Compositional Model Checking

Orna Grumberg
Technion, Israel

Marktoberdorf 2015

Thanks to Sharon Shoham and Yael Meller
for their help with the presentation

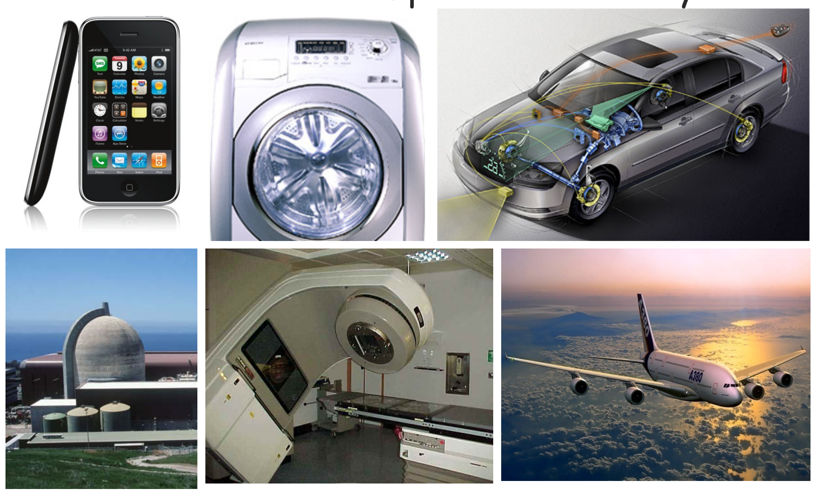
1

Lecture 1

2

Why (formal) Verification?

Computers are everywhere



Why (formal) verification?

- safety-critical applications: Bugs are unacceptable!
  - Air-traffic controllers
  - Medical equipment
  - Cars
- Bugs found in later stages of design are expensive
- Hardware and software systems grow in size and complexity: Subtle errors are hard to find by testing
- Pressure to reduce time-to-market

Automated tools for formal verification are needed

Formal Verification

Given

- a model of a (hardware or software) system and
- a formal specification

does the system model satisfy the specification?

Not decidable!

To enable automation, we restrict the problem to a decidable one:

- **Finite-state** reactive systems
- **Propositional** temporal logics

5

Finite state systems - examples

- Hardware designs
- Controllers (elevator, traffic-light)
- Communication protocols (when ignoring the message content)
- High level (abstracted) description of non finite state systems

6

Properties in propositional temporal logic - examples

- **mutual exclusion:**
  always $\neg(cs_1 \wedge cs_2)$
- **non starvation:**
  always (request $\Rightarrow$ **eventually** granted)
- **communication protocols:**
  ($\neg$ get-message) **until** send-message

7

Model Checking [CE81,QS82]

An efficient procedure that receives:

- A finite-state model describing a system
- A temporal logic formula describing a property

It returns

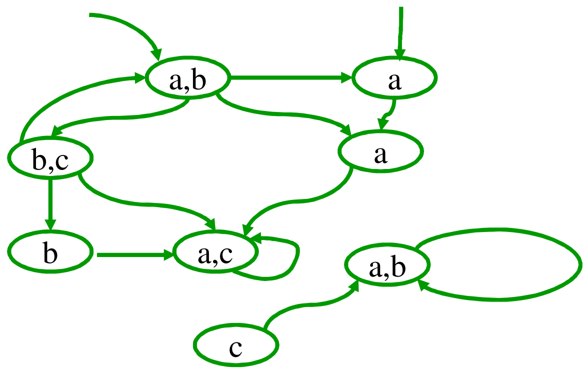
yes, if the system has the property

no + Counterexample, otherwise

8

Model of a system

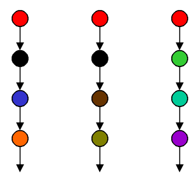
Kripke structure / transition system



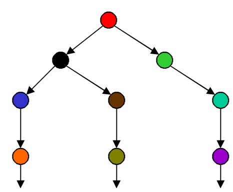
Temporal Logics

- Temporal Logics
  - Express properties of event orderings in time

- Linear Time
  - Every moment has a unique successor
  - Infinite sequences (words)
  - Linear Time Temporal Logic (LTL)



- Branching Time
  - Every moment has several successors
  - Infinite tree
  - Computation Tree Logic (CTL)



Propositional temporal logic

AP – a set of atomic propositions

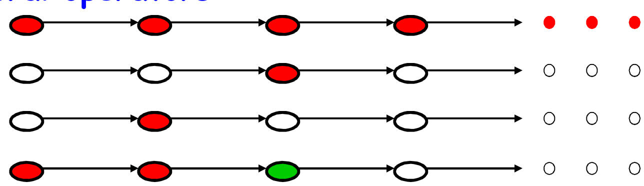
Temporal operators:

Gp

Fp

Xp

pUq



Path quantifiers: **A** for **all** path

E there **exists** a path

11

CTL formulas: Example

- **mutual exclusion:** $AG \neg(cs_1 \wedge cs_2)$
- **non starvation:** $AG($ request $\Rightarrow AF$ grant$)$
- **“sanity” check:** EF request

12

Example to demonstrate:

- Building a model from a program
- Properties
- Model checking

13

Mutual Exclusion Example

- Two processes with a joint Boolean signal sem
- Each process P_i has a variable v_i describing its state:
  - $v_i = N$ Non critical
  - $v_i = T$ Trying
  - $v_i = C$ Critical

Mutual Exclusion Example

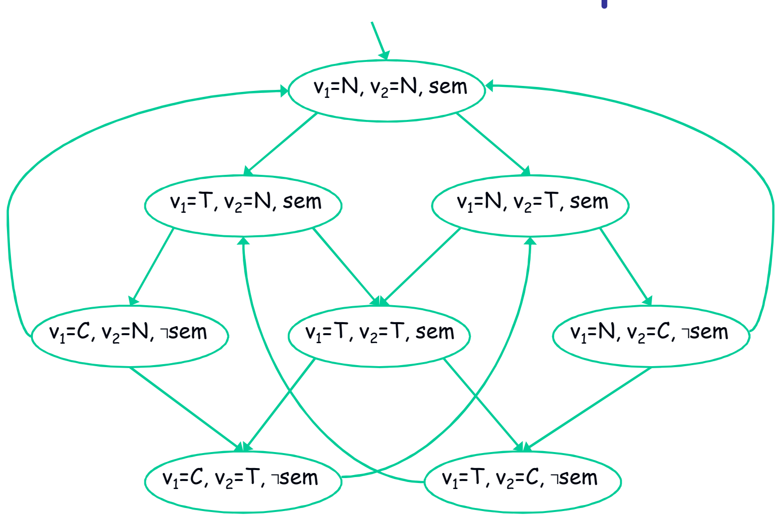
- Each process runs the following program:

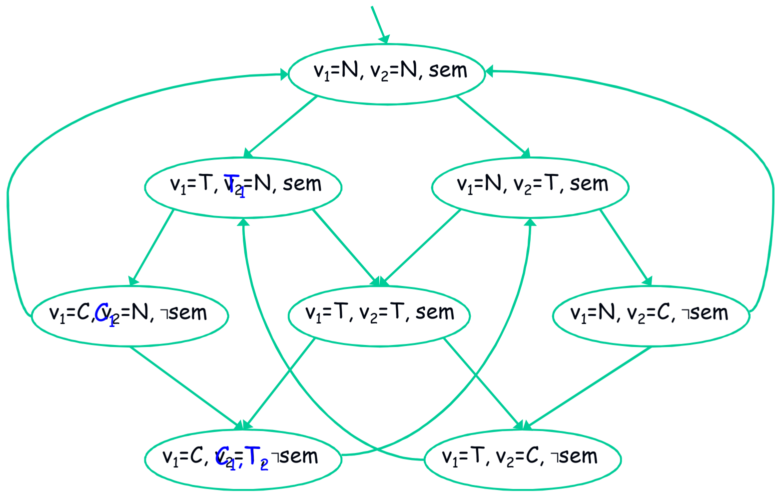
```
Pi :: while (true) {
          if (vi == N)  vi = T;
          else if (vi == T && sem)  { vi = C; sem = 0; }
          else if (vi == C)  {vi = N; sem = 1; }
      }
```

Atomic action

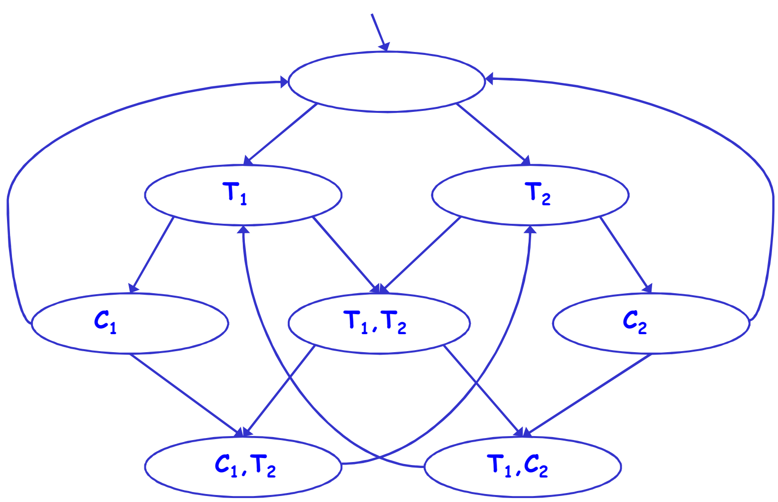
- The full program is: $P_1||P_2$
- Initial state: (v_1=N, v_2=N, sem)
- The execution is interleaving

Mutual Exclusion Example

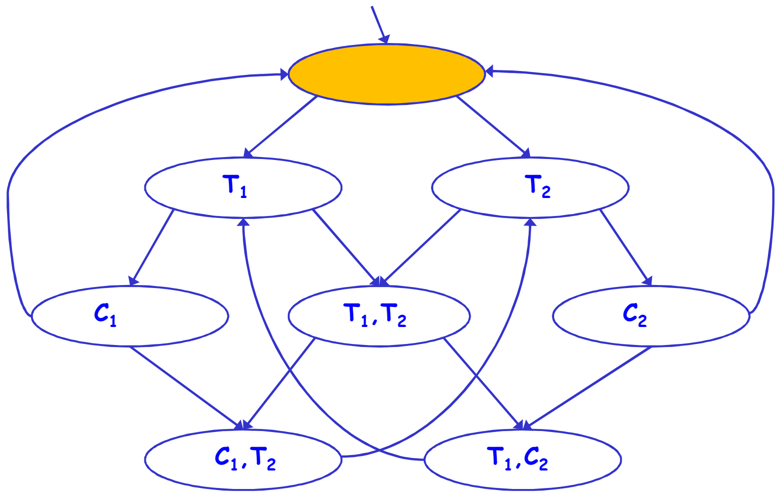




- We define atomic propositions: $AP=\{C_1,C_2,T_1,T_2)$
- A state is marked with T_i if v_i=T
- A state is marked with C_i if v_i=C

