Compositional Model Checking

Orna Grumberg
Technion, Israel

Marktoberdorf 2015

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

Lecture 1
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

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
Properties in propositional temporal logic - examples

- mutual exclusion:
  \[ \text{always } \neg (cs_1 \land cs_2) \]

- non starvation:
  \[ \text{always } (\text{request} \Rightarrow \text{eventually granted}) \]

- communication protocols:
  \[ (\neg \text{get-message}) \text{ until send-message} \]

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
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:**

- $\mathcal{G}p$ (always)
- $\mathcal{F}p$ (eventually)
- $\mathcal{X}p$ (next)
- $p \mathcal{U} q$ (eventually $q$)

**Path quantifiers:**

- $A$ for all path
- $E$ there exists a path

**CTL formulas: Example**

- *mutual exclusion:* $A\mathcal{G}( \sim (cs_1 \land cs_2))$
- *non starvation:* $A\mathcal{G}(request \Rightarrow A\mathcal{F}grant)$
- *“sanity” check:* $E\mathcal{F}request$
Example to demonstrate:
- Building a model from a program
- Properties
- Model checking

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:
  
  \[
  P_i :: \text{ while (true) } \{
  \text{ if (} v_i == N \text{) } v_i = T; \\
  \text{ else if (} v_i == T \&\& \text{ sem) } \{ v_i = C; \text{ sem} = 0; \} \\
  \text{ else if (} v_i == C \text{) } \{ v_i = N; \text{ sem} = 1; \}
  \}
  \]

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

---

**Mutual Exclusion Example**

```
(\( v_1=\text{N}, v_2=\text{N}, \text{sem} \))
```

```
(\( v_1=\text{T}, v_2=\text{N}, \text{sem} \))
```

```
(\( v_1=\text{N}, v_2=\text{T}, \text{sem} \))
```

```
(\( v_1=\text{C}, v_2=\text{N}, \neg \text{sem} \))
```

```
(\( v_1=\text{T}, v_2=\text{T}, \text{sem} \))
```

```
(\( v_1=\text{N}, v_2=\text{C}, \neg \text{sem} \))
```

```
(\( v_1=\text{C}, v_2=\text{T}, \neg \text{sem} \))
```

```
(\( v_1=\text{T}, v_2=\text{C}, \neg \text{sem} \))
```
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: $AG(T_1 \land T_2)$
• Property 1: $\textbf{AG}^{-1}(C_1 \wedge C_2)$

$S_0$

• Property 1: $\textbf{AG}^{-1}(C_1 \wedge C_2)$

$S_1$
• Property 1: $AG^{-}(C_1 \land C_2)$

$S_2$

• Property 1: $AG^{-}(C_1 \land C_2)$

$S_3$
• $M \models AG \rightarrow (C_1 \land C_2)$

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

• Property 2: $AG \models (T_1 \land T_2)$
• Property 2: $AG\neg(T_1 \land T_2)$
• $\mathcal{M} \not\models AG \neg (T_1 \land T_2)$

• A violating state has been found

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

Learning Assumptions for Compositional Verification

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

TACAS 2003
Compositional Verification

• **Inputs:**
  - composite system $M_1 \parallel M_2$
  - property $P$

• **Goal:** check if $M_1 \parallel M_2 \models P$

First attempt: Divide and Conquer

• 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?

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) $P$
Useful AG Rule

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

\[
M_1 \parallel M_2 \models P \Rightarrow \text{true} \quad M_1 || M_2 \models P
\]

Useful AG Rule for Safety Properties

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

\[
\begin{align*}
\langle A \rangle M_1 &\langle P \rangle \\
\text{true} M_2 &\langle A \rangle \\
\text{true} \quad M_1 || M_2 &\models P
\end{align*}
\]
Outline

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

Labeled Transition Systems (LTS)

\( \text{LTS } M = (Q, q^0, \alpha_M, \delta) \)

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

Observable actions

Internal action

Diagram:

- States: 0, 1, 2
- Actions: in, send, ack
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 $M$

Parallel Composition $M_1 \parallel M_2$

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

Example:
Safety Properties

Expressed as deterministic LTSs

For a safety LTS, \( P \):

- \( L(P) \) describes the set of legal (acceptable) behaviors over \( \alpha P \)

\[ M \models P \text{ iff } \forall \sigma \in L(M) : (\sigma \uparrow_{\alpha P}) \in L(P) \]

