

# Compositional Reasoning

Yih-Kuen Tsay

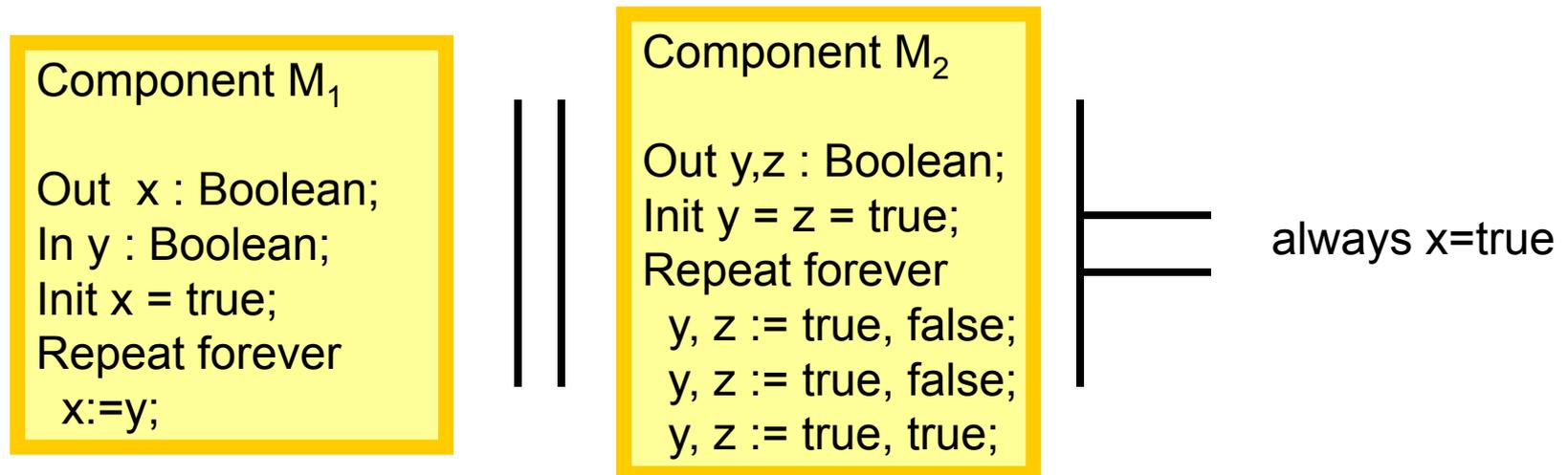
(original created by Yu-Fang Chen)

Dept. of Information Management

National Taiwan University

# Verification of Parallel Compositions

- **Verification Task:** verify if the system composed of components  $M_1$  and  $M_2$  satisfies a property  $P$ , i.e.,  $M_1 \parallel M_2 \models P$ .
- $M_1$  and  $M_2$  may rely on each other to satisfy  $P$ .

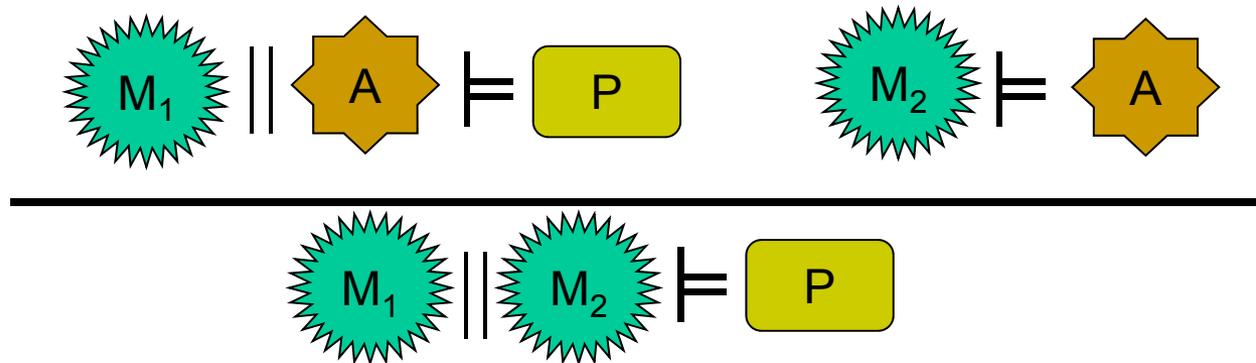


$M_1$  alone does not guarantee “always  $x = \text{true}$ ”!

- Can the construction of  $M_1 \parallel M_2$  be avoided?

# Compositional Reasoning

- An **Assume-Guarantee (A-G)** rule:



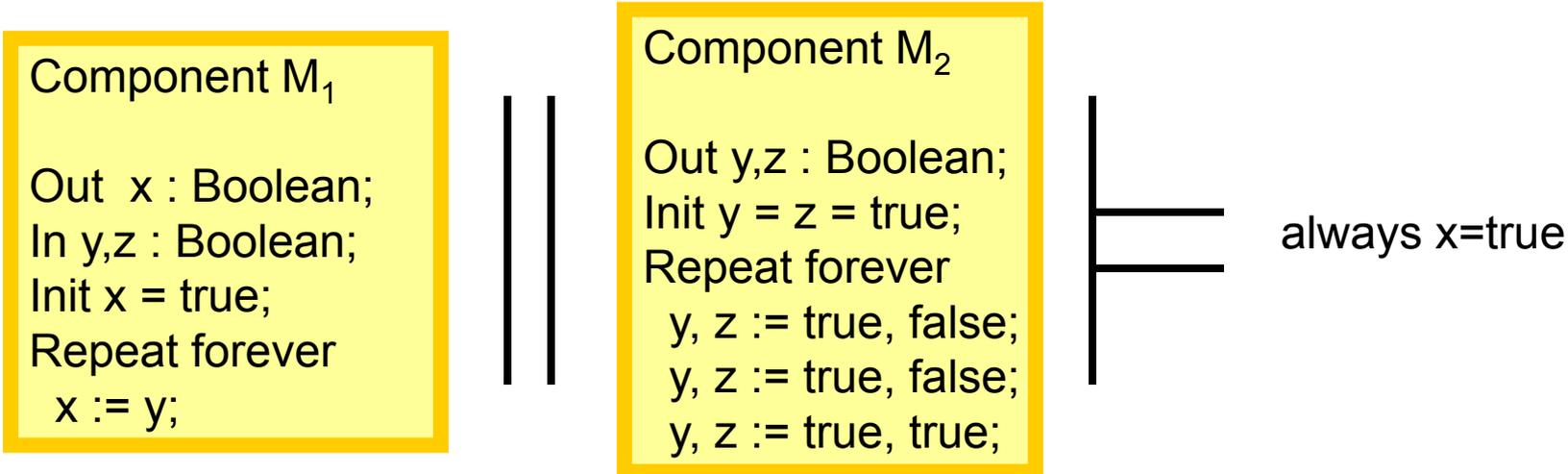
- If a small *contextual assumption*  $A$  (an abstraction of  $M_2$ ) exists, then the overall verification task may become easier.



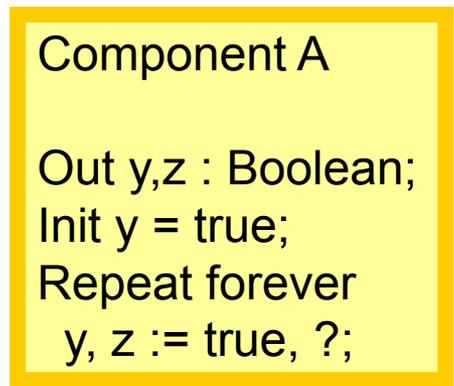
But, how to find  $A$  automatically?

- It is possible when  $M_1$ ,  $M_2$ ,  $A$ , and  $P$  are finite automata.