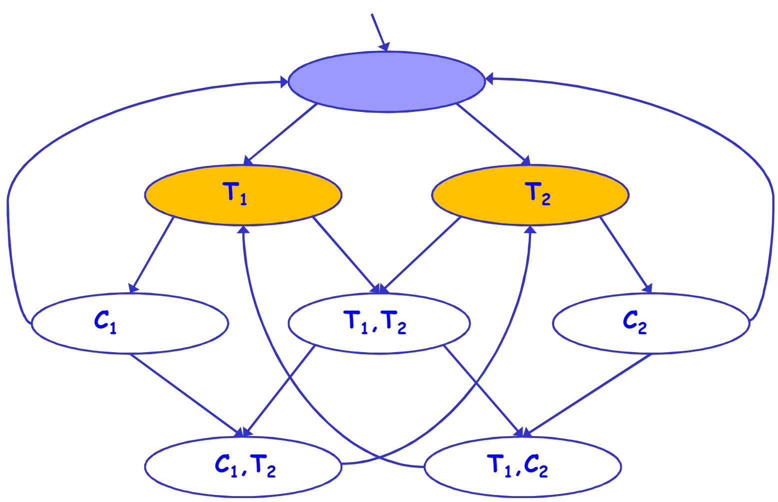


- Property 1: $\boldsymbol{AG}\neg(C_1 \wedge C_2)$



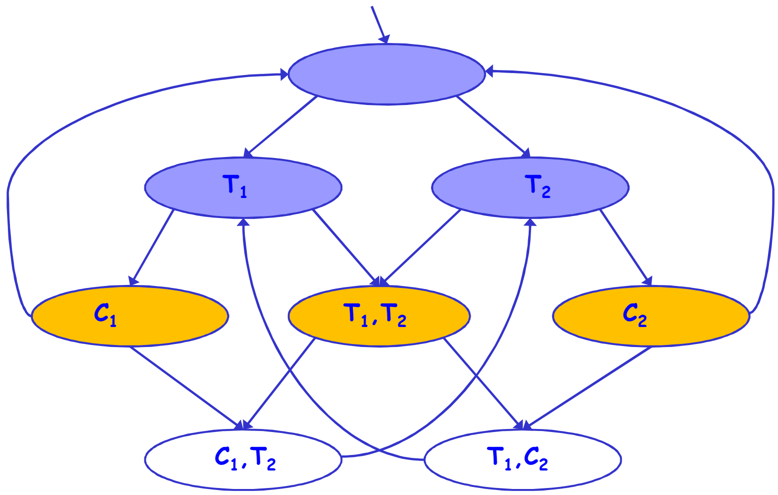
- Property 1: $\boldsymbol{AG}\neg(C_1 \wedge C_2)$

S_0



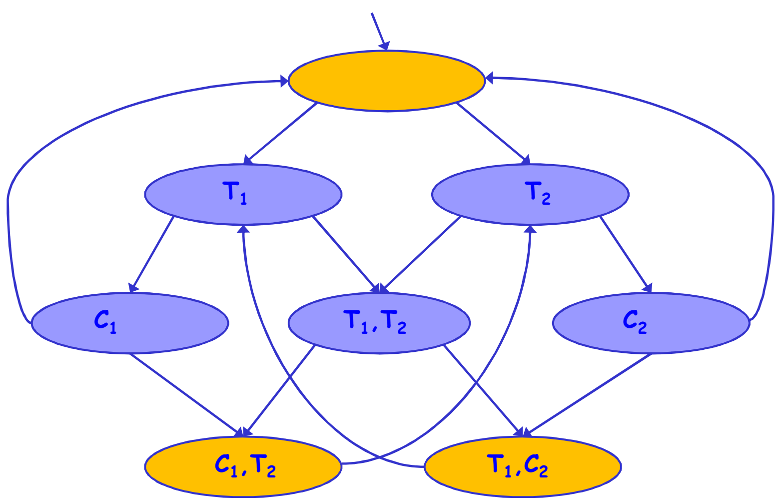
- Property 1: $\boldsymbol{AG}\neg(C_1 \wedge C_2)$

S_1



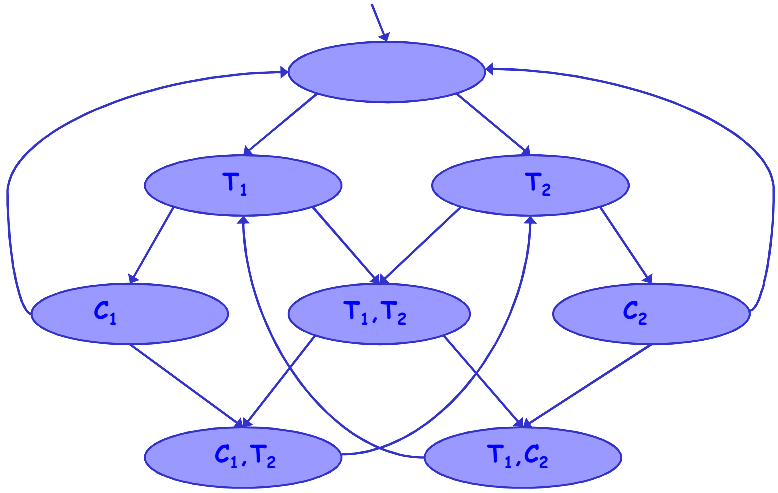
- Property 1: $\boldsymbol{AG}\neg(C_1 \wedge C_2)$

S_2



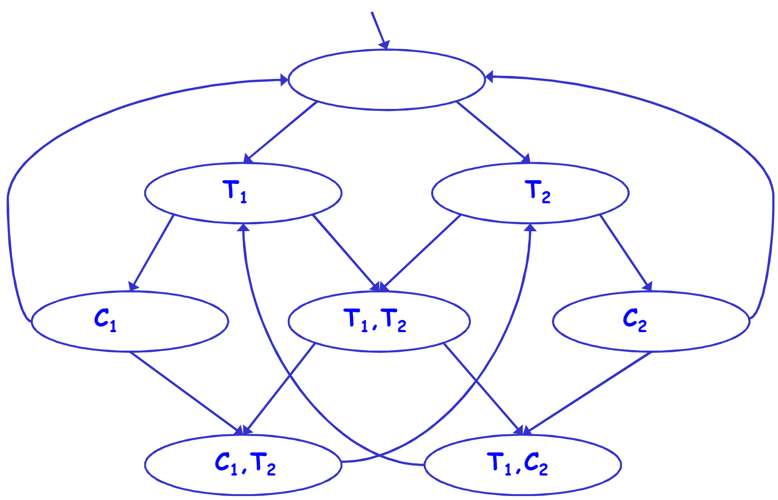
- Property 1: $\boldsymbol{AG}\neg(C_1 \wedge C_2)$

S_3

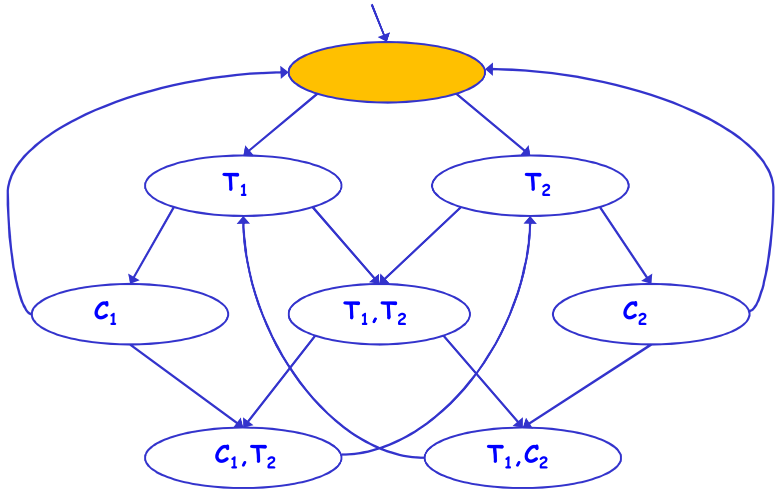


- $M \models AG \neg (C_1 \wedge C_2)$

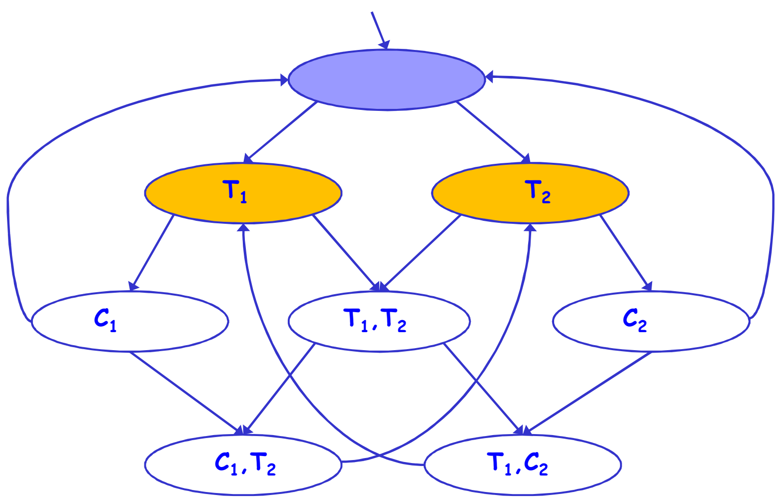
$S_4 \subseteq S_0 \cup S_1 \cup S_2 \cup S_3$



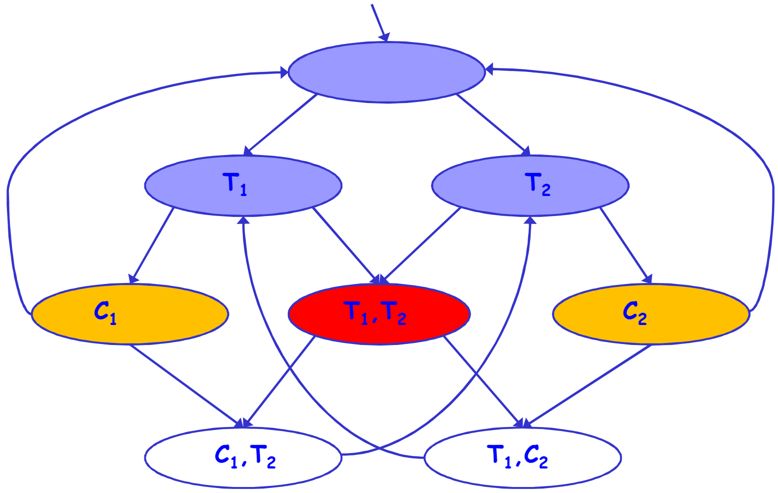
- Property 2: $AG\neg(T_1 \wedge T_2)$



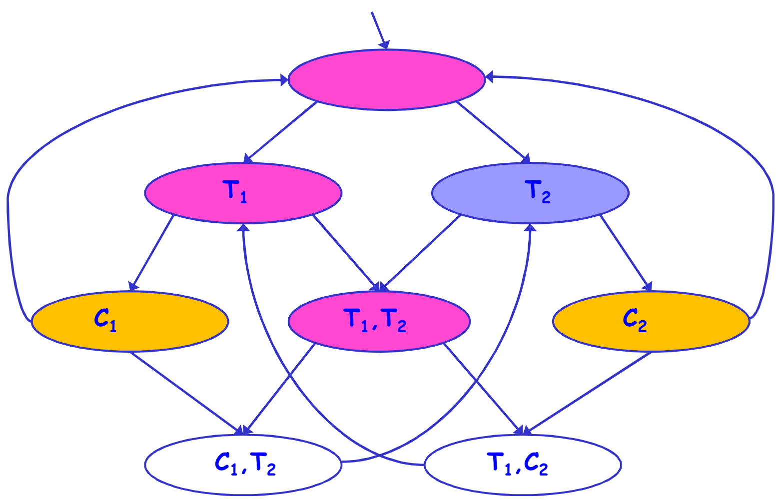
- Property 2: $AG\neg(T_1 \wedge T_2)$



- Property 2: $AG\neg(T_1 \wedge T_2)$



- $M \not\models AG \neg (T_1 \wedge T_2)$
- *A violating state has been found*



- $M \not\models AG \neg (T_1 \wedge T_2)$

Model checker returns a counterexample

Main Limitation of Model Checking:

The state explosion problem

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

29

Learning Assumptions for Compositional Verification

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

TACAS 2003

30

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

$M_1 \| M_2 \models P$

- try to verify each component separately
- usually inapplicable due to dependencies
  - a component is typically designed to satisfy its requirements in **specific environments** (contexts)

Assume-Guarantee (AG) Paradigm

- Introduces **assumptions** representing a component's environment

Instead of: Does **component** satisfy **property**?