Note that, since we check \( M \models P, \alpha P \subseteq \alpha M \)
Example

```
<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,a</td>
<td>in</td>
</tr>
<tr>
<td>1,a</td>
<td>send</td>
</tr>
<tr>
<td>2,a</td>
<td>out</td>
</tr>
<tr>
<td>0,b</td>
<td>in</td>
</tr>
<tr>
<td>1,b</td>
<td>out</td>
</tr>
<tr>
<td>2,b</td>
<td>out</td>
</tr>
<tr>
<td>0,c</td>
<td>in</td>
</tr>
<tr>
<td>1,c</td>
<td>in</td>
</tr>
<tr>
<td>2,c</td>
<td>out</td>
</tr>
</tbody>
</table>
```

\[ \text{Pref}(<\text{in,send,out,ack}^*>) \uparrow \{\text{in, out}\} \]

\[ \subseteq \]

\[ \sqrt{\text{Pref(<in,out>*')}} \]

Order

\[ \text{in} \rightarrow \text{out} \]

\[ \text{Pref(<in,out>*')} \subseteq \text{Pref(<in,send,out,ack>*')} \]

\[ \Rightarrow \]

\[ \sqrt{\text{Pref(<in,out>*')}} \]

Lecture 2
Labeled Transition Systems (LTS)

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

$L(M) = \text{the Language of } M$: the set of all traces of $M$

Parallel Composition $M_1 \parallel M_2$

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

Example:
Safety Properties

Expressed as deterministic LTSs

For a safety LTS, \( P \):

- \( L(P) \) describes the set of legal (acceptable) behaviors over \( \alpha P \)

\[ M \models P \iff \forall \sigma \in L(M) : (\sigma, \alpha P) \in L(P) \]

Note that, since we check \( M \models P, \alpha P \subseteq \alpha M \)
Example

\[ \text{Pref}(\langle \text{in}, \text{send}, \text{out}, \text{ack} \rangle^*) \subseteq \text{Pref}(\langle \text{in}, \text{out} \rangle^*) \]

Model Checking \( M \models P \)

Safety LTS \( P \Rightarrow \) an Error LTS, \( P_{\text{err}} \):
- “traps” violations with special error state \( \pi \)
- Error LTS is complete
- \( \pi \) is a deadend state: has no outgoing transitions
Model Checking $M \models P$

**Theorem:**

- $M \models P$ iff $\pi$ 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 $\alpha P$

---

**Example**

Input $\parallel$ Output

```
0,a
1,a
2,a
0,b
1,b
2,b
0,c
1,c
2,c
```

Order$_{err}$

```
In
Out
Out
Out
Out
Out
```

In automata:

$M \models \phi$ iff $L(A_M \cap A_{\phi}) = \emptyset$
Example

Input || Output

Order_{err}

Order_{err}
Assume Guarantee Reasoning

- **Assumptions**: also expressed as safety LTSs.

- \(<A> M <P>\) is true iff \(A \parallel M \models P\)
  i.e. \(\pi\) is unreachable in \(A \parallel M \parallel P_{err}\)

\[
\begin{array}{ccc}
\langle A \rangle M_1 \langle P \rangle \\
\langle \text{true} \rangle M_2 \langle A \rangle \\
\langle \text{true} \rangle M_1 \parallel M_2 \langle P \rangle
\end{array}
\rightarrow
\begin{array}{ccc}
A \parallel M_1 \models P \\
M_2 \models A \\
M_1 \parallel M_2 \models P
\end{array}
\]

- \(M_1, M_2 : \text{LTSs}\)
- \(P, A : \text{safety LTSs}\)

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 $\alpha A_w \subseteq \alpha M_2'$
  $<true> M_1 || M_2' <P>$ iff $<true> M_2' <A_w>$

Note that:
- $<true> M_1 || A_w <P>$ holds
- $A_w$ describes exactly all traces $\sigma$ over $\Sigma_I$ such that in the context of $\sigma$, $M_1$ satisfies $P$
Observation

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

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

Given a Candidate Assumption $A_i$

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 $\sigma$ such that “$\sigma \parallel M_1” \neq P
- $\sigma$ should be removed from $A_i$
Given a Candidate Assumption $A_i$

Suppose $A_i \parallel M_1 \models P$ holds:
- If $M_2 \models A_i$ also holds $\Rightarrow M_1 \parallel M_2 \models P$ \textbf{verified}!

Otherwise: $\sigma \in L(M_2)$ but $(\sigma\uparrow\Sigma_I) \notin L(A_i)$ -- $\text{cex}$ $P$ violated by $M_1$ in the context of $\text{cex}=(\sigma\uparrow\Sigma_I)$?
- \textbf{Yes: real violation} $\Rightarrow M_1 \parallel M_2 \models P$ \textbf{falsified}!
- \textbf{No: spurious violation} $\Rightarrow$ need to find better approximation of $A_w$

---

"$\text{cex} \parallel M_1 \models P$?"

