Automating Compositional Verification

Dimitra Giannakopoulou, NASA Ames
collaborators

- Corina Păsăreanu (CMU / NASA Ames)

- and talented students / visitors:
  - Howard Barringer (Univ. of Manchester)
  - Colin Blundell (UPenn)
  - Jamieson Cobleigh (UMass, now MathWorks)
  - Michael Emmi (UCLA)
  - Mihaela Gheorgiu (Univ. of Toronto)
  - Chang-Seo Park (UC Berkeley)
  - Suzette Person (Univ. of Nebraska)
  - Rishabh Singh (MIT)
state-explosion problem
compositional verification

does system made up of $M_1$ and $M_2$ satisfy property $P$?

- check $P$ on entire system: too many states!
- use system’s natural decomposition into components to break-up the verification task
- check components in isolation:

```
M_1
```

```
M_2
```

```
A
```

does $M_1$ satisfy $P$?
“when we try to pick out anything by itself, we find it hitched to everything else in the universe”

John Muir
introduces assumptions / reasons about triples:

\[ \langle A \rangle M \langle P \rangle \]

is true if whenever M is part of a system that satisfies A, then the system must also guarantee P

simplest assume-guarantee rule (ASYM):

1. \[ \langle A \rangle M_1 \langle P \rangle \]
2. \[ \langle true \rangle M_2 \langle A \rangle \]

\[ \langle true \rangle M_1 || M_2 \langle P \rangle \]

“discharge” the assumption

how do we come up with the assumption?
given component $M$, property $P$, and the interface $\Sigma$ of $M$ with its environment, generate the **weakest** environment assumption $WA$ such that: $\langle WA \rangle M \langle P \rangle$ holds

weakest means that for all environments $E$:

$\langle true \rangle M \parallel E \langle P \rangle$ IFF $\langle true \rangle E \langle WA \rangle$
weakest assumption in AG reasoning

1. \( \langle A \rangle M_1 \langle P \rangle \)
2. \( \langle true \rangle M_2 \langle A \rangle \)

\[
\langle true \rangle M_1 \parallel M_2 \langle P \rangle
\]

weakest assumption makes rule complete

for all \( E \), \( \langle true \rangle M \parallel E \langle P \rangle \) IFF \( \langle true \rangle E \langle WA \rangle \)

\[
\langle true \rangle M_1 \parallel M_2 \langle P \rangle \) IFF \( \langle true \rangle M_2 \langle WA \rangle \\
\text{in other words:} \\
\langle true \rangle M_2 \langle WA \rangle \) holds implies \( \langle true \rangle M_1 \parallel M_2 \langle P \rangle \) holds \\
\langle true \rangle M_2 \langle WA \rangle \) not holds implies \( \langle true \rangle M_1 \parallel M_2 \langle P \rangle \) not holds
formalisms

- components modeled as **finite state machines** (FSM)
  - FSMs assembled with parallel composition operator “||”
    - synchronizes shared actions, interleaves remaining actions

- a safety property P is a **FSM**
  - P describes all legal behaviors in terms of its alphabet
  - P\(_{err}\) — complement of P
    - determinize & complete P with an “error” state;
    - bad behaviors lead to error
  - component M satisfies P iff error state unreachable in (M || P\(_{err}\))

- **assume-guarantee** reasoning
  - assumptions and guarantees are FSMs
  - \(\langle A \rangle M \langle P \rangle\) holds iff error state unreachable in (A || M || P\(_{err}\))
require in and out to alternate (property Order)

Input

Output

Order_{err}

Input

Output
parallel composition
crex. 1: \((I_0, O_0)\) out \((I_0, O_{\text{error}})\)
crex. 2: \((I_0, O_0)\) in \((I_1, O_1)\) send \((I_2, O_1)\) out \((I_2, O_0)\) out \((I_2, O_{\text{error}})\)
assume-guarantee reasoning

Input

Assumption

crex 1: \((I_0, A_0, O_0)\) out \(X\)
crex 2: \((I_0, A_0, O_0)\) in \((I_1, A_0, O_1)\) send \((I_2, A_1, O_1)\) out \((I_2, A_0, O_0)\) out \(X\)
iterative solution +
intermediate results

$L^*$ learns unknown regular language $U$ (over alphabet $\Sigma$) and produces minimal DFA $A$ such that $L(A) = U$

($L^*$ originally proposed by Angluin)
L* learner

queries:
should word w be included in L(A)?

conjectures:
here is an A – is L(A) = U?

the oracle

yes / no

yes!