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

33

Assume-Guarantee (AG) Paradigm

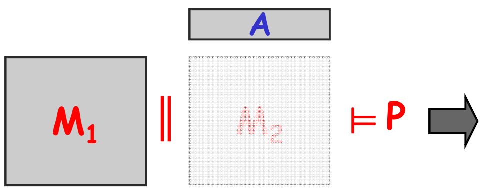
Notation: **<A> M <P>**

<A> M <P> is true if
whenever **M** is part of a system satisfying the **assumption** **A**, then the system also satisfies (**guarantee**) **P**

34

Useful AG Rule

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

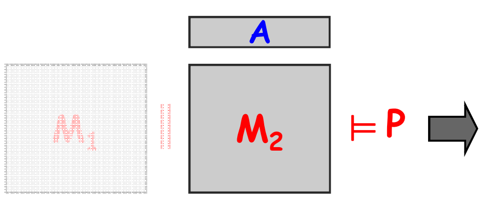


$$\frac{\langle A\rangle\ M_1\ \langle P\rangle}{\langle \text{true}\rangle\ M_1||M_2\ \langle P\rangle}$$

35

Useful AG Rule for Safety Properties

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



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

36

Outline

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

37

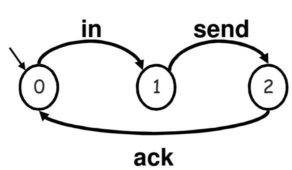
Labeled Transition Systems (LTS)

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

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

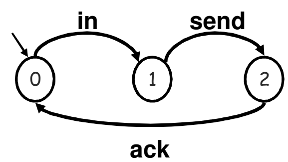
Observable actions

Internal action



38

Labeled Transition Systems (LTS)



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

Trace of an LTS **M** : finite sequence of observable actions that M can perform starting at the **initial state**

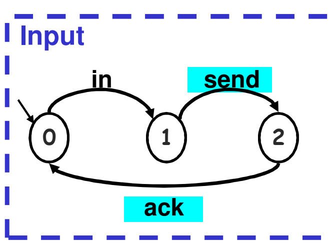
L(M) = the **Language** of **M** : the set of all traces of M

39

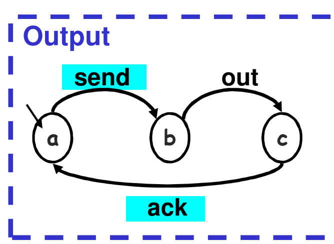
Parallel Composition $M_1 \| M_2$

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

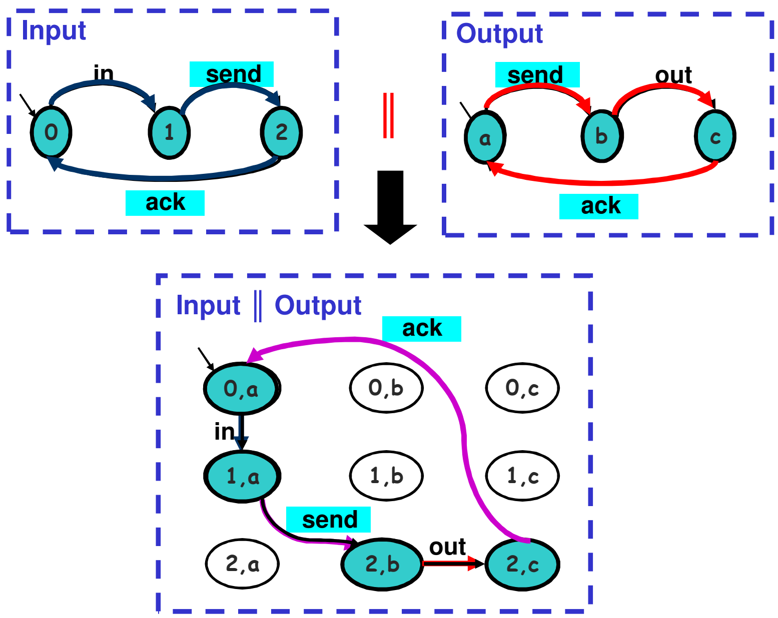
Example:



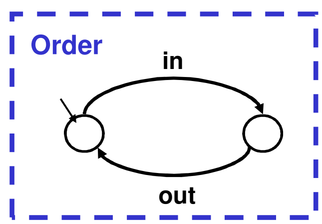
‖



40



Safety Properties



Expressed as deterministic LTSs

For a **safety LTS, P**:

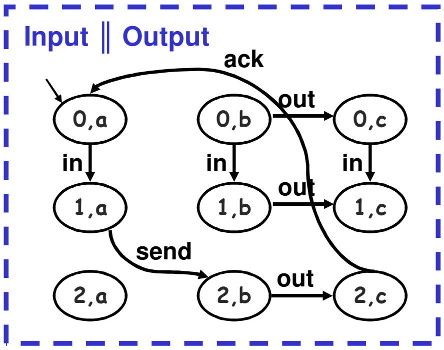
- **L(P)** describes the set of **legal** (acceptable) behaviors over αP

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

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

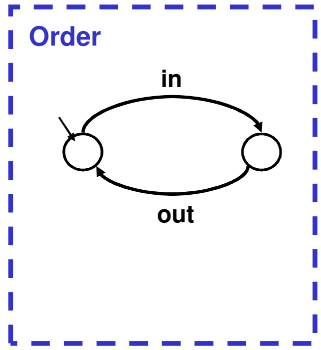
Note that, since we check $M \models P$, $\alpha P \subseteq \alpha M$

Example



$\overset{?}{\models}$

P



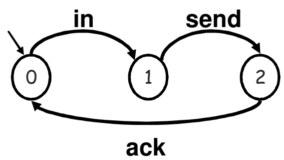
$\mathrm{Pref}(\langle in,send,out,ack\rangle^*)\uparrow\{in,out\} \overset{?}{\subseteq} \mathrm{Pref}(\langle in,out\rangle^*)$ √

43

Lecture 2

44

Labeled Transition Systems (LTS)



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

Trace of an LTS **M** : finite sequence of observable actions that M can perform starting at the **initial state**

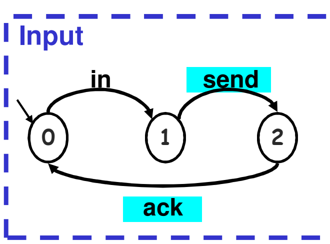
L(M) = the **Language** of **M** : the set of all traces of M

45

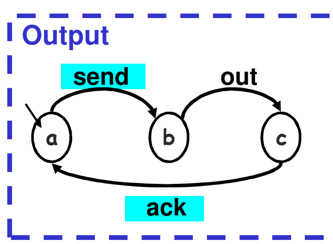
Parallel Composition $M_1 \| M_2$

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

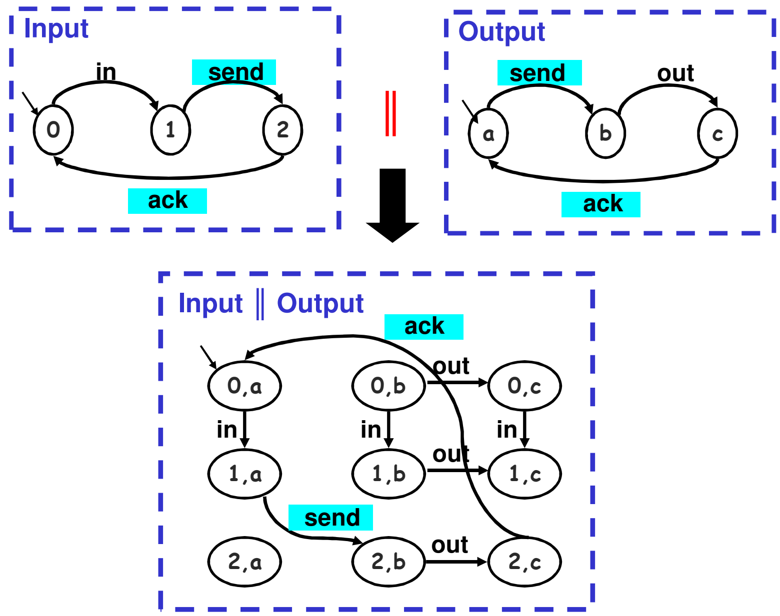
Example:



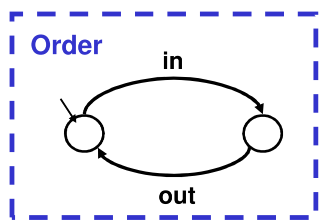
‖



46



Safety Properties



Expressed as deterministic LTSs

For a **safety LTS, P**:

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

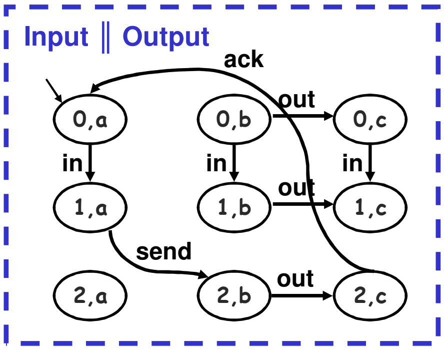
Language Containment
$L(M)\uparrow\alpha P \subseteq L(P)$

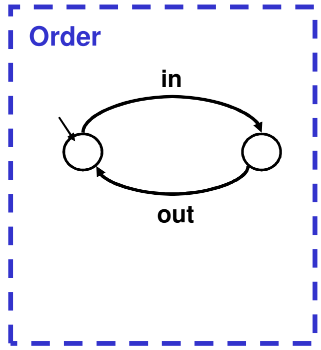
$\mathbf{M} \models \mathbf{P}$ iff $\forall \sigma \in \mathbf{L(M)} : (\sigma \uparrow \alpha \mathbf{P}) \in \mathbf{L(P)}$

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

Example

P





$\overset{?}{\models}$

Pref(<in,send,out,ack>*) $\overset{?}{\subseteq}$ Pref(<in,out>*)

↑{in,out}

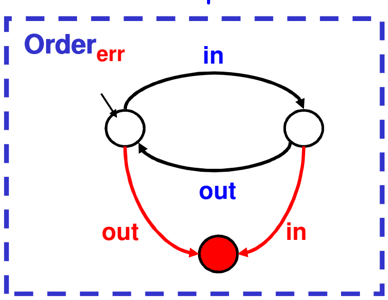
√

49

Model Checking M ⊨ P

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

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



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

50

Model Checking $M \models P$

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

Theorem:

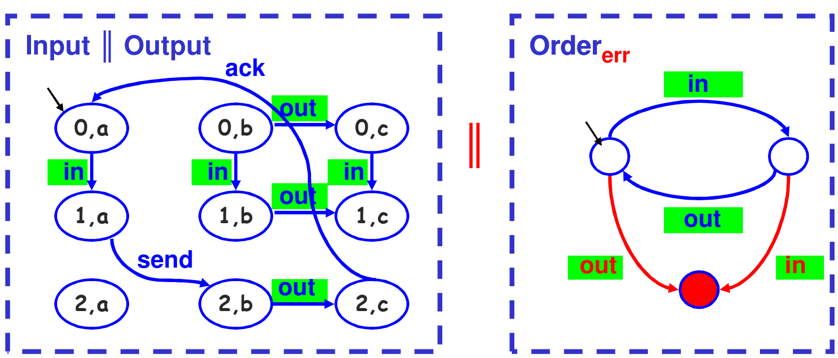
- $M \models P$ iff π is **unreachable** in $M \parallel P_{err}$

Recall that,

- $M \models P$ iff $\forall \sigma \in L(M) : (\sigma \uparrow \alpha P) \in L(P)$
- $M \parallel P_{err}$ synchronizes on αP