# Compositional Reasoning (cont.)

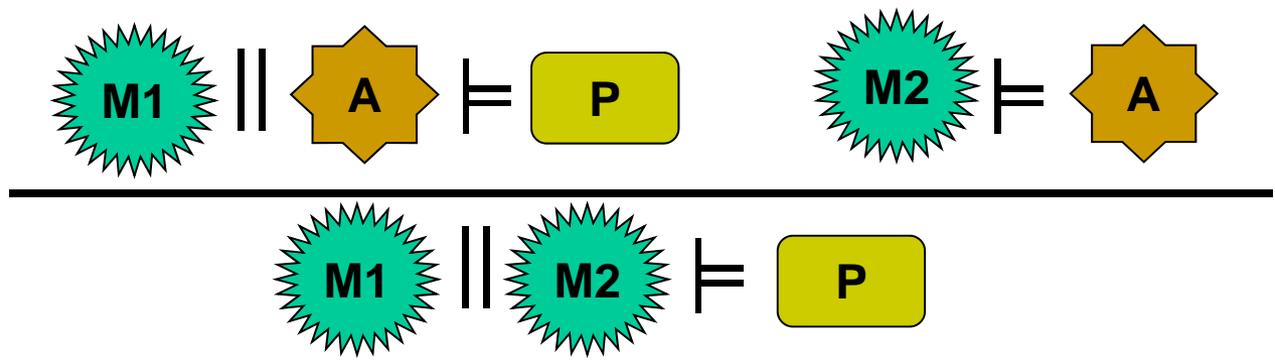


A suitable contextual assumption  :

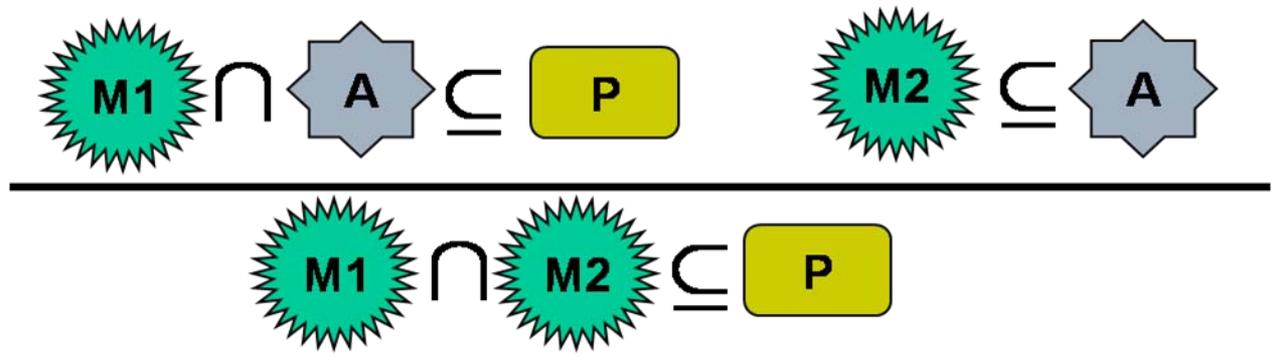


Component A has fewer states (automaton locations) than  $M_2$ .

# Setting the Stage



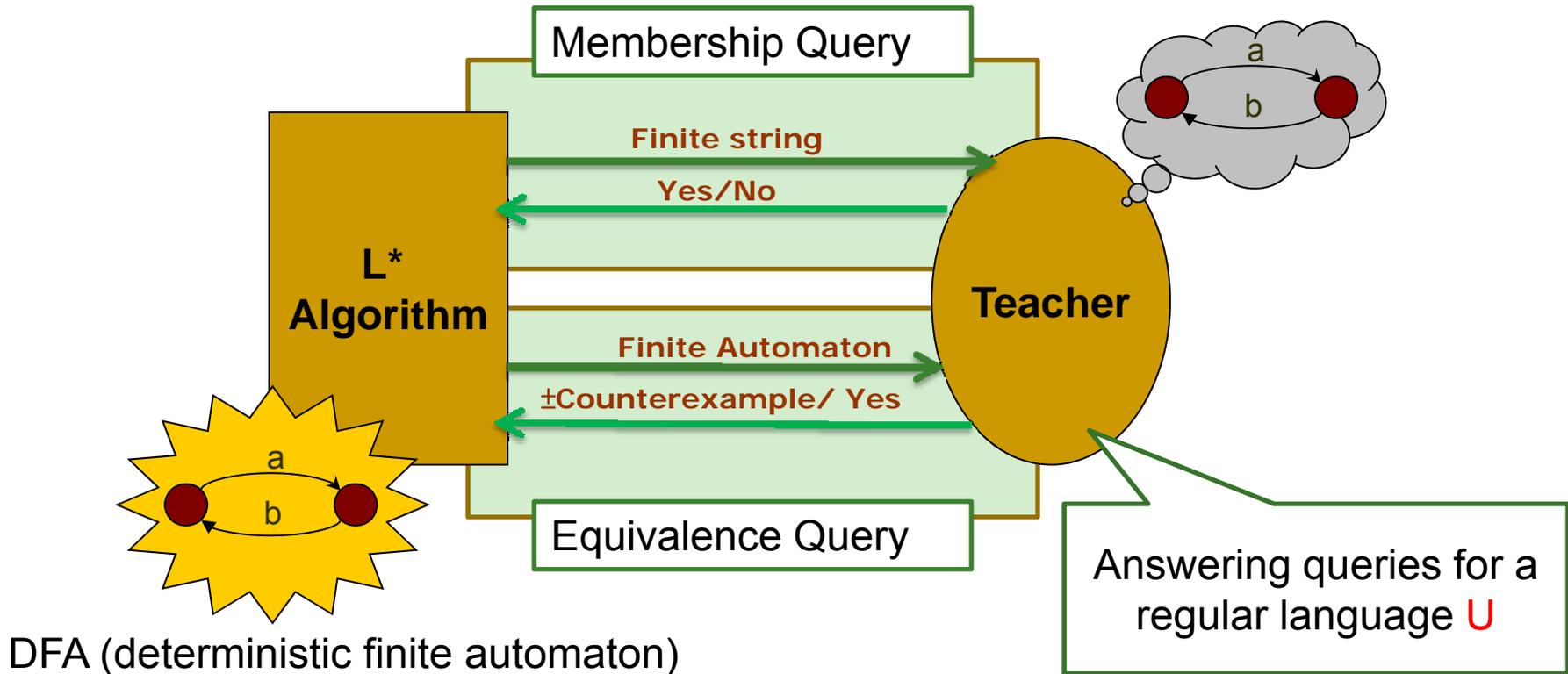
- ❑ The **behaviors of components** and **properties** are described as **regular languages**.
- ❑ **Parallel composition** is presented by **the intersection of the languages**.
- ❑ **A system satisfies a property** if **the language of the system is a subset of the language of the property**.



# Outline

- Learning-Based Compositional Model Checking:
  - Automation by Learning
  - The  $L^*$  Algorithm
  - The Problem of  $L^*$ -Based Approaches
  
- Learning Minimal Separating DFA's:
  - The  $L^{\text{SEP}}$  Algorithm
  - Comparison with Another Algorithm
  - Adapt  $L^{\text{SEP}}$  for Compositional Model Checking

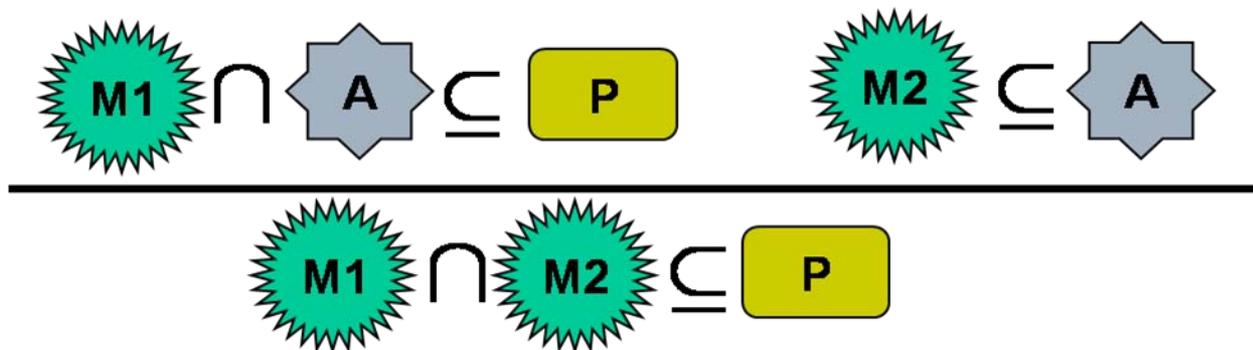
# Overview of the L\* Algorithm



If such a teacher is provided, L\* guarantees to produce a DFA that recognizes  $U$  using a polynomial number of queries.

