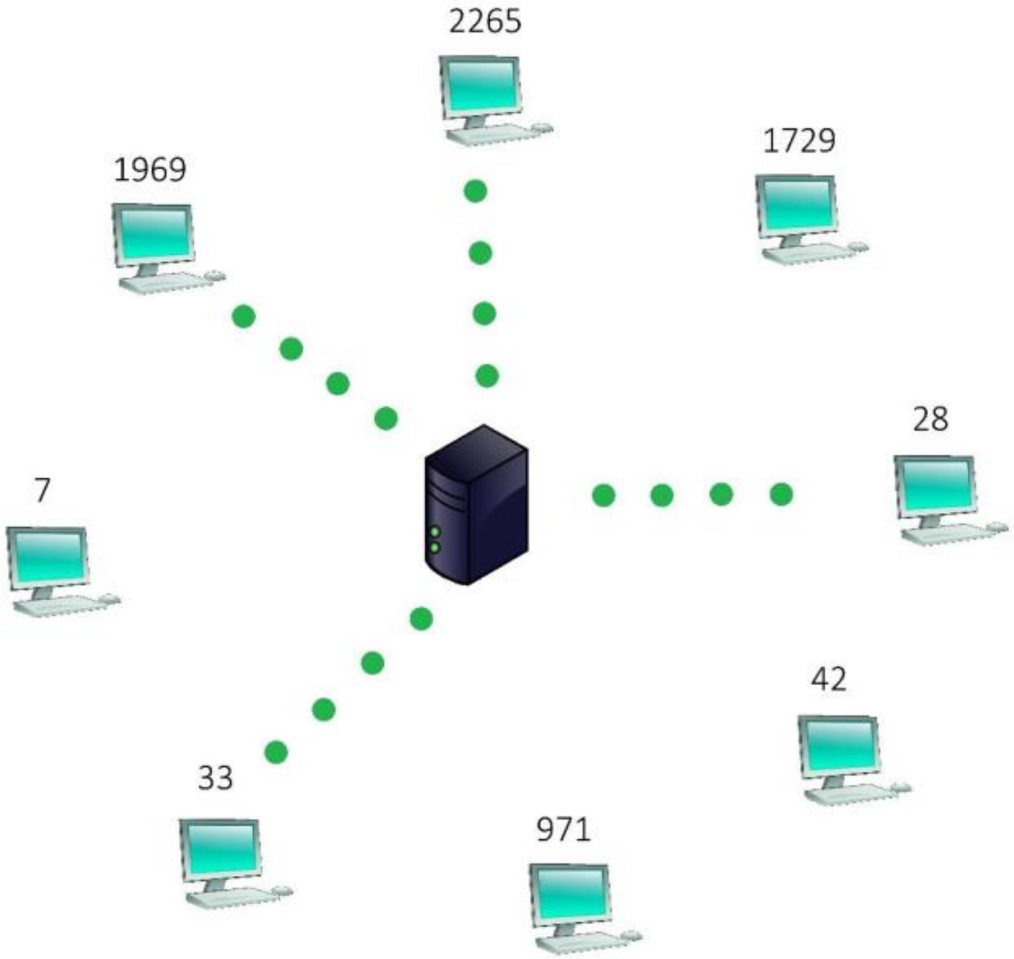
Verification of Systems over Infinite Data Domains

- LTL cannot express the property

"every client is eventually active"

Variable LTL (VLTL) [GKS12]

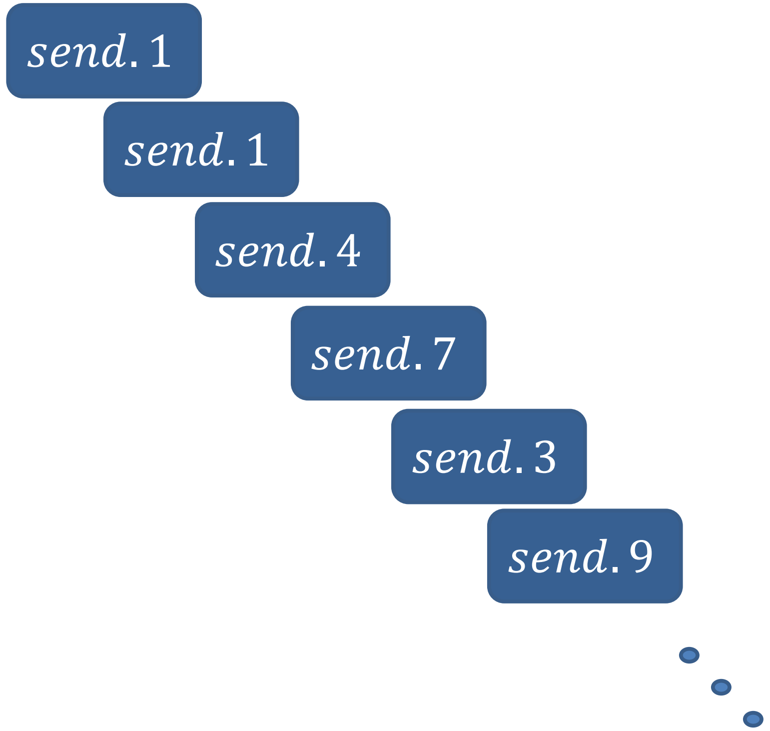
- $\forall x: F\ active.x$
- AP - finite set of (parameterized) propositions
- V - finite set of quantified variables



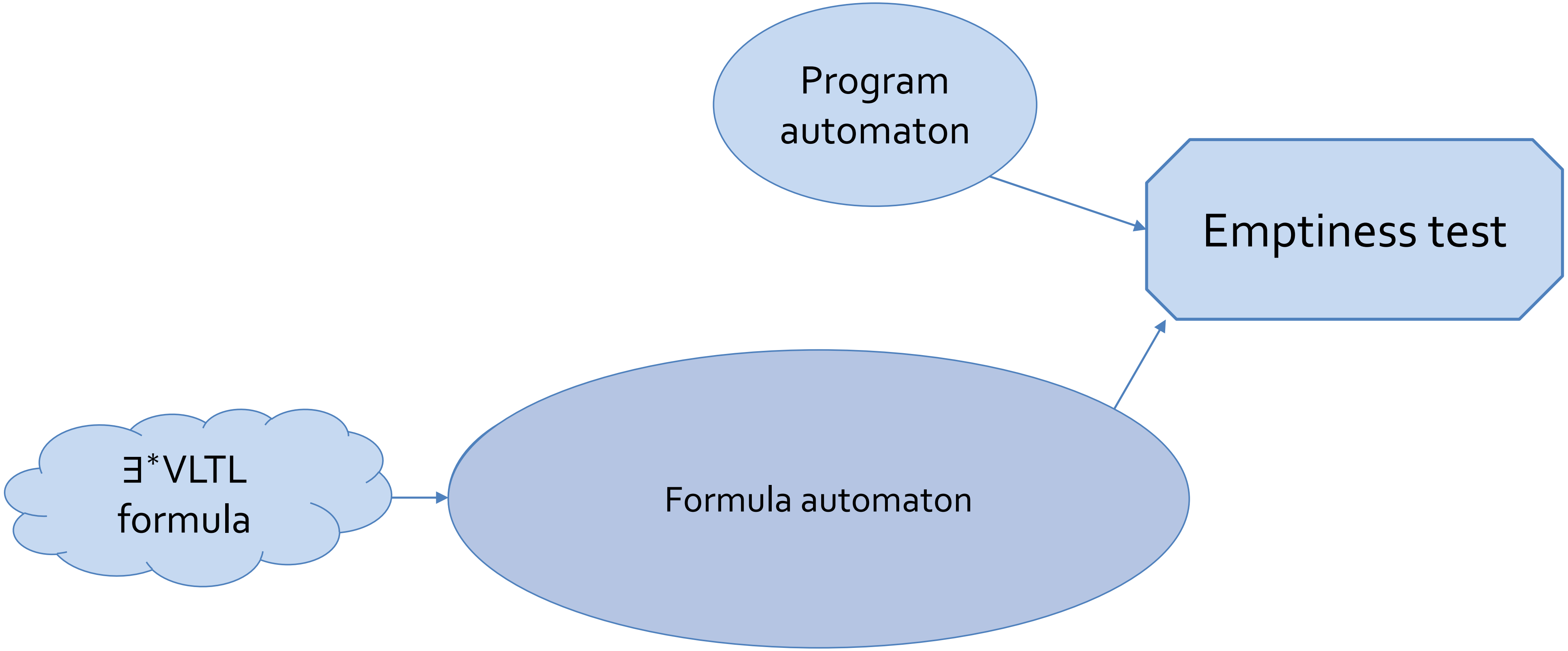
$\exists^*$VLTL [GKS12]

- VLTL with only existential quantifiers
- $G\ \exists x{:}\ send.x$
- A possible satisfying computation

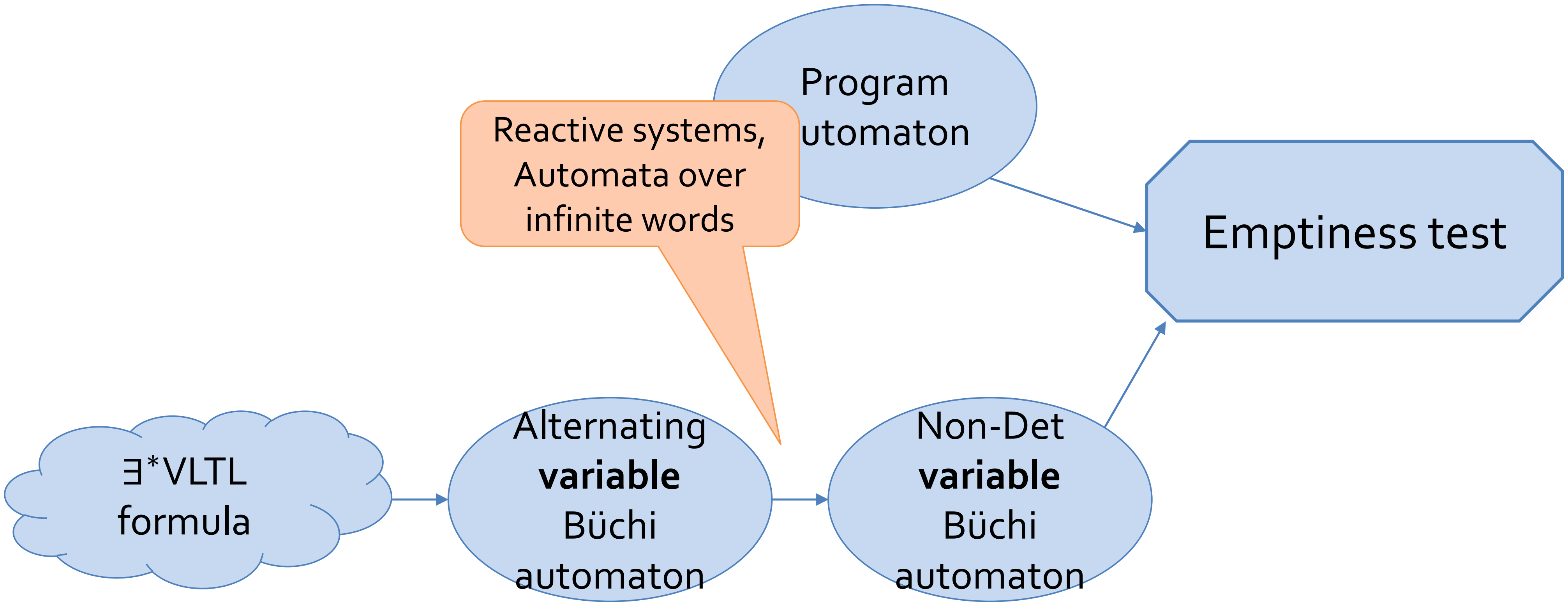
- We are interested in verifying universal properties, the negation that describes a bad behavior is existential



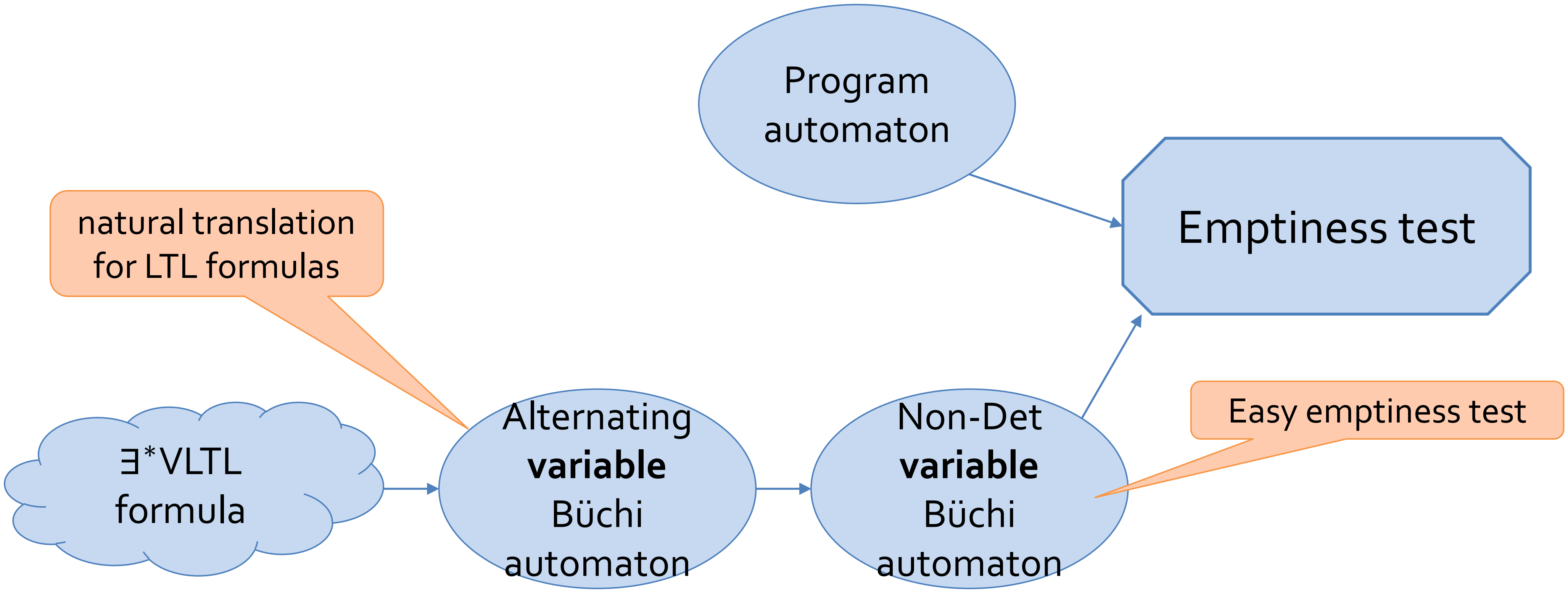
Model Checking - Infinite Data Domains



Model Checking - Infinite Data Domains

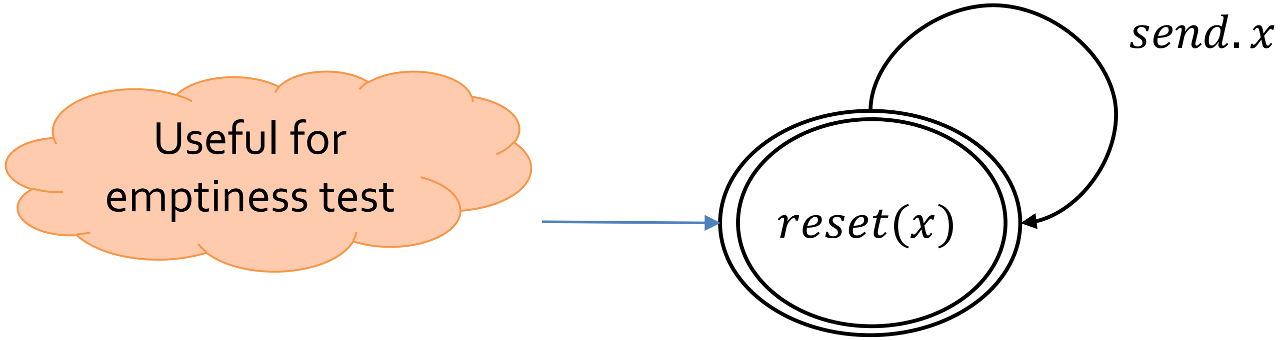


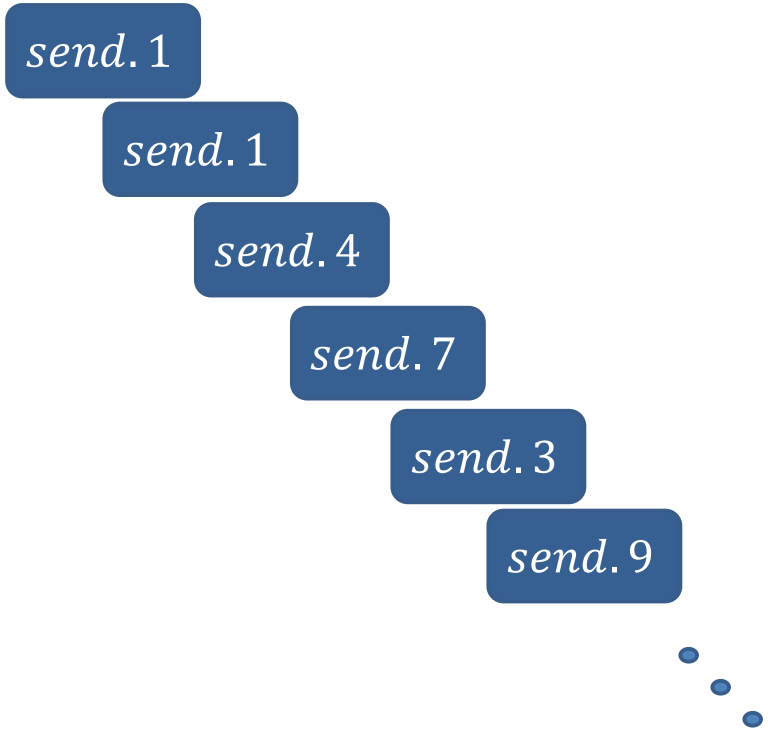
Model Checking - Infinite Data Domains



Non-Deterministic Variable Büchi Automata (NVBW) [GKS13]

- $G\ \exists x: send.x$
- Alphabet is parameterized propositions
- Ability to reset a variable and to assign it a new value
- As long as there is no reset - the value cannot be changed





NVBW Cannot Express all $\exists^*$VLTL

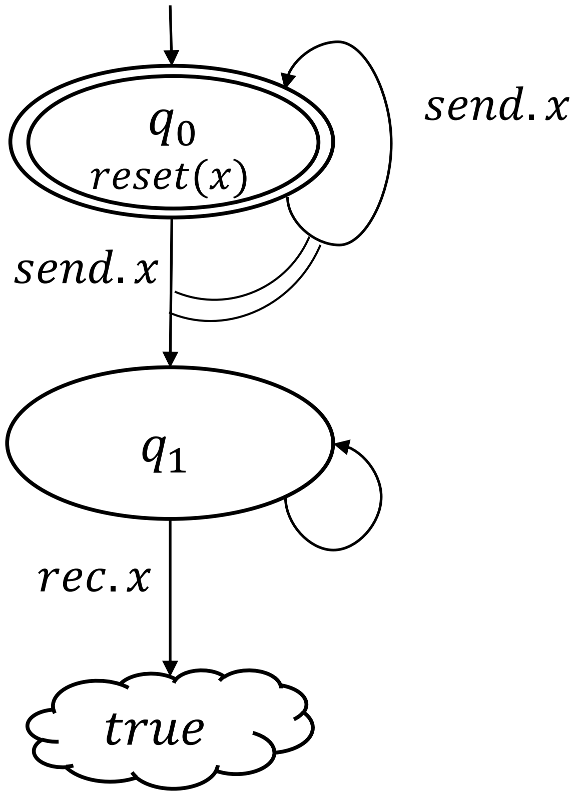
- $G\ (\exists x: send.x \wedge XF\ receive.x)$

$send.1$ | $send.2$, $rec.1$ | $send.3$ | $send.4$, $rec.2$ | $send.5$ | $send.6$, $rec.3$ | $send.7$ | $send.8$, $rec.4$ | . . .

- Increasing gaps between $send.x, receive.x$.
- Not enough variables and states to remember all values

Alternating Variable Büchi Automata (AVBW)

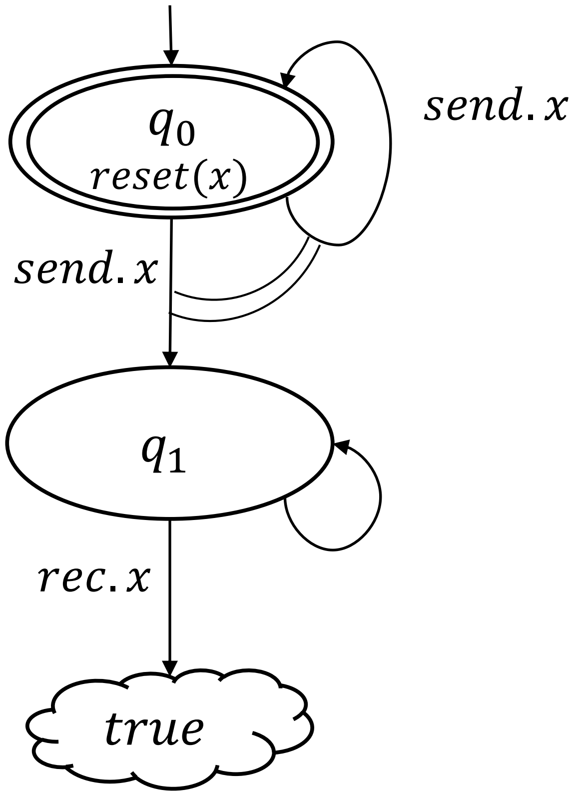
- $G\ (\exists x{:}\ send.x \land XF\ receive.x)$



Alternating Variable Büchi Automata (AVBW)

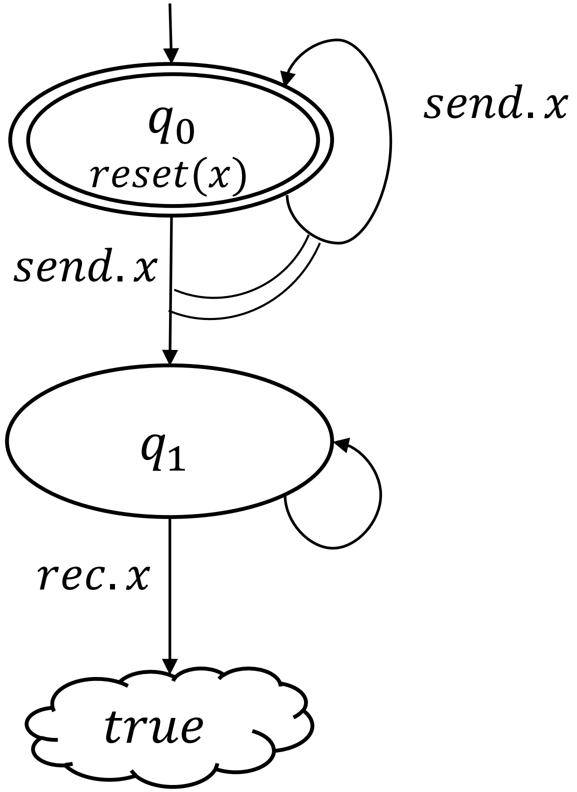
- $G\ (\exists x\!: send.x \wedge XF\ receive.x)$

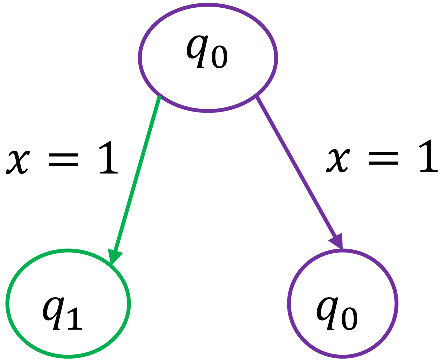
Easy construction from $\exists^*$ VLTL



Alternating Variable Büchi Automata (AVBW)

- $G\ (\exists x\colon send.x \wedge XF\ receive.x)$

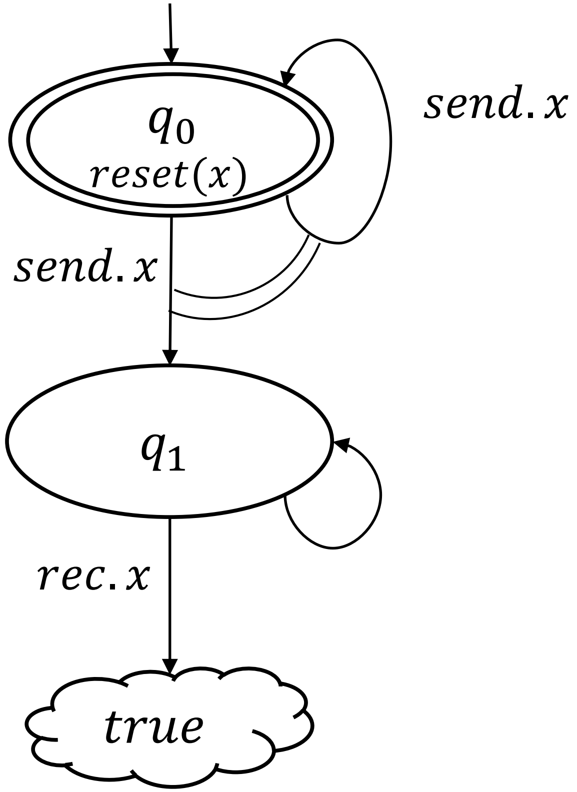


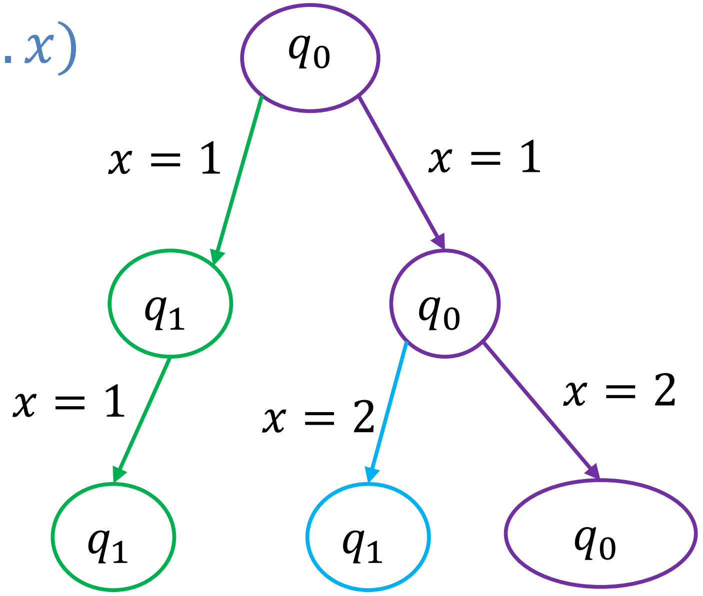


$send.1$

Alternating Variable Büchi Automata (AVBW)