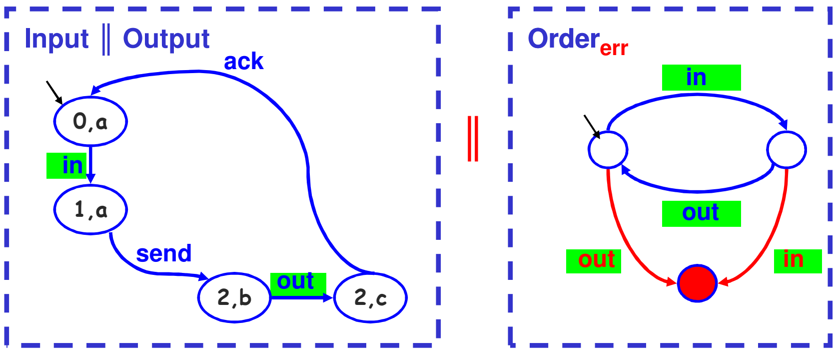
51

Example

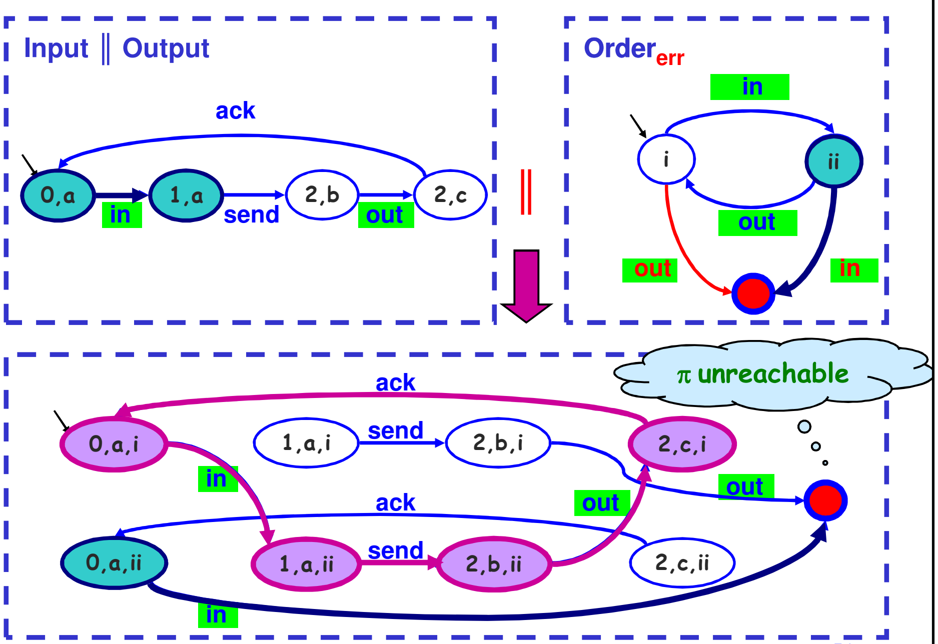


52

Example



53



54

Assume Guarantee Reasoning

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

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

M_1, M_2 : LTSs
P, A : safety LTSs

Outline

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

Important concept: Weakest assumption for M_1, M_2, P

Definition:

The weakest assumption A_w is a deterministic LTS such that:

- $\alpha A_w = \Sigma_I = \alpha M_2 \cap (\alpha P \cup \alpha M_1)$
- For every M'_2 such that $\alpha A_w \subseteq \alpha M'_2$
  $\langle true \rangle\ M_1 || M'_2\ \langle P \rangle$ iff $\langle true \rangle\ M'_2\ \langle A_w \rangle$

57

Important concept: Weakest assumption for M_1, M_2, P

Note that:

- $\langle true \rangle\ M_1 || A_w\ \langle P \rangle$ holds
- A_w describes exactly **all traces** σ over Σ_I such that in the context of σ, M_1 satisfies P

58

Observation

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

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

➔ Instead of finding A_w:

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

59

Given a Candidate Assumption A_i

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

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

Assumption A_i is not tight enough

➔ need to **strengthen** A_i

- We found a trace σ such that "$\sigma \parallel M_1$" $\not\models P$
- σ should be **removed** from A_i

60

Given a Candidate Assumption A_i

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

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

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

Otherwise: $\sigma \in L(M_2)$ but $(\sigma \uparrow \Sigma_I) \notin L(A_i)$ -- **cex**

P violated by M_1 in the context of **cex**=$(\sigma \uparrow \Sigma_I)$?

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

Yes: real violation !

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

No: spurious violation

➔ need to find better approximation of A_w

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

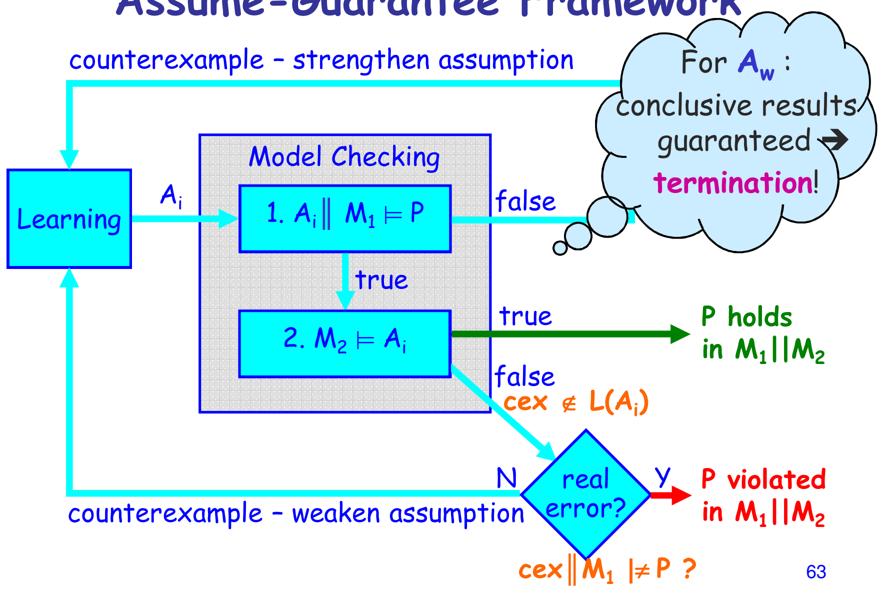
"cex $\in L(A_w)$" ?

- Check if π **reachable** in $A_{cex} \parallel M_1 \parallel P_{err}$

Alternatively,

- **Simulate cex** on $M_1 \parallel P_{err}$
  - **cex** $\parallel M_1 \models P$ **iff** when **cex** is simulated on $M_1 \parallel P_{err}$ it **cannot lead** to π (**error**) state.

Assume-Guarantee Framework



63

Lecture 3

64

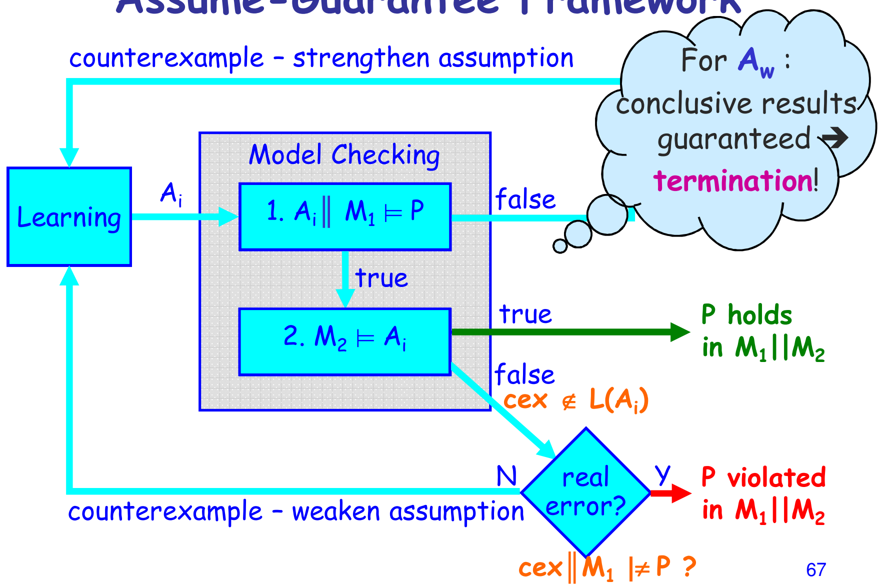
Goal:

Automatically learn an assumption for the Assume Guarantee Rule for $M_1 \| M_2 \models P$

Important concept: Weakest assumption A_w for M_1, M_2, P

- $\langle true \rangle\ M_1 || A_w\ \langle P \rangle$ holds
- A_w describes exactly **all traces** σ over Σ_I such that in the context of σ, **M_1** satisfies **P**

Assume-Guarantee Framework



67

Outline

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

68

Learning Algorithm for DFA – L*

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

DFA M = (Q, q^0, αM, δ, F) :

- **Q, q^0, αM,** δ : as in deterministic LTS
- **F** ⊆ **Q** : accepting states
- **L(M) = {σ | δ(q^0, σ) ∈ F}**

Learning Algorithm for DFA – L*

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

Conjectures C_1, C_2, … converge to **C**

Equivalence Queries

Outline

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

71

Assume-Guarantee with Learning

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

In order to **use L*** to produce assumptions A_i:
- Show that **$L(A_w)$** is **regular**
- Translate **DFA** to **safety LTS** (assumption)
- Implement the **teacher**