# Automation by Learning

- **First developed** by Cobleigh, Giannakopoulou, and Păsăreanu [TACAS 2003]
- Apply the  $L^*$  learning algorithm for regular languages to find an  for the A-G rule:



# Basic Understanding

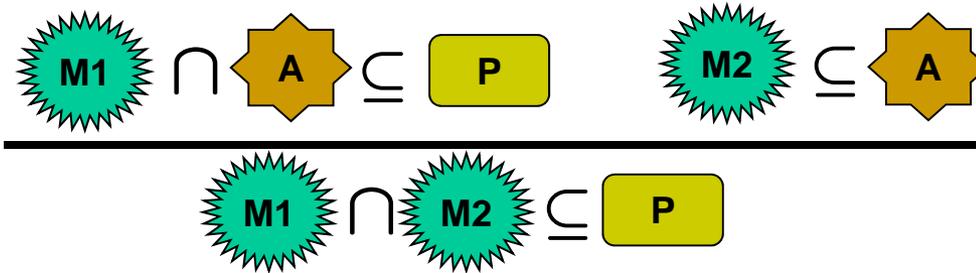
- A closer look at the A-G rule:

$$M1 \cap A \subseteq P \Leftrightarrow$$

$$M1 \cap A \cap \bar{P} = \emptyset \Leftrightarrow$$

$$A \cap \overline{(P \cup M1)} = \emptyset \Leftrightarrow$$

$$A \subseteq \overline{P \cup M1}$$



$$\text{When } A = \overline{P \cup M1},$$

$$M2 \subseteq A \Leftrightarrow$$

$$M2 \subseteq \overline{P \cup M1} \Leftrightarrow$$

$$M2 \cap \overline{(P \cup M1)} = \emptyset \Leftrightarrow$$

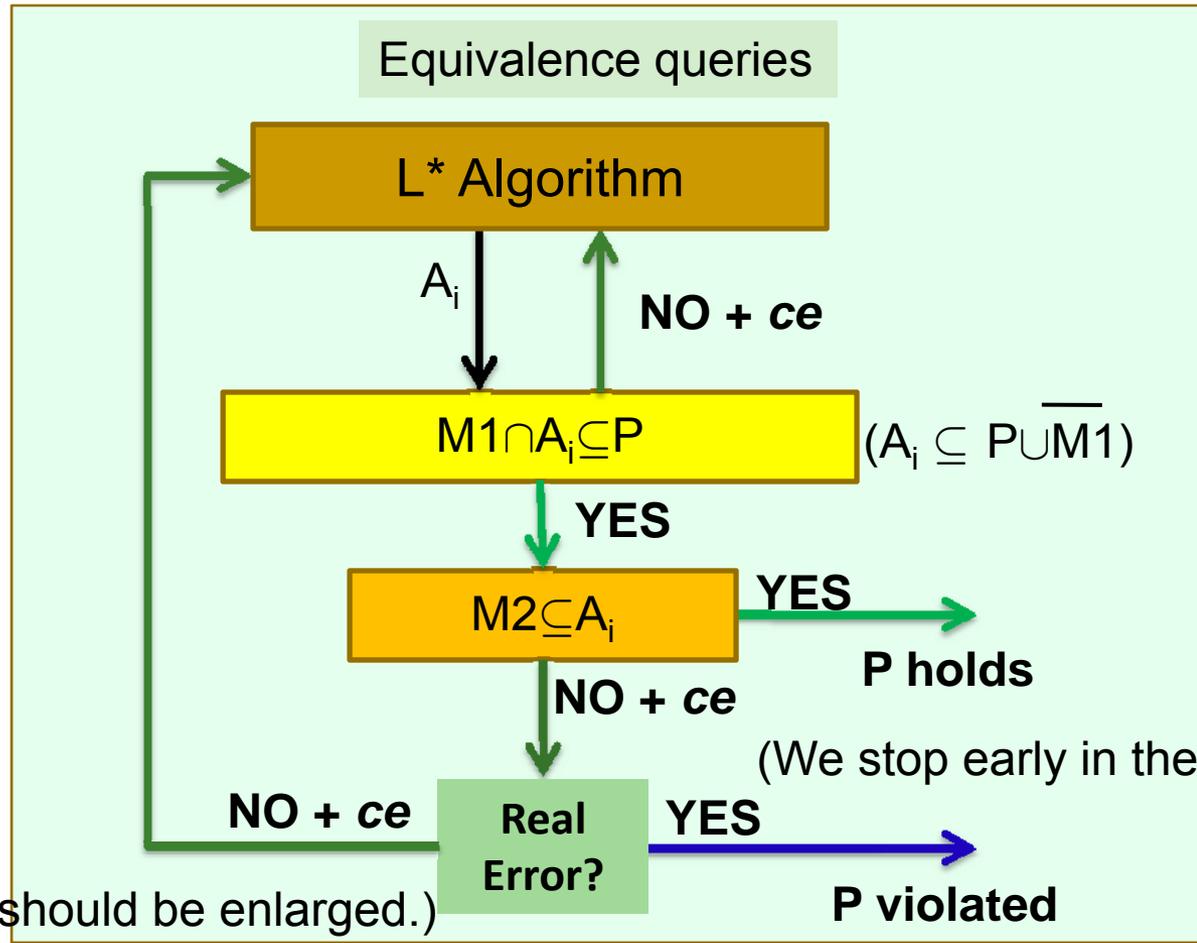
$$M2 \cap \bar{P} \cap M1 = \emptyset \Leftrightarrow$$

$$M1 \cap M2 \subseteq P$$

- Conceptually, the target language is  $\overline{P \cup M1}$ , the *weakest assumption* for the premise  $M1 \cap A \subseteq P$ .
- Actually reaching the target would be even worse than checking  $M1 \cap M2 \subseteq P$  directly.
- It really pays off when we can stop earlier ...

# The Algorithm of Cobleigh *et al.*

Target:  $\overline{P \cup M1}$



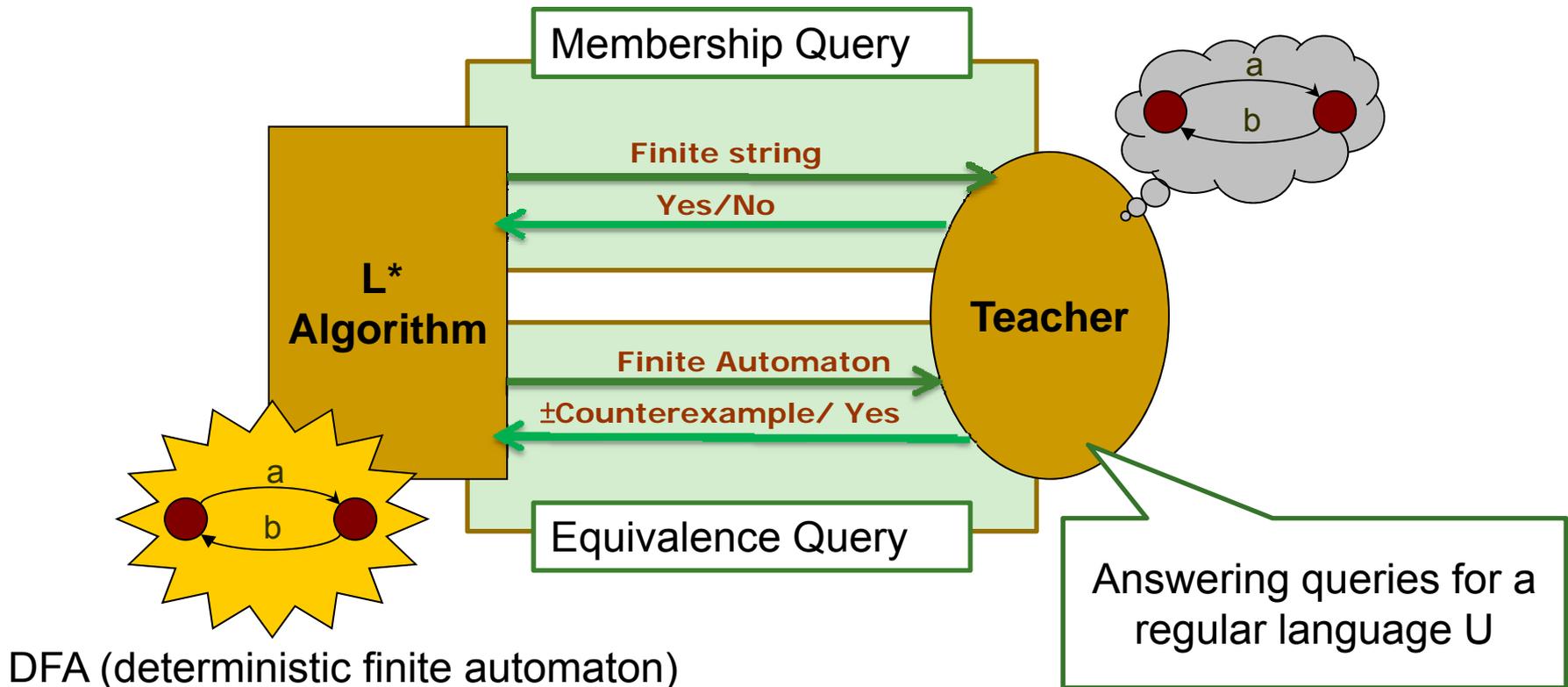
(ce tells how  $A_i$  should be enlarged.)