- $G\ (\exists x\colon send.x \wedge XF\ receive.x)$



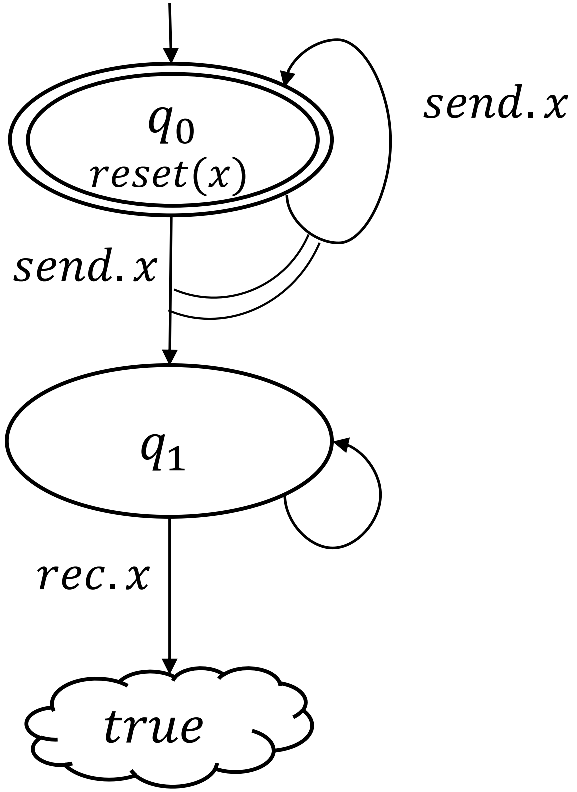


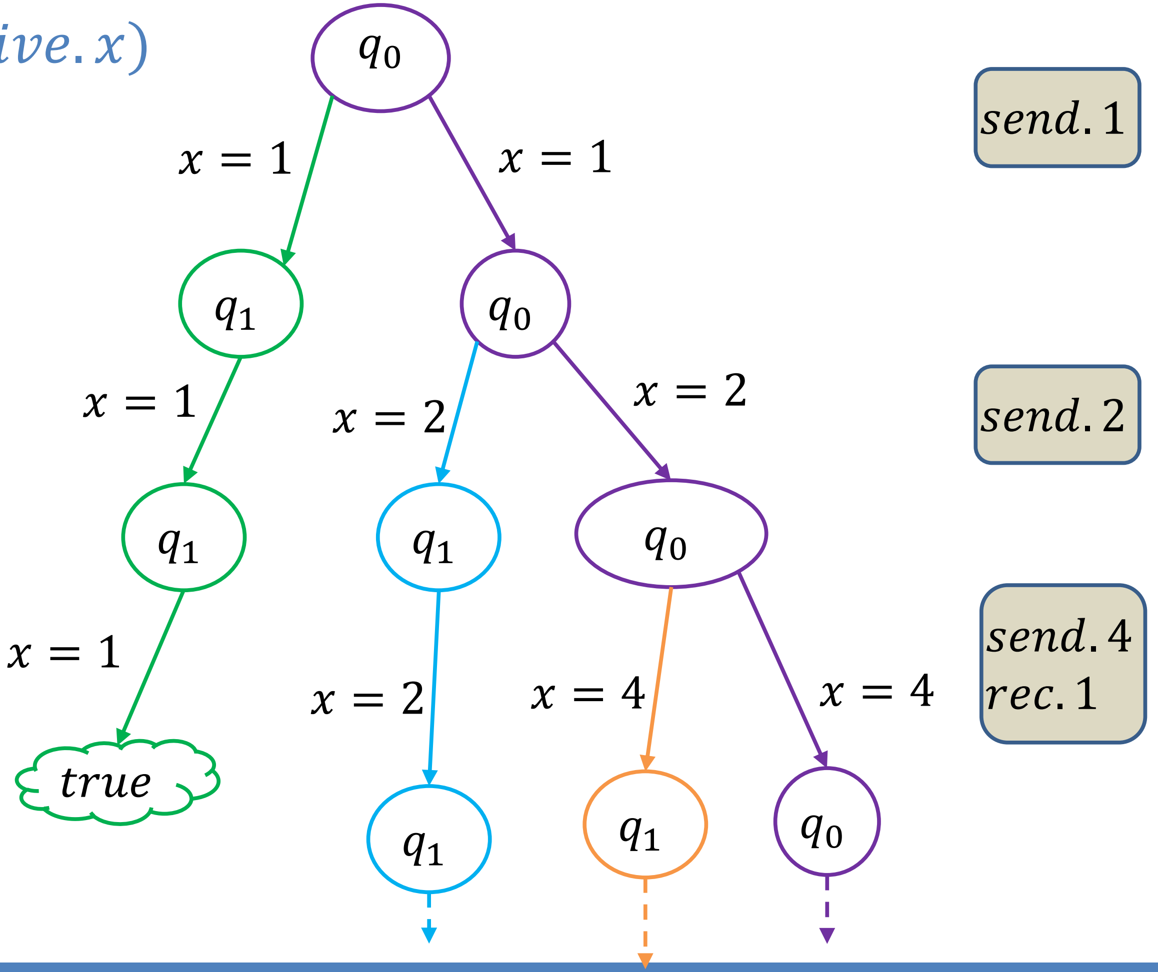
$send.1$

$send.2$

Alternating Variable Büchi Automata (AVBW)

- $G\ (\exists x{:}\ send.x \wedge XF\ receive.x)$



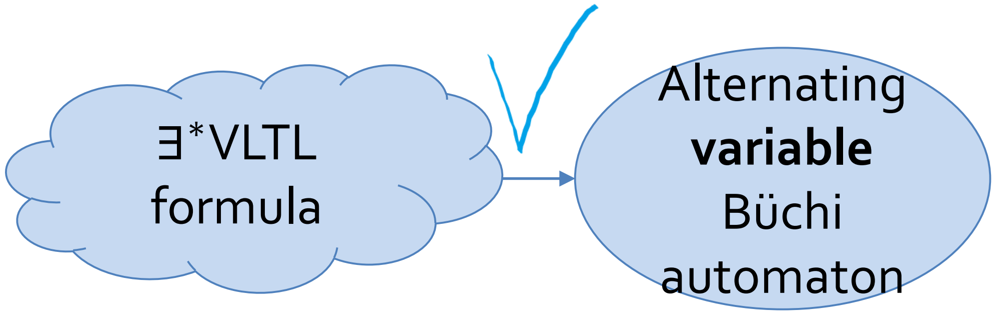


VLTL to AVBWs

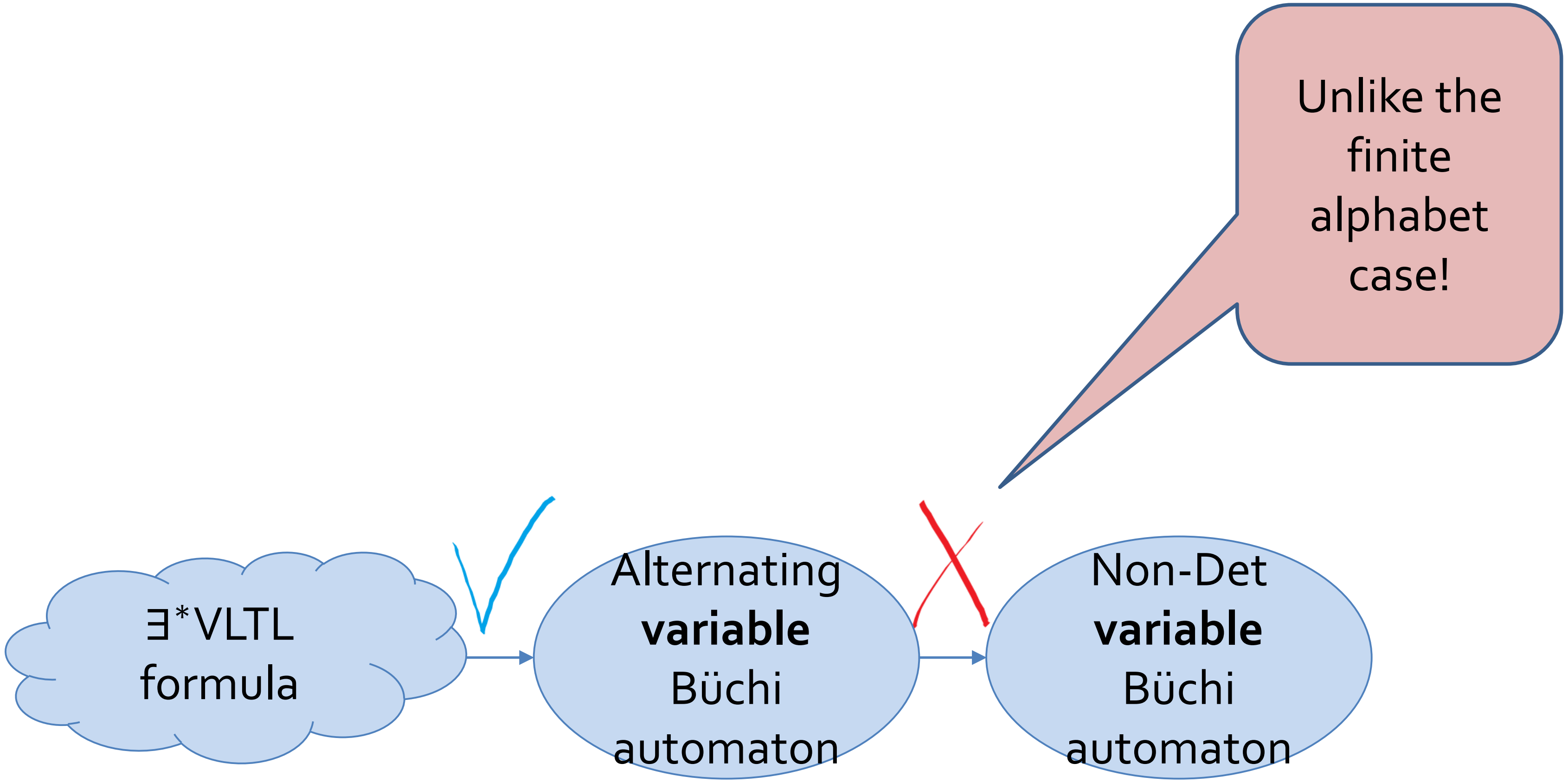
- Similar to [V95]
- Special care of resets
- $X = vars(\varphi) \cup \{x_p | p \in \mathrm{AP}\}$
- $Q = sub(\varphi)$
- Reset
  - x_p varaibles
  - variables under $\exists$
- $x \neq y$ for $\neg a.x \in sub(\varphi)$

- $\delta(a.x, A) = \mathit{true}$ if $a.x \in A$ and $\delta(a.x, A) = \mathit{false}$, otherwise.
- $\delta(\neg a.x, A) = \neg\delta(a.x, A)$.[4]
- $\delta(\eta \wedge \psi, A) = \delta(\eta, A) \wedge \delta(\psi, A)$.
- $\delta(\eta \vee \psi, A) = \delta(\eta, A) \vee \delta(\psi, A)$
- $\delta(\mathbf{X}\, \eta, A) = \eta$
- $\delta(\eta\, \mathbf{U}\, \psi, A) = \delta(\psi, A) \vee (\delta(\eta, A) \wedge \eta\, \mathbf{U}\, \psi)$
- $\delta(\eta\, \mathbf{V}\, \psi, A) = \delta(\eta \wedge \psi, A) \vee (\delta(\psi, A) \wedge \eta\, \mathbf{V}\, \psi)$
- $\delta(\exists x \eta, A) = \delta(\eta, A)$

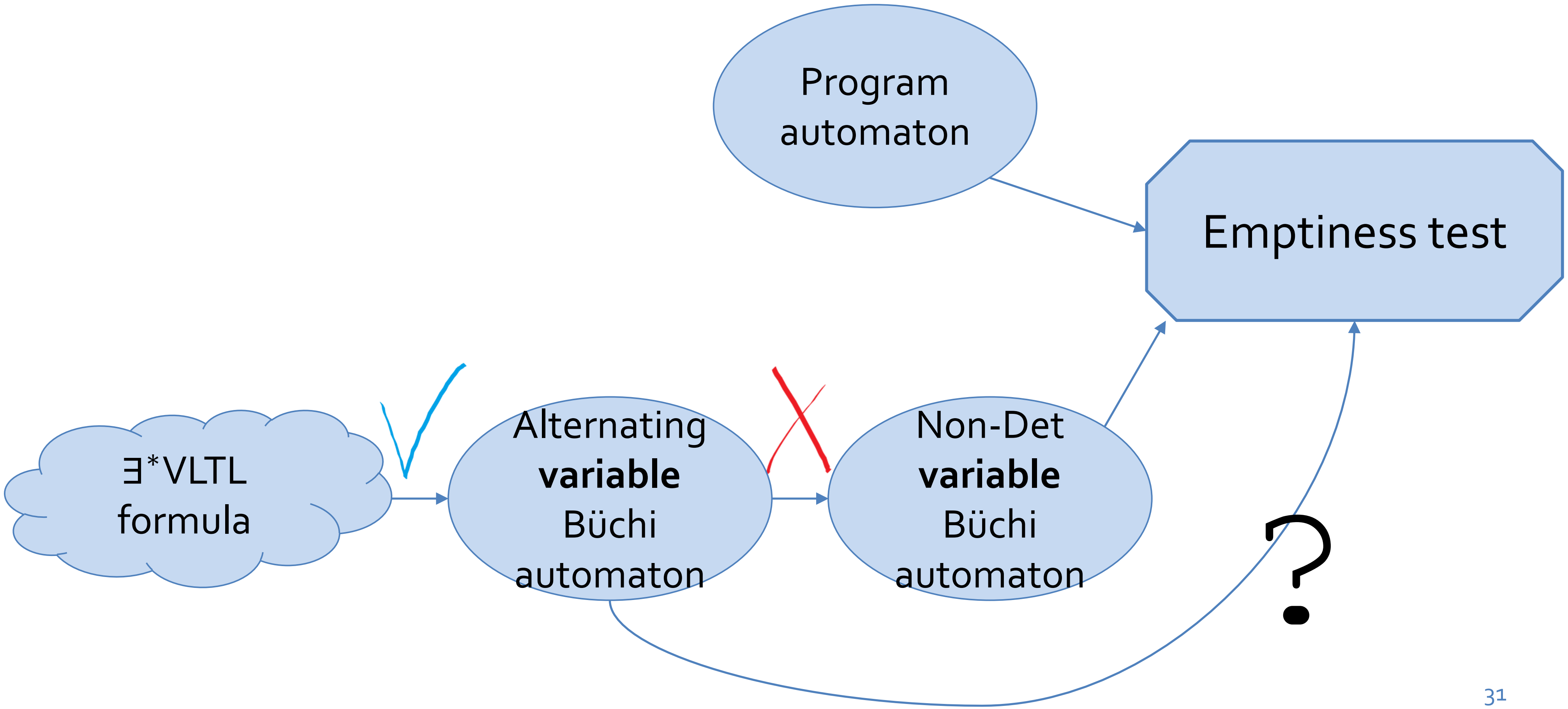
Model Checking - Infinite Data Domains



Model Checking - Infinite Data Domains



Model Checking - Infinite Data Domains



Model Checking - Infinite Data Domains

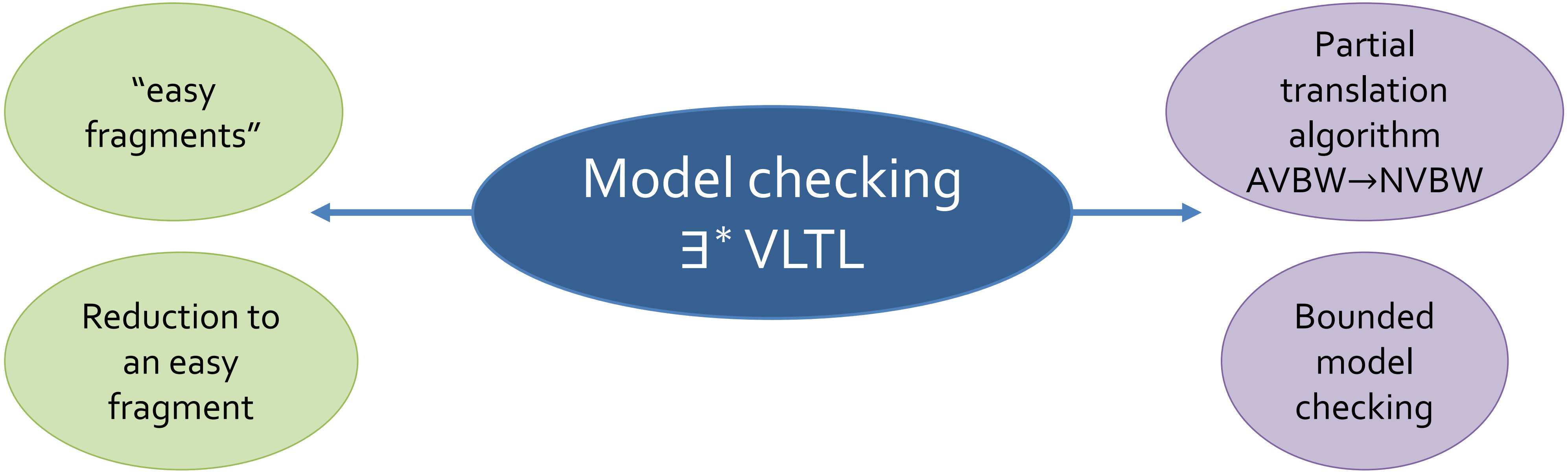
- Emptiness of AVBWs is <u>**undecidable**</u>
- Satisfiability problem of $\exists^*$VLTL formulas is undecidable [SW14]
- $\exists^*$VLTL $\equiv$ AVBW, thus
  - Satisfiability problem $\equiv$ emptiness problem

Emptiness test

Alternating **variable** Büchi automaton

?

Solutions



“easy fragments”
Reduction to an easy fragment
Model checking $\exists^*$ VLTL
Partial translation algorithm AVBW→NVBW
Bounded model checking

$\exists^*$VLTL Formulas with a Direct Construction to NVBW

"easy fragments"

- PNF formulas $\exists x: G\ send.x$ $(send.7)^\omega$
- X, F formulas
- Quantifiers are at the beginning \ next to atomic propositions $\exists x_1: G\ send.x_1 \wedge G\ \exists x_2: rec.x_2$

Flattening

Reduction to an easy fragment

- A formula with no negations has an equisatisfiable formula in PNF

$$G\ (\exists x: send.x \wedge XF\ receive.x)$$

Always holds $\Uparrow$ $\Downarrow$ No negations

$$\exists x: G\ (send.x \wedge XF\ receive.x)$$

Translation Algorithm

Partial translation algorithm AVBW→NVBW

- A partial algorithm for translation
- Based on the Miyano-Hayashi construction [MH84]

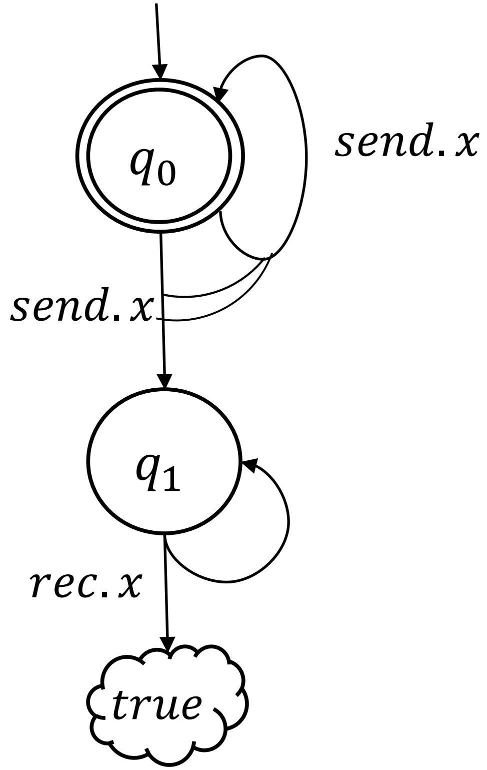
AND

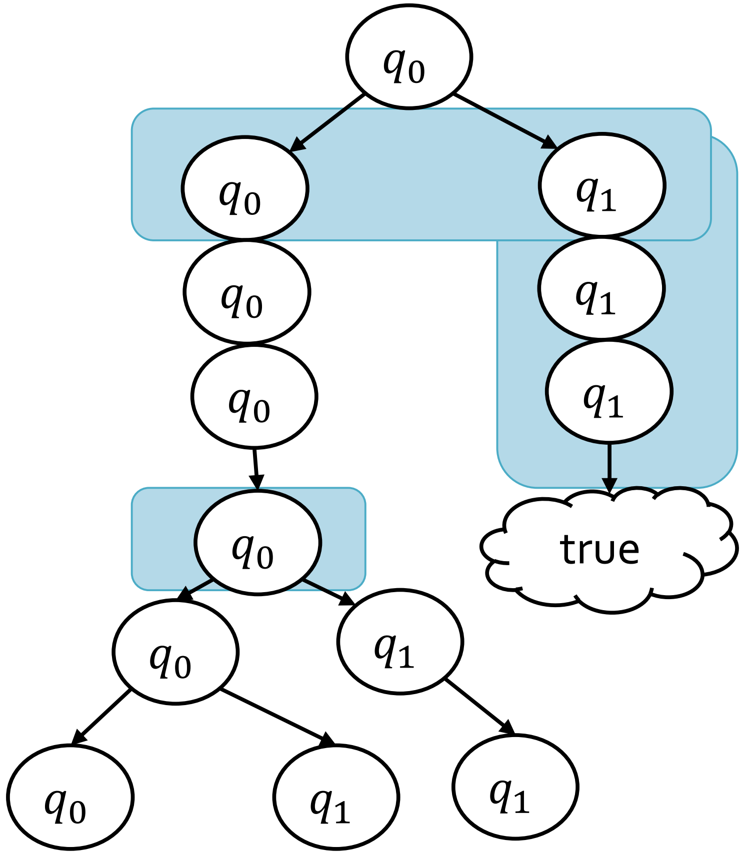
- Take care of variables, resets
- Map variables of alternating automaton to variables of non-deterministic automaton

$$\left(\begin{Bmatrix}(q_0, \emptyset)\\(q_1, x \to z_1)\\(q_1, x \to z_3)\end{Bmatrix}, \{(q_1, x \to z_1)\}\right)$$
$$reset(z_2)$$

Alternating to Non-Deterministic [MH84]

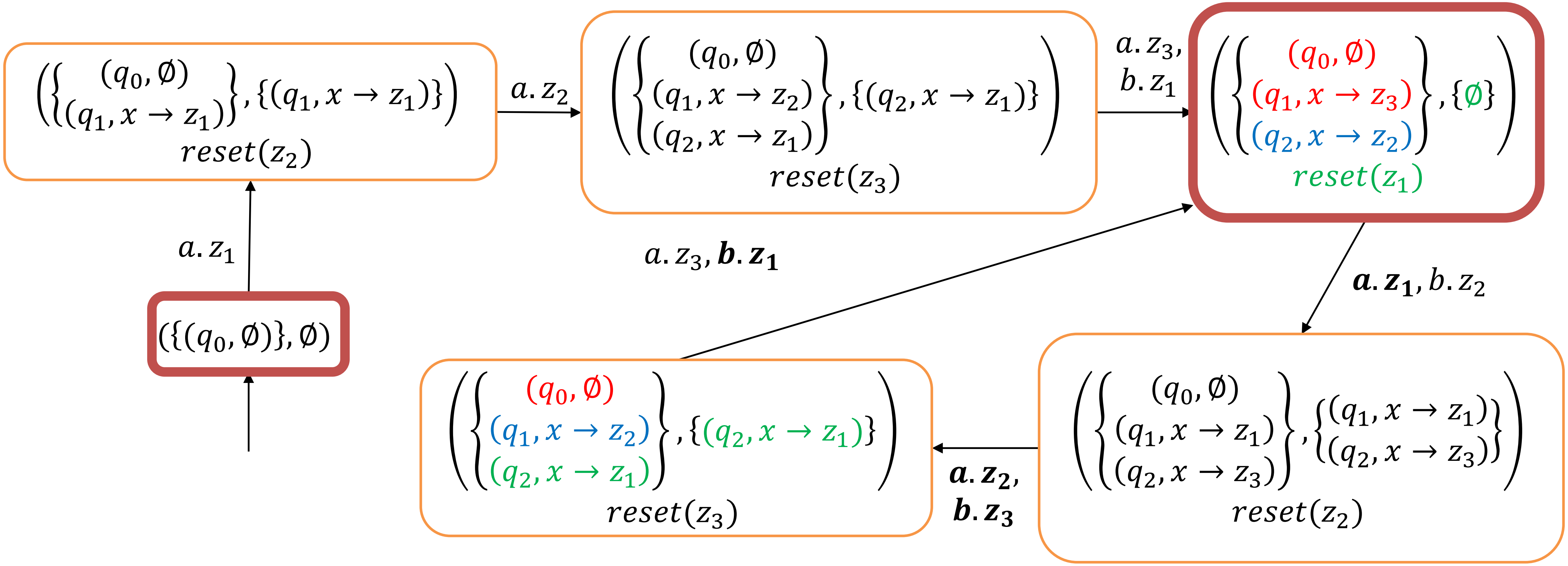
- $G\ (send \rightarrow XF\ receive)$





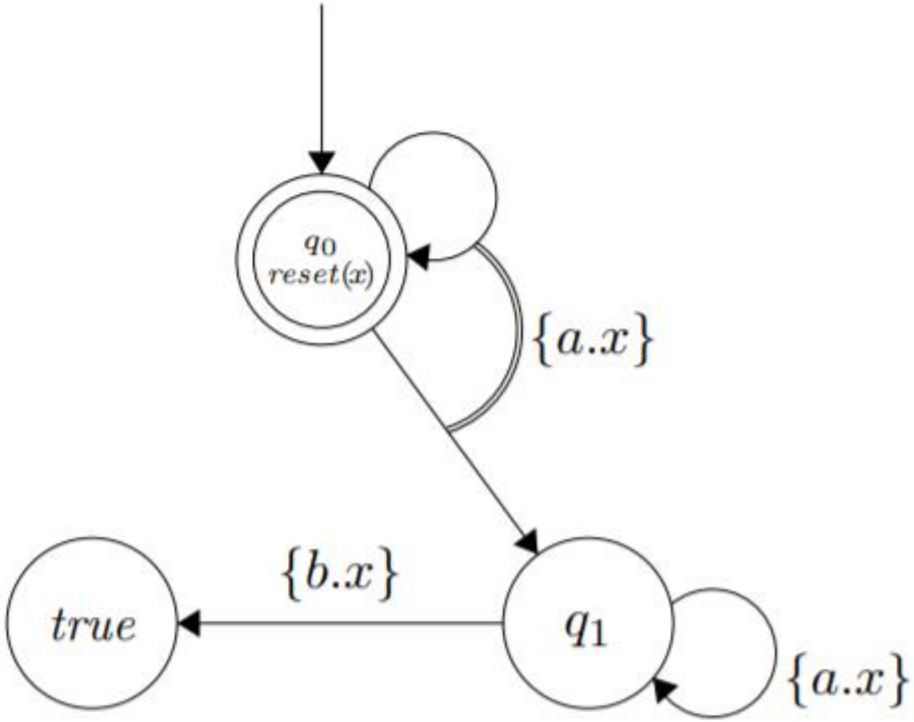
AVBW to NVBW

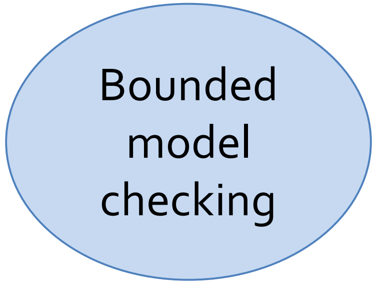
- $G \exists x{:}\, a.x \wedge XX\; b.x$



Incompleteness

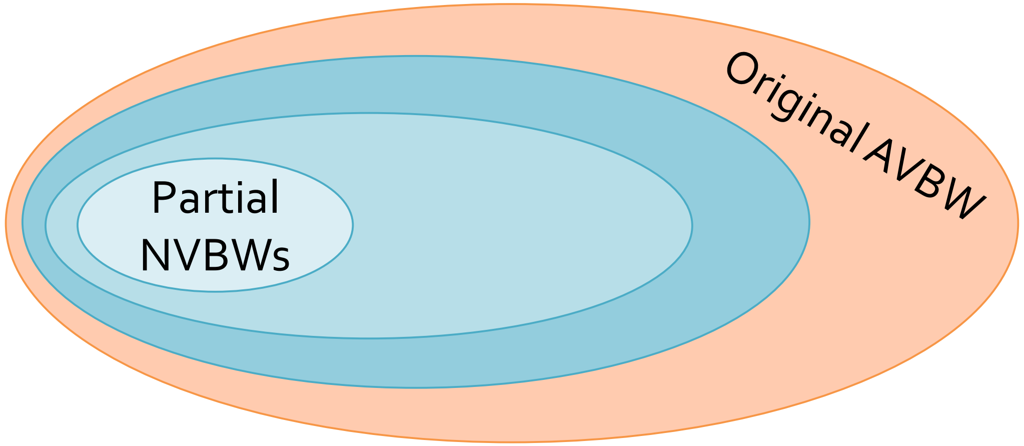
- The empty language
- Our algorithm does not halt