72

Implementing the Teacher

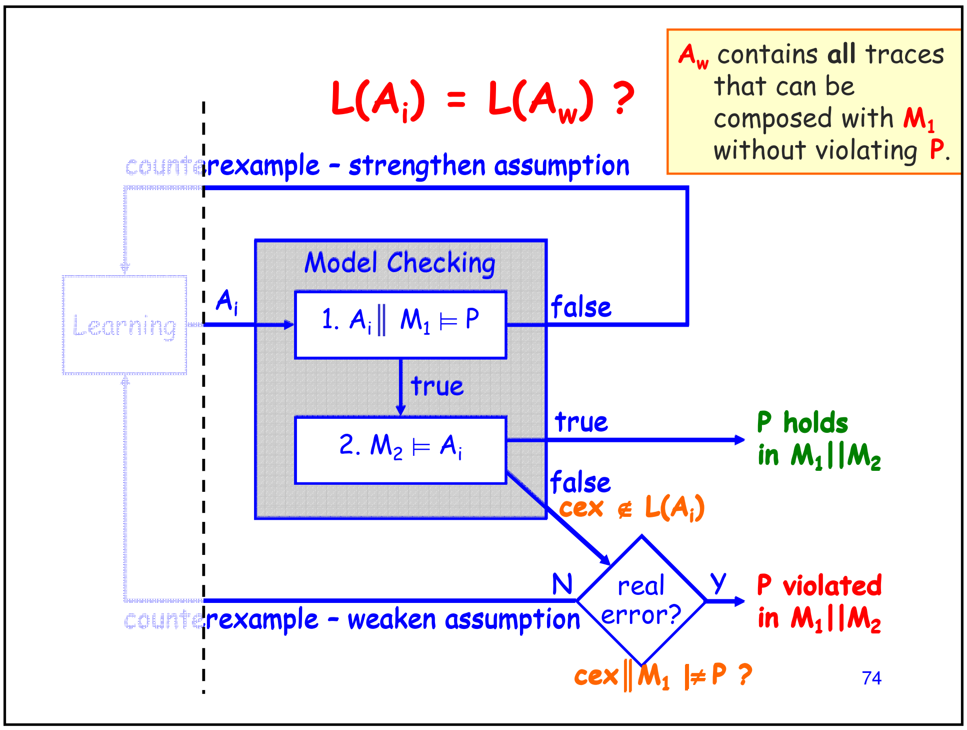
A_w is **unknown**…

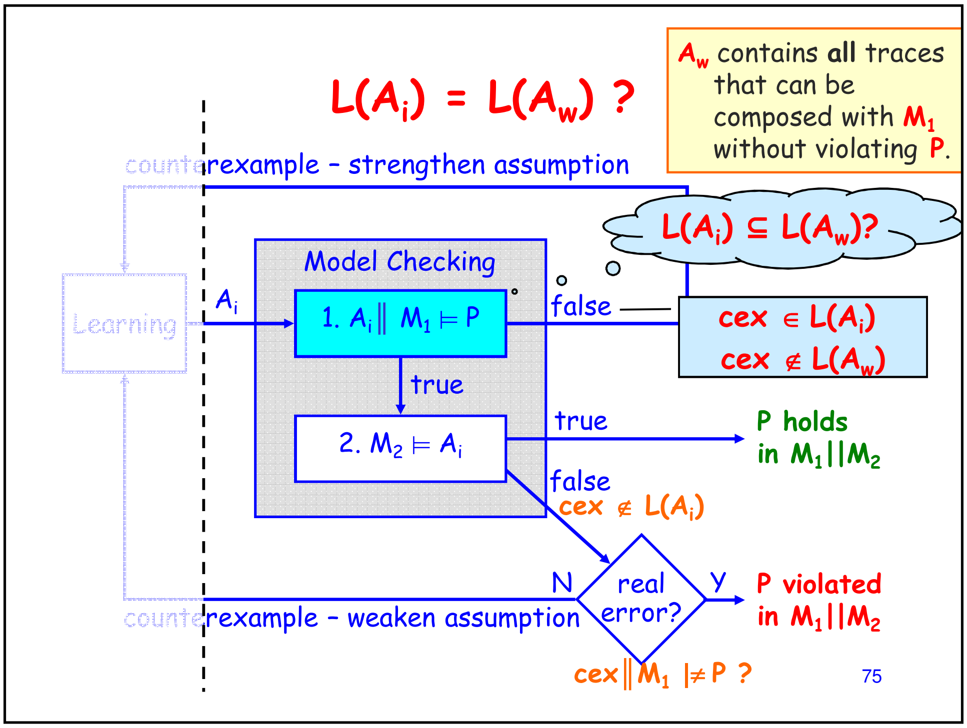
$\sigma \in L(A_w)$ iff in context of σ, M_1 satisfies P

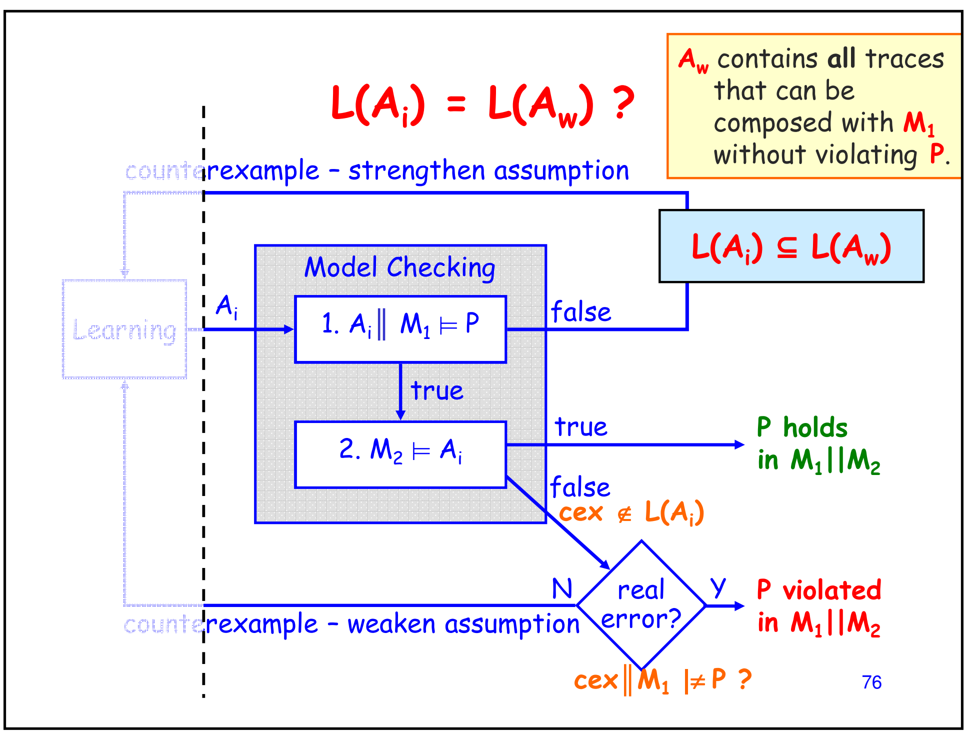
- **Membership** query: $\sigma \in L(A_w)$?
  ➔ Check if "$\sigma \parallel M_1$" $\models P$:
  - **Model checking**: is π **reachable** in $A_\sigma \parallel M_1 \parallel P_{err}$?

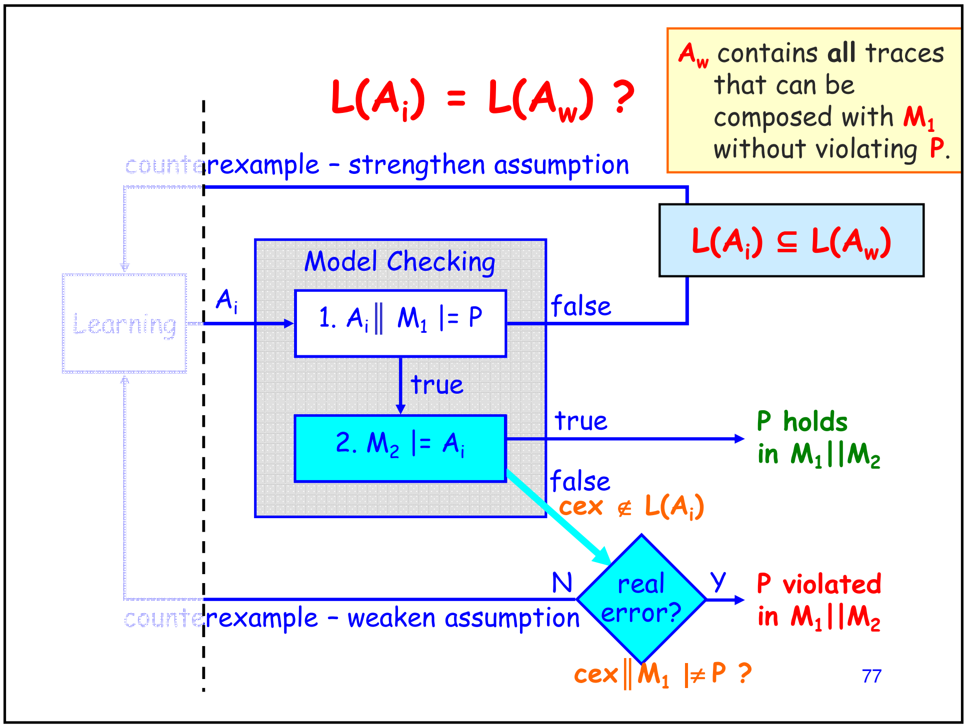
  or – **Simulation**: is π (**error**) state **reachable** when simulating σ on $M_1 \parallel P_{err}$?
- **Equivalence** query: $L(C_i) = L(A_w)$?
  - Translate C_i into a safety LTS A_i
  - Use it as candidate assumption for ***AG* rule**

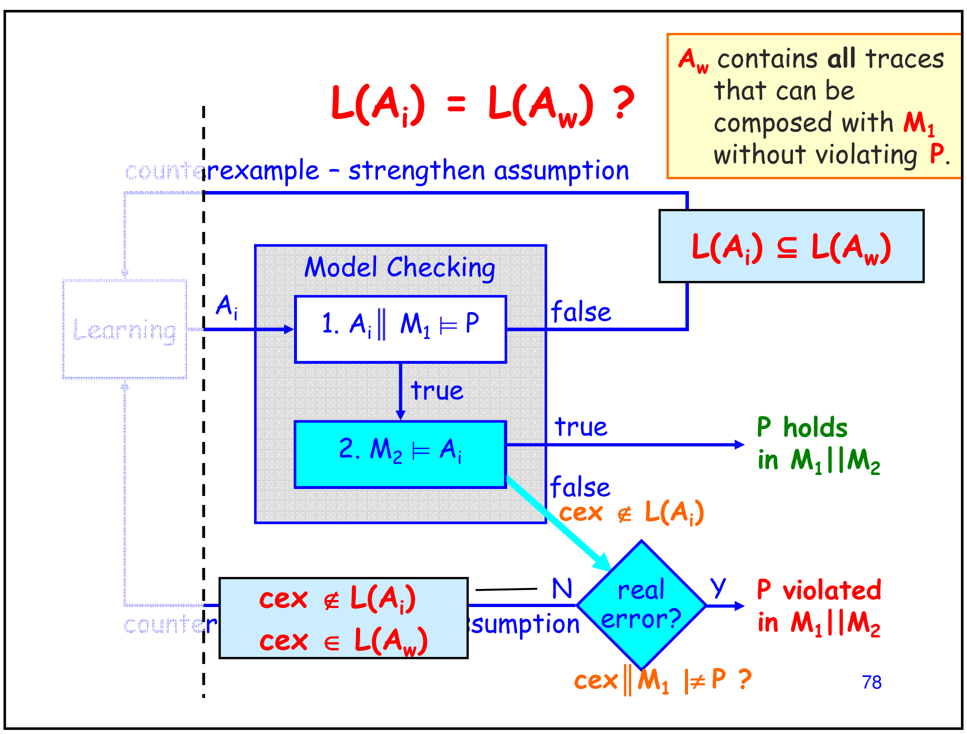
73




L(A_i) = L(A_w) ?
A_w contains all traces that can be composed with M_1 without violating P.
counterexample – strengthen assumption
L(A_i) ⊆ L(A_w)?
Learning
A_i
Model Checking
1. A_i ‖ M_1 ⊨ P
true
2. M_2 ⊨ A_i
false
cex ∈ L(A_i)
cex ∉ L(A_w)
true
P holds in M_1||M_2
false
cex ∉ L(A_i)
N
real error?
Y
P violated in M_1||M_2
counterexample – weaken assumption
cex‖M_1 |≠ P ?
75

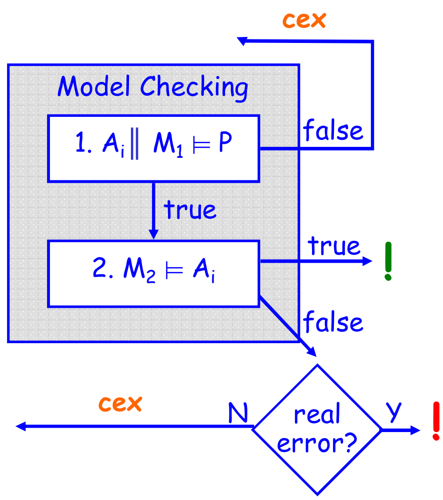

L(A_i) = L(A_w) ?
A_w contains all traces that can be composed with M_1 without violating P.
counterexample – strengthen assumption
L(A_i) ⊆ L(A_w)
Learning
A_i
Model Checking
1. A_i ‖ M_1 ⊨ P
false
true
2. M_2 ⊨ A_i
true
P holds in M_1||M_2
false
cex ∉ L(A_i)
N
real error?
Y
P violated in M_1||M_2
counterexample – weaken assumption
cex‖M_1 |≠ P ?
76


L(Ai) = L(Aw) ?
Aw contains all traces that can be composed with M1 without violating P.
counterexample – strengthen assumption
L(Ai) ⊆ L(Aw)
Model Checking
Learning
Ai
1. Ai ‖ M1 |= P
false
true
2. M2 |= Ai
true
P holds in M1||M2
false
cex ∉ L(Ai)
N
real error?
Y
P violated in M1||M2
counterexample – weaken assumption
cex‖M1 |≠ P ?
77


L(Ai) = L(Aw) ?
Aw contains all traces that can be composed with M1 without violating P.
counterexample – strengthen assumption
L(Ai) ⊆ L(Aw)
Model Checking
Learning
Ai
1. Ai ‖ M1 ⊨ P
false
true
2. M2 ⊨ Ai
true
P holds in M1||M2
false
cex ∉ L(Ai)
cex ∉ L(Ai)
cex ∈ L(Aw)
N
real error?
Y
P violated in M1||M2
counter
sumption
cex‖M1 |≠ P ?
78

Equivalence Query - Summary

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



79

Characteristics of Framework

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

- L* terminates
  ➔ Framework guaranteed to **terminate**:
  - **At latest** terminates when A_w is produced.
  - **Possibly** terminates **before** A_w is produced!