no: word w should (not) be in L(A)
oracle for WA in assume-guarantee reasoning

1. \( \langle A \rangle M_1 \langle P \rangle \)
2. \( \langle true \rangle M_2 \langle A \rangle \)
   \[ \langle true \rangle M_1 \parallel M_2 \langle P \rangle \]

\[
\langle WA \rangle M_1 \langle P \rangle \text{ holds}
\]
\[
\langle true \rangle M_2 \langle WA \rangle \text{ holds implies } \langle true \rangle M_1 \parallel M_2 \langle P \rangle \text{ holds}
\]
\[
\langle true \rangle M_2 \langle WA \rangle \text{ does not hold implies } \langle true \rangle M_1 \parallel M_2 \langle P \rangle \text{ does not hold}
\]
characteristics

assumptions conjectured by L* are not comparable semantically

- terminates with *minimal* automaton \( A \) for \( U \)
- generates DFA candidates \( A_i: |A_1| < |A_2| < \ldots < |A| \)
- produces at most \( n \) candidates, where \( n = |A| \)
- \# queries: \( O(kn^2 + n \log m) \),
  - \( m \) is size of largest counterexample, \( k \) is size of alphabet
- for assume-guarantee reasoning, may terminate early with a smaller assumption than the weakest
we check: \( \langle \text{true} \rangle \) Input || Output \( \langle \text{Order} \rangle \)

\(M_1 = \text{Input}, \ M_2 = \text{Output}, \ P = \text{Order} \)

assumption alphabet: \{send, out, ack\}
queries

\[ S = \text{set of prefixes} \]
\[ E = \text{set of suffixes} \]

### Table \( T \)

<table>
<thead>
<tr>
<th>( S \cdot \Sigma )</th>
<th>( E )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda )</td>
<td>true</td>
</tr>
<tr>
<td>out</td>
<td>false</td>
</tr>
<tr>
<td>ack</td>
<td></td>
</tr>
<tr>
<td>out, ack</td>
<td></td>
</tr>
<tr>
<td>out, out</td>
<td></td>
</tr>
<tr>
<td>out, send</td>
<td></td>
</tr>
</tbody>
</table>

\[ S = \{ \lambda, \text{out} \} \]

**Order**

\[ \text{Input} \]

\[ \text{Output} \]
candidate construction

\[ S = \text{set of prefixes} \]
\[ E = \text{set of suffixes} \]

**Table T**

<table>
<thead>
<tr>
<th>( S )</th>
<th>( \lambda )</th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( S \cdot \Sigma )</th>
<th>( \text{out, ack} )</th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{true, false} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \text{ack, send} )</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>( \text{out, out} )</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>( \text{out, send} )</td>
<td>false</td>
</tr>
</tbody>
</table>

\[ \text{2 states – error state omitted} \]

**Assumption A_1**

\[ \text{counterexamples add to } S \]
conjectures

Oracle 1: \(\langle A_1 \rangle\) Input \langle Order \rangle

Order_{err}

Counterexample:
\(c = \langle \text{in,send,ack,in} \rangle\)

Oracle 1:
\(\langle A_2 \rangle\) Input \langle Order \rangle

Oracle 2:
\(\langle \text{true} \rangle\) Output \langle A_2 \rangle

property \text{Order} holds on Input \| Output
more than 2 components [TACAS03, FMSD09]
symmetric rules: motivation

\[ M_1 = \text{Input}, \ M_2 = \text{Output}, \ P = \text{Order} \]

\[ A_1: \quad \text{ack, send} \]

\[ \text{Order}_{err} \]

\[ M_2 = \text{Output}, \ M_1 = \text{Input}, \ P = \text{Order} \]

\[ A_1: \quad \text{ack, in} \]

\[ A_2: \quad \text{in, send} \]

\[ A_4: \quad \text{ack, out, send} \]
symmetric learning framework [SAVCBS05]

\[
\begin{align*}
L^* & \Rightarrow A_1 \Rightarrow \langle A_1 \rangle M_1 \langle P \rangle \\
L^* & \Rightarrow A_2 \Rightarrow \langle A_2 \rangle M_2 \langle P \rangle \\
L(\text{co}A_1 \parallel \text{co}A_2) & \subseteq L(P) \\
\text{counterexample analysis} & \\
P \text{ holds in } M_1 \parallel M_2 & \\
P \text{ violated in } M_1 \parallel M_2
\end{align*}
\]
- beyond syntactic interfaces
  - *(open file before close)*
- document implicit assumptions
- **safe:** accept NO illegal sequence of calls
- **permissive:** accept ALL legal sequences of calls
- **safe & permissive interface = weakest assumption**
should word \( w \) be included in \( L(A) \)?

yes / no

is \( A \) safe and permissive?

yes!

no: word \( w \) should (not) be in \( L(A) \)
checkSafe(interface A, FSM M)
checkPermissive(interface A, FSM M)

if M is non-deterministic, permissiveness check requires subset construction
permissiveness heuristics [FASE 2009]

\[(A_{\text{err}} \parallel M)\]