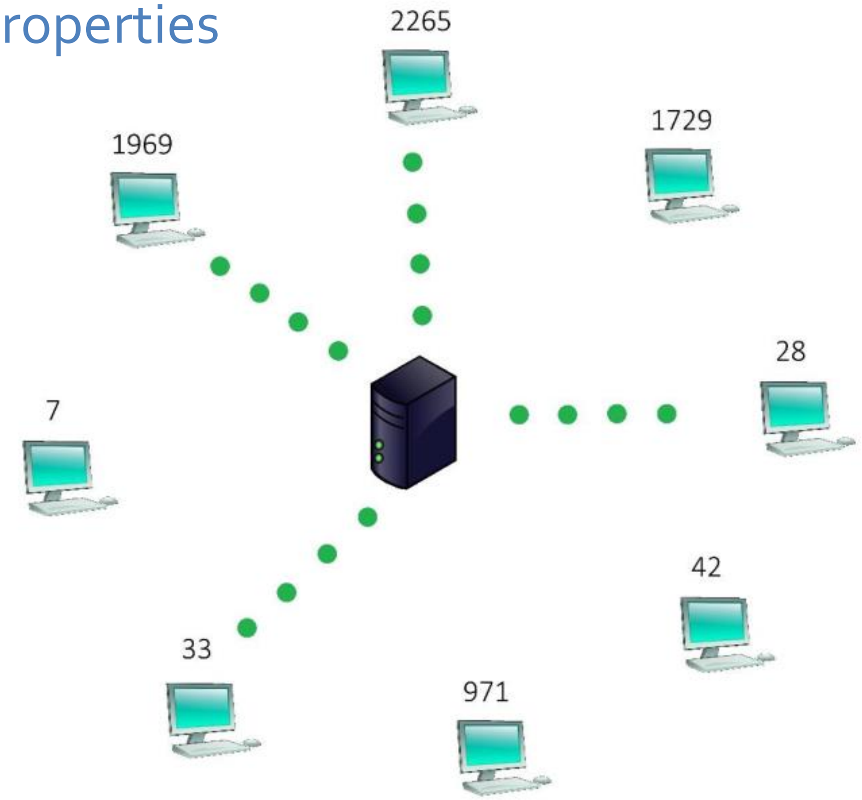
BMC Algorithm

- Based on the translation algorithm
- We are looking for a *witness* to non-emptiness
- Test emptiness with a partial NVBW
- Might find "more interesting" witnesses as the algorithm continues



VLTL Summary

- Using alternating variable automata to model VLTL properties
- Translation algorithm from AVBWs to NVBWs
- Bounded model-checking procedure for $\exists^*$VLTL
- Easy fragments for model-checking

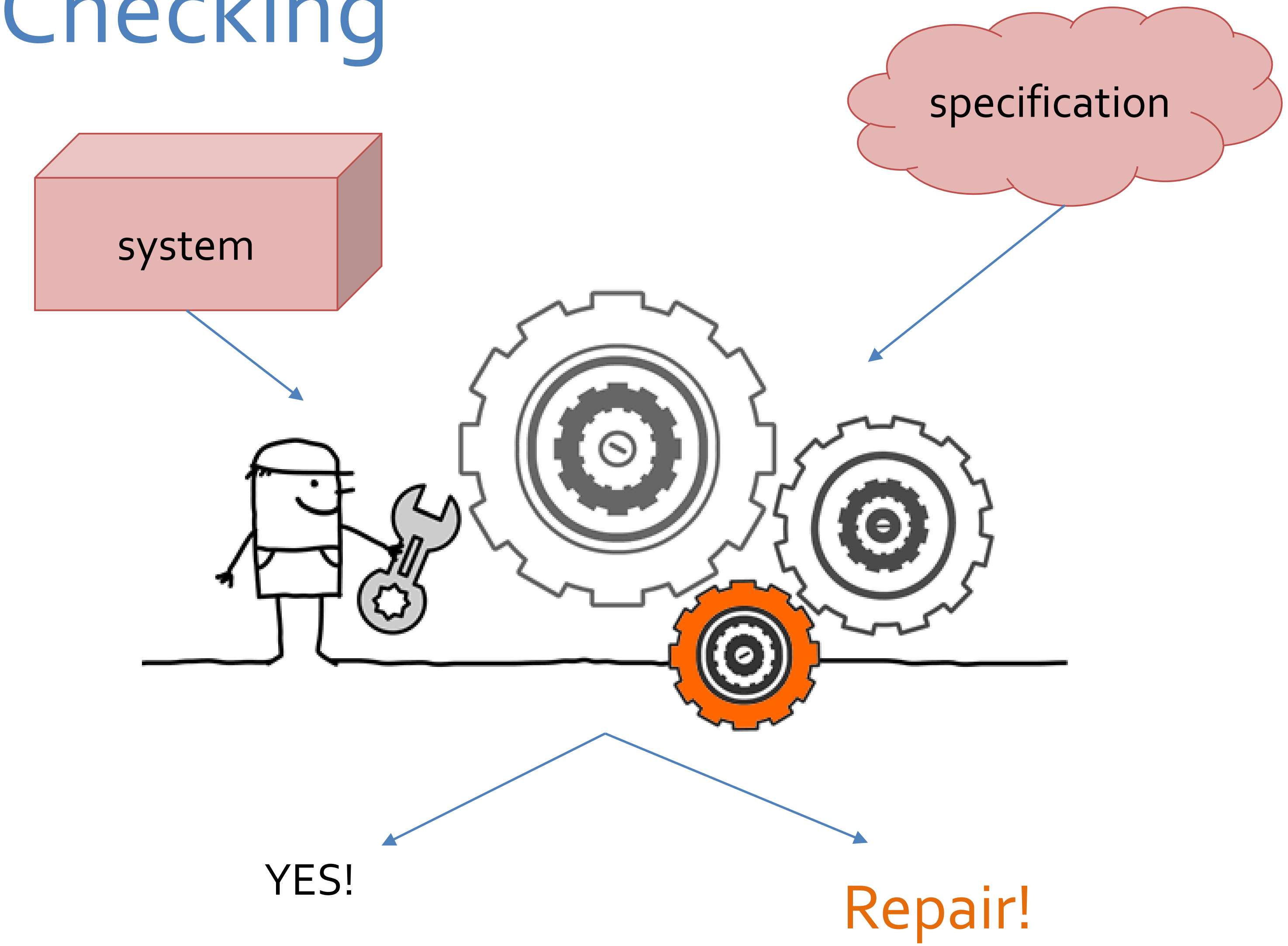


COMPOSITIONAL VERIFICATION AND REPAIR

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

@TACAS 2020

Model Checking



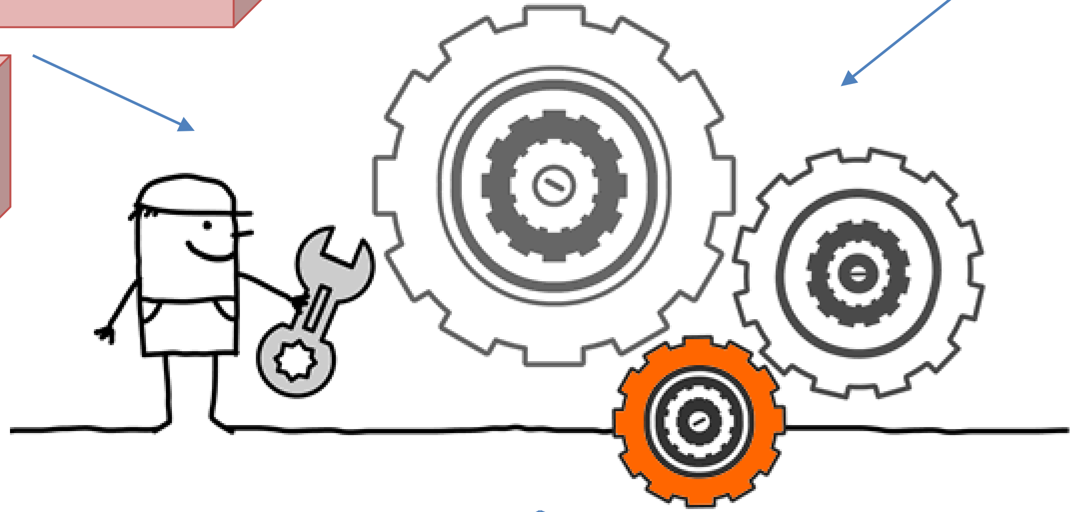
Model Checking

component

component

component

specification



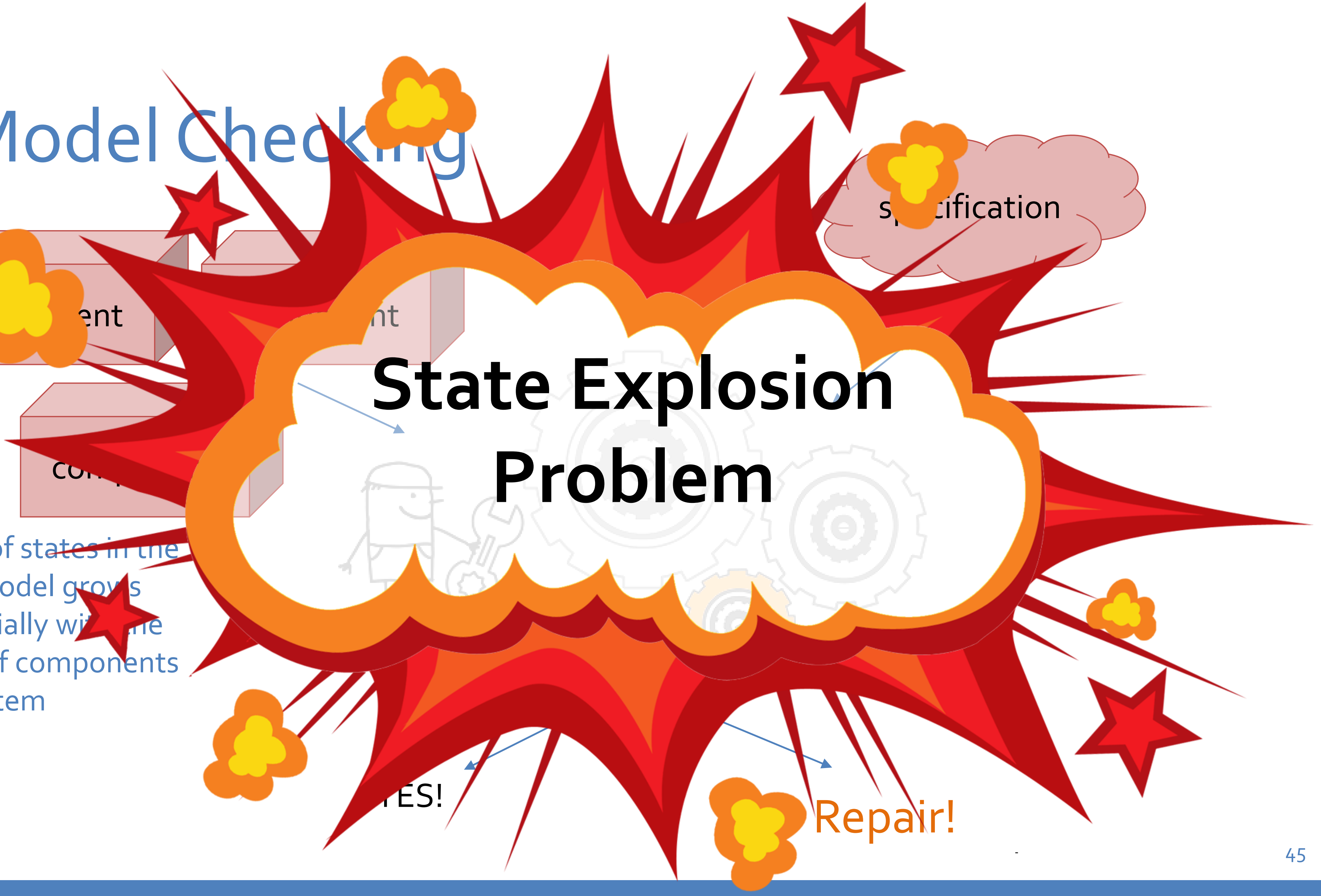
Number of states in the system model grows exponentially with the number of components in the system

YES!

Repair!

Model Checking

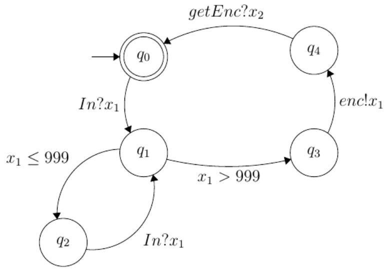
Number of states in the system model grows exponentially with the number of components in the system



COMPOSITIONAL VERIFICATION AND REPAIR OF C-LIKE PROGRAMS

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

Setting – Communicating Systems



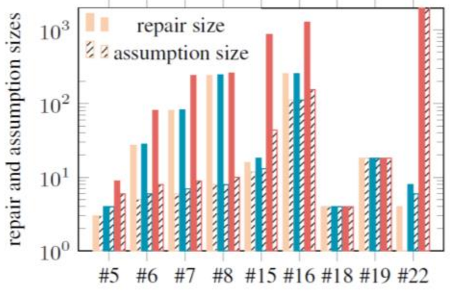
Assume-Guarantee (AG)

$M_1 || A \vDash P$
$M_2 \vDash A$

$M_1 || M_2 \vDash P$

AG rule & Automata Learning

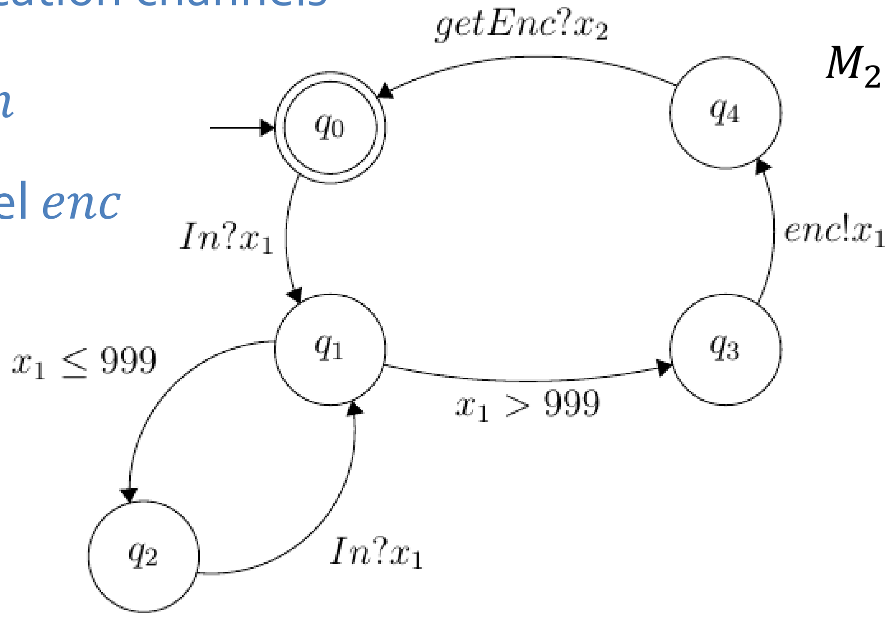
Repair & Results



Communicating Systems

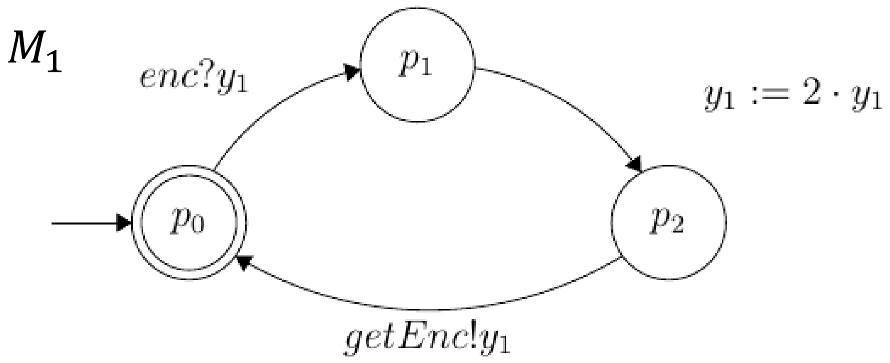
- C-like programs
- Each component is described as a control-flow graph (automaton)
  - Alphabet: program statements & communication channels
- $In?\,x_1$ – reads a value to x_1 through channel In
- $enc!\,x_1$ – sends the value of x_1 through channel enc

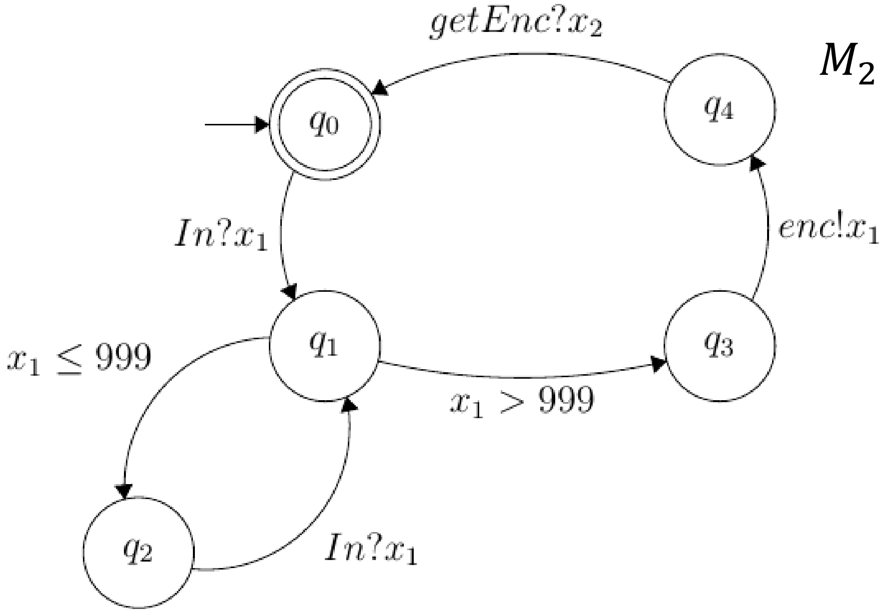
```
while (true)
    pass = readInput;
    while (pass ≤ 999)
        pass = readInput;
    pass2 = encrypt(pass);
```



Example

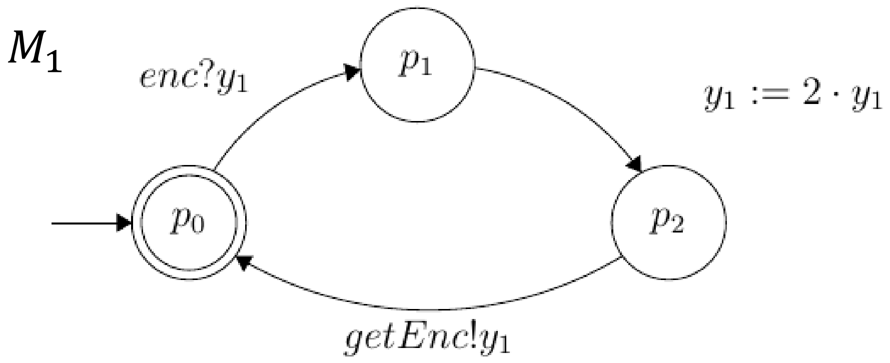
Synchronization using read-write channels, Interleaving on all other alphabet

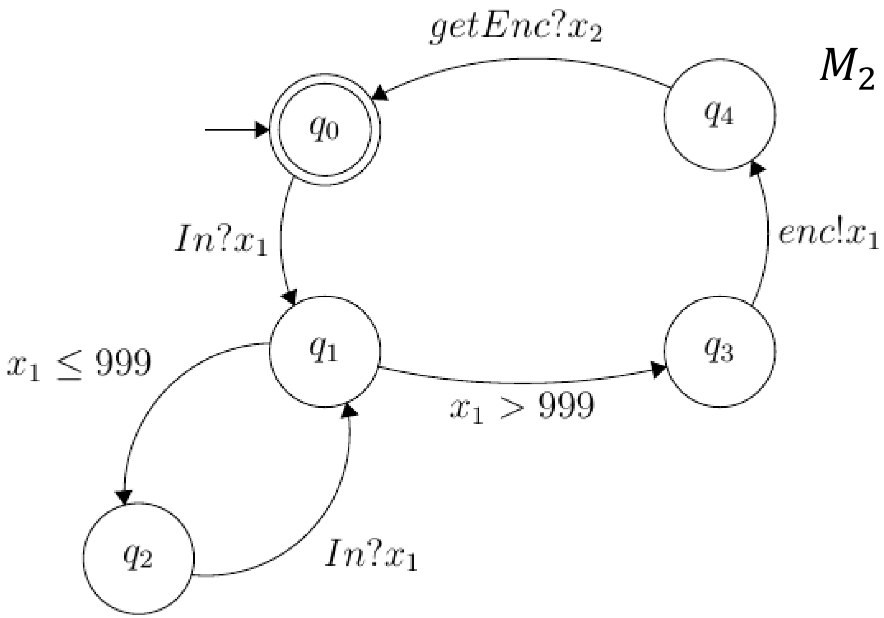




Example

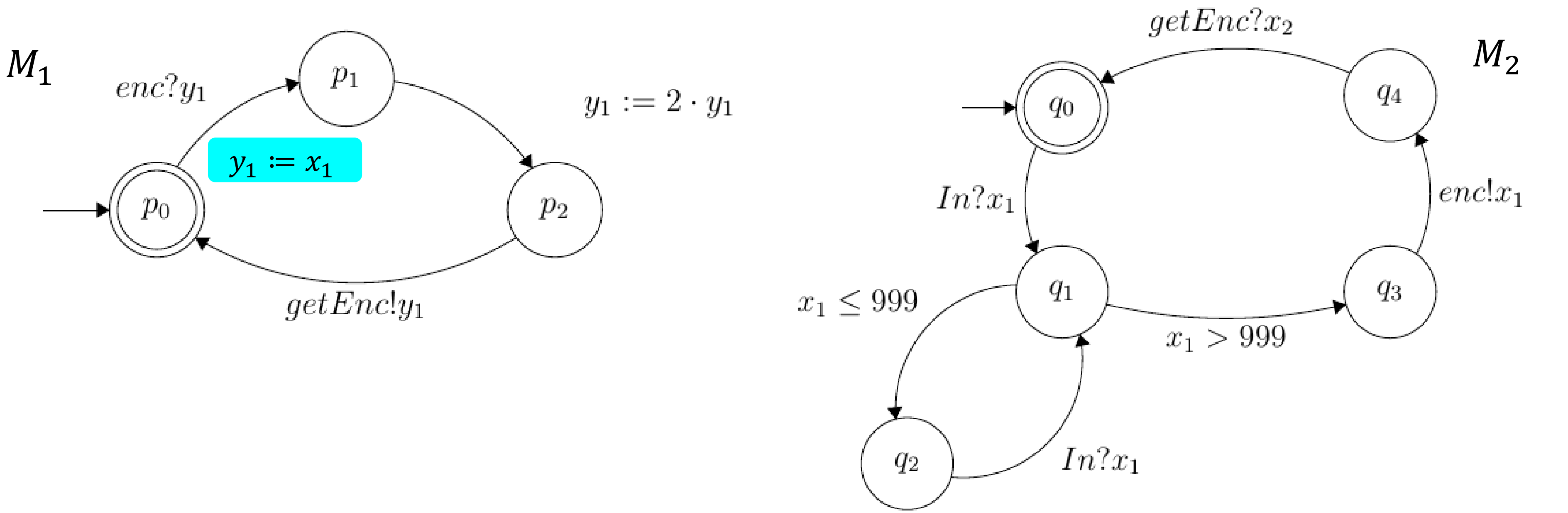
Synchronization using read-write channels, Interleaving on all other alphabet





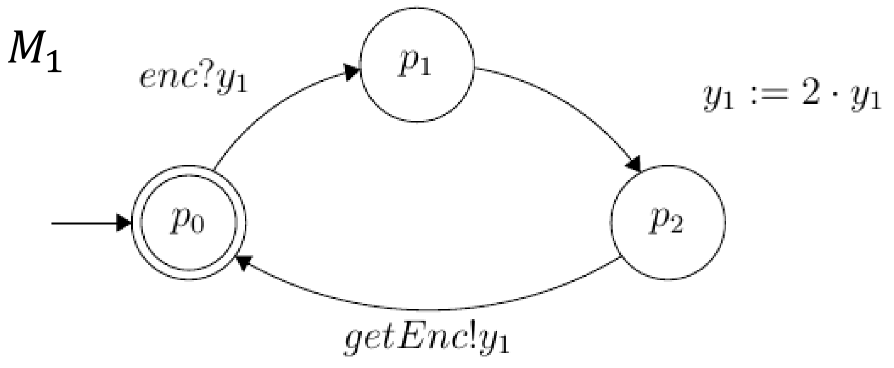
Example

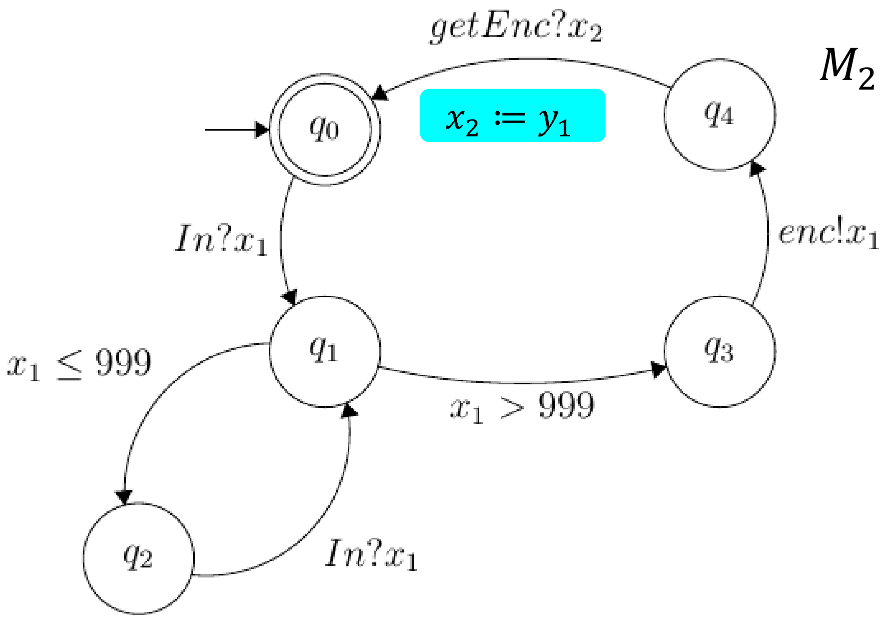
Synchronization using read-write channels, Interleaving on all other alphabet