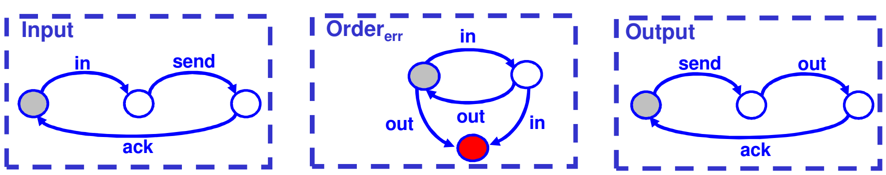
80

Outline

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

81

Example



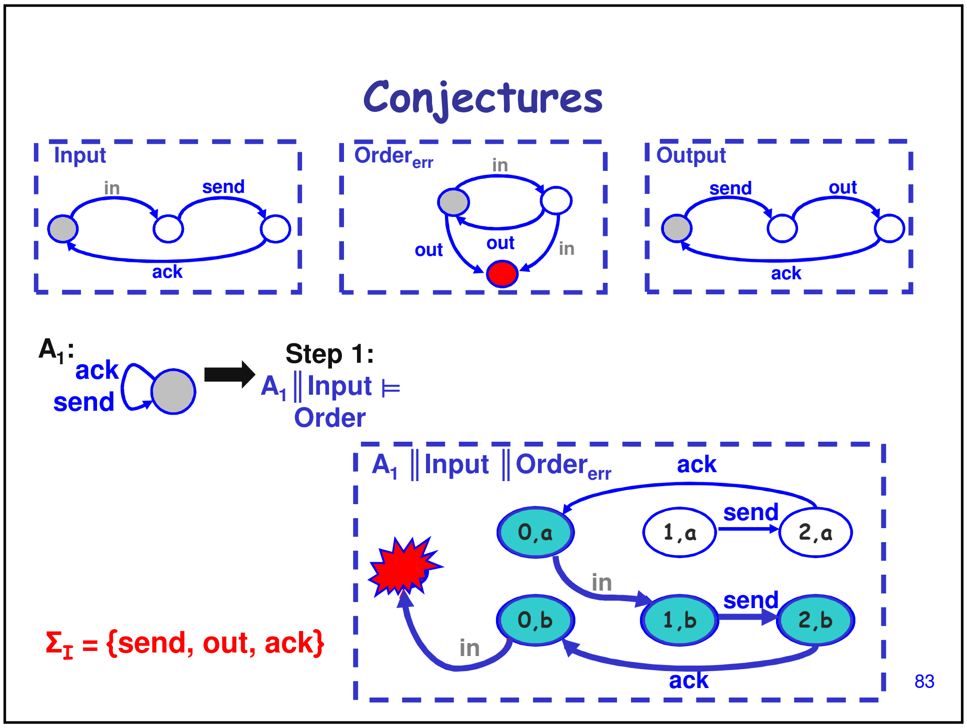
Check: **Input $\|$ Output $\models$ Order ?**

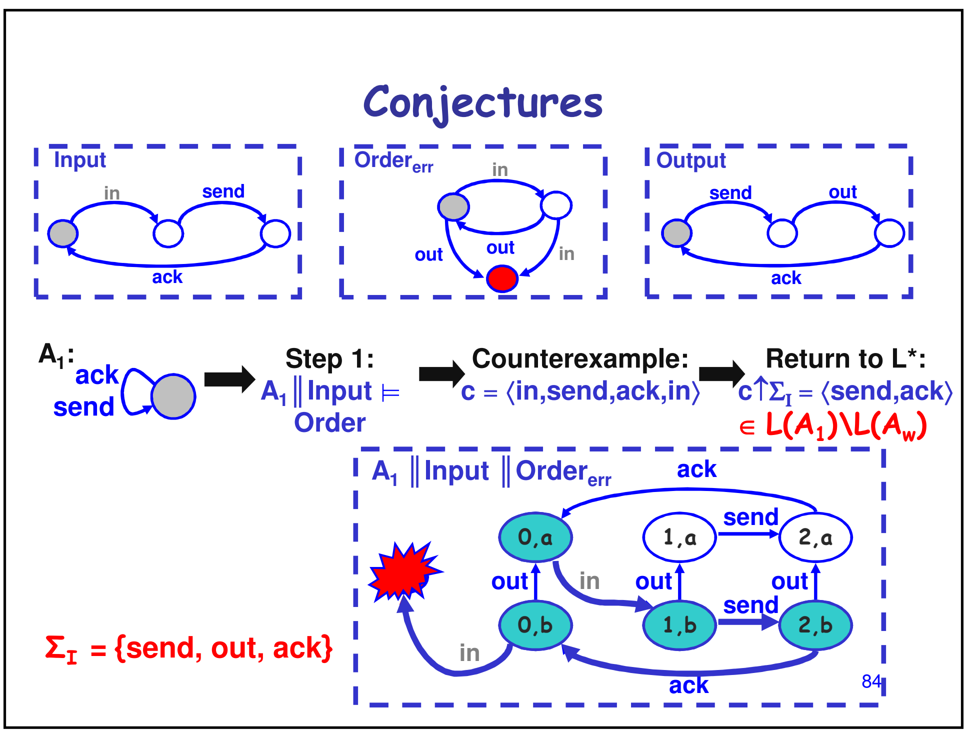
M_1 $\quad$ M_2 $\quad$ P

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

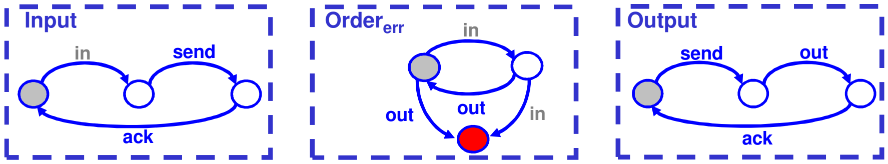
$\Sigma_I = (\alpha M_1 \cup \alpha P) \cap \alpha M_2$

82

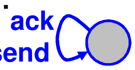

Conjectures
Input
in
send
ack
Order$_{err}$
in
out
out
in
Output
send
out
ack
A$_1$:
ack
send
Step 1:
A$_1$ ‖ Input ⊨
Order
A$_1$ ‖ Input ‖ Order$_{err}$
ack
0,a
1,a
send
2,a
in
0,b
1,b
send
2,b
in
ack
Σ_I = {send, out, ack}
83


Conjectures
Input
in
send
ack
Order$_{err}$
in
out
out
in
Output
send
out
ack
A$_1$:
ack
send
Step 1:
A$_1$ ‖ Input ⊨
Order
Counterexample:
c = ⟨in,send,ack,in⟩
Return to L*:
c↑Σ_I = ⟨send,ack⟩
∈ L(A$_1$)\L(A$_w$)
A$_1$ ‖ Input ‖ Order$_{err}$
ack
0,a
1,a
send
2,a
out
in
out
out
0,b
1,b
send
2,b
in
ack
Σ_I = {send, out, ack}
84

Conjectures



A_1:



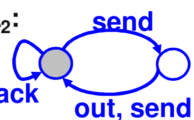
Step 1: $A_1 \| Input \models Order$

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

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

Queries

A_2:



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

Step 1: $A_2 \| Input \models Order$ True

Step 2: $Output \models A_2$ True

property Order holds on Input || Output

Σ_I = {send, out, ack}

85

Conclusion of Learning-Based AG

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

86

Learning-Based Compositional Verification of Behavioral UML Systems

Yael Meller, Orna Grumberg,
and Karen Yorav

87

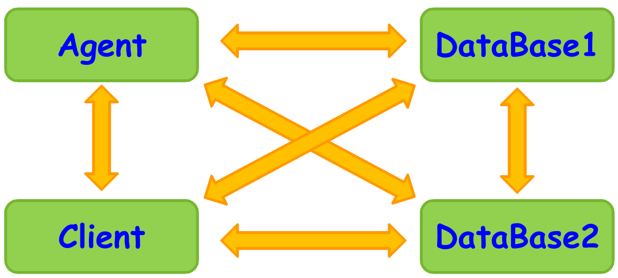
UML – Unified Modeling Language

- UML – object oriented modeling language
- Used for visualizing, specifying, and constructing systems
- Becoming dominant modeling language for embedded systems
  - E.g. car industry
- Used at early stages of the design

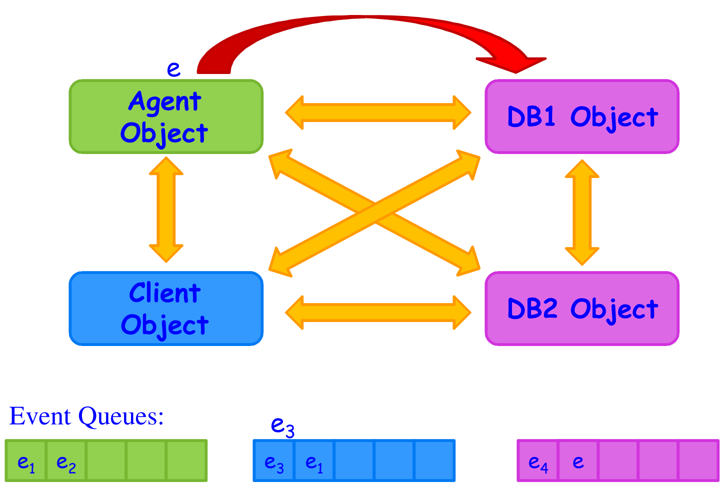
Verification is crucial

88

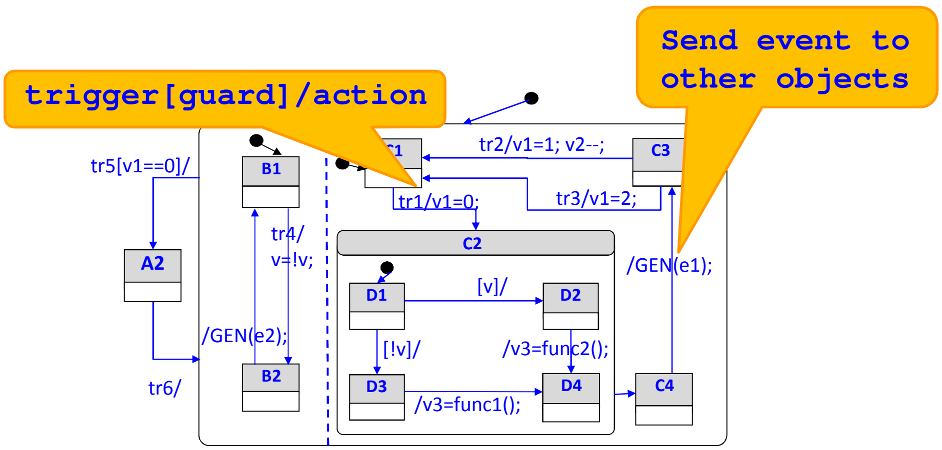
Behavioral UML Model – Objects and Their Connection