- Check if $\pi$ reachable in $A_{\text{cex}} \parallel M_1 \parallel P_{\text{err}}$

Alternatively,

- \textbf{Simulate} $\text{cex}$ on $M_1 \parallel P_{\text{err}}$
  - $\text{cex} \parallel M_1 \models P$ iff when $\text{cex}$ is simulated on $M_1 \parallel P_{\text{err}}$
    - it cannot lead to $\pi$ (error) state.
Assume-Guarantee Framework

Learning

Model Checking

1. $A_i \models M_1 \models P$

2. $M_2 \models A_i$

counterexample - strengthen assumption

true

false

counterexample - weaken assumption

real

error?

true

false

cex \not\in L(A_i)

P holds in $M_1 || M_2$

P violated in $M_1 || M_2$

For $A_w$: conclusive results guaranteed

termination!

Lecture 3
Goal:
Automatically learn an assumption for the Assume Guarantee Rule for $M_1 || M_2 \models P$

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

- $<\text{true}> M_1 || A_w <P>$ holds
- $A_w$ describes exactly all traces $\sigma$ over $\Sigma_I$ such that in the context of $\sigma$, $M_1$ satisfies $P$
Assume-Guarantee Framework

Learning

Model Checking

1. $A_i \models M_1 \models P$
   - true
   - false

2. $M_2 \models A_i$
   - false
   - cex $\notin L(A_i)$
   - real error?

For $A_w$: conclusive results guaranteed → termination!

P holds in $M_1 \parallel M_2$
P violated in $M_1 \parallel M_2$

Outline

✓ Motivation
✓ Setting
✓ Automatic Generation of Assumptions for the AG Rule
  • Learning algorithm (briefly)
  • Assume-Guarantee with Learning
  • Example
Learning Algorithm for DFA – L*

• L*: by Angluin, improved by Rivest & Schapire
• learns an unknown regular language $\mathcal{U}$
• produces a minimal Deterministic Finite-state Automaton (DFA) $\mathcal{C}$ such that $L(\mathcal{C}) = \mathcal{U}$

DFA $\mathcal{M} = (Q, q^0, \alpha_M, \delta, F)$:
• $Q, q^0, \alpha_M, \delta$: as in deterministic LTS
• $F \subseteq Q$: accepting states
• $L(\mathcal{M}) = \{\sigma \mid \delta(q^0, \sigma) \in F\}$

Learning Algorithm for DFA – L*

• L* interacts with a Teacher to answer two types of questions:
  - Membership queries: is string $\sigma$ in $\mathcal{U}$?
  - Conjectures: for a candidate DFA $\mathcal{C}_i$, is $L(\mathcal{C}_i) = \mathcal{U}$?
    - answers are (true) or (false + counterexample)

Conjectures $\mathcal{C}_1, \mathcal{C}_2, \ldots$ converge to $\mathcal{C}$

Equivalence Queries
Outline

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

Assume-Guarantee with Learning

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

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

A_w is unknown...

- **Membership** query: \( \sigma \in L(A_w) \)?
  
  \( \Rightarrow \) Check if "\( \sigma \ || \ M_1 \) satisfies P:
  
  - **Model checking**: is \( \pi \) reachable in \( A_\sigma \ || \ M_1 \ || \ P_{err} \)?
  
  or
  
  - **Simulation**: is \( \pi \) (error) state reachable when simulating \( \sigma \) on \( M_1 \ || \ P_{err} \)?

- **Equivalence** query: \( L(C_i) = L(A_w) \)?
  
  - Translate \( C_i \) into a safety LTS \( A_i \)
  
  - Use it as candidate assumption for AG rule

\[ L(A_i) = L(A_w) ? \]
Model Checking

1. \( A_i \parallel M_1 \models P \)

2. \( M_2 \models A_i \)

A_w contains all traces that can be composed with \( M_1 \) without violating \( P \).

Learning

real error?

\( L(A_i) \subseteq L(A_w) \)?

counterexample – strengthen assumption

\( L(A_i) = L(A_w) \)?

counterexample – weaken assumption

\( L(A_i) \subseteq L(A_w) \)?