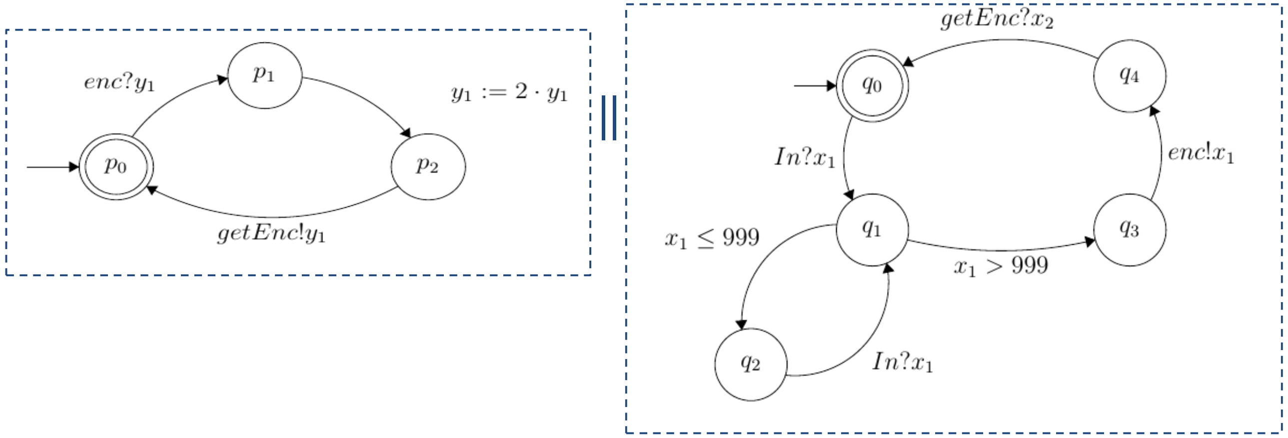
Example

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

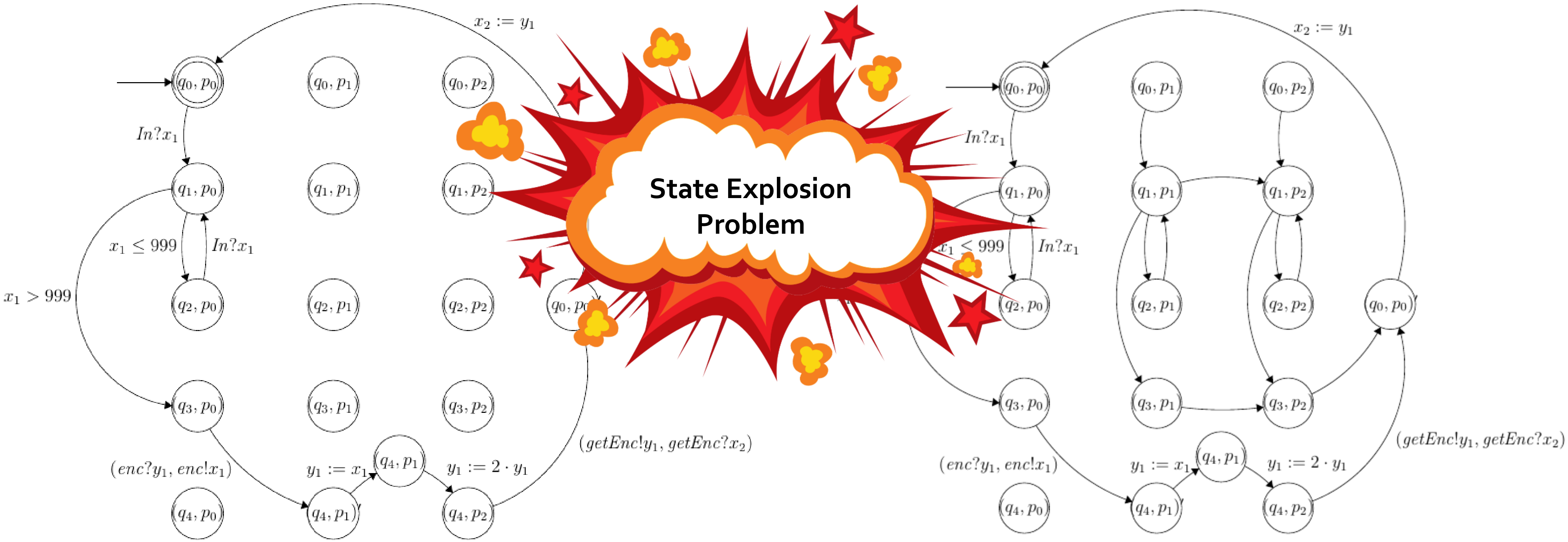




Example



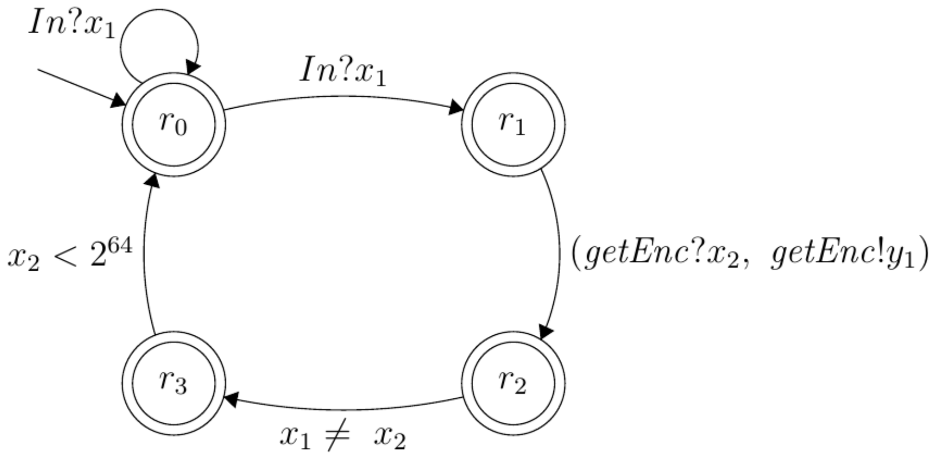
enc?y1
p1
y1 := 2 · y1
p0
p2
getEnc!y1
||
getEnc?x2
q0
q4
In?x1
enc!x1
x1 ≤ 999
q1
q3
x1 > 999
q2
In?x1



x2 := y1
In?x1
x1 ≤ 999
In?x1
x1 > 999
(enc?y1, enc!x1)
y1 := x1
y1 := 2 · y1
(getEnc!y1, getEnc?x2)
State Explosion Problem

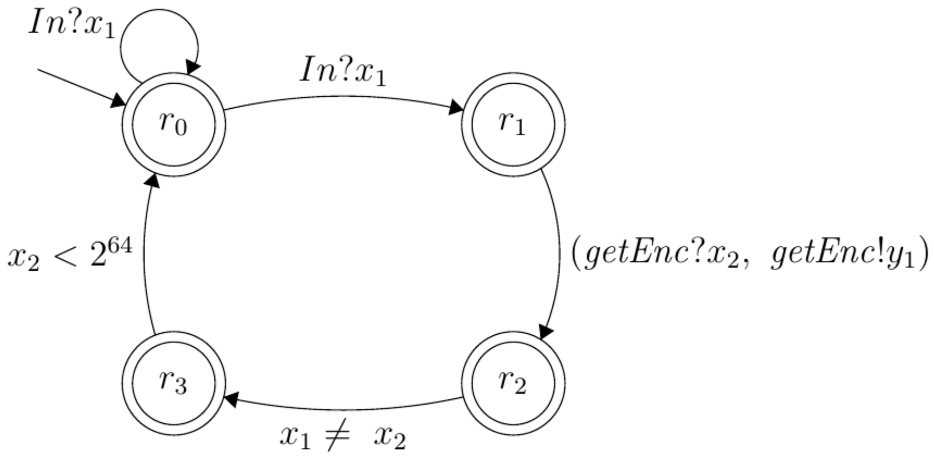
Specifications

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



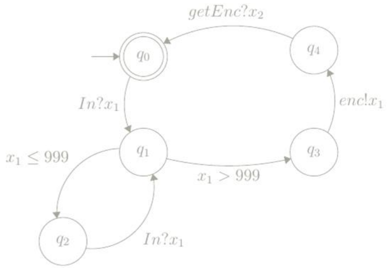
Specifications

- 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”



Reasoning About the Smaller Components

Setting – Communicating Systems



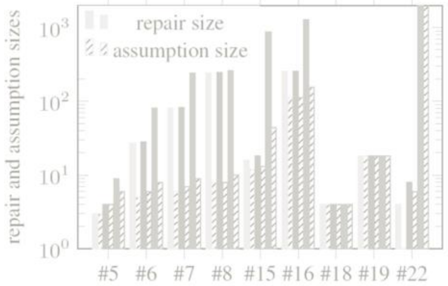
Assume-Guarantee (AG)

$M_1 || A \vDash P$
$M_2 \vDash A$

$M_1 || M_2 \vDash P$

AG rule & Automata Learning

Repair & Results



Compositional Verification

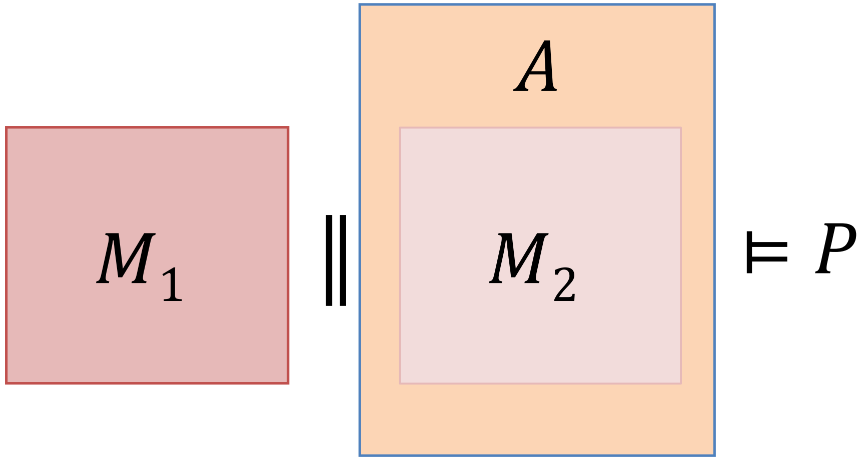
- Inputs:
  - composite system $M_1 \| M_2$
  - property P
- Goal: check if $M_1 \| M_2 \models P$
- First attempt: "divide and conquer"
  - Problem: usually impossible to verify each component separately
  - Components are designed to satisfy requirements in specific contexts

$$\boxed{M_1} \models P_1$$

$$\boxed{M_2} \models P_2$$

Compositional Verification

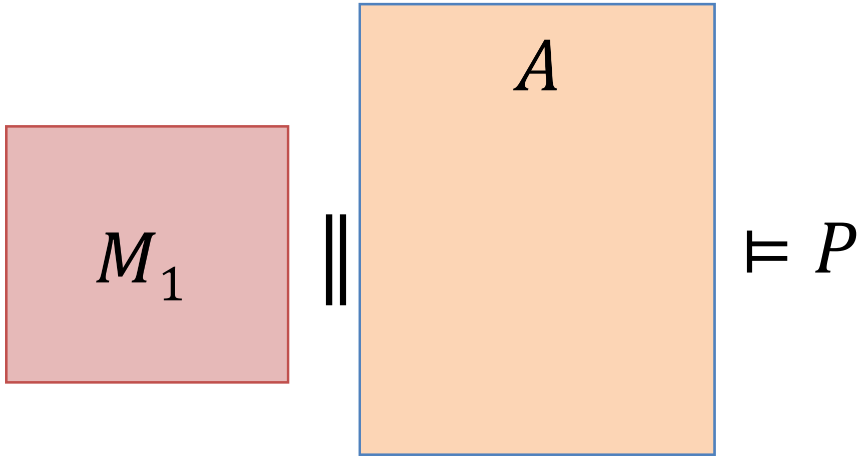
- **Assume-Guarantee** (AG) paradigm [Pnueli, 1985]:
  - assumptions represent component's environment
- Under assumption A on its environment, does the component guarantee the property?



AG Rule for Safety Properties

1. check if a component M_1 **guarantees** P when it is a part of a system satisfying **assumption** A

$$M_1 || A \models P$$



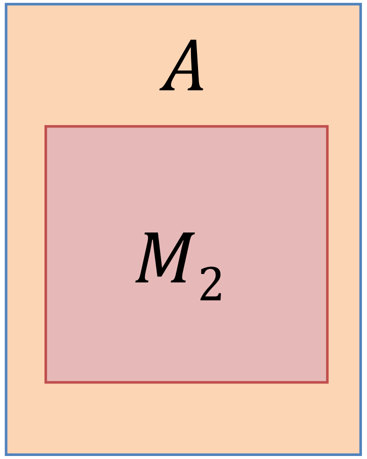
AG Rule for Safety Properties

1. check if a component M_1 **guarantees** P when it is a part of a system satisfying **assumption** A

   $M_1 || A \models P$

2. **discharge** assumption: show that the remaining component M_2 satisfies A

   $M_2 \models A$



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

   $\boxed{M_1 \| A \models P}$

2. **discharge** assumption: show that the remaining component M_2 satisfies A

   $\boxed{M_2 \models A}$

3. Conclude that $M_1 \| M_2 \models P$

$$\boxed{M_1} \;\|\; \boxed{M_2} \models P$$

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

$M_1 || A \models P$

2. **discharge** assumption: show that the remaining component M_2 satisfies A

$M_2 \models A$

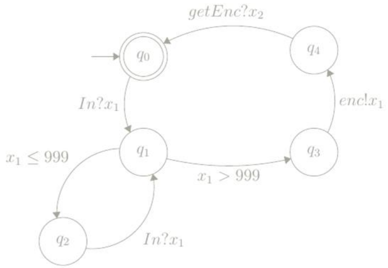
3. Conclude that $M_1 || M_2 \models P$

$M_1 \; || \; M_2 \; \models P$

Can we automatically construct A?

Automatic Assumption Generation

Setting – Communicating Systems

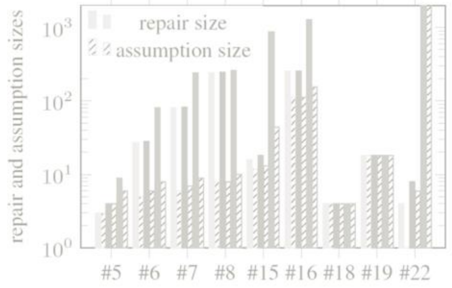


Assume-Guarantee (AG)

$$M_1 || A \vDash P$$
$$M_2 \vDash A$$
$$\overline{M_1 || M_2 \vDash P}$$

AG rule & Automata Learning

Repair & Results



L* Algorithm for Learning Regular Languages [Angluin87]

- Learning assumptions for compositional verification [CGP03]
- Given a regular language L, we learn a DFA A such that $\mathcal{L}(A) = L$

Teacher

Learner

L* Algorithm for Learning Regular Languages [Angluin87]

- Learning assumptions for compositional verification [CGP03]
- Given a regular language L, we learn a DFA A such that $\mathcal{L}(A) = L$
- Membership queries

Teacher

Yes / No

Is $w \in L$?

Learner

L* Algorithm for Learning Regular Languages [Angluin87]

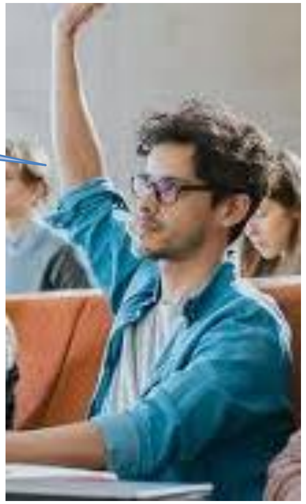
- Learning assumptions for compositional verification [CGP03]
- Given a regular language L, we learn a DFA A such that $\mathcal{L}(A) = L$
- Equivalence queries, for a candidate A_i

Teacher

Yes – Done!

No +
cex $\in \mathcal{L}(A_i)\Delta L$

Is $\mathcal{L}(A_i) = L$?



Learner

L^* Algorithm for Learning Regular Languages [Angluin87]

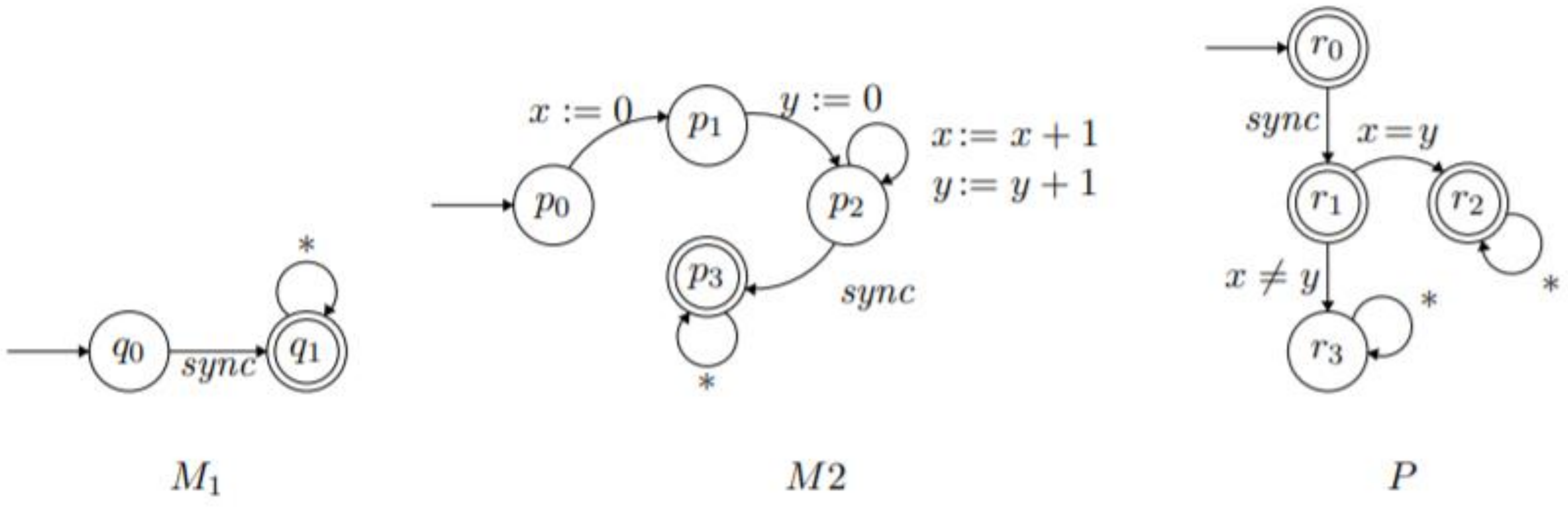
- Learning assumptions for compositional verification [CGP03]
- Given a regular language L, we learn a DFA A such that $\mathcal{L}(A) = L$
- Equivalence queries, for a candidate A_i
- Try to use intermediate candidates A_i as assumptions for AG rule
- But, the weakest assumption is not regular in our case

Weakest assumption

$$\frac{M_1 || A_i \vDash P \qquad M_2 \vDash A_i}{M_1 || M_2 \vDash P}$$

Weakest Assumption is not always regular

- By a way of contradiction
- A_w is over $\alpha M_2 = \{x \coloneqq 0,\ y \coloneqq 0,\ x \coloneqq x+1,\ y \coloneqq y+1,\ sync\}$
- Consider $L = \{x \coloneqq 0\} \cdot \{y \coloneqq 0\} \cdot \{x \coloneqq x+1, y \coloneqq y+1\}^* \cdot \{sync\}$



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

$M_1 || M_2 \vDash P$

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

A New Goal for Learning

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

$M_1 \| M_2 \vDash P$

- 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

Membership Queries - $T(M_2)$

$M_1 \| A \vDash P$
$M_2 \vDash A$

$M_1 \| M_2 \vDash P$

Is $w \in L$?

$w \notin T(M_2)$
NO!

$w \in T(M_2) \wedge$
$M_1 \| w \vDash P$
YES!

$w \in T(M_2) \wedge$
$M_1 \| w \nvDash P$
w is a real cex!

Teacher

Learner

Equivalence Queries - $T(M_2)$

$M_1 || A \vDash P$
$M_2 \vDash A$

$M_1 || M_2 \vDash P$

$M_2 \subseteq A_i$

$M_1 || A_i \vDash P$

YES!

Is $\mathcal{L}(A_i) = L$?

$M_1 || A_i \nvDash P$
$t \in A_i$: $M_1 || t \nvDash P$

$t \in M_2$
t is a real cex!