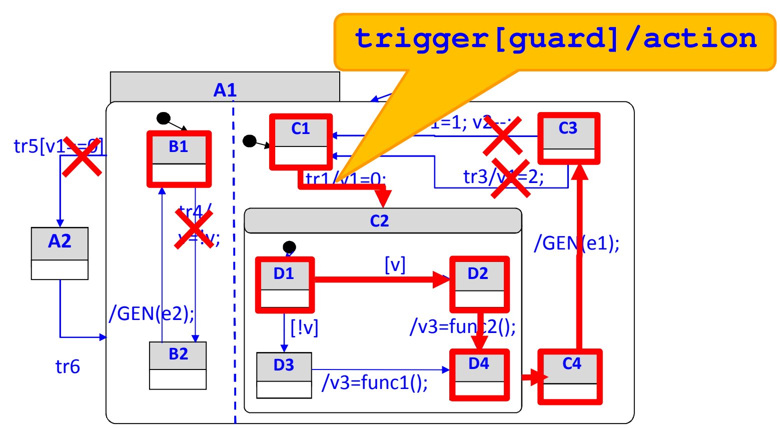
Behavioral UML System – Objects and Their Connection



Object Behavior Defined by State Machines



Object Behavior Defined by State Machines – RTC steps



Compositional Verification for Behavioral UML Models

$$\frac{\begin{array}{c}[A]M_1\langle P\rangle \\ \langle true\rangle M_2\,[A]\end{array}}{\langle true\rangle M_1||M_2\langle P\rangle}$$

Define framework at the UML level

- The notation [A] emphasizes that the assumption is a UML state machine

Compositional Verification for Behavioral UML Models

Advantages of framework at the UML level:

- Avoid blowup due to translation to lower level representation
- Enable use of UML model checkers
  - Useful information to the user

Lecture 4

95

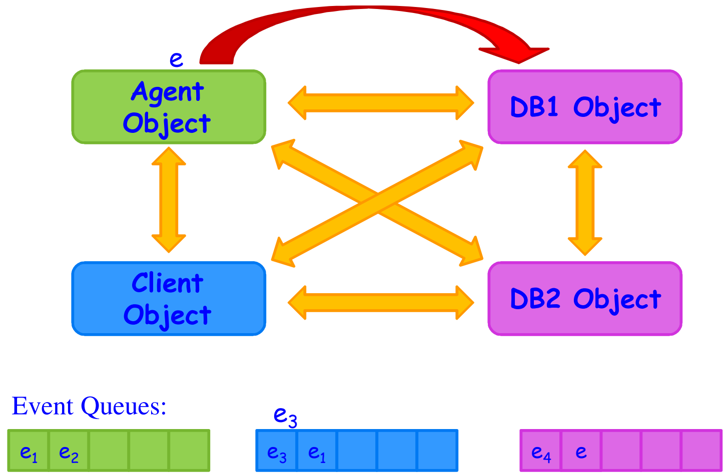
Goal

Develop a learning-based Assume Guarantee reasoning for UML models

- Components and assumptions are UML state machine
- Specification is AGp :
  - p holds for every reachable **configuration** of the system

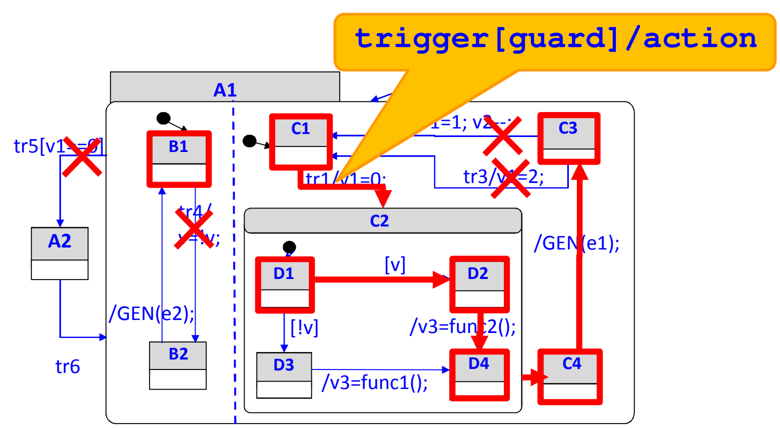
96

Behavioral UML System – Objects and Their Connection



97

Object Behavior Defined by State Machines – Run to Completion (RTC) steps



98

Compositional Verification for Behavioral UML Models

$$\frac{[A]M_1\langle P\rangle \quad \langle true\rangle M_2\,[A]}{\langle true\rangle M_1||M_2\langle P\rangle}$$

Define framework at the UML level

- The notation [A] emphasizes that the assumption is a UML state machine

99

Compositional Verification for Behavioral UML Models - Semantics

For every execution ex of $A||M_1$: **$ex \models P$**

Every execution ex of M_2 has a "representative" in A **$ex\downarrow_A \in EX(A)$**

$$\frac{[A]M_1\langle P\rangle \quad \langle true\rangle M_2[A]}{\langle true\rangle M_1||M_2\langle P\rangle}$$

For every execution ex of $M_1||M_2$: **$ex \models P$**

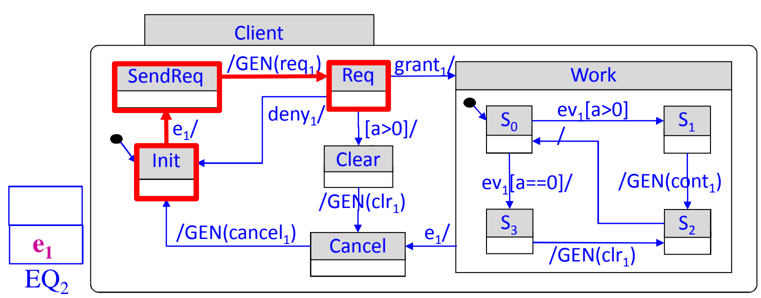
Need to define executions of UML

100

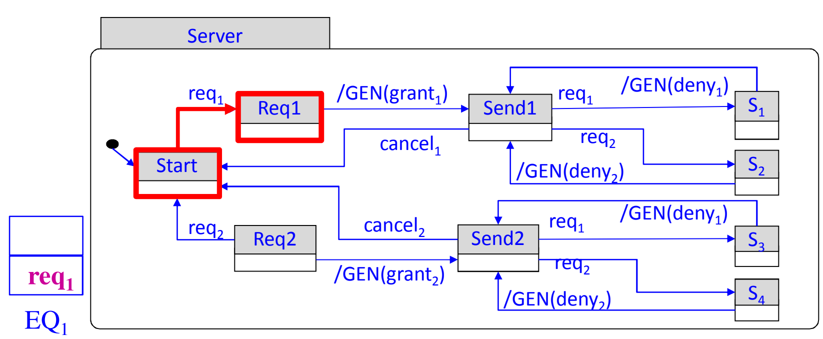
(Abstract) executions of UML models

- Describe the behavior w.r.t. **event manipulation**
  - gen(e) – represents **generation** of event **e**
  - tr(e) – represents **sending e** to its target state machine (from the event queue)

101



ex=gen(e_1),tr(e_1),gen(req_1),tr(req_1),gen(e_1),…



Properties to be checked

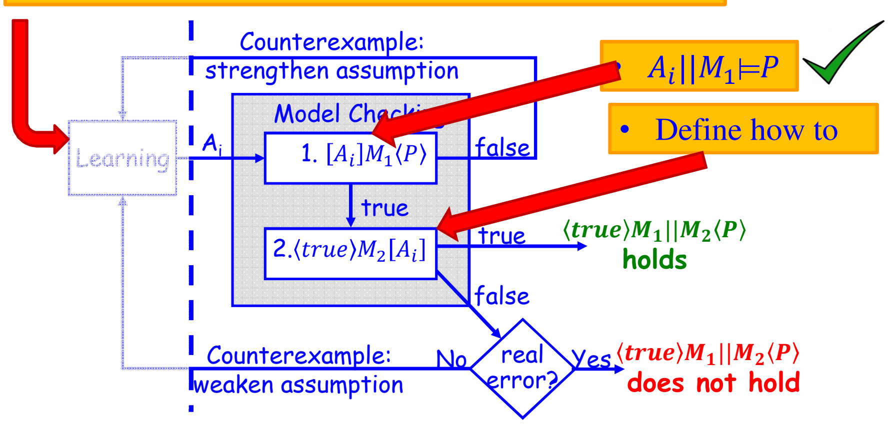
P is a safety property defined over events, based on predicates such as

- InQ(e): true when event e is in EQ
- Before(e,e'): true when e is before e' in EQ
- Gen(e): true when e is generated
- tr(e): true when e is sent to target

103

Learning-Based Compositional Verification of Behavioral UML Systems

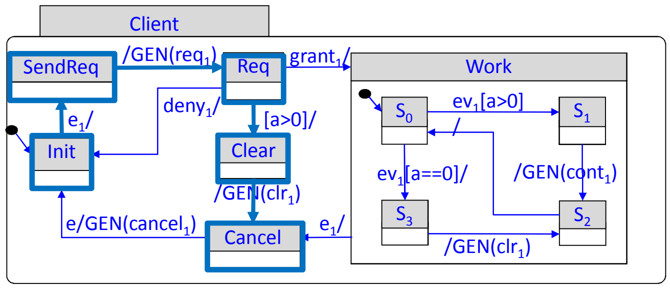
- Define alphabet and Teacher
- Translatate words/automata to state machines



Define Alphabet

$$\frac{[A]M_1\langle P\rangle \quad \langle true\rangle M_2[A]}{\langle true\rangle M_1||M_2\langle P\rangle}$$

M_2::



RTC of M_2: **(tr(e_1),gen(req_1),gen(clr_1))**

105

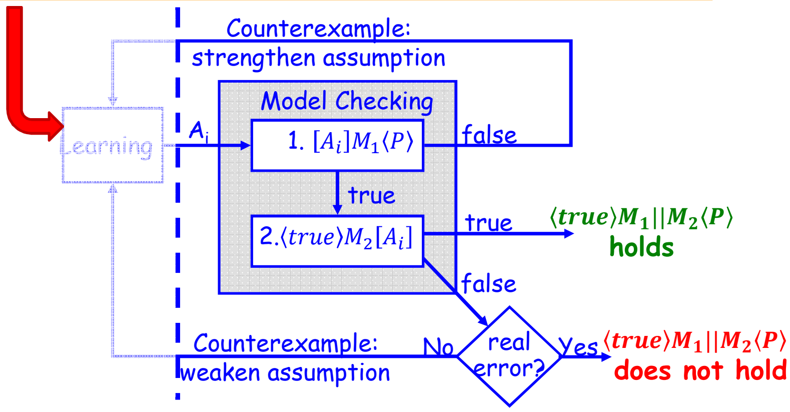
Letters in the alphabet

- Every RTC of M_2 induces a letter in the alphabet
- RTC is described as a sequence of events
  - e.g. **(tr(e_1),gen(req_1),gen(clr_1))**
- Alphabet of A is defined based on the interface of M_2
  - e.g. **(tr(e_1),gen(req_1))** in alphabet of A

106

Learning-Based Compositional Verification of Behavioral UML Systems

- Alphabet – based on interface events
- Define a Teacher

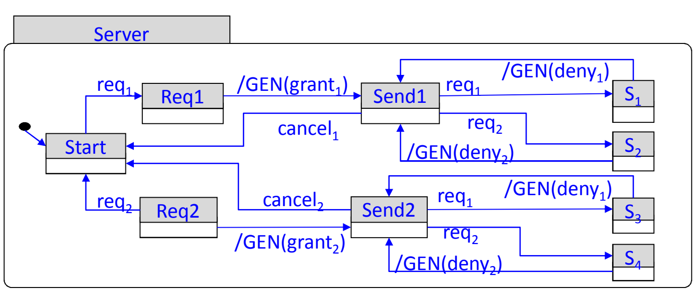


Checking Membership Queries

$$\frac{\begin{array}{c}[A]M_1\langle P\rangle \\ \langle true\rangle M_2[A]\end{array}}{\langle true\rangle M_1||M_2\langle P\rangle}$$

– Is **w=(tr(e$_1$),gen(req$_1$)),(tr(grant$_1$))** in A?