(We stop early in these two cases.)

(ce is a real error if ce is in  $M2$ , but not in  $\overline{P \cup M1}$ , implying  $M2 \not\subseteq \overline{P \cup M1}$ , i.e.,  $M1 \cap M2 \not\subseteq P$ .)

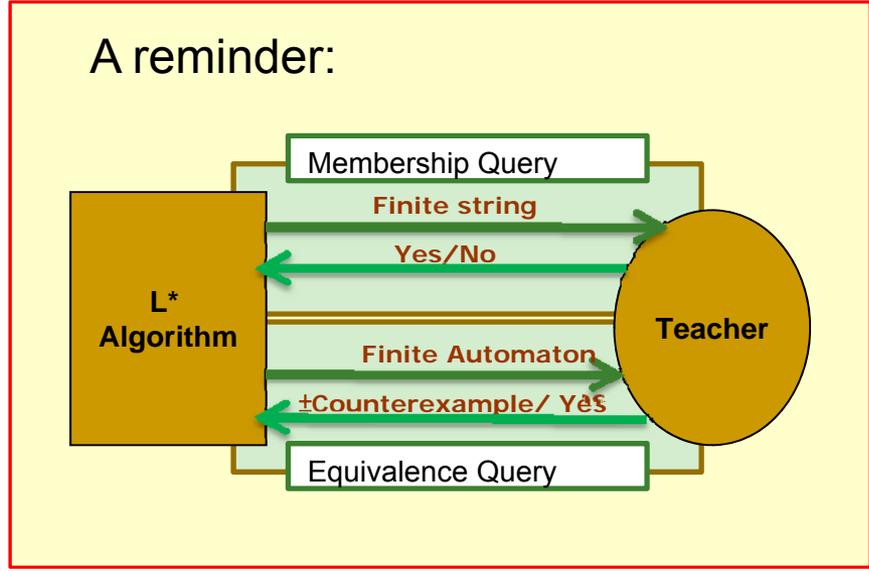
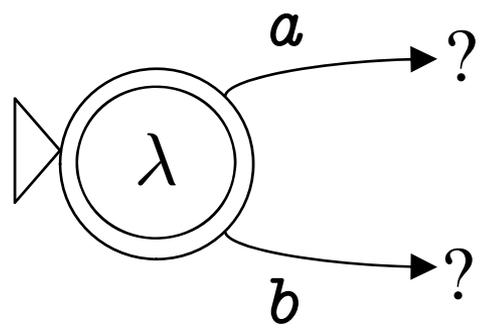
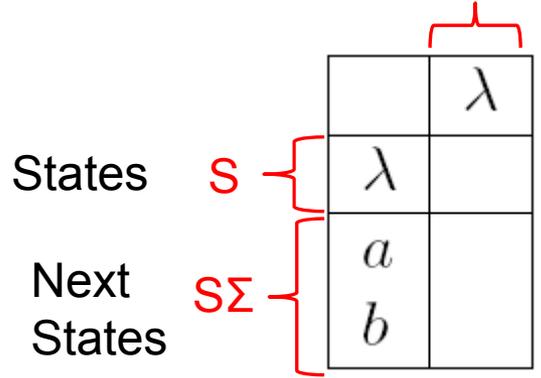
# The L\* Learning Algorithm

- Proposed by D. Angluin [Info.&Comp. 1987] and improved by Rivest and Schapire [Info.&Comp. 1993]



# L\*: Initial Setting

E: Distinguishing Experiments



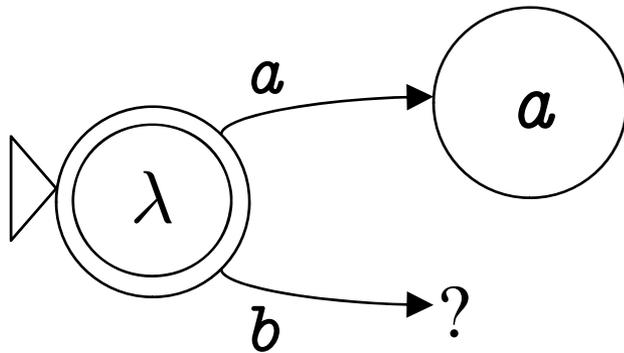
Target:  $(ab+aab)^*$

# L\*: Fill Up the Table by Membership Queries

	$\lambda$
$\lambda$	T
$a$	F
$b$	F

Fill up the table using **membership queries**.

$a$  represents a new equivalence class, because its **row** is different from all of those in the current S set.



Target:  $(ab+aab)^*$

# L\*: Table Expansion

Move  $a$  to the S set and expand the table with elements  $aa$  and  $ab$ .

	$\lambda$
$\lambda$	T
$a$	F
$b$	F
$aa$	
$ab$	

Target:  $(ab+aab)^*$

# $L^*$ : A Closed Table

	$\lambda$
$\lambda$	T
$a$	F
$b$	F
$aa$	F
$ab$	T

Again, fill up the table using membership queries.

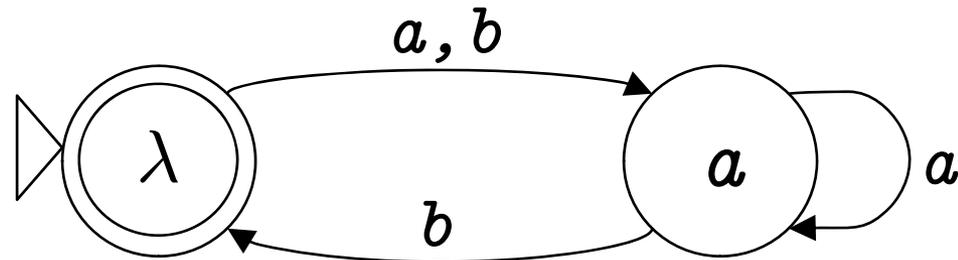
We say that the table is **closed** because every row in the  $S\Sigma$  set appears somewhere in the  $S$  set.

Target:  $(ab+aab)^*$

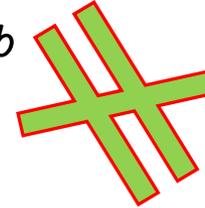
# L\*: Making a Conjecture

	$\lambda$
$\lambda$	T
$a$	F
$b$	F
$aa$	F
$ab$	T

Construct a DFA from the learned equivalence classes.



Counterexample:  $bb$



$\delta(s, a) = s'$  iff  $sa$  and  $s'$  have the same row.