$t \notin M_2$
NO!
take t off A_{i+1}

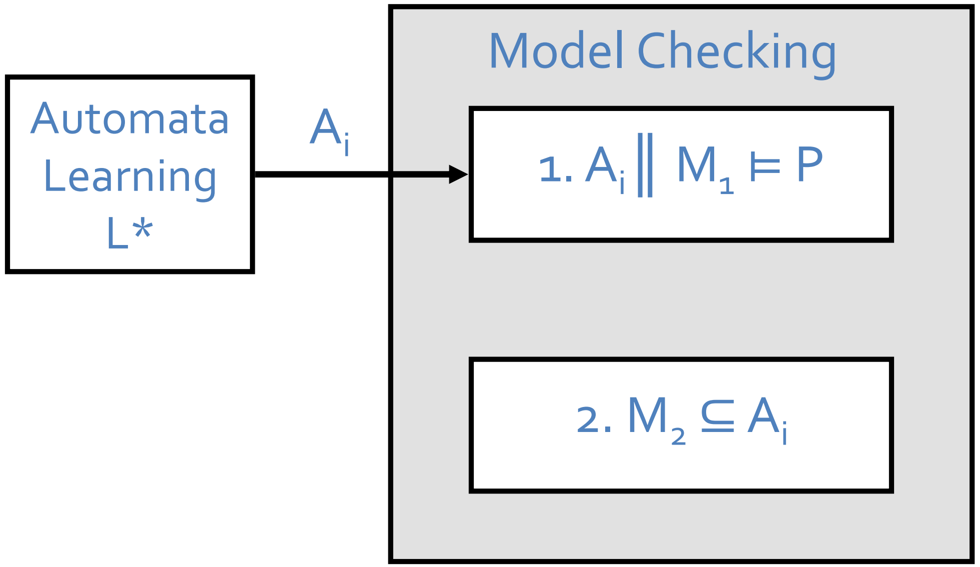
$M_2 \nsubseteq A_i$

$t \in M_2 \setminus A_i$
NO!
add it to A_{i+1}

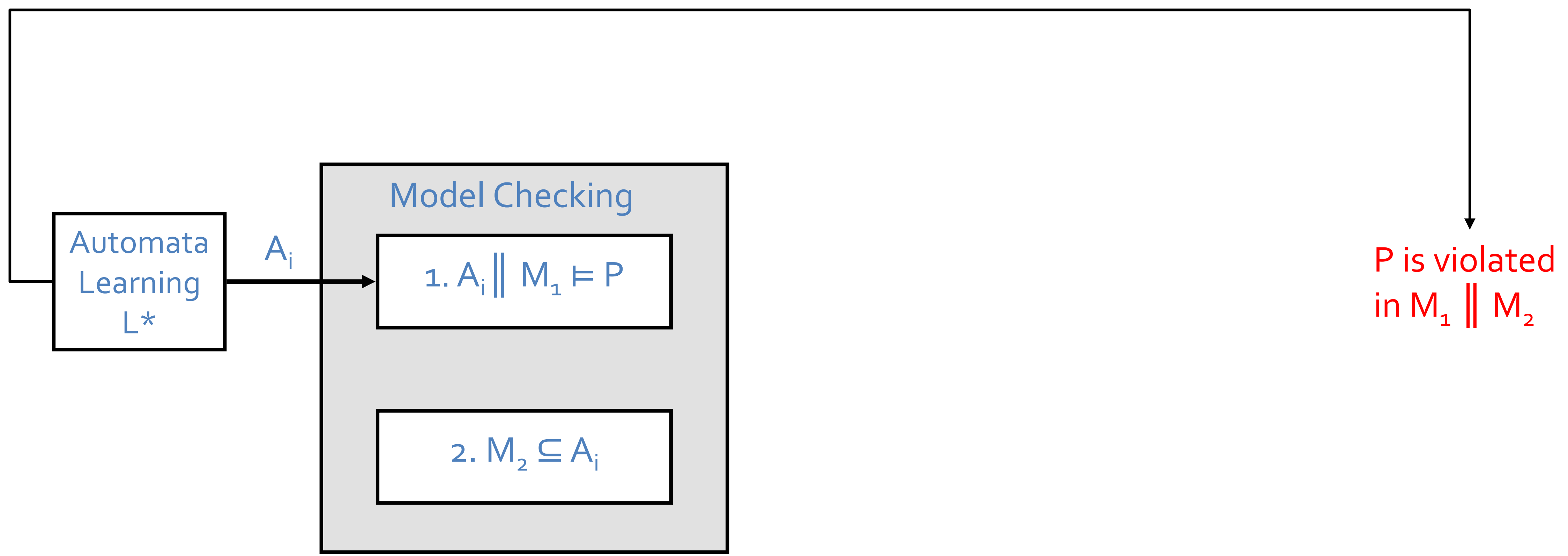
Teacher

Learner

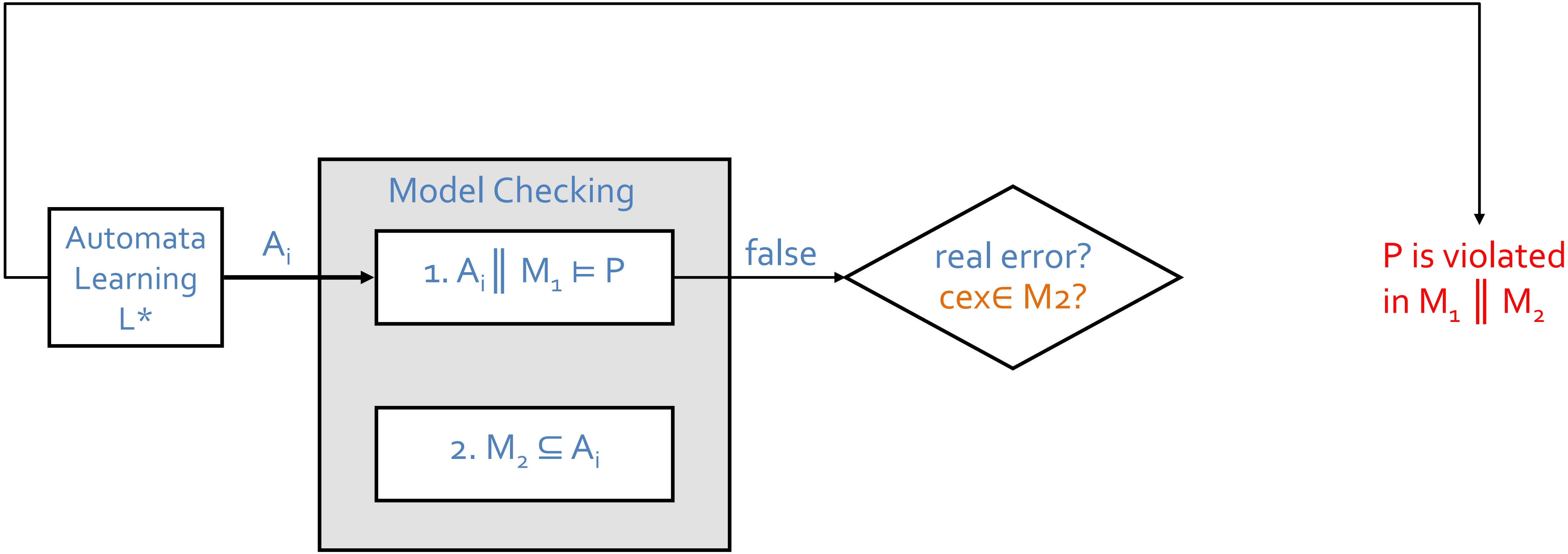
AG rule with learning



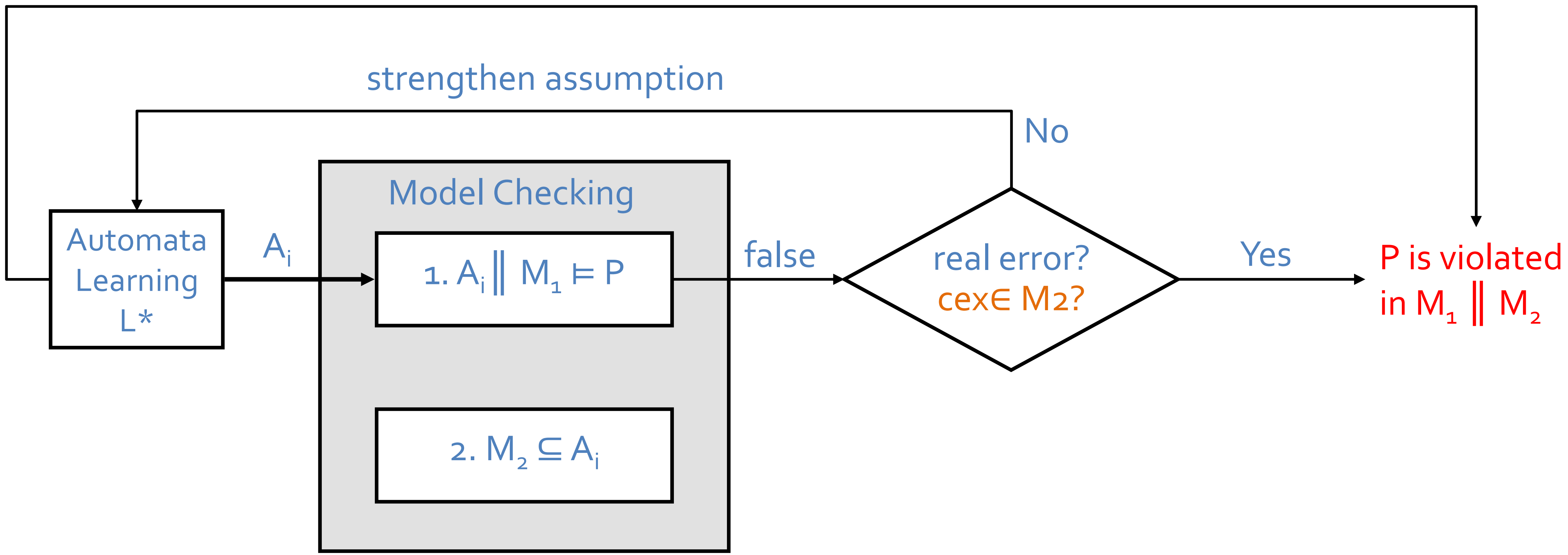
AG rule with learning



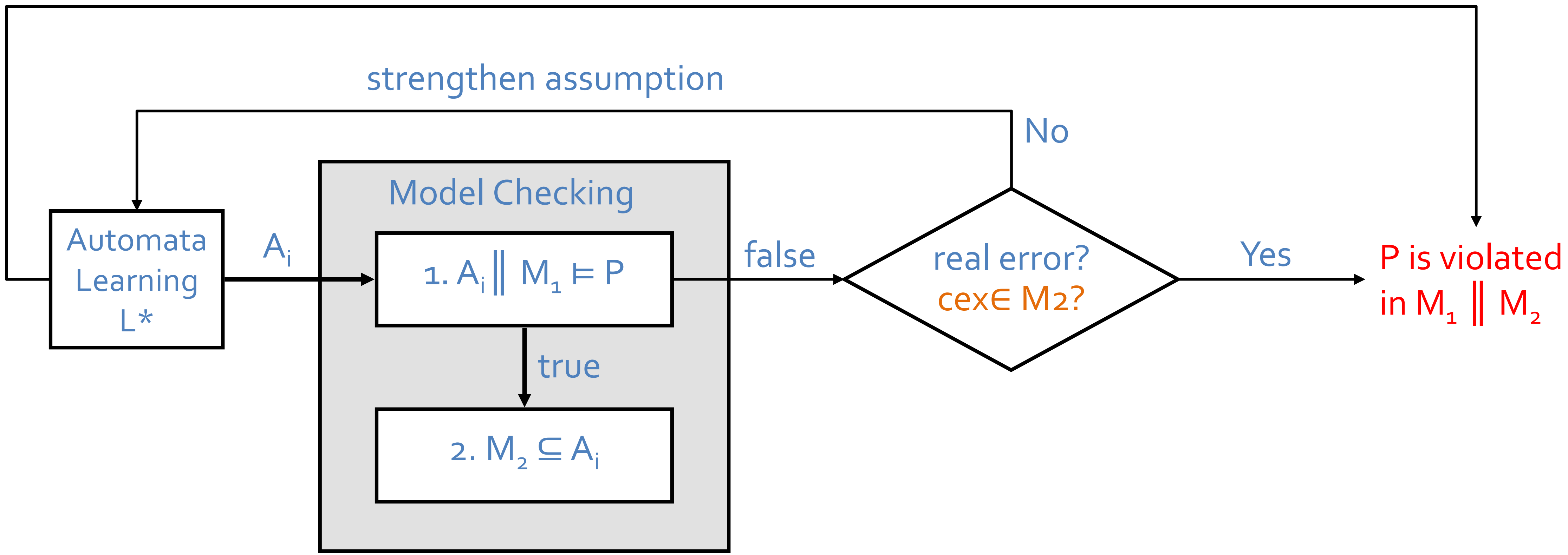
AG rule with learning



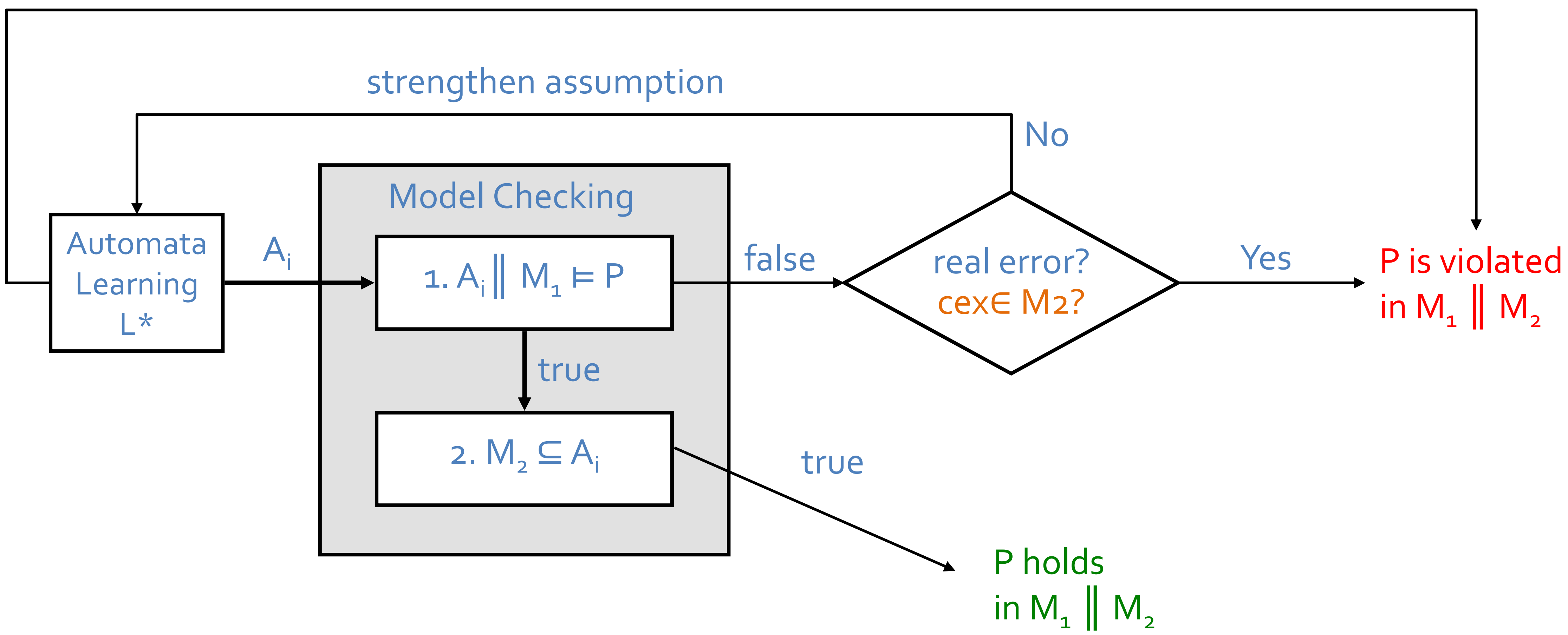
AG rule with learning



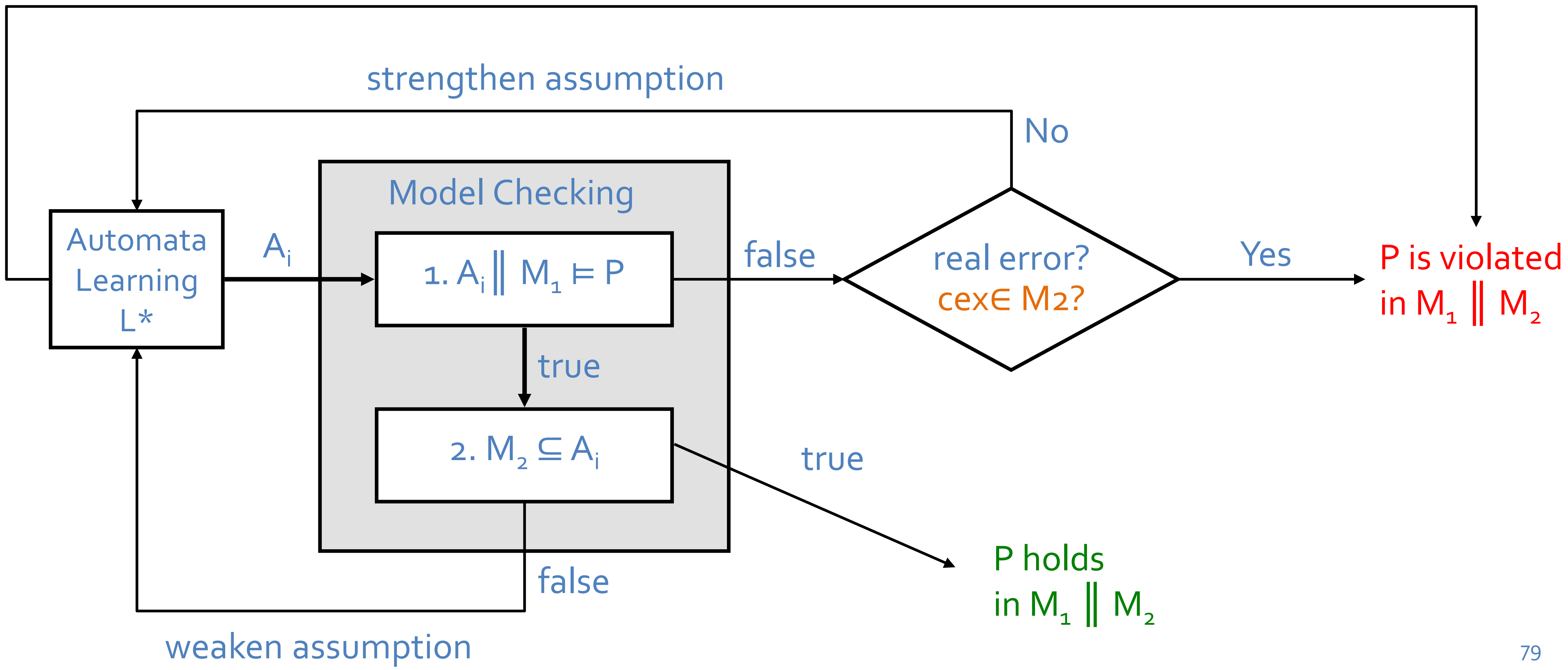
AG rule with learning



AG rule with learning

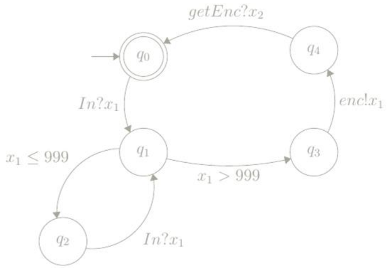


AG rule with learning



Repair

Setting – Communicating Systems

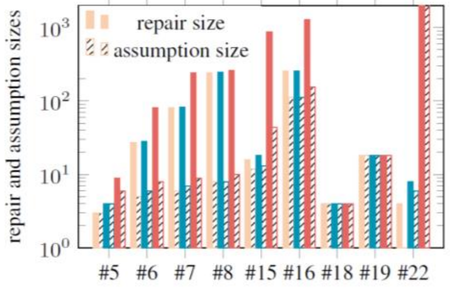


Assume-Guarantee (AG)

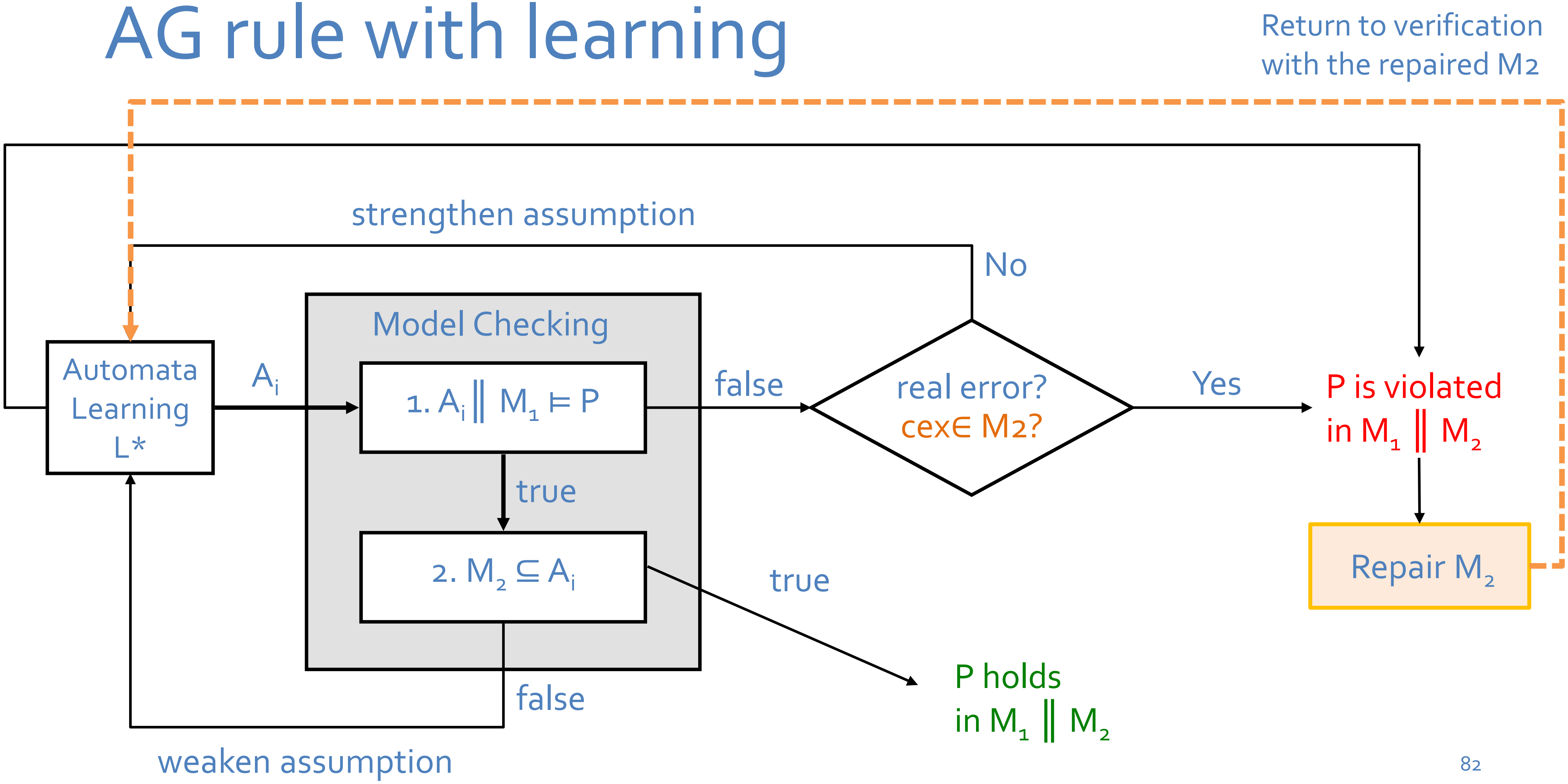
$$M_1 || A \models P$$
$$M_2 \models A$$
$$M_1 || M_2 \models P$$

AG rule & Automata Learning

Repair & Results



AG rule with learning



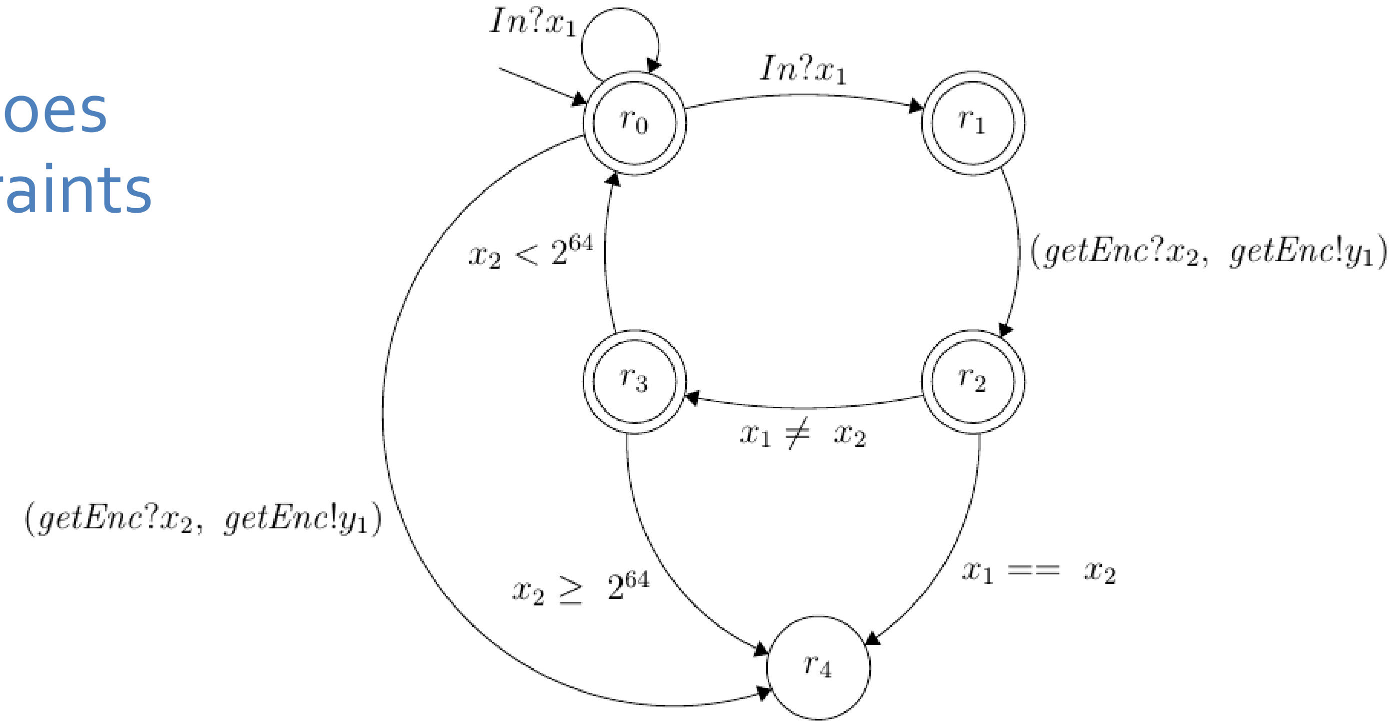
Assume Guarantee or Repair

- 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

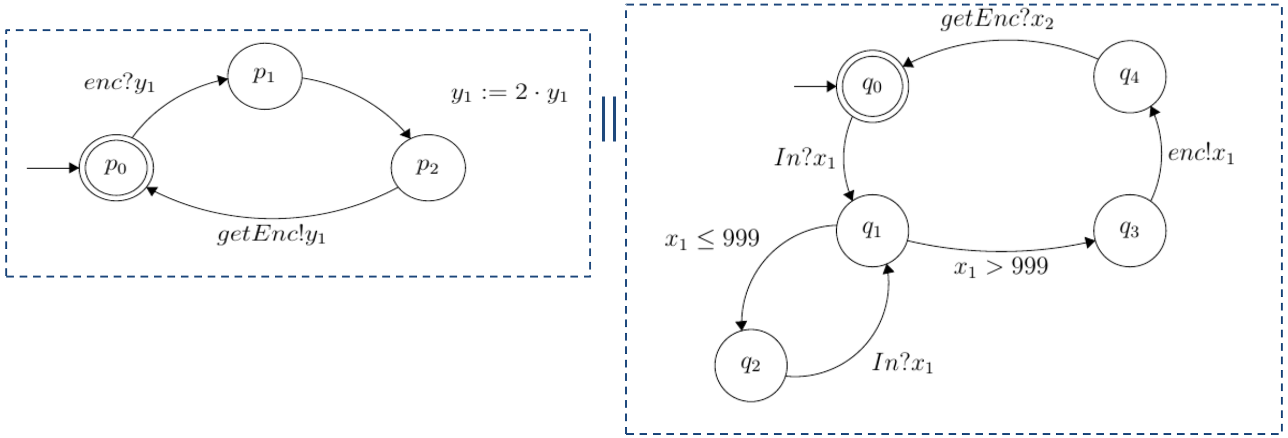


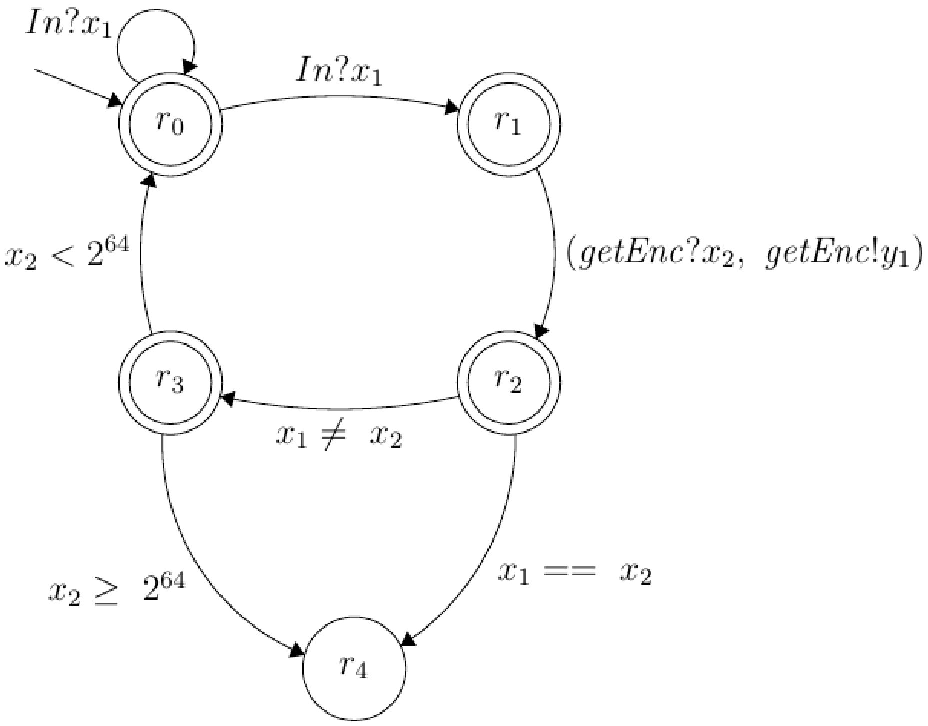
Syntactic Repair

- 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

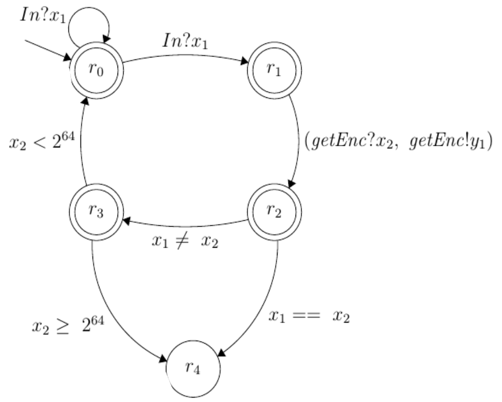
Assume Guarantee or Repair

Semantic repair – counterexample contains violated constraints of the specification





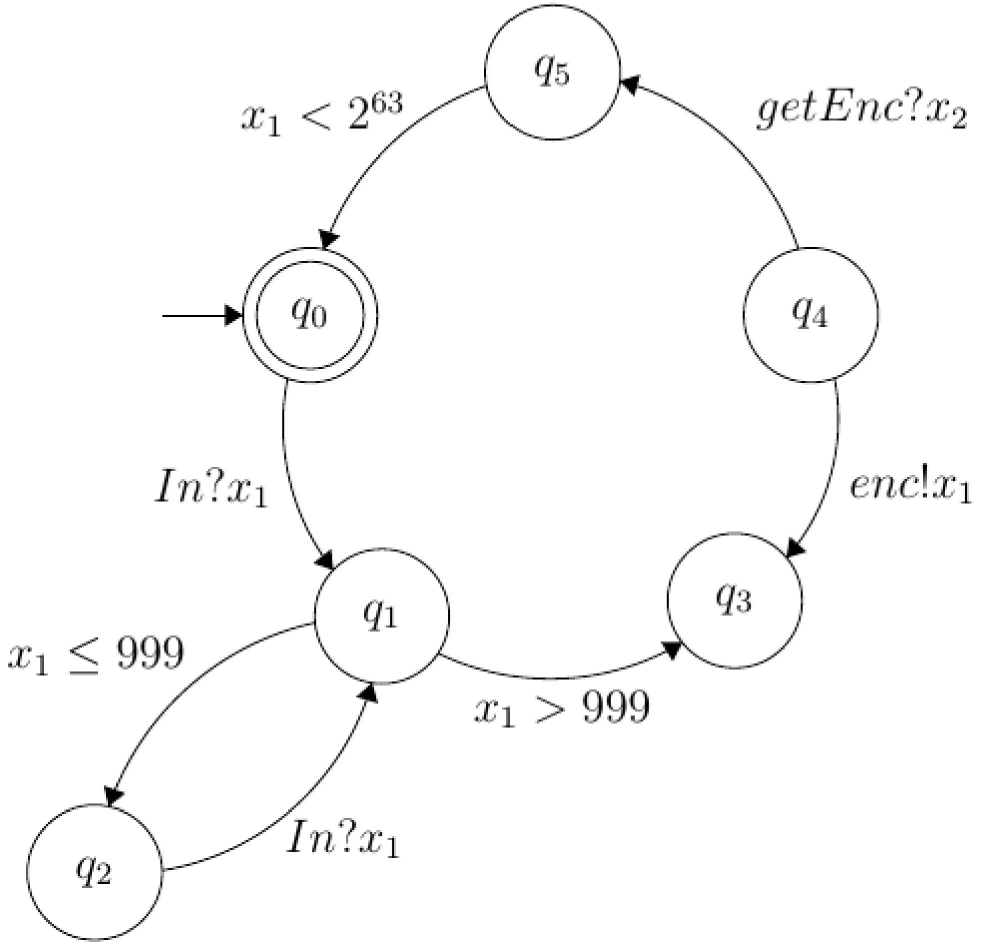
Semantic Repair



- 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 $\boldsymbol{\mathcal{C}} = (x_1 < 2^{63})$