A_w contains all traces that can be composed with \( M_1 \) without violating \( P \).

false

cex \( \notin L(A_i) \)

true

cex \( \notin L(A_w) \)

true

P holds in \( M_1 || M_2 \)

false

cex \( \notin L(A_i) \)

false

cex \( \notin L(A_i) \)

true

cex \( \notin L(A_w) \)

true

P violated in \( M_1 || M_2 \)

false

cex \( \notin L(A_i) \)

false

cex \( \notin L(A_i) \)

true

P violated in \( M_1 || M_2 \)

false

cex \( \notin L(A_i) \)

false

cex \( \notin L(A_i) \)

true

P holds in \( M_1 || M_2 \)

false

cex \( \notin L(A_i) \)

false

cex \( \notin L(A_i) \)

true

P violated in \( M_1 || M_2 \)

false

cex \( \notin L(A_i) \)

false

cex \( \notin L(A_i) \)

true

P holds in \( M_1 || M_2 \)
Model Checking

1. $A_i \parallel M_1 \models P$
   - false
2. $M_2 \models A_i$
   - true
   - false
   - $cex \notin L(A_i)$
   - real error?

Learning

- $P$ holds in $M_1 \parallel M_2$
- $P$ violated in $M_1 \parallel M_2$
- $L(A_i) \subseteq L(A_w)$
- $A_w$ contains all traces that can be composed with $M_1$ without violating $P$.

---

L($A_i$) = L($A_w$) ?

Model Checking

1. $A_i \parallel M_1 \models P$
   - false
2. $M_2 \models A_i$
   - true
   - false
   - $cex \notin L(A_i)$
   - real error?

Learning

- $P$ holds in $M_1 \parallel M_2$
- $P$ violated in $M_1 \parallel M_2$
- $L(A_i) \subseteq L(A_w)$
- $A_w$ contains all traces that can be composed with $M_1$ without violating $P$. 
Equivalence Query - Summary

2 applications of model checking + 1 model checking or simulation

Model Checking

1. $A_i \parallel M_1 \models P$
2. $M_2 \models A_i$

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!
Outline

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

Example

Check: Input || Output ⊨ Order ?

\[ \Sigma_I (\text{assumption's alphabet}) : \{ \text{send, out, ack} \} \]

\[ \Sigma_I = (aM_1 \cup aP) \cap aM_2 \]
\( \Sigma_I = \{ \text{send, out, ack} \} \)
### Conjectures

**A₁:**
- `ack`
- `send`

**A₂:**
- `send`
- `ack`
- `out`, `send`

**Queries**
- `c ⇒ Σ ⇒ L(A₂)`
- `Σ_I = {send, out, ack}`

**Step 1:**
- `A₁ ⊦ Input ⊨ Order`

**Counterexample:**
- `c = (in, send, ack, in)`
- `c ⇒ Σ ⇒ L(A₁) \ L(A_w)`

**Return to L⁺:**
- `c ⇒ Σ ⇒ L(A₁) \ L(A_w)`

**Step 2:**
- `A₂ ⊦ Input ⊨ Order`
- `Output ⊨ A₂`

**Property**
- `Order` holds on `Input ⊦ Output`

---

### 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
Learning-Based Compositional Verification of Behavioral UML Systems

Yael Meller, Orna Grumberg, and Karen Yorav

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
Object Behavior Defined by State Machines

trigger[guard]/action

Send event to other objects

Object Behavior Defined by State Machines - RTC steps

trigger[guard]/action
Compositional Verification for Behavioral UML Models

\[
[A]M_1 \models P \\
\langle \text{true} \rangle M_2 [A] \\
\langle \text{true} \rangle M_1 || M_2 \models P
\]

Define framework at the UML level

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

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
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
Behavioral UML System - Objects and Their Connection

Event Queues:

Object Behavior Defined by State Machines - Run to Completion (RTC) steps
Compositional Verification for Behavioral UML Models

\[
[A]M_1\{P\} \\
\langle\text{true}\rangle M_2[A] \\
\langle\text{true}\rangle M_1 \parallel M_2\{P\}
\]

Define framework at the UML level

• The notation \([A]\) emphasizes that the assumption is a UML state machine

Compositional Verification for Behavioral UML Models - Semantics

For every execution ex of A||M_1: ex=P

\[
[A]M_1\{P\} \\
\langle\text{true}\rangle M_2[A] \\
\langle\text{true}\rangle M_1 \parallel M_2\{P\}
\]

Every execution ex of M_2 has a “representative” in A

\[\text{ex} \in \{\text{executions of } M_2}\in E(X(A))\]

Need to define executions of UML
(Abstract) executions of UML models

- Describe the behavior w.r.t. event manipulation
  - $\text{gen}(e)$ - represents generation of event $e$
  - $\text{tr}(e)$ - represents sending $e$ to its target state machine (from the event queue)
Properties to be checked