A suffix  $b$  is extracted from  $bb$  as a valid distinguishing experiment

Target:  $(ab+aab)^*$

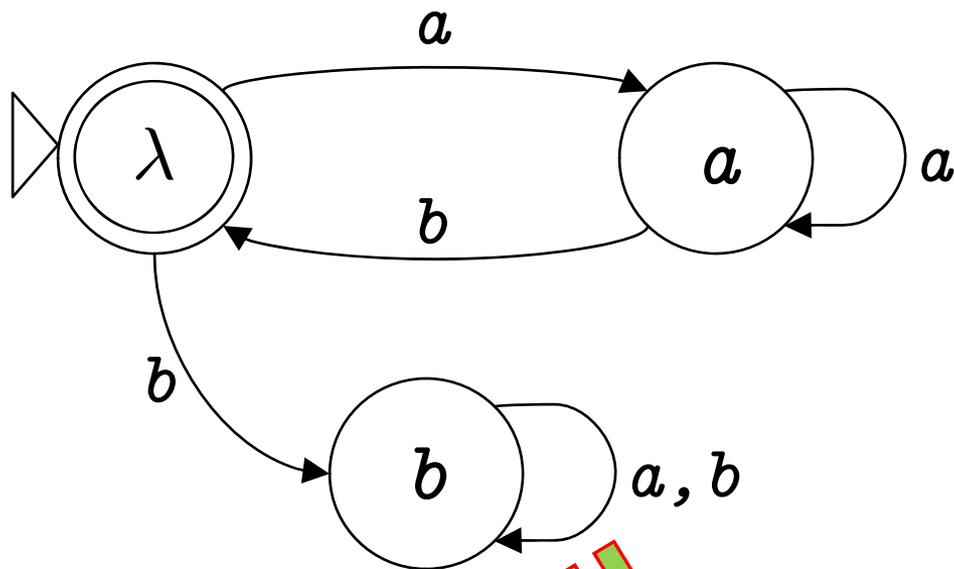
## Theorem:

At least one suffix of the counterexample is a valid distinguishing experiment.

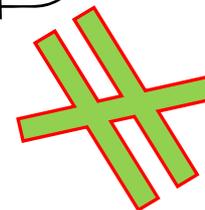
# L\*: 2<sup>nd</sup> Iteration

Add  $b$  to the E set, fill up and expand the table following the same procedure.

	$\lambda$	$b$
$\lambda$	T	F
$a$	F	T
$b$	F	F
$aa$	F	T
$ab$	T	F
$ba$	F	F
$bb$	F	F



Counterexample:  $aaab$



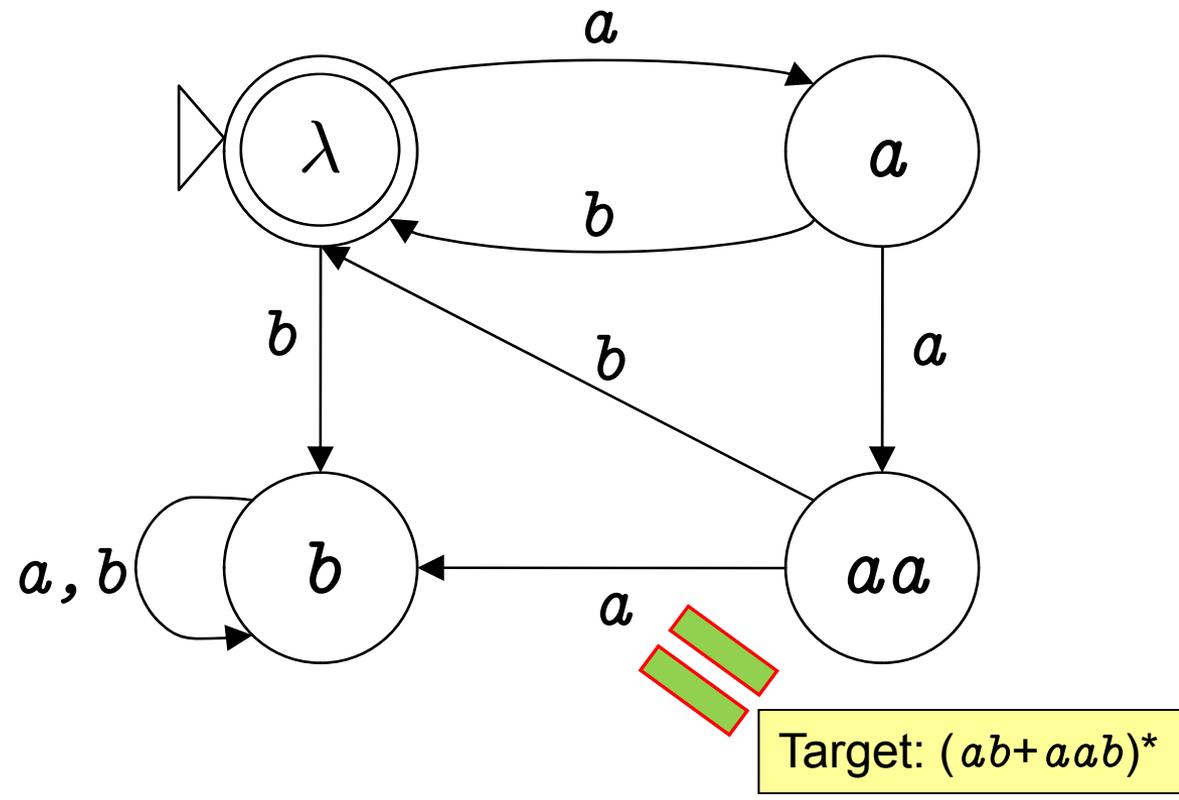
A suffix  $ab$  is extracted from  $aaab$  as a valid distinguishing experiment.

Target:  $(ab+aab)^*$

# L\*: 3<sup>rd</sup> Iteration (Completed)

Add *ab* to the E set, fill up and expand the table following the same procedure.

	$\lambda$	$b$	$ab$
$\lambda$	T	F	T
$a$	F	T	T
$b$	F	F	F
$aa$	F	T	F
$ab$	T	F	T
$ba$	F	F	F
$bb$	F	F	F
$aaa$	F	F	F
$aab$	T	F	T



**Theorem:**  
 The DFA produced by L\* is the minimal DFA that recognizes that target language.

# L\*: Complexity

## ■ Complexity:

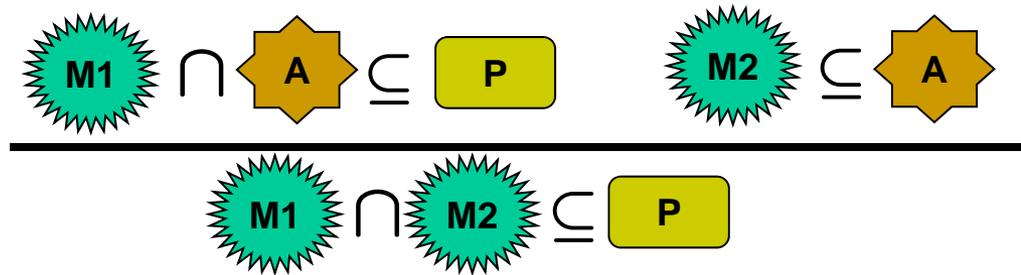
- Equivalence query: at most  $n$
- Membership query:  $O(|\Sigma|n^2 + n \log m)$

	$\lambda$	$b$	$ab$
$\lambda$	T	F	T
$a$	F	T	T
$b$	F	F	F
$aa$	F	T	F
$ab$	T	F	T
$ba$	F	F	F
$bb$	F	F	F
$aaa$	F	F	F
$aab$	T	F	T

Note:  $n$  is the size of the minimal DFA that recognizes  $U$ ,  $m$  is the length of the longest counterexample returned from the teacher.

# The Problem

- The  $L^*$ -based approaches cannot guarantee finding the **minimal assumption** (in size), even if there exists one.



- The smaller the size of  $A$  is, the easier it is to check the correctness of the two premises.
- $L^*$  targets a single language, however, there exists a range of languages that satisfy the premises of an A-G rule.