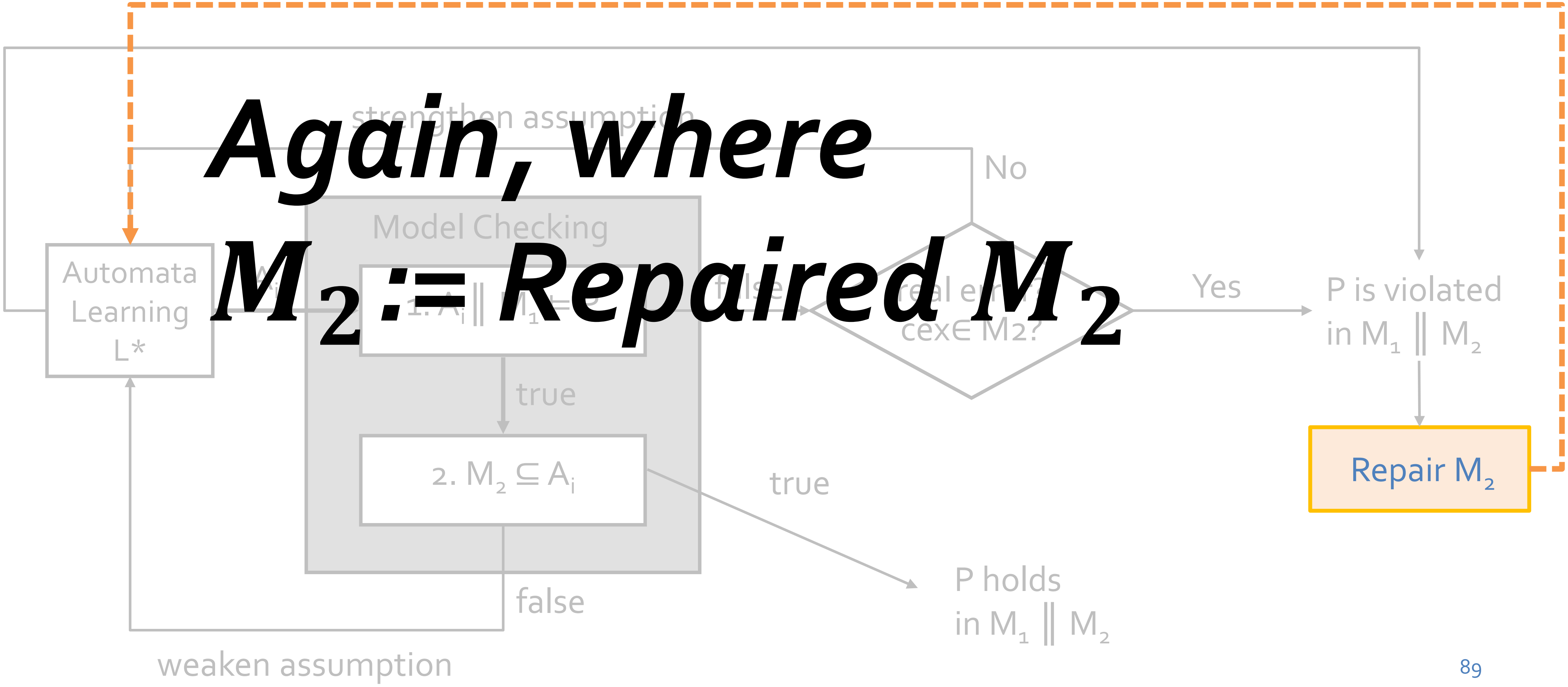
Result

```
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 $M_2 :=$ 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

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

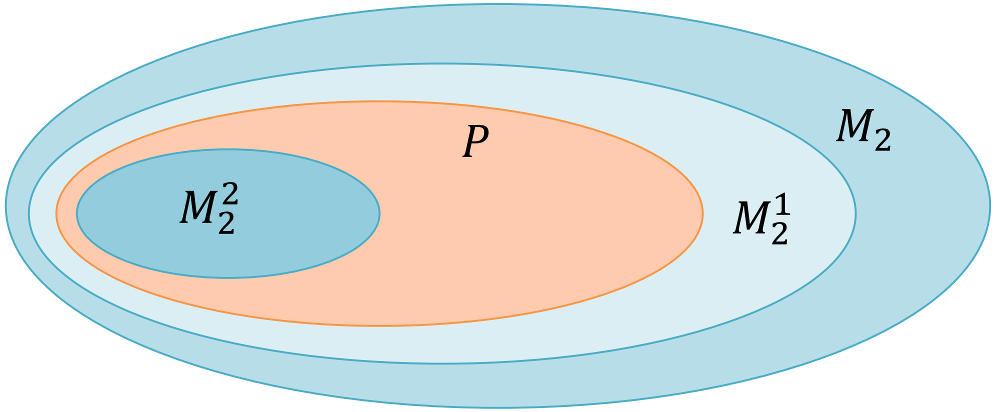
$$M_1||M_2 \vDash P$$

- 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

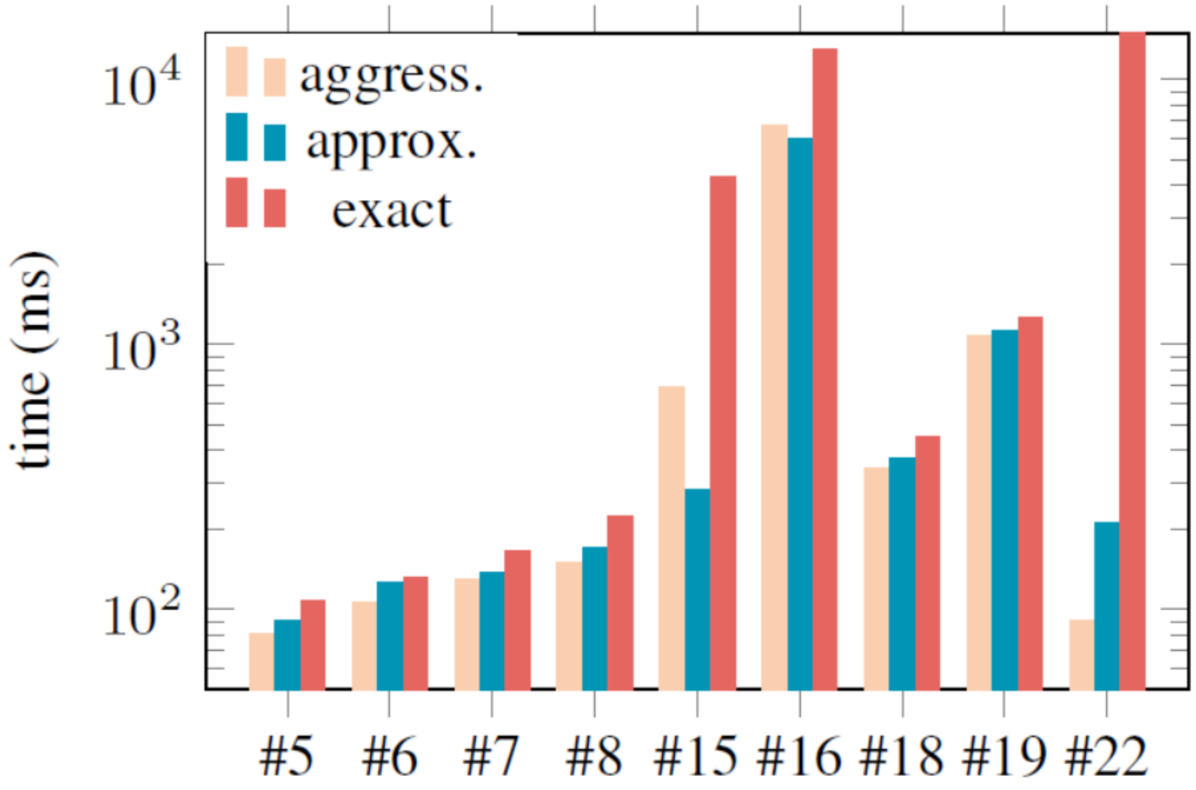
Correctness and Termination

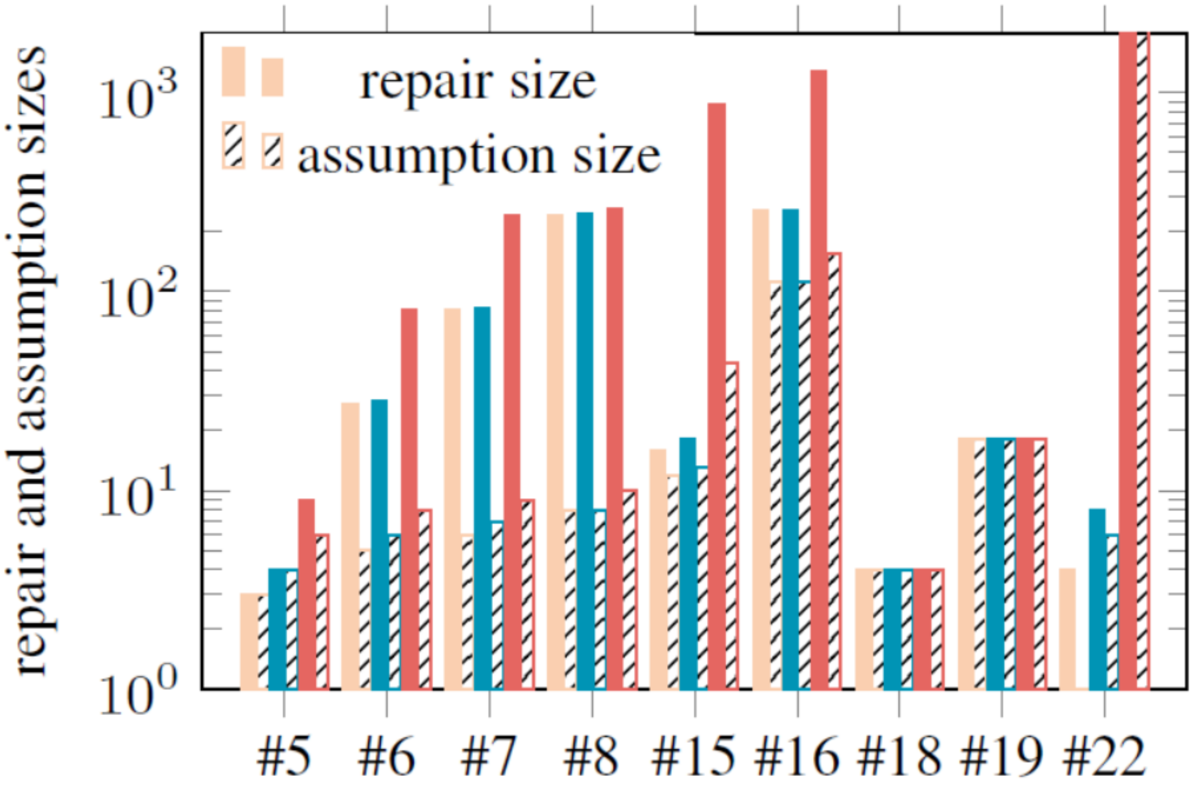
- Correctness of Repair
- All questions relate to language containment
- Repair only eliminates traces



- Incremental
- Previous answers to the learner’s questions are still correct
- Can use the same table for L^*

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

LEARNING SYMBOLIC AUTOMATA

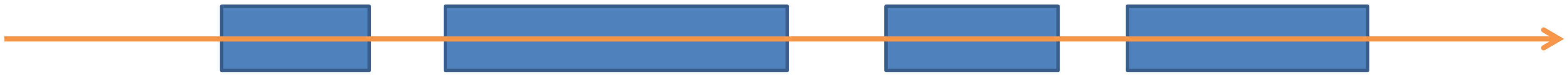
Joint work with Dana Fisman and Sandra Zilles

Symbolic Finite-State Automata (SFAs)

- Finite state automata
- Defined with respect to a Boolean algebra
- The transition relation is over predicates from the Boolean algebra

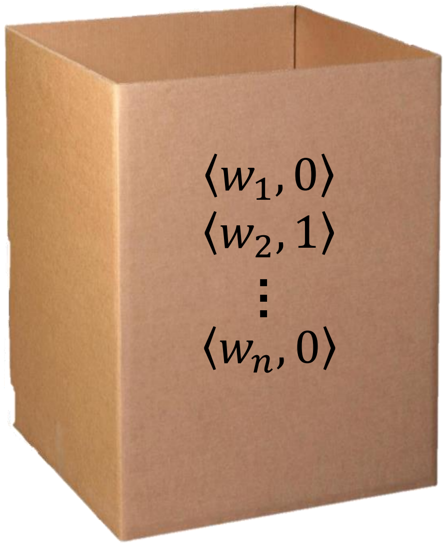
Monotonic Algebras

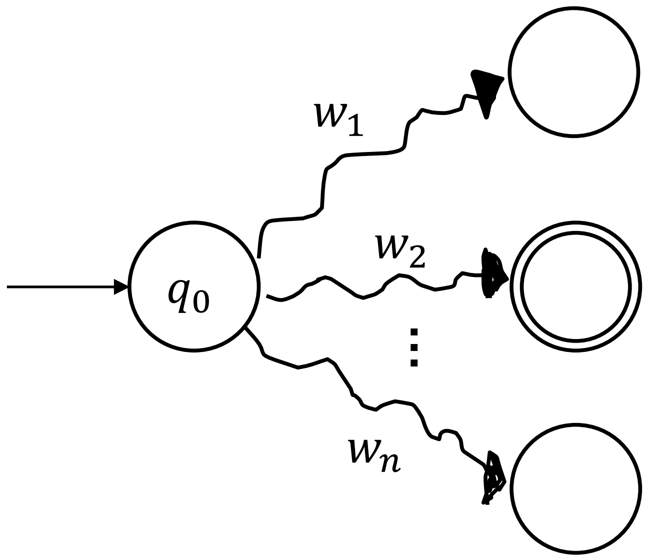
- Predicates correspond to a total order over the domain elements
- $[\![\psi]\!] = \{\, d \mid a \leq d \leq b \,\}$
- Interval algebra over $\mathbb{N}, \mathbb{Z}, \mathbb{R}$



Identification in the Limit using polynomial time and data [G78, dlH97]

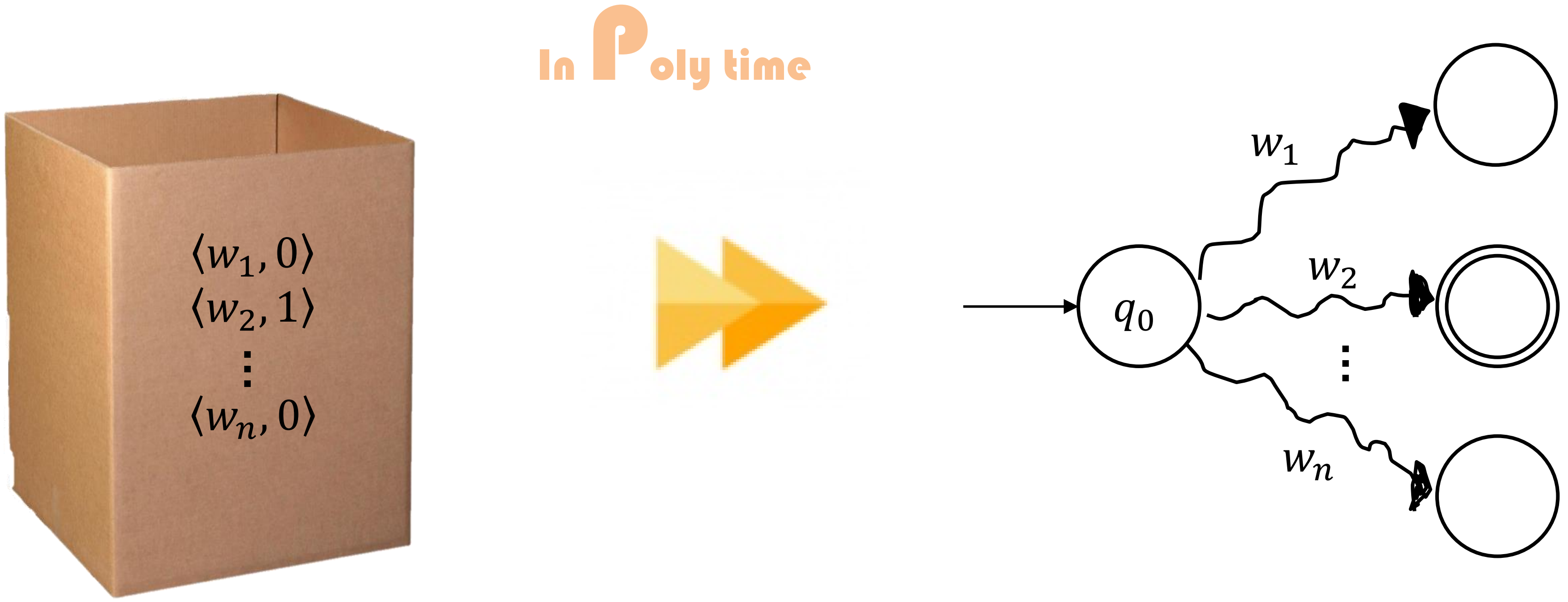
- Passive learning (vs. active learning in L*)





Identification in the Limit using polynomial time and data [G78, dlH97]

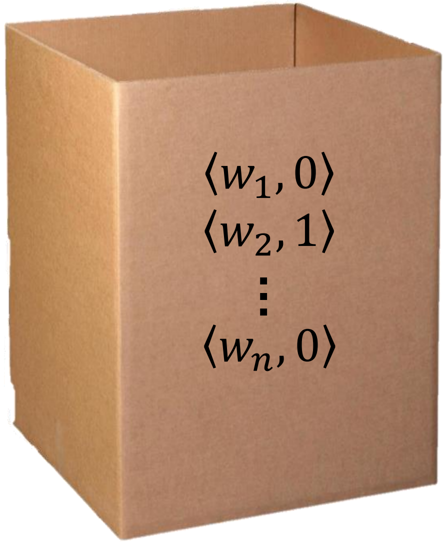
- Passive learning (vs. active learning in L^*)
- Given a set S of labeled words, build an automaton that agrees with S

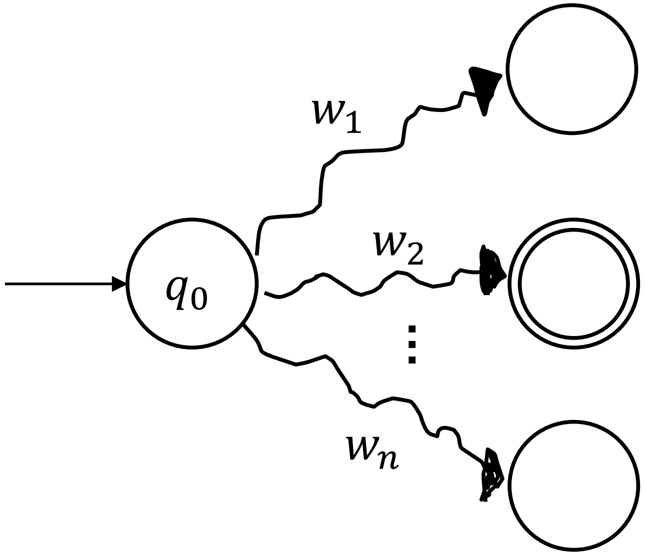


Identification in the Limit using polynomial time and data [G78, dlH97]

- Given an automaton A, build a characteristic sample S

In Poly data

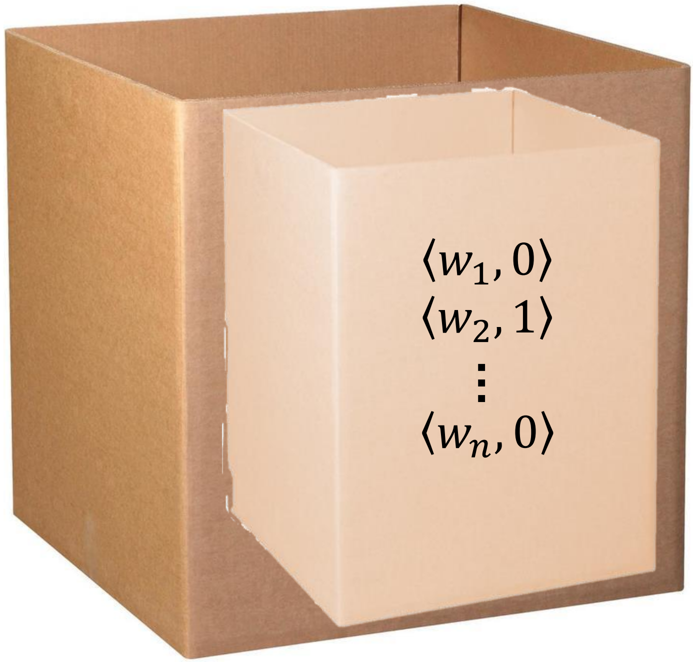


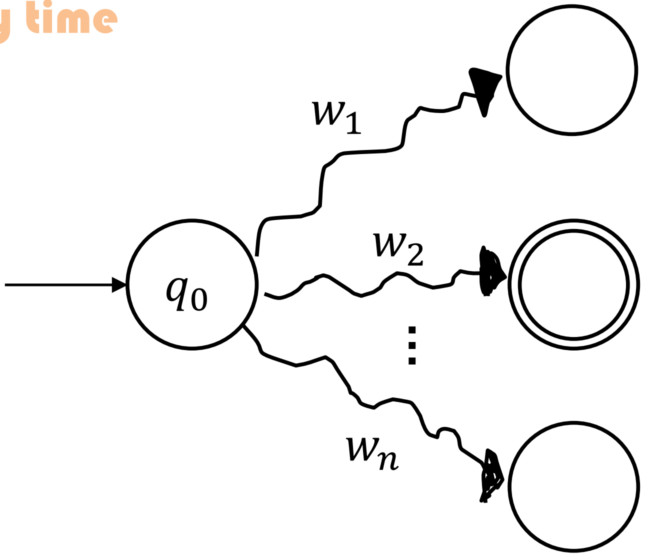


Identification in the Limit using polynomial time and data [G78, dlH97]

- Given an automaton A, build a characteristic sample S
- For every sample $S' \supseteq S$ that agrees with A, infer an equivalent automaton to A

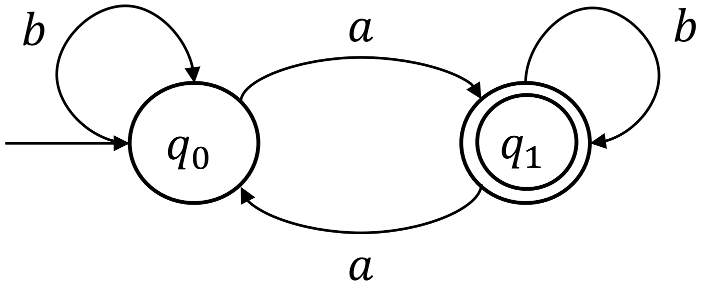
In Poly time





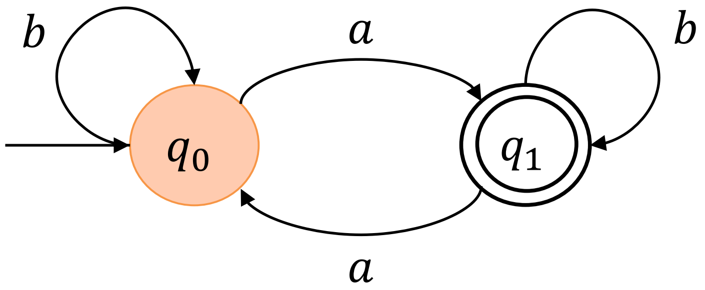
Identification in the Limit for DFAs [OG92]

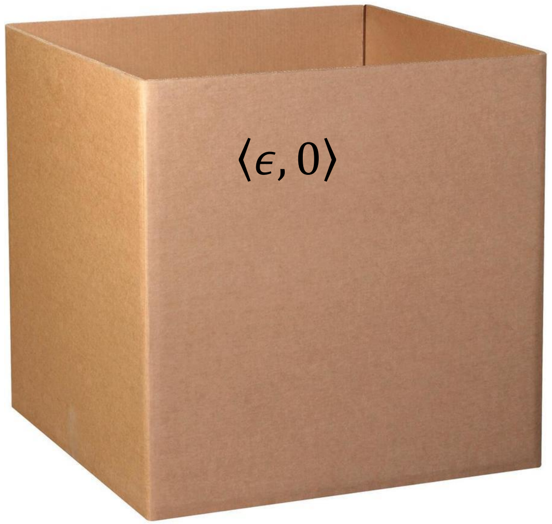
- Constructing a characteristic sample
- Every state is represented by an access word



Identification in the Limit for DFAs [OG92]

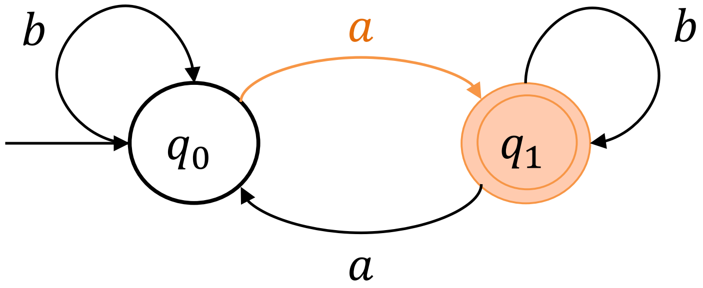
- Constructing a characteristic sample
- Every state is represented by an access word

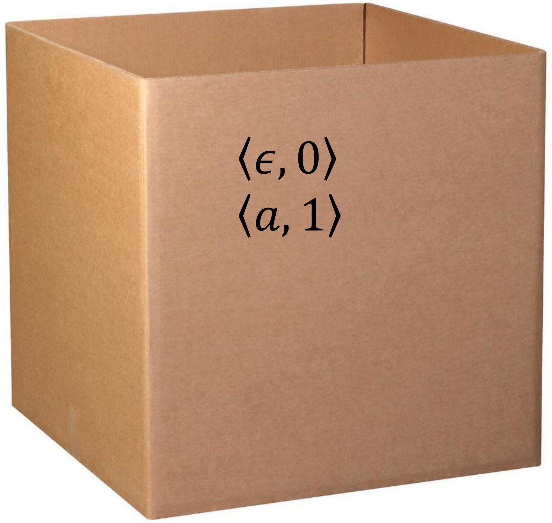




Identification in the Limit for DFAs [OG92]

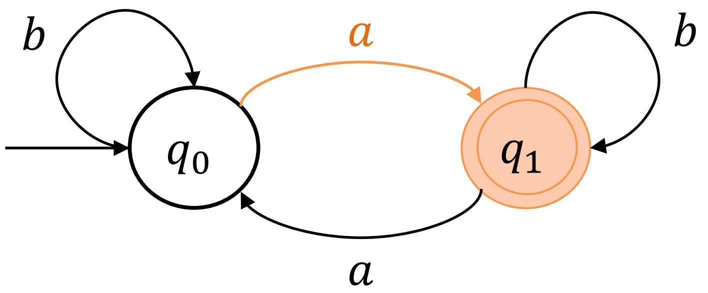
- Constructing a characteristic sample
- Every state is represented by an access word

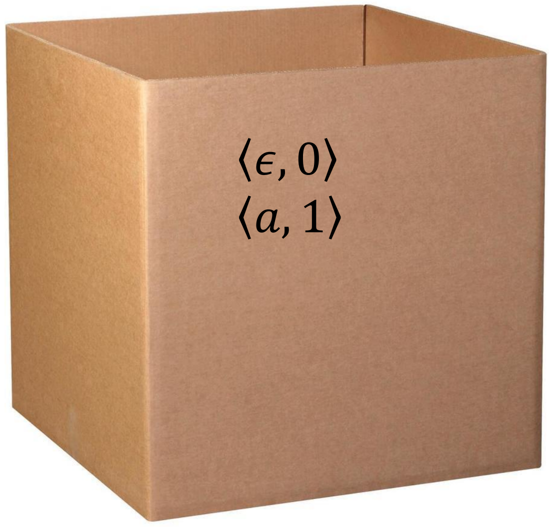




Identification in the Limit for DFAs [OG92]

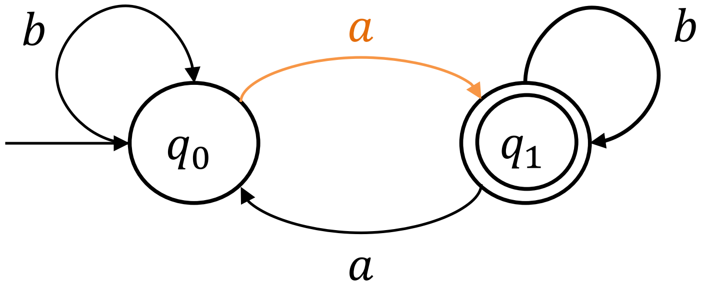
- Constructing a characteristic sample
- Distinctive suffixes between states:
  - If $\delta(q_0, w) \neq \delta(q_0, u)$
  - there exists a suffix z such that $w \cdot z \in L(A), u \cdot z \notin L(A)$
  - Add $w \cdot z, u \cdot z$

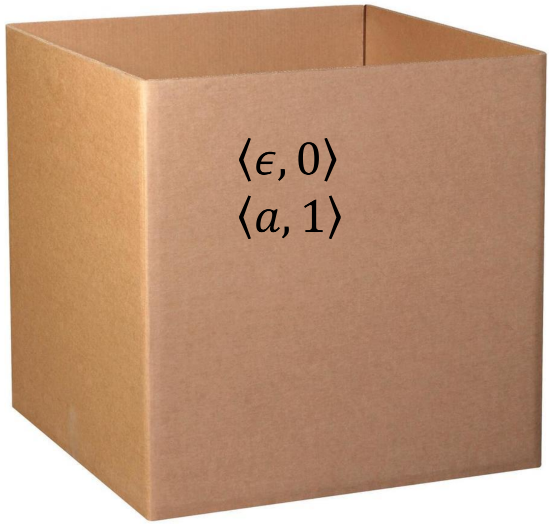




Identification in the Limit for DFAs [OG92]

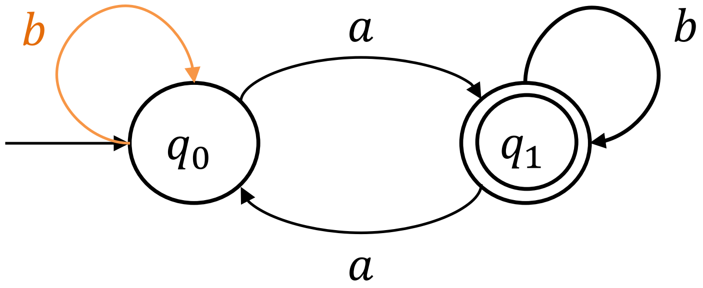
- Constructing a characteristic sample
- Representing the transition relation

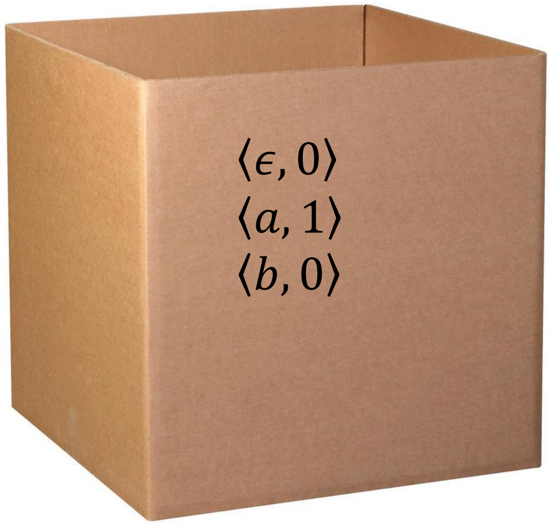




Identification in the Limit for DFAs [OG92]

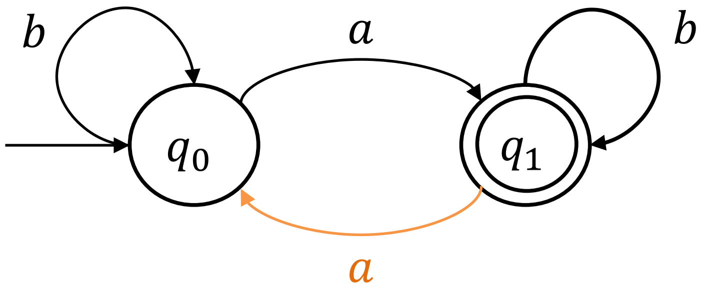
- Constructing a characteristic sample
- Representing the transition relation

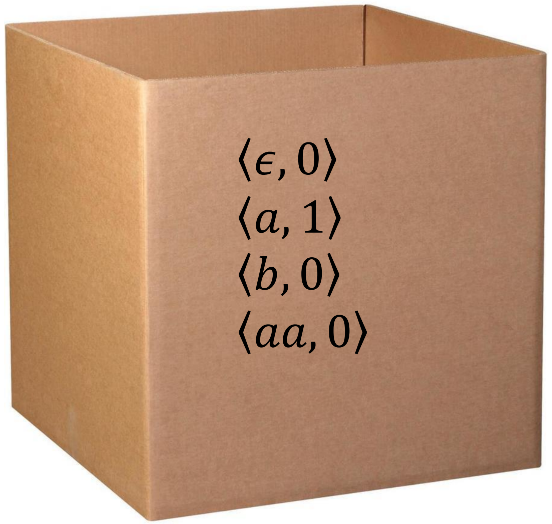




Identification in the Limit for DFAs [OG92]

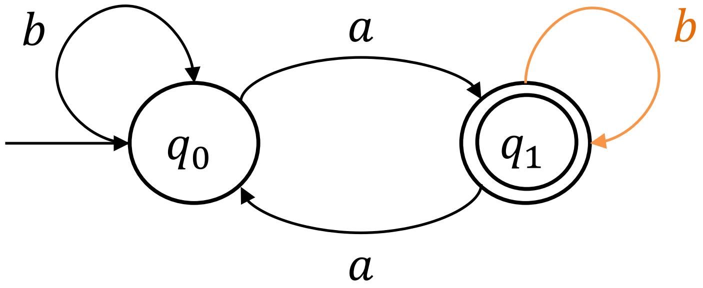
- Constructing a characteristic sample
- Representing the transition relation

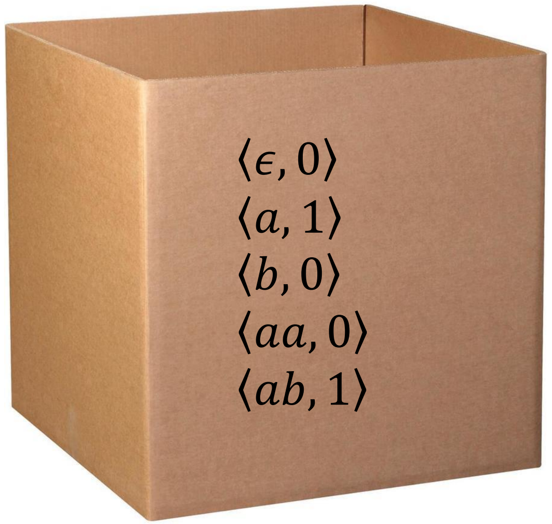




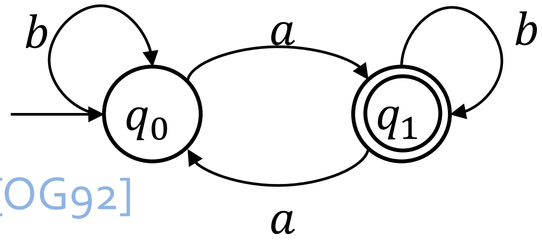
Identification in the Limit for DFAs [OG92]

- Constructing a characteristic sample
- Representing the transition relation

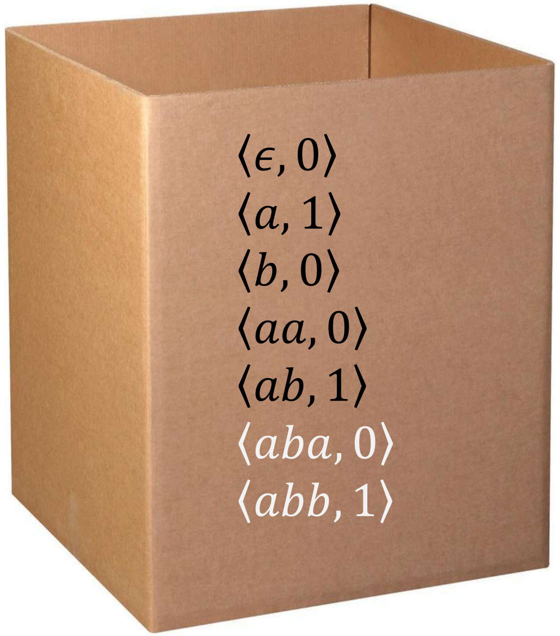


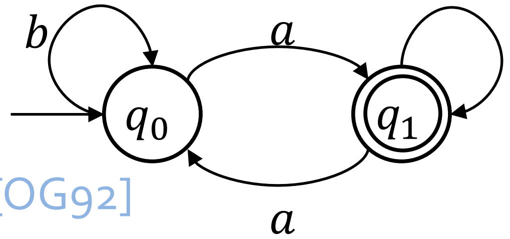


Identification in the Limit for DFAs [OG92]



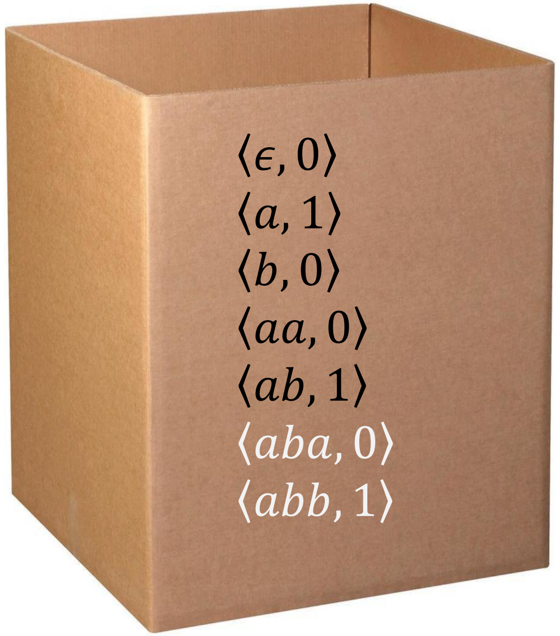
- Constructing a DFA

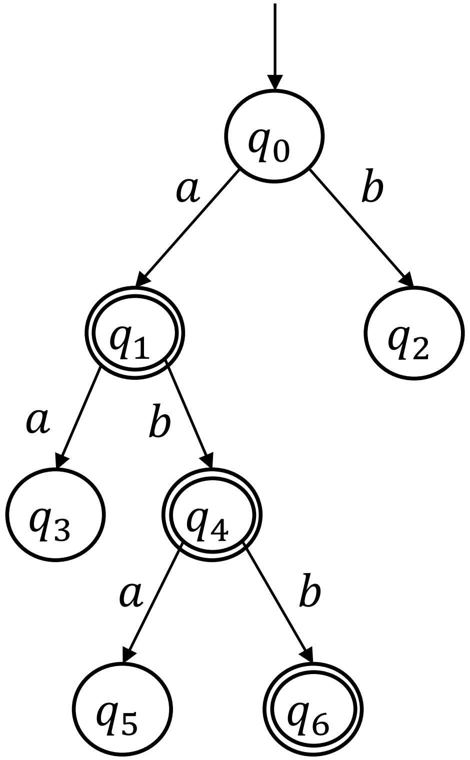




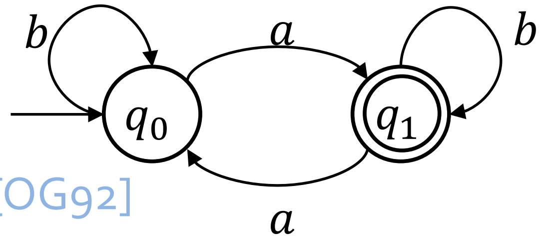
Identification in the Limit for DFAs [OG92]

- Constructing a DFA
- Prefix-tree automaton

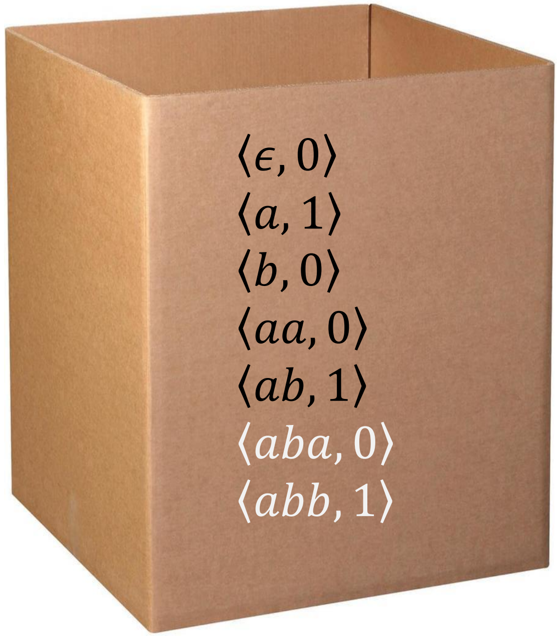


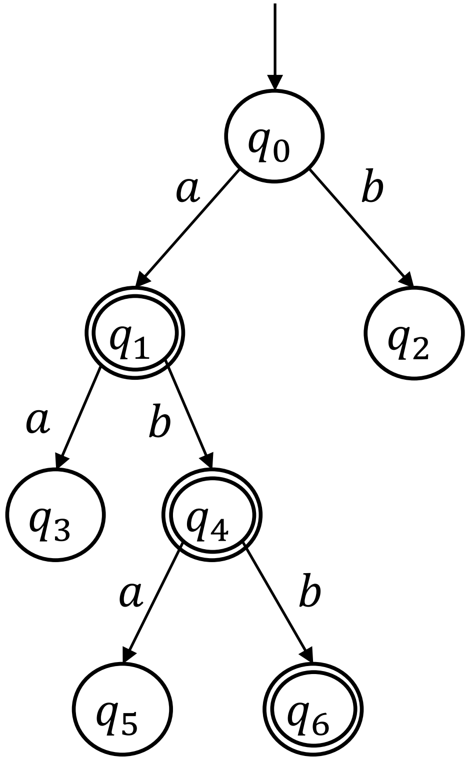


Identification in the Limit for DFAs [OG92]

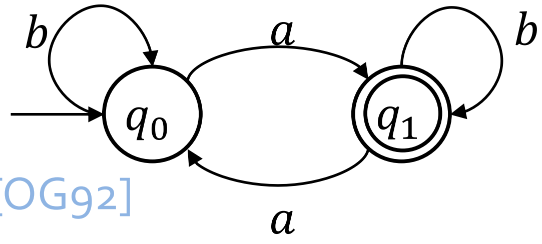


- Constructing a DFA
- Prefix-tree automaton
- Join states according to S'

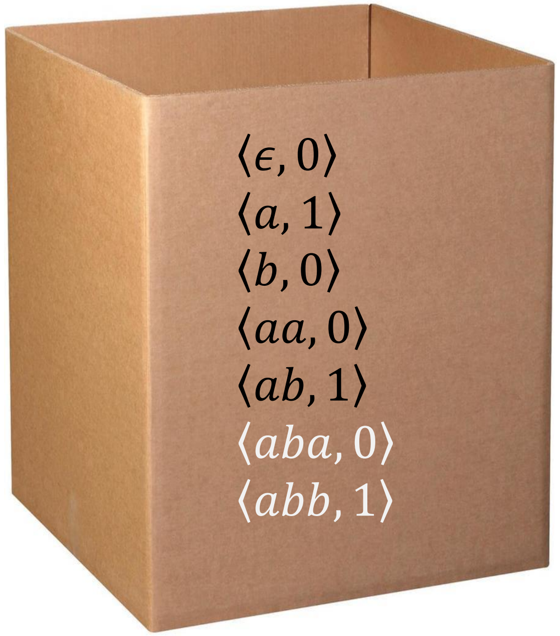


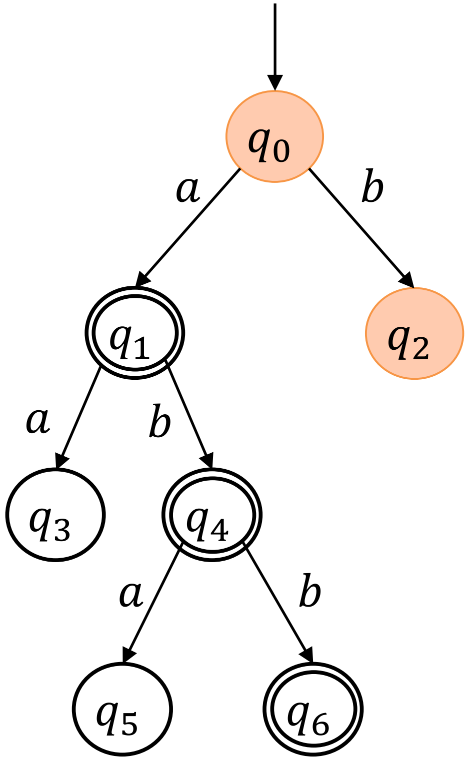


Identification in the Limit for DFAs [OG92]

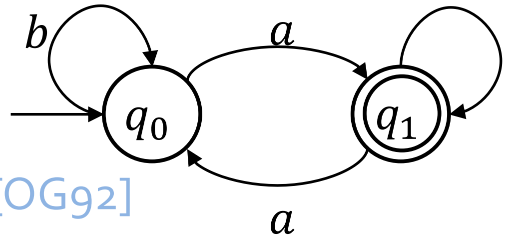


- Constructing a DFA
- Prefix-tree automaton
- Join states according to S'

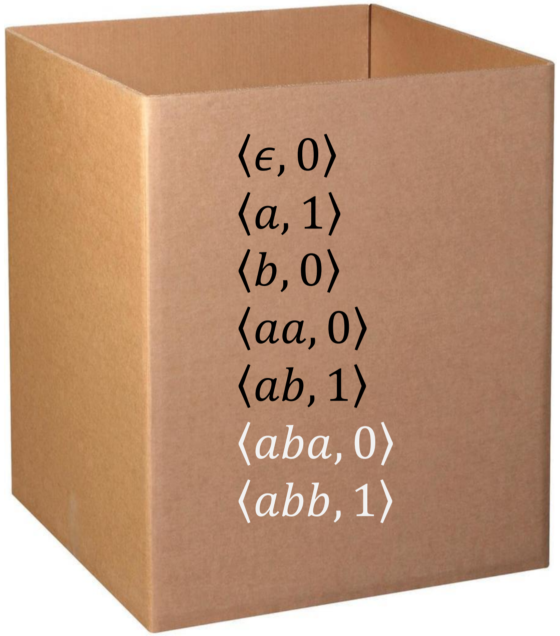


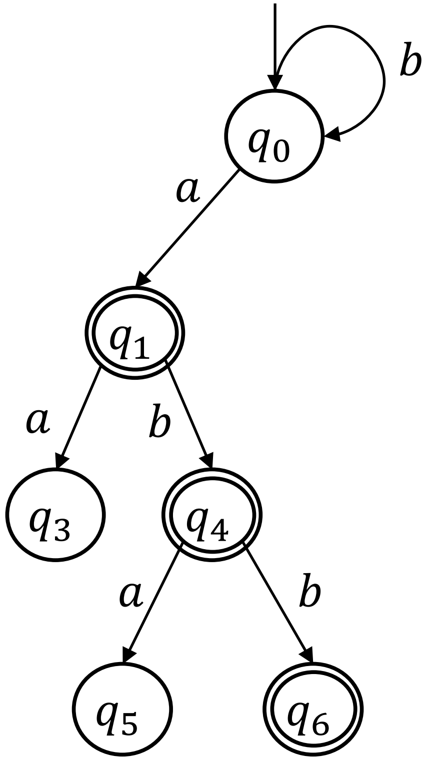


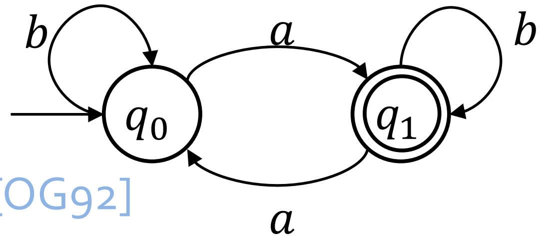
Identification in the Limit for DFAs [OG92]



- Constructing a DFA
- Prefix-tree automaton
- Join states according to S'

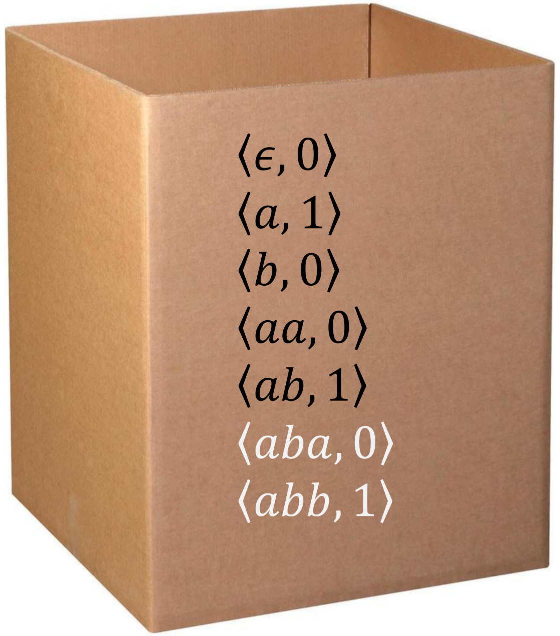