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

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

Learning-Based Compositional Verification of Behavioral UML Systems

- Define alphabet and Teacher
- Translate words/automata to state machines
- Define how to
Define Alphabet

$M_2::

RTC of $M_2$: $(\text{tr}(e_1), \text{gen(req_1)}, \text{gen(clr_1}))$

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. $(\text{tr}(e_1), \text{gen(req_1)}, \text{gen(clr_1}))$
- Alphabet of $A$ is defined based on the interface of $M_2$
  - e.g. $(\text{tr}(e_1), \text{gen(req_1)})$ in alphabet of $A$
Learning-Based Compositional Verification of Behavioral UML Systems

- Alphabet – based on interface events
- Define a Teacher

Model Checking

1. \([A_1]M_1(P)\)
2. \((true)M_2[A_1]\)

Counterexample: strengthen assumption

Counterexample: weaken assumption

Checking Membership Queries

- Is \(w=(tr(e_1), gen(req_1), (tr(grant_1))\) in \(A\)?
Checking Membership Queries

- \( w = (\text{tr}(e_1), \text{gen}(\text{req}_1)), (\text{tr}(\text{grant}_1)) \) in \( A \) iff on every concrete execution of \( A || M_1 \) that matches \( w \), \( P \) holds.

\[
\text{M}(w) ::
\begin{align*}
S_1 & \xrightarrow{e_1} S_2 \xrightarrow{\text{GEN}(\text{req}_1)} S_3 \xrightarrow{\text{grant}_1} S_4
\end{align*}
\]

\[
\text{M}_1 ::
\begin{align*}
\text{Start} & \xrightarrow{\text{req}_1} \text{Req}_1 \xrightarrow{\text{GEN}(\text{grant}_1)} \text{Send1} \xrightarrow{\text{req}_1} S_2 \xrightarrow{\text{GEN}(\text{deny}_1)} S_3 \\
& \quad \text{cancel}_1 \xrightarrow{\text{req}_2} \text{send2} \xrightarrow{\text{req}_2} \text{req}_2 \xrightarrow{\text{GEN}(\text{deny}_1)} S_4 \\
& \quad \text{req}_2 \xrightarrow{\text{GEN}(\text{grant}_2)} \text{req}_2 \xrightarrow{\text{cancel}_2} S_5
\end{align*}
\]

\[
[A] M_1(P) \\
\text{(true) } M_2[A] \\
\text{(true) } M_1 || M_2 \models P
\]
• In fact, $M(w)$ is somewhat more complex...
**Verify** \( \langle \text{true} \rangle M_2[A] \)

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

\[ \text{checking inclusion: } M(A, M_2) = \text{reach}(\text{RTC}Err) \]

**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.
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

\[
\begin{align*}
\langle P_2 \rangle M_1 & \langle P_1 \rangle \\
\langle P_1 \rangle M_2 & \langle P_2 \rangle \\
M_1 || M_2 & \models P_1 || P_2
\end{align*}
\]

Unsound!

Need to break circularity

**Induction:** over formulas, time, or both

Inductive Properties [McMillan 99]

\[ \mathcal{M} \models A \triangleright P : \]

Every trace \( \sigma \) of \( \mathcal{M} \):

- Initially satisfies \( P \), and
- If \( \sigma \) satisfies \( A \) up to \( k \), then it also satisfies \( P \) up to step \( k + 1 \)
Inductive Properties \cite{McMillan99}

\( M \models A \models P : \)

Every trace \( \sigma \) of \( M \):

- Initially satisfies \( P \), and
- If \( \sigma \) satisfies \( A \) up to \( k \), then it also satisfies \( P \) up to step \( k + 1 \)

Circular AG Rule

Rule \text{Circ-AG}

\[
\begin{align*}
M_1 &\models g_2 \models g_1, \\
M_2 &\models g_1 \models g_2.
\end{align*}
\]
Circular AG Rule

Rule Circ-AG

\[ M_1 \models g_2 \models g_1 \]
\[ M_2 \models g_1 \models g_2 \]
\[ g_1 || g_2 \models P \]
\[ M_1 || M_2 \models P \]

Rule Circ-AG is sound and complete

Automated Circular AG Reasoning

Rule Circ-AG

\[ M_1 \models g_2 \models g_1 \]
\[ M_2 \models g_1 \models g_2 \]
\[ g_1 || g_2 \models P \]
\[ M_1 || M_2 \models P \]

How to automatically construct assumptions?
Challenge: \( g_1 \) and \( g_2 \) depend on each other
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**
Thank You