# Finding a Minimal Assumption

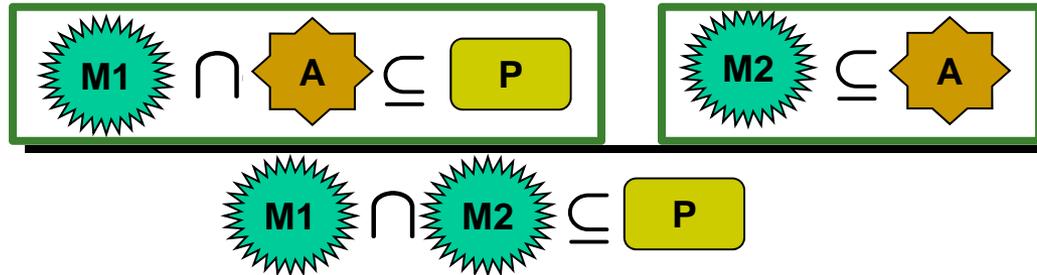
- **A reminder:** we use the following Assume-Guarantee rule for decomposition.

$$M1 \cap A \subseteq P \Leftrightarrow$$

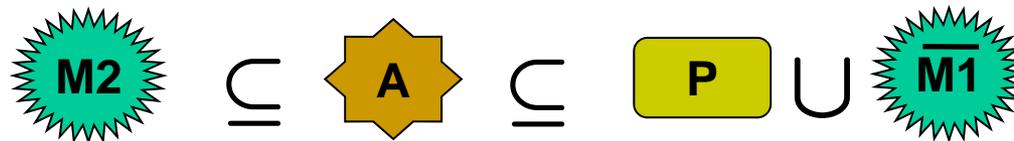
$$M1 \cap A \cap \overline{P} = \emptyset \Leftrightarrow$$

$$A \cap \overline{(P \cup \overline{M1})} = \emptyset \Leftrightarrow$$

$$A \subseteq P \cup \overline{M1}$$



- The two **premises** can be rewritten as follows:



# Finding a Minimal Assumption (cont.)

- To apply the A-G rule is to find an  satisfying the following constraint:

$$\text{M2} \subseteq \text{A} \subseteq \text{P} \cup \overline{\text{M1}}$$

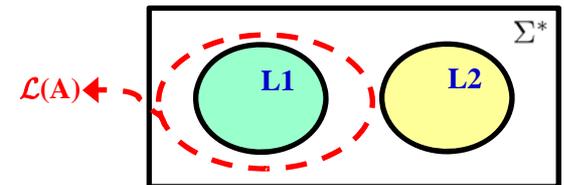
- So, the problem of finding a minimal assumption for the A-G rule reduces to **finding a minimal separating DFA** that
  - **accepts** every string in **M2** and
  - **rejects** every string not in **P**  $\cup$   $\overline{\text{M1}}$ .

First observed by Gupta, McMillan, and Fu

# Learning a Minimal Separating DFA

- **Contribution of [Chen et al. TACAS 2009]:** a **polynomial-query** learning algorithm,  $L^{\text{Sep}}$ , for minimal separating DFA's.
- **Problem:** given two disjoint regular languages **L1** and **L2**, we want to find a minimal DFA **A** that satisfies

$$L1 \subseteq \mathcal{L}(A) \subseteq \overline{L2}$$

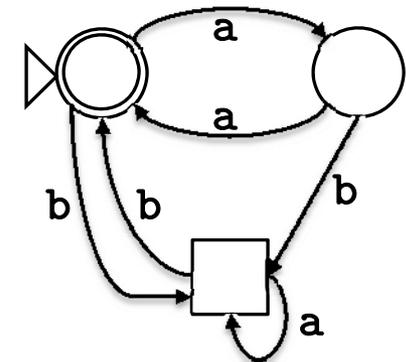


We say that **A** is a **separating DFA** for L1 and L2

- **Assumption:** a teacher for L1 and L2:
  - Membership query: if a string **s** is in L1 (resp. L2)
  - Containment query:  $? \subseteq L1$ ,  $? \supseteq L1$ ,  $? \subseteq L2$ , and  $? \supseteq L2$

# 3-Value DFA (3DFA)

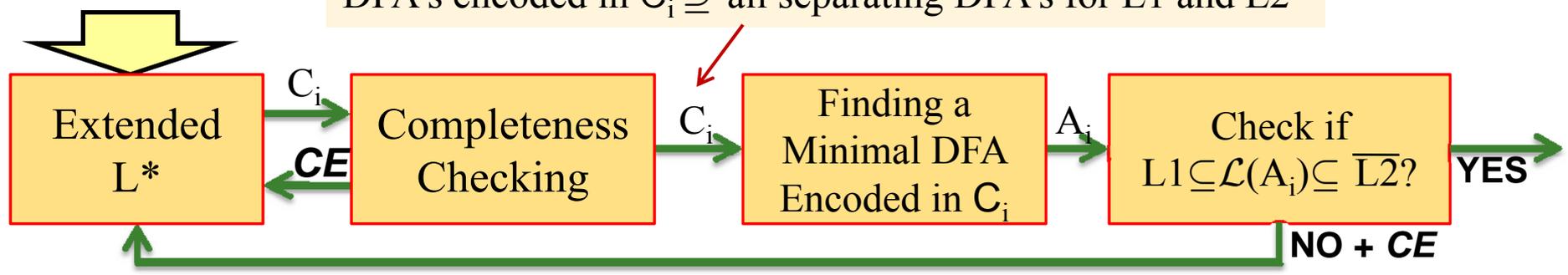
- A 3DFA is a tuple  $C = (\Sigma, S, s_0, \delta, Acc, Rej, Dont)$ .
- A **DFA A** is **encoded in** a **3DFA C** iff **A**
  - accepts all strings that **C** accepts and
  - rejects all strings that **C** rejects.
  - A don't care string in **C** can be either accepted or rejected by **A**.



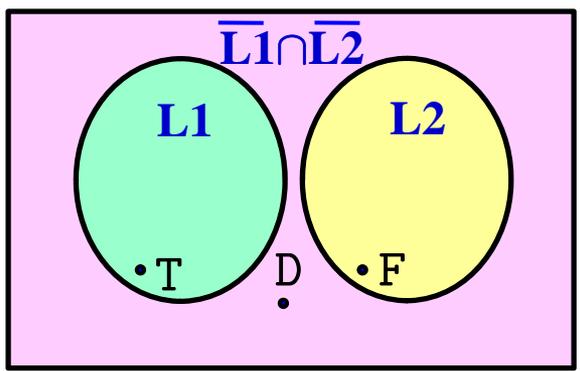
An example of a 3DFA

# The $L^{Sep}$ Algorithm: Overview

DFA's encoded in  $C_i \supseteq$  all separating DFA's for  $L1$  and  $L2$



Target:



	$\lambda$	$a$
$\lambda$	<b>T</b>	<b>F</b>
$a$	F	T
$ab$	<b>D</b>	D
$b$	D	D
$aa$	T	F
$aba$	D	D
$abb$	T	F

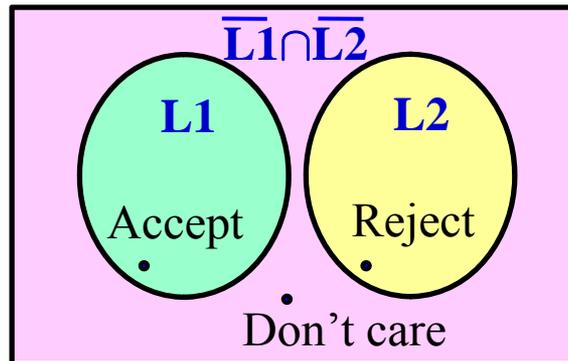
Extend the  $L^*$  algorithm to allow **don't care** values.

# The Target 3DFA

## ■ The target 3DFA $C$

- **accepts** every string in  $L1$ , and
- **rejects** every string in  $L2$ .
- Strings in  $\overline{L1} \cap \overline{L2}$  are **don't care** strings.