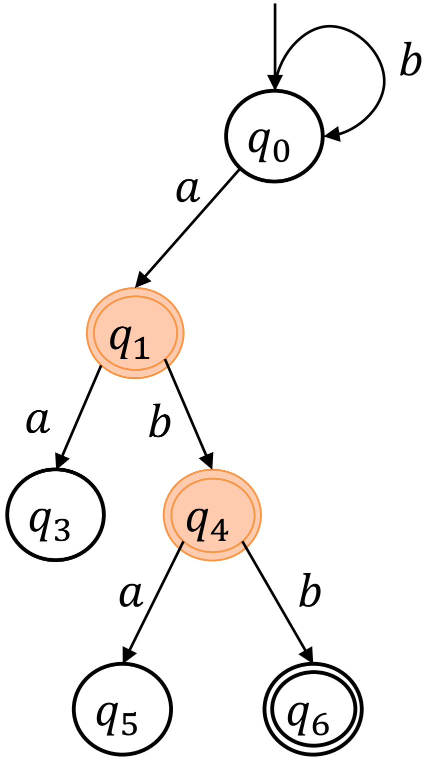




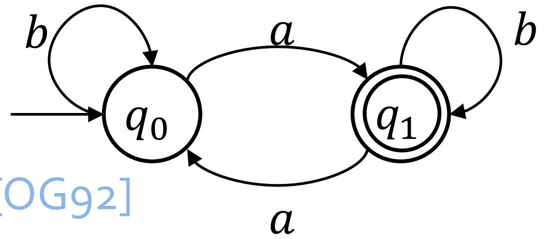
Identification in the Limit for DFAs [OG92]

- Constructing a DFA
- Prefix-tree automaton
- Join states according to S'

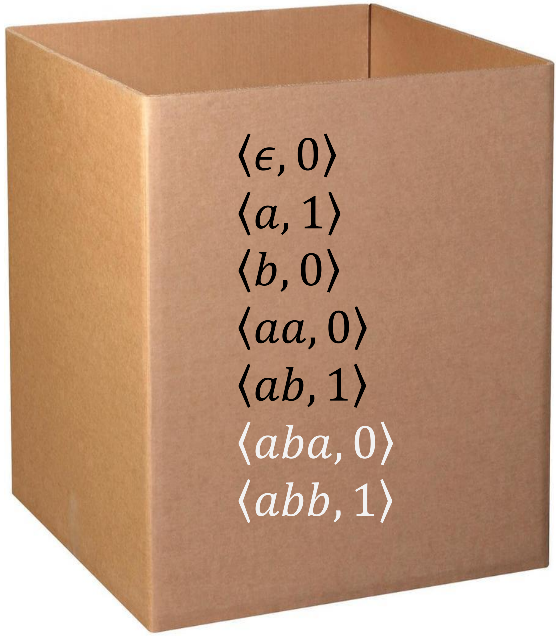


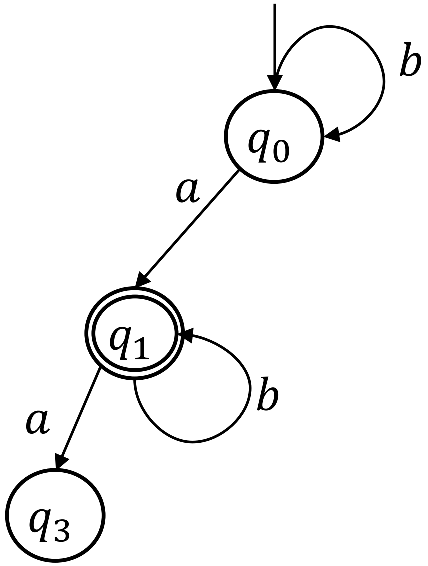


Identification in the Limit for DFAs [OG92]

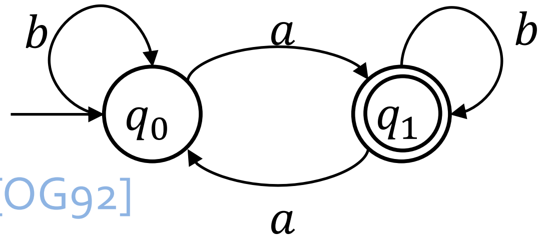


- Constructing a DFA
- Prefix-tree automaton
- Join states according to S'

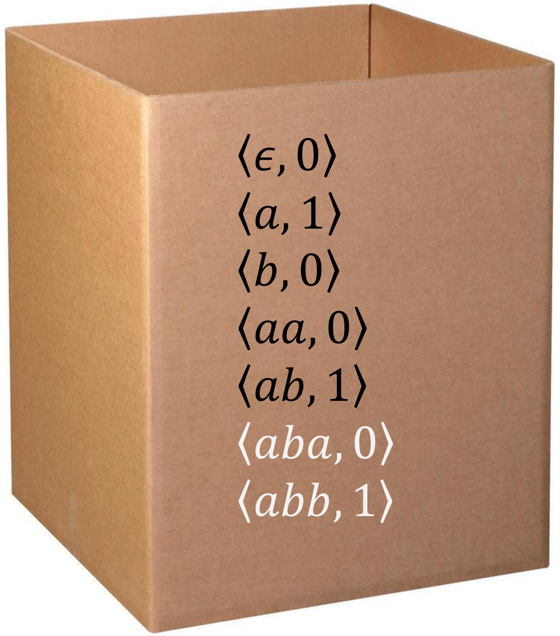


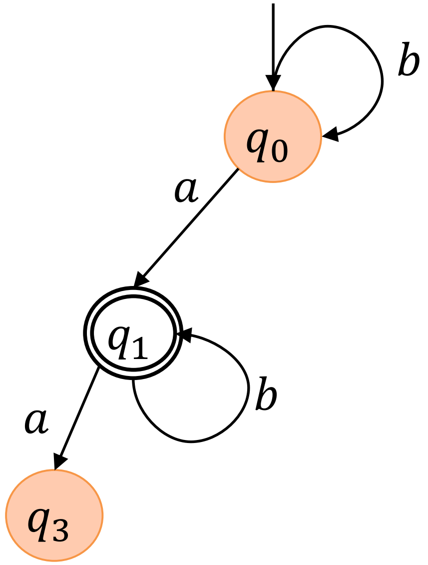


Identification in the Limit for DFAs [OG92]

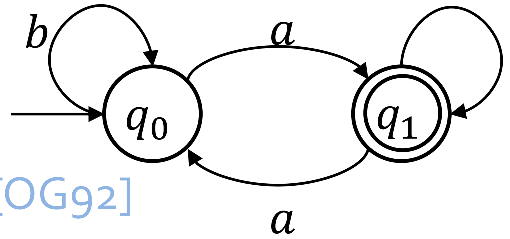


- Constructing a DFA
- Prefix-tree automaton
- Join states according to S'

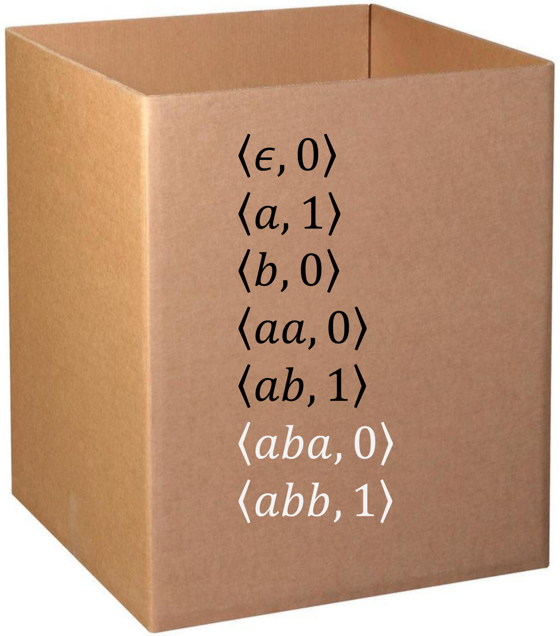


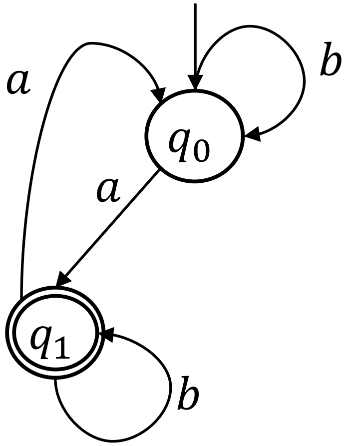


Identification in the Limit for DFAs [OG92]



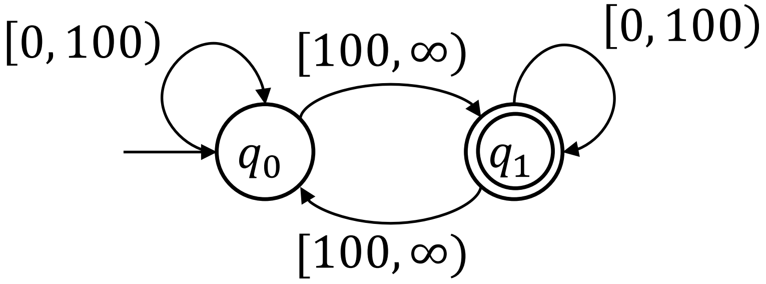
- Constructing a DFA
- Prefix-tree automaton
- Join states according to S'

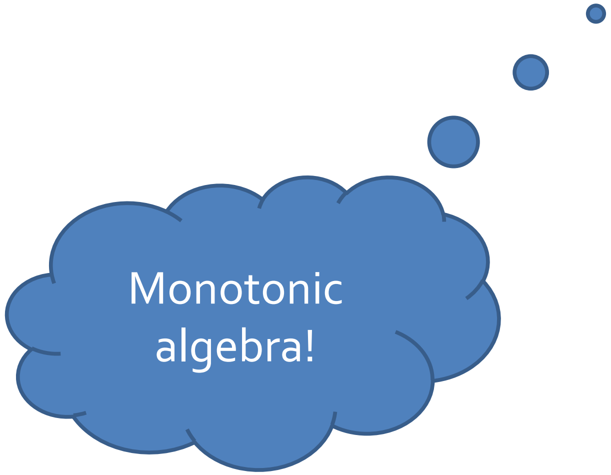




Identification in the Limit for SFAs

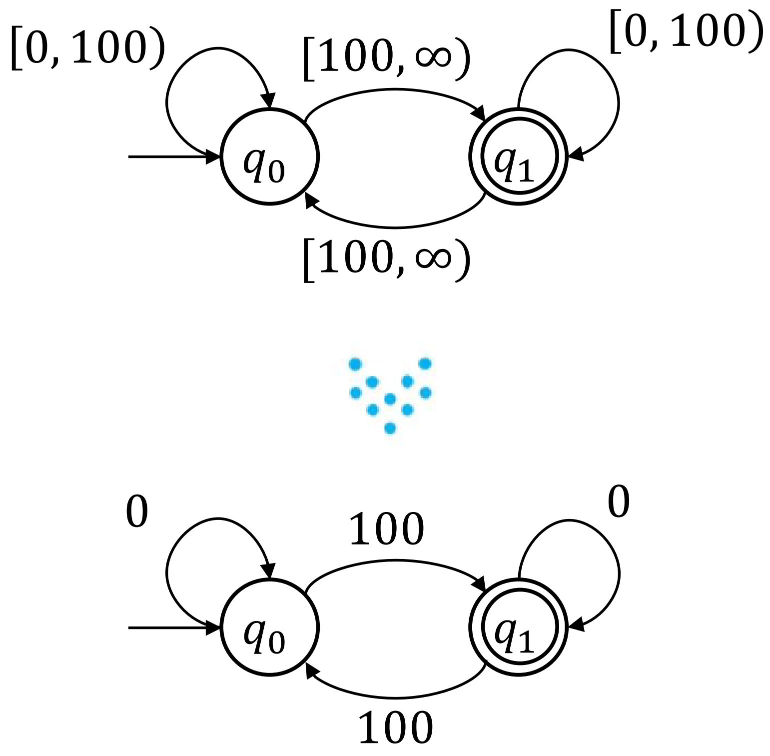
- Learn the SFA out of a set of concrete words
- **Creating a set of concrete words**
- concretize $(\langle \psi_1, \dots, \psi_n \rangle) = \langle \Gamma_1, \dots, \Gamma_n \rangle$
- concretize $([0{,}100), [100, \infty)) = \langle \{0\}, \{100\} \rangle$





Identification in the Limit for SFAs

- Learn the SFA out of a set of concrete words
- **Creating a set of concrete words**
- concretize $(\langle \psi_1, \dots, \psi_n \rangle) = \langle \Gamma_1, \dots, \Gamma_n \rangle$
- concretize $([0,100), [100, \infty)) = \langle \{0\}, \{100\} \rangle$



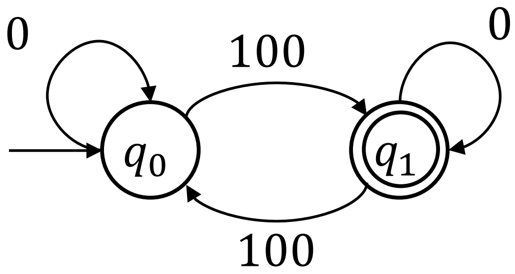
Identification in the Limit for SFAs

- Learn the SFA out of a set of concrete words
- **Creating a set of concrete words**

Identification in the Limit for SFAs

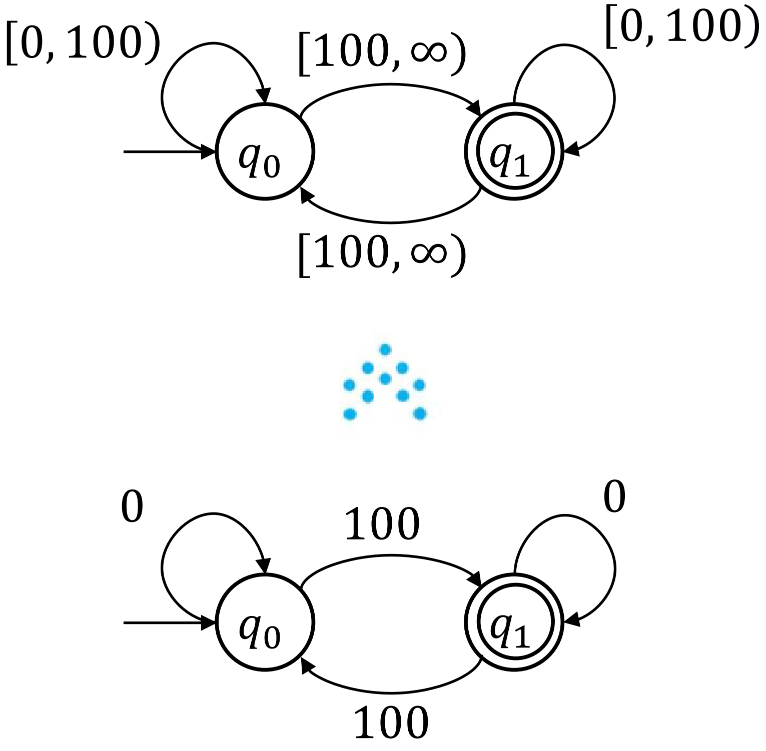
- Learn the SFA out of a set of concrete words
- **Construct an SFA**
- $\text{generalize}(\langle \Gamma_1, \dots, \Gamma_n \rangle) = \langle \psi_1, \dots, \psi_n \rangle$
- $\text{generalize}(\{0\}, \{100\}) = \langle [0,100), [100, \infty) \rangle$

Monotonic algebra!



Identification in the Limit for SFAs

- Learn the SFA out of a set of concrete words
- **Construct an SFA**
- $\text{generalize}(\Gamma_1, \dots, \Gamma_n) = \langle \psi_1, \dots, \psi_n \rangle$
- $\text{generalize}(\{0\}, \{100\}) = \langle [0{,}100), [100, \infty) \rangle$



Identification in the Limit for SFAs

- generalize
- $\Gamma_1 = \{0, 50, 400\}$ $\quad \Gamma_2 = \{100, \ 800\}$ $\quad \Gamma_3 = \{2048\}$

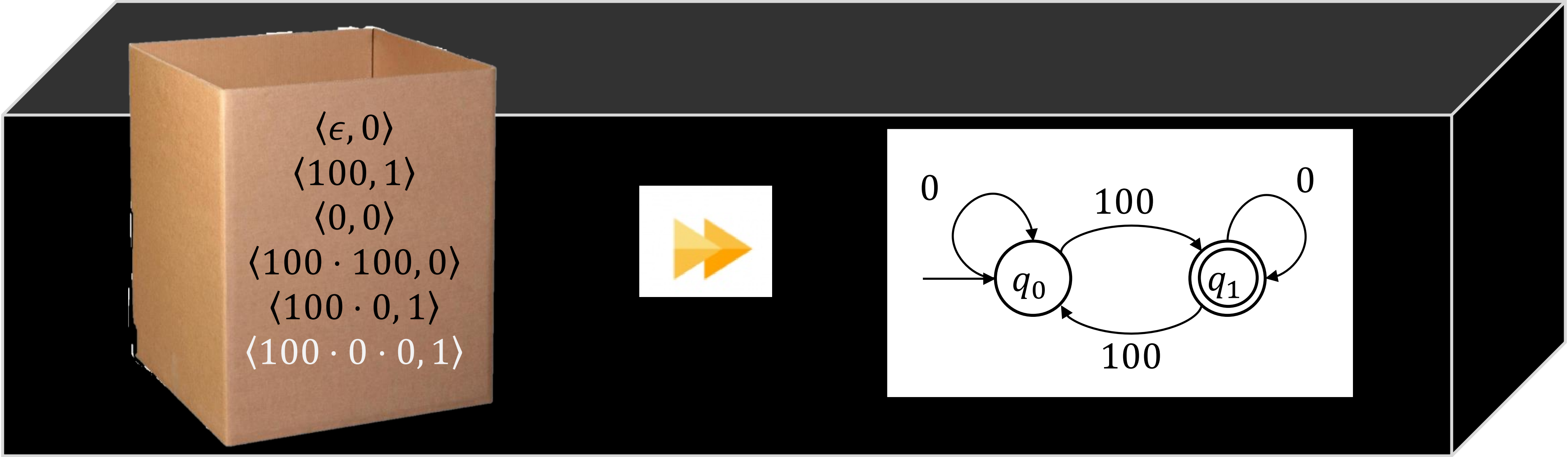
0 100 400 800 2048

$[0{,}100)$ $[100, 400)$ $[400{,}800)$ $[800, 2048)$ $[2048, \infty)$

- $\text{generalize}(\langle \Gamma_1, \Gamma_2, \Gamma_3 \rangle) = \langle [0{,}100) \vee [400{,}800), \ \ [100{,}400) \vee [800{,}2048), [2048, \infty) \rangle$

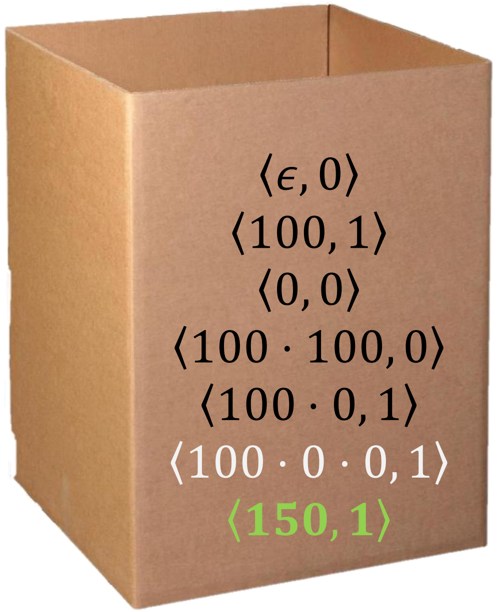
Identification in the Limit for SFAs

- Learn the SFA out of a set of concrete words
- **Construct an SFA**



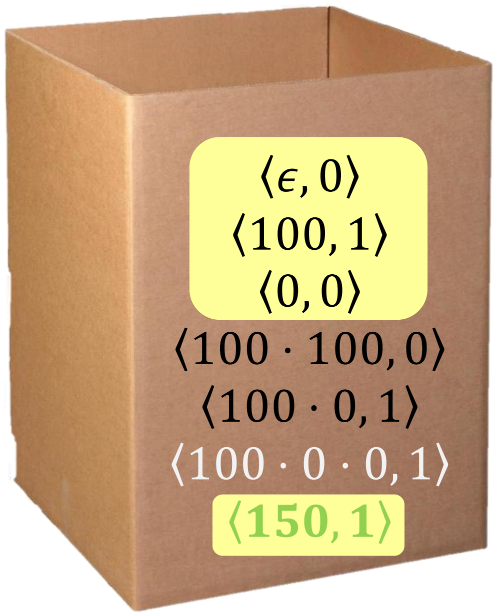
Identification in the Limit for SFAs

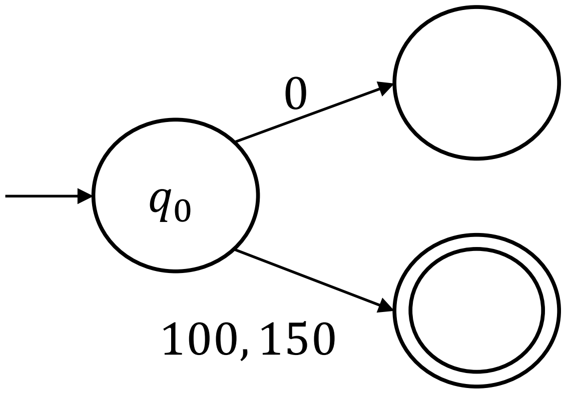
- Learn the SFA out of a set of concrete words
- **Construct an SFA**
- $\mathrm{decontaminate}(\Sigma) = \Sigma'$
- $\Sigma' \subseteq \Sigma$ and contains exactly the alphabet of concretizations



Identification in the Limit for SFAs

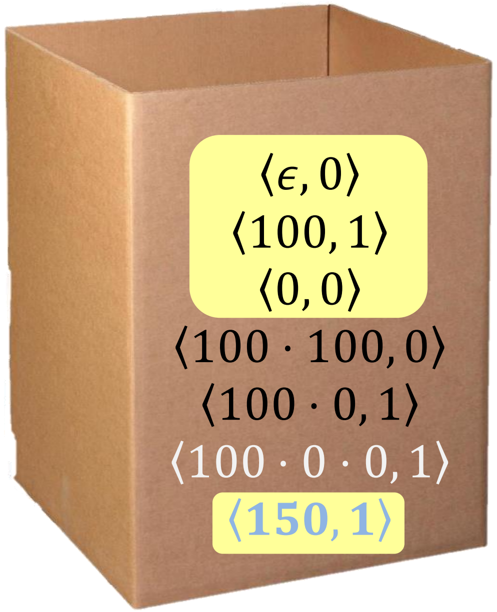
- Travers words by lexicographic order
- Add letters that are needed for access words and for transitions relation

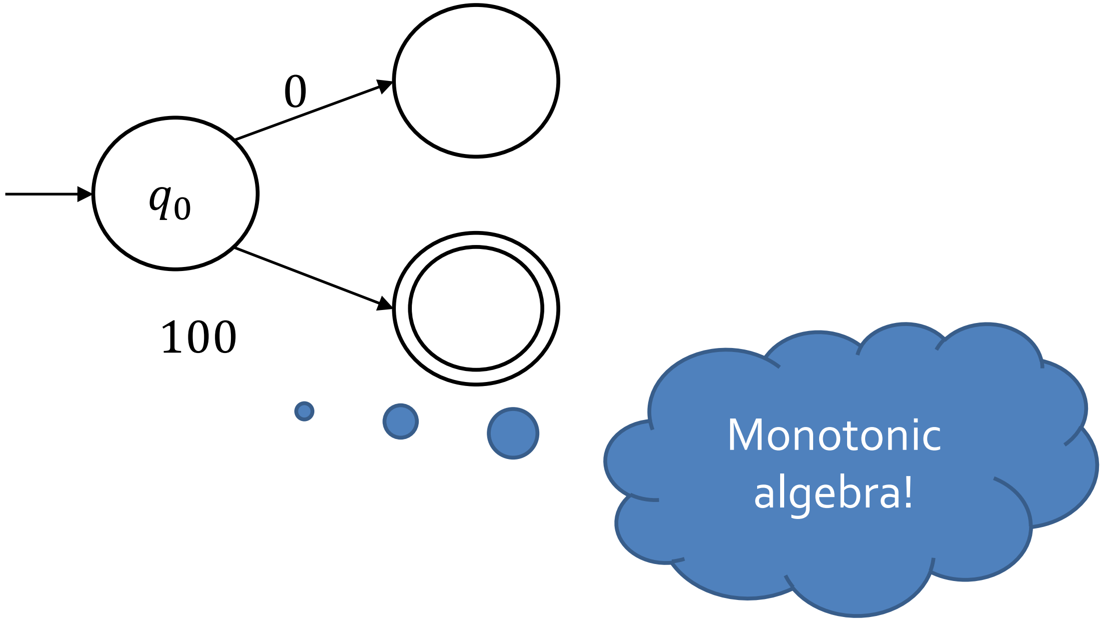




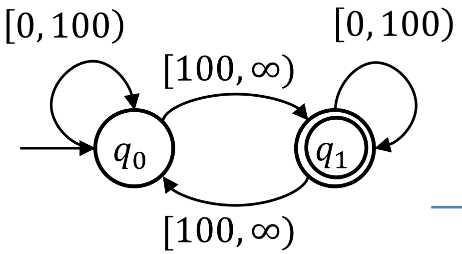
Identification in the Limit for SFAs

- Travers words by lexicographic order
- Add letters that are needed for access words and for transitions relation





Identification in the Limit for SFAs



SFA A

concretize

DFA D_A

Black box characteristic sample for DFAs

Sample S_A

Sample $S \supseteq S_A$

decontaminate

Sample $S \supseteq S' \supseteq S_A$

Black box infer a DFA

DFA $D_{S'}$

generalize

SFA $A_{S'}$

Necessary Condition

$$\langle \psi_1, \dots, \psi_n \rangle \xrightarrow{\text{concretize}} \langle \Gamma_1, \dots, \Gamma_n \rangle$$

$$\langle \varphi_1, \dots, \varphi_n \rangle \xleftarrow{\text{generalize}} \langle \Delta_1, \dots, \Delta_n \rangle$$

Necessary Condition

$$\langle \psi_1, \ldots, \psi_n \rangle \xrightarrow{\text{concretize}} \langle \Gamma_1, \ldots, \Gamma_n \rangle$$

If $[\![\Delta_i]\!] \supseteq [\![\Gamma_i]\!]$

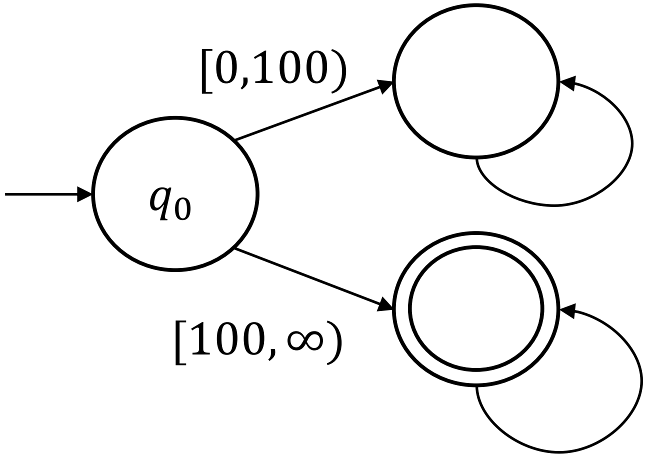
Then $[\![\varphi_i]\!] = [\![\psi_i]\!]$

$$\langle \varphi_1, \ldots, \varphi_n \rangle \xleftarrow{\text{generalize}} \langle \Delta_1, \ldots, \Delta_n \rangle$$

Necessary Condition

Otherwise, we cannot learn outgoing transitions of a single state

$\langle \psi_1, \dots, \psi_n \rangle \xrightarrow{\text{concretize}} \langle \Gamma_1, \dots, \Gamma_n \rangle$



Poly time and data

If $[\![\Delta_i]\!] \supseteq [\![\Gamma_i]\!]$

Then $[\![\varphi_i]\!] = [\![\psi_i]\!]$

$\langle \varphi_1, \dots, \varphi_n \rangle \xleftarrow{\text{generalize}} \langle \Delta_1, \dots, \Delta_n \rangle$

Propositional Algebra

- Predicates are defined over $\{p_1, \dots, p_k\}$