M_1::

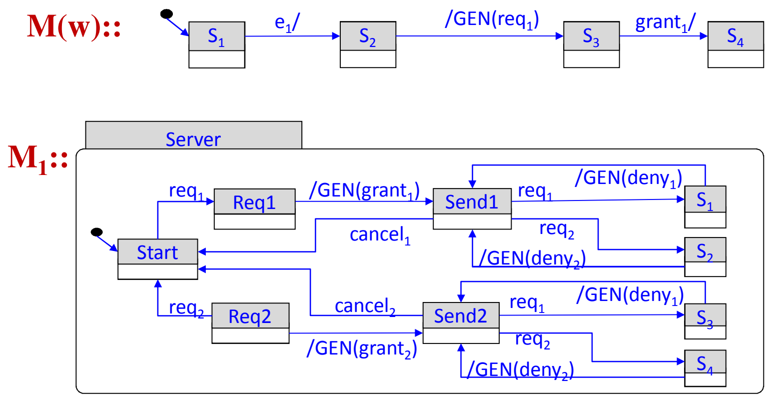


108

Checking Membership Queries

$$\frac{[A]M_1\langle P\rangle \quad \langle true\rangle M_2[A]}{\langle true\rangle M_1||M_2\langle P\rangle}$$

- w=(tr(e_1),gen(req_1)),(tr($grant_1$)) in A **iff** on every concrete execution of $A||M_1$ that matches w, P holds

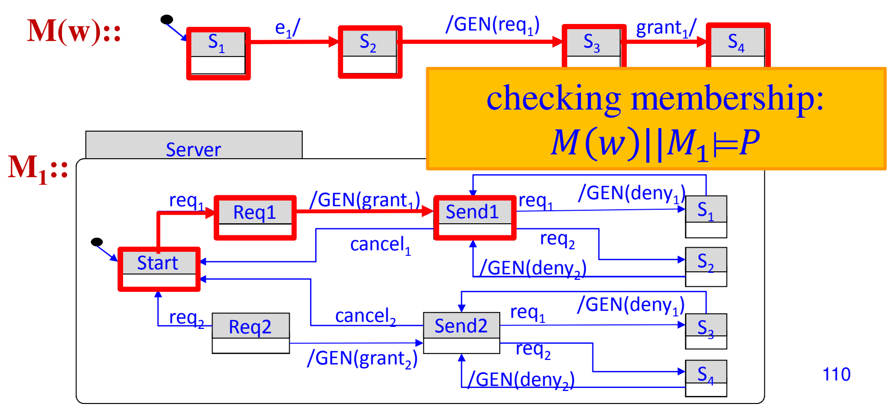


109

Checking Membership Queries

$$\frac{[A]M_1\langle P\rangle \quad \langle true\rangle M_2[A]}{\langle true\rangle M_1||M_2\langle P\rangle}$$

- w=(tr(e_1),gen(req_1)),(tr($grant_1$)) in A **iff** on every concrete execution of $A||M_1$ that matches w, P holds



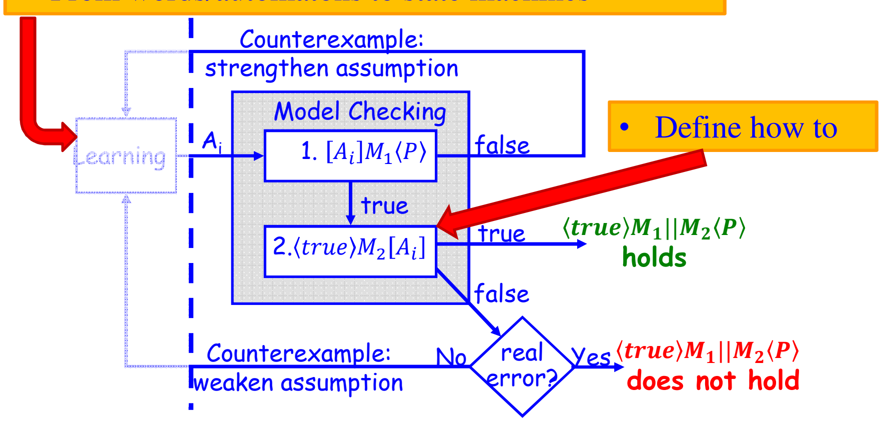
110

- In fact, M(w) is somewhat more complex…

111

Learning-Based Compositional Verification of Behavioral UML Systems

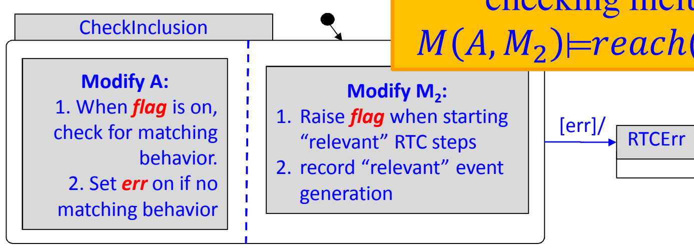
- Alphabet – based on interface events
- Define a Teacher
- From words/automatons to state machines



Verify $\langle true \rangle M_2 [A]$

- $\langle true \rangle M_2 [A]$ means execution inclusion: $EX(M_2)\downarrow_A \subseteq EX(A)$
- Create a new state machine for monitoring M_2 w.r.t A and model check result

checking inclusion: $M(A, M_2) \vDash reach(RTCErr)$



113

Summary of AG Verification for UML

- Present a framework for learning-based AG reasoning on behavioral UML models
- Model checking remains at the UML level
- Framework can also be defined for the case where the system includes more than 2 objects.

114

Automated Circular Assume-Guarantee Reasoning

Karam Abd Elkader, Orna Grumberg,
Corina Pasareanu, and Sharon Shoham

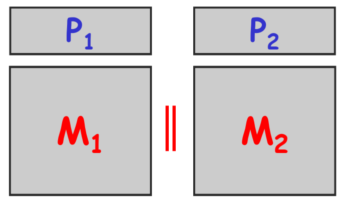
Formal Methods (FM) 2015

Motivation

- AG rule is asymmetric w.r.t M_1 and M_2
- Sometimes the components mutually depend on each other for their correctness

Naïve Circular Rule

$$\frac{\langle P_2 \rangle\ M_1\ \langle P_1 \rangle \qquad \langle P_1 \rangle\ M_2\ \langle P_2 \rangle}{M_1 || M_2 \models P_1 || P_2}$$



Unsound!

Need to break circularity

Induction: over formulas, time, or both

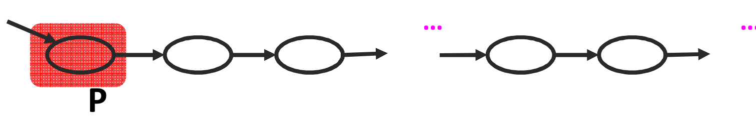
117

Inductive Properties [McMillan 99]

M $\models$ **A** $\triangleright$ **P** :

Every trace σ of M:

- Initially satisfies **P**, and
- If σ satisfies **A** up to k, then it also satisfies **P** up to step $k+1$



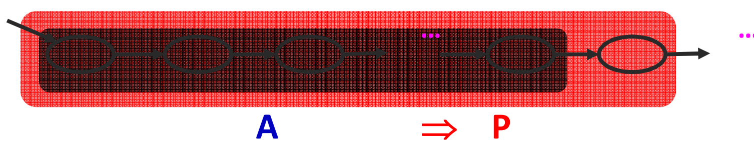
118

Inductive Properties [McMillan 99]

$M \vDash A \triangleright P$:

Every trace σ of M:

- Initially satisfies **P**, and
- If σ satisfies **A** up to k, then it also satisfies **P** up to step $k+1$

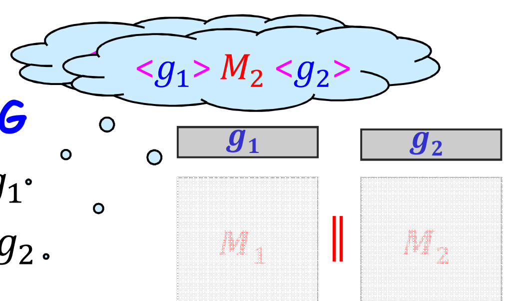


Circular AG Rule

Rule Circ-AG

$M_1 \vDash g_2 \triangleright g_1$.

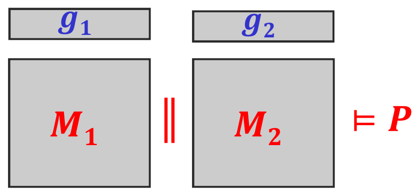
$M_2 \vDash g_1 \triangleright g_2$.



Circular AG Rule

Rule Circ-AG

$$\frac{\begin{array}{c} M_1 \models g_2 \triangleright g_1 \\ M_2 \models g_1 \triangleright g_2 \\ g_1 || g_2 \models P \end{array}}{M_1 || M_2 \models P}$$



Rule Circ-AG is sound and complete

P is verified with **some g_1 and g_2** → P is **correct**

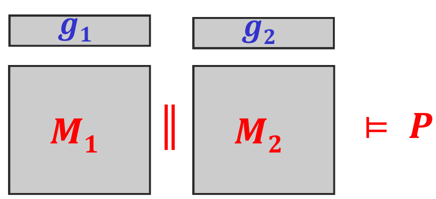
P is **correct** → **Exist g_1 and g_2** s.t. rule applicable

121

Automated Circular AG Reasoning

Rule Circ-AG

$$\frac{\begin{array}{c} M_1 \models g_2 \triangleright g_1 \\ M_2 \models g_1 \triangleright g_2 \\ g_1 || g_2 \models P \end{array}}{M_1 || M_2 \models P}$$

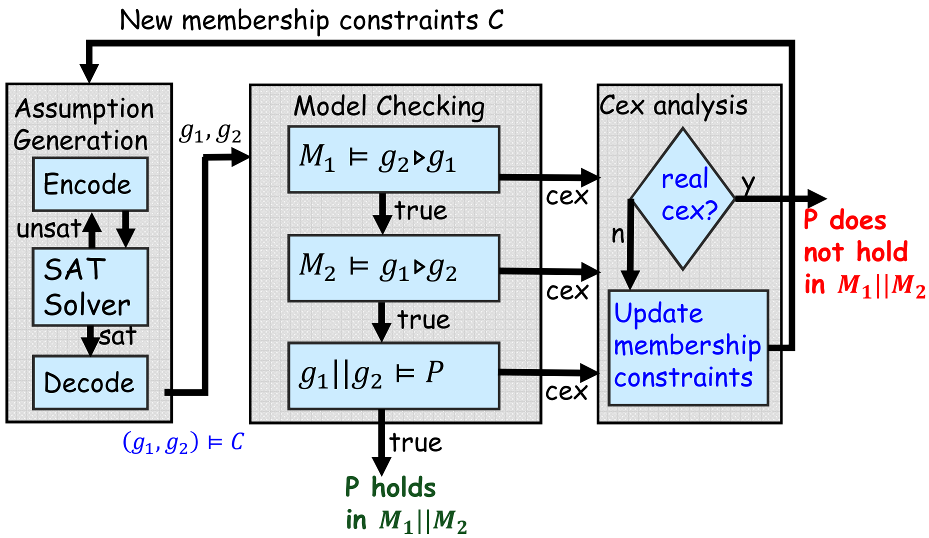


How to automatically construct assumptions ?

Challenge: g_1 and g_2 depend on each other

122

ACR Algorithm



123

Summary of ACR

- Automated circular assume-guarantee reasoning
  - Assumptions depend on each other
  - Uses joint disjunctive constraints
- Assumptions are significantly smaller than those obtained by the non-circular rule
- ACR outperforms L* based algorithms for non-circular rule

124

Thank You

125