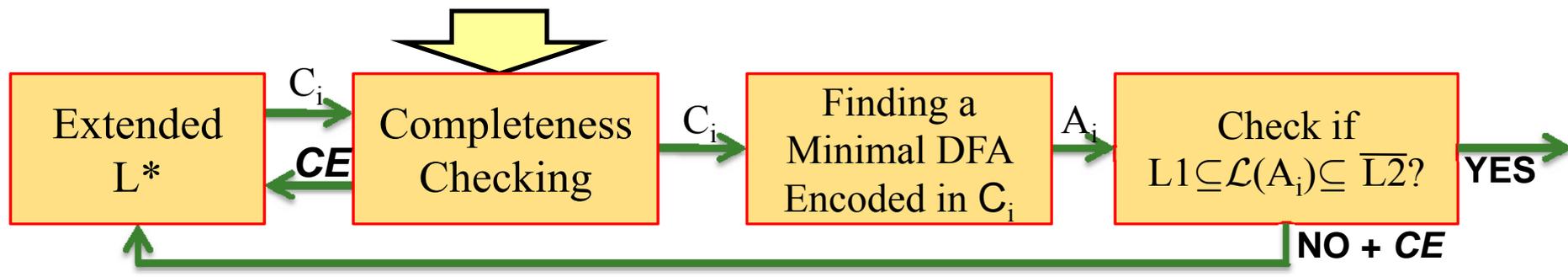
DFA's encoded in  $C$  =  
all separating DFA's for  $L1$  and  $L2$



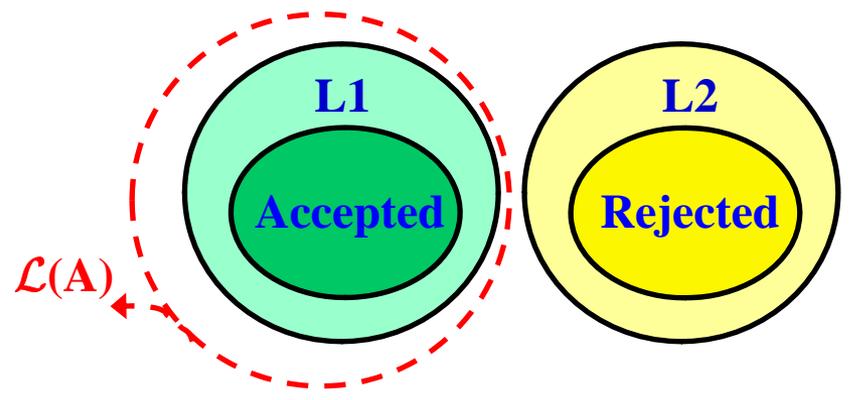
Definition:

- A DFA  $A$  is **encoded in** a 3DFA  $C$  iff  $A$ 
    - accepts all strings that  $C$  accepts and
    - rejects all strings that  $C$  rejects.
  - A DFA  $A$  **separates**  $L1$  and  $L2$  iff  $A$ 
    - accepts all strings in  $L1$  and
    - rejects all strings in  $L2$ .
- A minimal DFA encoded in  $C$  is a minimal separating DFA of  $L1$  and  $L2$ .

# The $L^{Sep}$ Algorithm

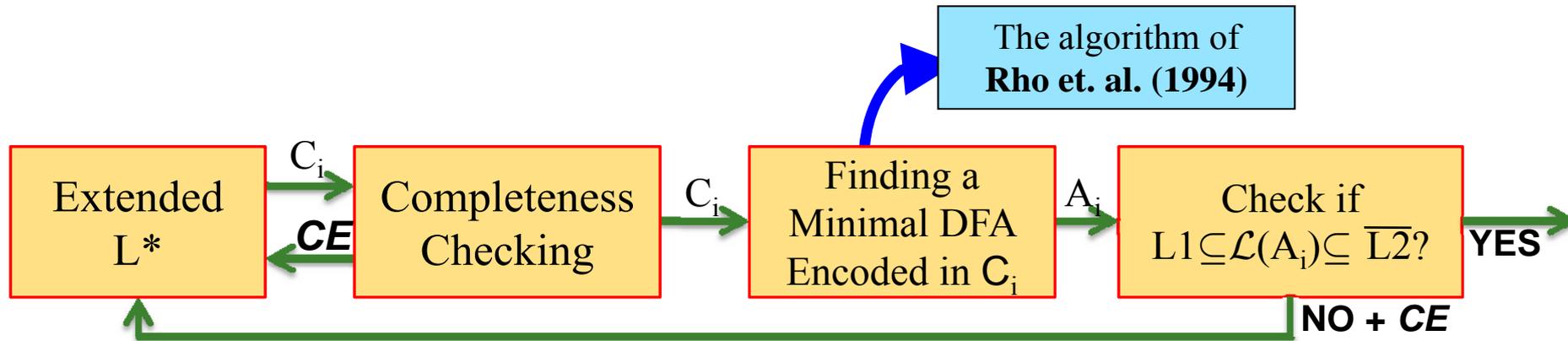


Check if all of the **separating** DFA's of  $L1$  and  $L2$  are **encoded** in  $C_i$ , which can be done by checking the following conditions:



- Definition:
- A **DFA A** is **encoded in** a **3DFA C** iff **A**
    - accepts all strings that **C** accepts and
    - rejects all strings that **C** rejects.
  - A **DFA A** **separates L1** and **L2** iff **A**
    - accepts all strings in **L1** and
    - rejects all strings in **L2**.

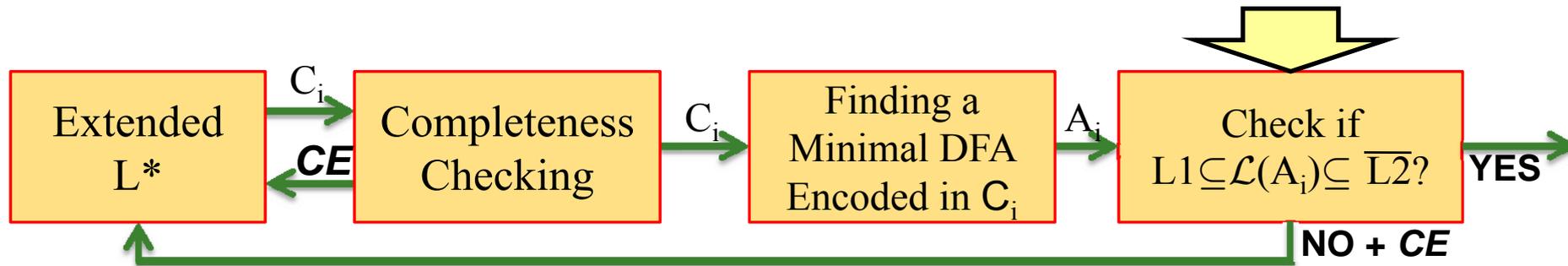
# The $L^{\text{Sep}}$ Algorithm



## LEMMA:

The size of **minimal separating DFA** of  $L_1$  and  $L_2 \geq |A_i|$ , the size of the **minimal DFA encoded in  $C_i$** .

# The $L^{\text{Sep}}$ Algorithm



If  $L1 \subseteq \mathcal{L}(A_i) \subseteq \overline{L2}$ :

$A_i$  is a minimal separating DFA.

If  $L1 \not\subseteq \mathcal{L}(A_i)$  or  $\mathcal{L}(A_i) \not\subseteq \overline{L2}$ :

Counterexample CE is a witness for  $C_i$  not being the target 3DFA.

## LEMMA:

The size of **minimal separating DFA** of  $L1$  and  $L2 \geq |A_i|$ , the size of the **minimal DFA encoded in  $C_i$** .

# The Algorithm of Gupta *et al.*

Begin with an empty **sample set**

Requires an **exponential** number of iterations in the worst case

Make a **minimal DFA** that consistent with the current **sample set**

Check if  $L1 \subseteq \mathcal{L}(A_i) \subseteq \overline{L2}$ ?

**YES**

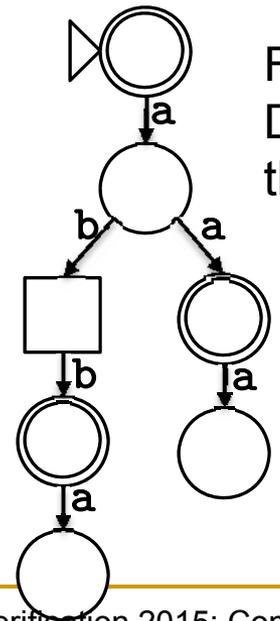
**NO + CE**  
Add **CE** to the sample set

## An Example

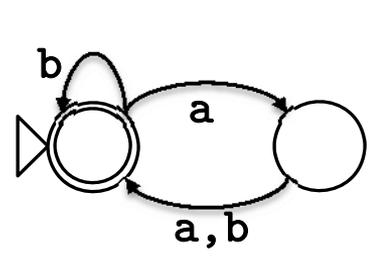
+ SAMPLES :  
 $\lambda, aa, abb$

- SAMPLES :  
 $a, aaa, abba$

Make a 3DFA



Find a minimal DFA encoded in the 3DFA (NP-hard)



# The $L^{Sep}$ Algorithm

Extend the  $L^*$  algorithm to manage the collected samples.