- model check for (err, ok)
- reached (err, ok) by “p” query “p”
- no (“p” should not be in A)
- backtrack & continue search…

resolves non-determinism

dynamically & selectively;
remember, it’s a heuristic
JavaPathfinder
UML statecharts

assume-guarantee reasoning

interface generation / discharge

jpf-cv

http://babelfish.arc.nasa.gov/trac/jpf
infinite components [CAV 2010]

- use predicate abstraction (e.g., \(x \geq 0\), \(x < 0\))
- generate may and must abstraction

an interface **safe** w.r.t. \(C_{\text{may}}\) and **permissive** w.r.t. \(C_{\text{must}}\)

**is safe** and **permissive** w.r.t. concrete component \(C\)
1. if `checkSafe(\sigma, C^{\text{must}})` \(!=\) null
2. \quad return “no”
3. \(cex = checkSafe(\sigma, C^{\text{may}})\)
4. if `cex` == null
5. \quad return “yes”
6. \(\text{Preds} = \text{Preds} \cup \text{Refine}(cex)\)
7. `Query(\sigma, C)`

*If concrete component is deterministic, so is the must abstraction…*

*ARMC model checker: Java2SDK library classes, OpenSSL, NASA CEV model*
related work

- learning with alphabet refinement (TACAS 2007; also Chaki et al.)
- learning assumptions for interface automata (FM 2008)
- assume-guarantee abstraction refinement (CAV 2008)

- compositional verification in symbolic setting (Alur et al. 05)
- minimal assumptions as separating automata for languages $L(M_2)$ and $L(M_1) \cap L(coP)$ (Gupta et al. 07, Chen et al. 09)
- learning omega-regular languages for liveness (Farzan et al. 08)
- learning non-deterministic automata (Bollig et al. 09)
- learning Boolean functions (Chen et al. 10)
- assumption generation in probabilistic setting (Feng et al. 10)
summary and food for thought...

- techniques are generic
- better applied at design level
- not a panacea...
  - perform well when alphabets & assumptions are small

- what makes a system amenable to compositional techniques?
- design for compositional verification; combine with other design approaches
- how can we make it practical for real systems? what types of interfaces are useful in practice?
- discovering good system decompositions
- liveness, timed & probabilistic systems, non functional properties
- multi core / parallelization?
thank you!
invoke a model checker **within** a model checker?
permissiveness check

MC: model check for \((M_i, A_{\text{error}})\)

reached \((\text{err, ok})\) by trace \(t\)

if \((\text{memoized}(t) == \text{no})\) // \(t\) is spurious
    backtrack and continue search
else  // \(\text{memoized}(t) == \text{yes}\) or \(t\) not in memoized
    model checker produces \(t\)

if \((\text{query}(t) == \text{yes})\)
    return \(t\) to \(L^*\) // not permissive
else restart at MC
1. $\text{cex} = \text{checkSafe}(A, C^{\text{may}})$
2. if $\text{cex} == \text{null}$
3. invoke Oracle2
4. If $\text{Query}(\text{cex, C}) == \text{“false”}$
5. return $\text{cex}$ to $L^*$
6. else
7. goto 1
1. cex = checkPermissive(A, C^{must})
2. if cex == null
3. return A
4. If Query(cex, C) == "true"
5. return cex to L*
6. else
7. goto 1
example 1: Mars Exploration Rover

- tools: LTSA, SPIN
- model derived from JPL’s Mars Exploration Rover (MER) Resource Arbiter
  - local management of resource contention between resource consumers (e.g. science instruments, communication systems)
  - consists of \( k \) user threads and one server thread (arbiter)
- checked mutual exclusion between resources (e.g. driving while capturing a camera image are incompatible)
- compositional verification scaled to >5 users vs. monolithic verification ran out of memory [SPIN’ 06]
example 2: autonomous rendezvous & docking

- **tool**: LTSA
- consists of control software, state estimator, and 4 types of sensors
- input provided as UML state-charts, properties of type:
  - “you need at least two operational sensors to proceed to next mode”
- 3 bugs detected
- **scaling achieved with compositional verification:**
  - monolithic verification runs out of memory after > 13M states
  - compositional verification terminates successfully in secs. Largest state-space explored is less than 60K states, as opposed to > 13M.
example 3: K9 Rover Executive

- tools: LTSA, JavaPathfinder
- model of NASA Ames K9 Rover Executive
  - executes flexible plans for autonomy
  - consists of Executive thread and ExecCondChecker thread for monitoring state conditions
  - checked for specific shared variable: if Executive reads its value, ExecCondChecker should not read the variable before the Executive clears it

- generated assumption of 6 states for model in LTSA [TACAS 2003]
- used generated assumption to check 8K lines of JAVA code translated from 10K lines of C++ code using the JavaPathfinder model checker [ICSE 2004]
- reduced memory used by JavaPathfinder > 3 times
- used generated assumption to perform assume-guarantee testing of C++ code using Eagle runtime monitoring framework [SAVCBS 2005, IET Software 2009]