- Examples: $p_1 \vee p_2$, $(p_1 \wedge p_2) \vee p_3$
- Looking for efficient concretize and generalize

Propositional Algebra

- Predicates are defined over $\{p_1, \dots, p_k\}$
- Examples: $p_1 \vee p_2$, $(p_1 \wedge p_2) \vee p_3$
- Looking for efficient concretize and generalize

No one to one function from Υ to P

Υ

set of semantic Boolean functions over k propositions

$|\Upsilon| = 2^{2^k}$

Every function defines a set of sets of propositions satisfying the function

P

set of concrete partitions of polynomial size in k

$|\mathrm{P}| < |\Upsilon|$

Query Learning of SFAs

- L^* - style learning of SFA
- Goal: learn an SFA over a Boolean algebra, while asking queries over **concrete** letters
- [AD18] suggest MAT^* for learning SFAs

Query Learning of SFAs

- Learnability of the underlying algebra is a necessary condition
- Membership

Is $a \in [\![\varphi]\!]$?

Yes / No

Teacher

Learner

Query Learning of SFAs

- Learnability of the underlying algebra is a necessary condition
- Equivalence

Teacher

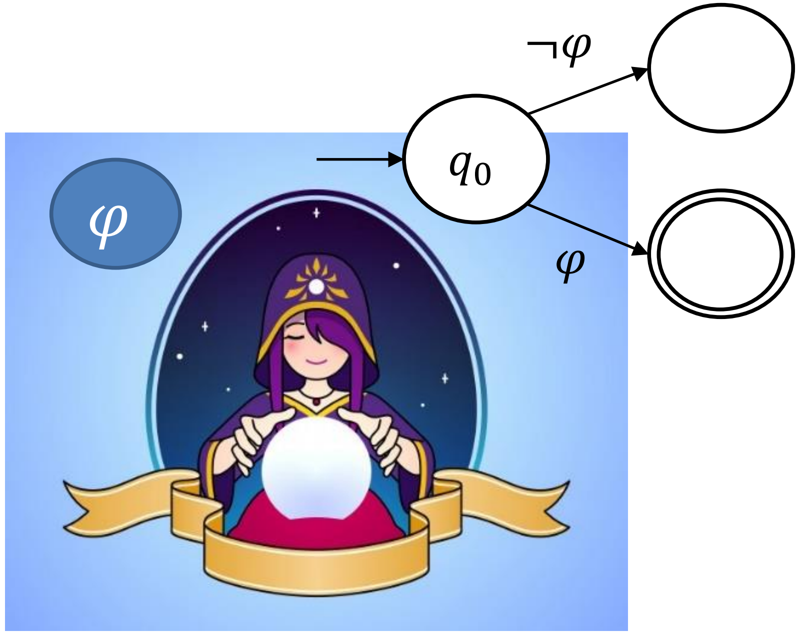
Yes /
No + cex

Is $[\![\psi]\!] = [\![\varphi]\!]$?

Learner

Query Learning of SFAs

- Learnability of the underlying algebra is a necessary condition
- Assume that we can learn SFA, then we can learn the algebra



Teacher


Learner

Query Learning of SFAs

- Concise SFA over the propositional algebra cannot be polynomially learned using MQ and EQ
- The teacher can force the learner to ask $2^k - 1$ queries
- Membership

No

Is $\langle 0,1,0,\ldots,1\rangle \in [\![\varphi]\!]$?

Teacher

Learner

Query Learning of SFAs

- Concise SFA over the propositional algebra cannot be polynomially learned using MQ and EQ
- The teacher can force the learner to ask $2^k - 1$ queries
- Equivalence

Teacher

No
+
$\bar{b} \notin [\![\psi]\!]$

Is $[\![\psi]\!] = [\![\varphi]\!]$?

Learner

Complexity of SFAs

- Usually, the **size** of DFA is measured by its number of states
- For SFAs, we need to consider:

$$\langle n, m, l \rangle$$

number of states

out-degree

size of the most complex predicate

Complexity of SFAs

Normalized SFA

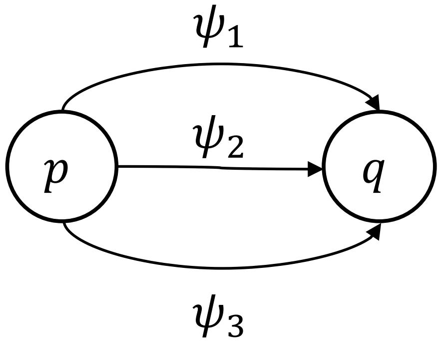
- One transition between each pair of states
- Predicates labeling the transitions can be very complex

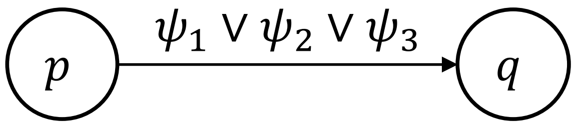
Neat SFA

- Only basic transitions
- Predicates labeling transitions are simple
- Can cause an exponential blowup in the number of transitions

Complexity of SFAs

- Converting to normalized
- Disjunction between all transition predicates



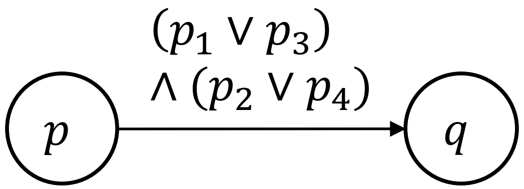


$\langle n, m, m \cdot l \rangle$

$\langle n, m, l \rangle$

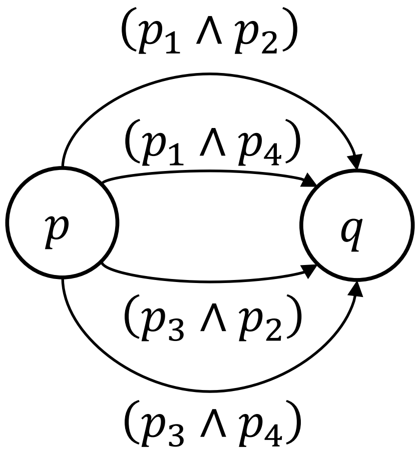
Complexity of SFAs

- Converting to neat
- Splitting into basic transitions, using DNF



$\langle n, m, l \rangle$

$(p_1 \wedge p_2) \vee$
$(p_1 \wedge p_4) \vee$
$(p_3 \wedge p_2) \vee$
$(p_3 \wedge p_4)$



$\langle n, m \cdot 2^l, l \rangle$

Complexity of SFAs

- **For monotonic algebras**, transforming to DNF is polynomial in the size of the original formula
- $$\begin{aligned}([0,100) \vee [200,500)) \wedge \big([0,300) \vee [400,600)\big) = \\ \big([0,100) \wedge [0,300)\big) \vee \big([0,100) \wedge [400,600)\big) \vee \\ \big([200,500) \wedge [0,300)\big) \vee \big([200,500) \wedge [400,600)\big) = \\ [0,100) \vee [200,300) \vee [400,500)\end{aligned}$$
- Then, over monotonic algebras, transforming to neat is polynomial

Complexity of SFAs – Automata Operations

Operation	$\langle \mathbf{n}, \mathbf{m}, \mathbf{l} \rangle$
product construction $\mathcal{M}_1, \mathcal{M}_2$	$\langle n_1 \times n_2,\ m_1 \times m_2,\ size^{\mathbb{P}}_{\wedge}(l_1, l_2) \rangle$
complementation of deterministic $\mathcal{M}_1$ [1]	$\langle n_1 + 1,\ m_1 + 1,\ size^{\mathbb{P}}_{\vee m_1}(l_1) \rangle$
determinization of $\mathcal{M}_1$	$\langle 2^{n_1},\ 2^{m_1},\ size^{\mathbb{P}}_{\wedge n_1 \times m_1}(l_1) \rangle$ [2]
minimization of $\mathcal{M}_1$	$\langle n_1, m_1,\ size^{\mathbb{P}}_{\wedge m_1}(l_1) \rangle$

Table 5.1: Analysis of standard automata procedures on SFAs.

Complexity of SFAs – Decision Procedures

Decision Procedures	**Time Complexity**
emptiness	linear in n, m
emptiness + feasibility	$n \times m \times sat^{\mathbb{P}}(l)$
membership of $\gamma_1 \cdots \gamma_t \in \mathbb{D}^*$	$\sum_{i=1}^{t} sat^{\mathbb{P}}(size_{\wedge}^{\mathbb{P}}(l, \|\psi_{\gamma_i}\|))$ 3
inclusion $\mathcal{M}_1 \subseteq \mathcal{M}_2$	$((n_1 \times n_2) \times (m_1 \times m_2) \times sat^{\mathbb{P}}(size_{\wedge}^{\mathbb{P}}(l_1, l_2)))$

Table 5.2: Analysis of times complexity of decision procedures for SFAs

SFA Summary

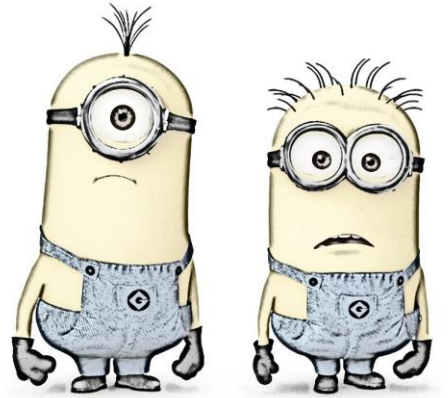
- Identification in the limit of SFA
  - Necessary and sufficient conditions
  - Algorithm for identification of SFAs over monotonic algebras

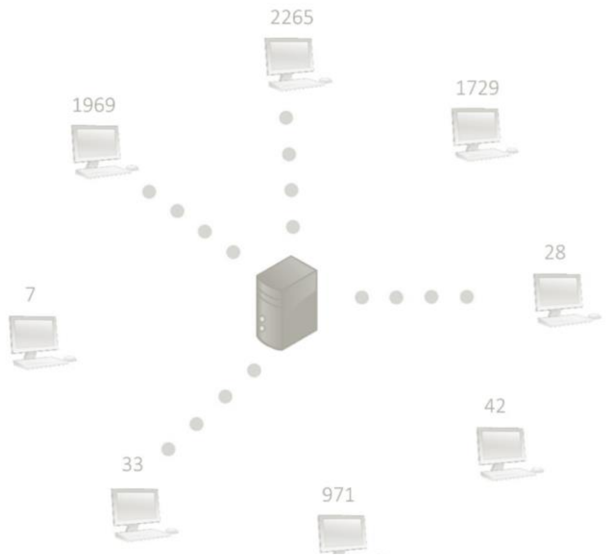
- Necessary condition for query learning of SFAs
  - SFAs over the propositional algebra are not efficiently learnable

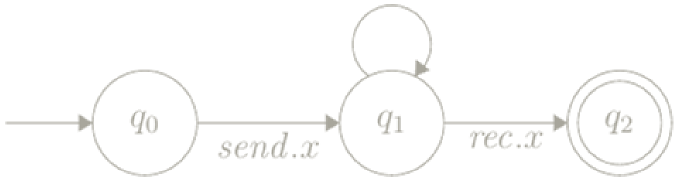
- Complexity of automata algorithms in terms of $\langle n, m, l \rangle$

number of states   out-degree   size of the most complex predicate

Thank you!
Questions?








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