$A_i$

Check if  $L1 \subseteq \mathcal{L}(A_i) \subseteq \overline{L2}$ ?

YES

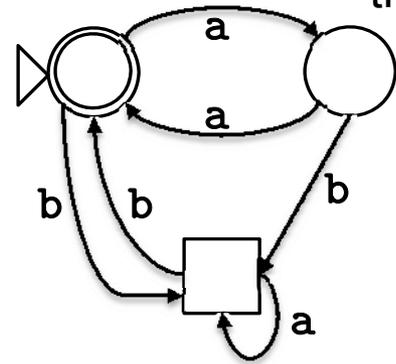
NO + CE

Requires a **polynomial** number of iterations in the worst case

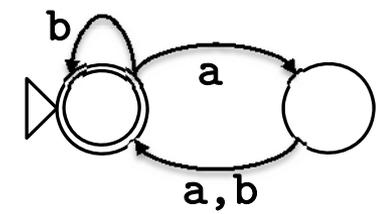
## An Example

	$\lambda$	$a$
$\lambda$	T	F
$a$	F	T
$ab$	D	D
$b$	D	D
$aa$	T	F
$aba$	D	D
$abb$	T	F

Make a 3DFA



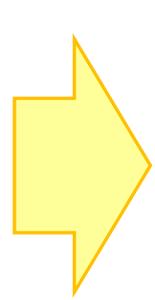
Find a minimal DFA encoded in the 3DFA (NP-hard)



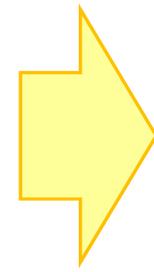
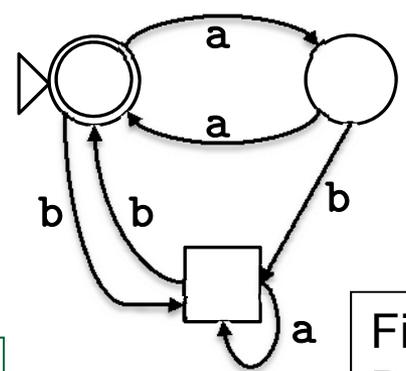
# Comparing the Two Algorithms

**L<sub>Sep</sub>:**

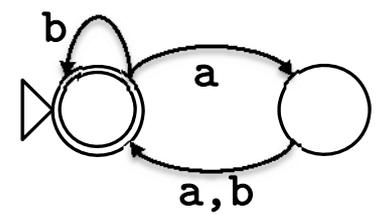
	$\lambda$	$a$
$\lambda$	T	F
$a$	F	T
$ab$	D	D
$b$	D	D
$aa$	T	F
$aba$	D	D
$abb$	T	F



Make a 3DFA



Find a minimal DFA encoded in the 3DFA (NP-hard)



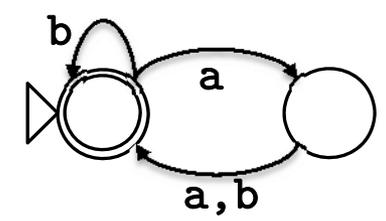
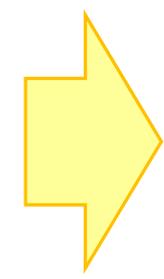
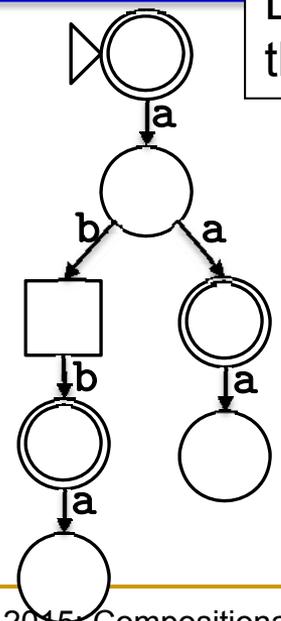
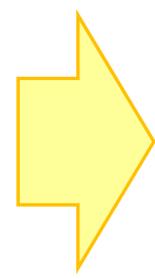
Same sample set!



**Gupta et al. :**

+ SAMPLES :  
 $\lambda, aa, abb$

- SAMPLES :  
 $a, aaa, abba$

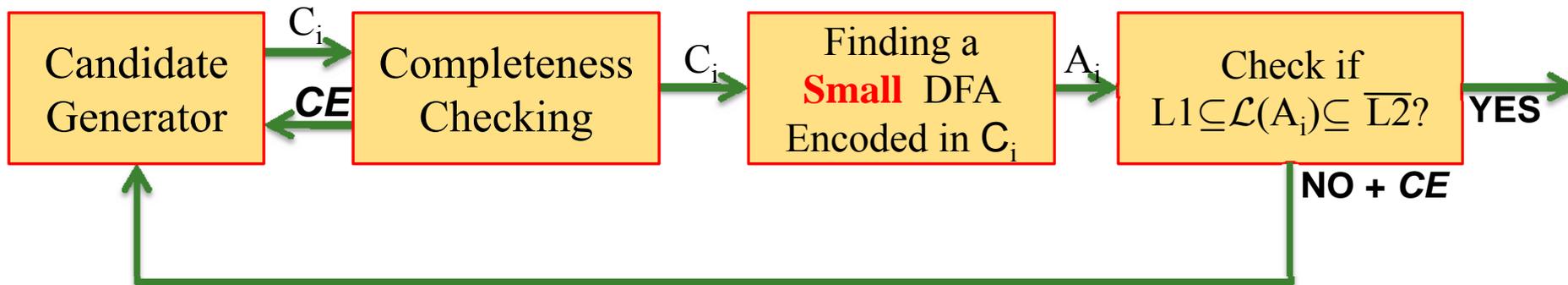


# Adapt $L^{\text{Sep}}$ for Compositional Verification

- Let  $L1 = M2$  and  $\bar{L}2 = P \cup \bar{M}1$ , use  $L^{\text{Sep}}$  to find a separating DFA for  $L1$  and  $L2$ .
- When  $M2 \not\subseteq P \cup \bar{M}1$  (i.e.,  $M1 \cap M2 \not\subseteq P$ ),  $L^{\text{Sep}}$  can be modified to guarantee finding a string in  $M2$ , but not in  $P \cup \bar{M}1$  (i.e.,  $M1 \cap M2 \setminus P$ ).

# Adapt $L^{\text{Sep}}$ for Compositional Verification

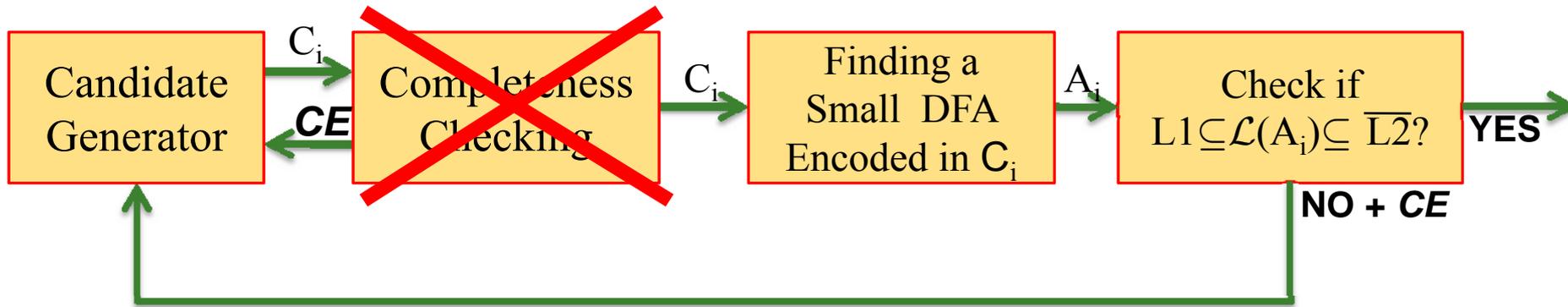
- Use heuristics to find a small consistent DFA:



Minimality is no longer guaranteed!

# Adapt $L^{\text{Sep}}$ for Compositional Verification

- Skip completeness checking:



Minimality is no longer guaranteed!