Automated Compositional Verification

Corina Păsăreanu
CMU Silicon Valley/ NASA Ames Research Center
Part I
- assume-guarantee reasoning
- computing assumptions
- learning assumptions
- multiple components
- different assume-guarantee rules

Part II
- alphabet refinement
- assume-guarantee abstraction refinement
- reasoning about code
- related work
- conclusion
Dimitra Giannakopoulou and … many collaborators
program / model

```java
void add(Object o) {
    buffer[head] = o;
    head = (head+1)%size;
}

Object take() {
    ...
    tail=(tail+1)%size;
    return buffer[tail];
}
```

property

always(φ or ψ)

model checker

YES (property holds)

NO + counterexample:

```
Line 5: ...
Line 12: ...
... Line 41: ...
Line 47: ...
```
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:
  - Does $M_1$ satisfy $P$?
    - typically a component is designed to satisfy its requirements in specific contexts
- Assume-guarantee reasoning
  - introduces assumption $A$ representing $M_1$’s context

[Misra&Chandy 81, Jones 83, Pnueli 84]
assume-guarantee reasoning

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 (\textit{ASYM}):  

1. \[ \langle A \rangle M_1 \langle P \rangle \]
2. \[ \langle true \rangle M_2 \langle A \rangle \]
\[ \frac{}{\langle true \rangle M_1 \parallel M_2 \langle P \rangle} \]

“discharge” the assumption
- no file “close” before “open”
- accesses to shared variable “X” must be protected by lock “L”
- *(rover executive)* whenever thread “A” reads variable “V”, no other thread can read “V” before thread “A” clears it first
- *(spacecraft flight phases)* a docking maneuver can only be invoked if the launch abort system has previously been jettisoned from the spacecraft
how do we compute assumptions?
components modeled as **finite state machines** (FSM)
- FSMs assembled with parallel composition operator “||”
- synchronizes shared actions, interleaves remaining actions

**alphabet of M**: \( \alpha M \)

**language of M**: \( \mathcal{L}(M) \)
- the set of all traces (with \( \tau \) removed) in \( M \)

**projection**
- \( t \uparrow \sum \) – the trace obtained from \( t \) by removing all actions not in \( \sum \)
- \( M \uparrow \sum \) – replace with \( \tau \) all actions not in \( \sum \)
a safety property $P$ is a FSM
  - $P$ describes all legal behaviors in terms of its alphabet

property satisfaction: $M \models P$ iff $L(M \uparrow_\alpha P) \subseteq L(P)$

alternative definition
  - $P_{err}$ — complement of $P$ (determinize & complete $P$ with “error” state)
  - bad behaviors lead to error

$M \models P$ iff error state unreachable in $(M \parallel P_{err})$

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

Input

\[\text{in} \quad \text{send} \quad \text{ack}\]

Output

\[\text{send} \quad \text{out} \quad \text{ack}\]
property satisfaction

cex. 1: \((I_0, O_0)\) out \((I_0, O_{\text{error}})\)
cex. 2: \((I_0, O_0)\) in \((I_1, O_1)\) send \((I_2, O_1)\) out \((I_2, O_0)\) out \((I_2, O_{\text{error}})\)
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**

- Prevents component to go to error (**safe**)
- Should be as **permissive** as possible
- Uses only **interface** actions
STEP 1: composition with $P_{err}$, hiding internals, minimization

STEP 2: backward propagation of error along $\tau$ transitions

STEP 3: property extraction (subset construction & completion)

property true! (all environments)

property false! (all environments)

assumption
Input $||\text{Order}_{\text{err}} \setminus \{\text{in}\}$
step 2: error propagation with $\tau$
step 3: subset construction
step 3: subset construction
step 3: property construction
weakest assumption in AG reasoning

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

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

weakest assumption makes rule complete

\(\langle WA \rangle M_1 \langle P \rangle\) holds (WA could be false)
\(\langle \text{true} \rangle M_2 \langle WA \rangle\) holds implies \(\langle \text{true} \rangle M_1 \parallel M_2 \langle P \rangle\) holds
\(\langle \text{true} \rangle M_2 \langle WA \rangle\) not holds implies \(\langle \text{true} \rangle M_1 \parallel M_2 \langle P \rangle\) not holds
learning assumptions

iterative solution

intermediate results
L* algorithm by Angluin, improved by Rivest & Schapire
Learns an unknown regular language $U$ (over alphabet $\Sigma$)
Produces a DFA $A$ such that $\mathcal{L}(A) = U$
Uses a teacher to answer two types of questions

**Query:**
- $s$: string $s$
  - Is $s$ in $U$?
  - True
  - False

**Conjecture:**
- $A_i$: conjecture
  - Is $\mathcal{L}(A_i) = U$?
    - True
    - False

**Actions:**
- Remove string $t$
- Add string $t$
Use \( L^* \) to generate candidate assumptions
\[
\alpha A = (\alpha M_1 \cup \alpha P) \cap \alpha M_2
\]
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\}
$S = \text{set of prefixes}$

$E = \text{set of suffixes}$
**Table T**

<table>
<thead>
<tr>
<th>$S$</th>
<th>$\lambda$</th>
<th>$\text{true}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{out}$</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

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

$S = \text{set of prefixes}$

$E = \text{set of suffixes}$

**Order$_{err}$**

Counterexamples add to $S$

2 states – error state omitted

Assumption $A_1$
A_1:

Oracle 1: \(<A_1>\) Input \(<Order>\)

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

Return to L*: \(c \uparrow \Sigma = \langle\text{send,ack}\rangle\)

A_2:

Oracle 1: \(<A_2>\) Input \(<Order>\)

Oracle 2: \(<\text{true}>\) Output \(<A_2>\)

property \text{Order holds on Input || Output}
Check if $M_1 \parallel M_2 \parallel \ldots \parallel M_n$ satisfies $P$

- decompose it into $M_1$ and $M_2 = \ldots = M_n$

1. $\langle A_1 \rangle M_1 \langle P \rangle$
2. $\langle true \rangle M_2 \langle A_1 \rangle$

$\langle true \rangle M_1 \parallel M_2 \ldots \parallel M_n \langle P \rangle$

apply learning framework recursively for 2nd premise of rule

1. $\langle A_2 \rangle M_2 \langle A_1 \rangle$
2. $\langle true \rangle M'_3 \langle A_2 \rangle$

$\langle true \rangle M_2 \ldots \parallel M_n \langle A_1 \rangle$

\[ \ldots \]
Mars Exploration Rover (MER) Resource Arbiter

- Local management of resource contention between resource consumers
- E.g. science instruments, communication systems
- k user threads and one server thread (arbiter)

Mutual exclusion between resources
- E.g. driving while capturing a camera image are mutually incompatible
Compute $A_1$ … $A_5$ s.t.
\[
\langle A_1 \rangle U_1 \langle P \rangle \& \langle \text{true} \rangle U_2 \| U_3 \| U_4 \| U_5 \| \text{ARB} \langle A_1 \rangle \\
\langle A_2 \rangle U_2 \langle A_1 \rangle \& \langle \text{true} \rangle U_3 \| U_4 \| U_5 \| \text{ARB} \langle A_2 \rangle \\
\langle A_3 \rangle U_3 \langle A_2 \rangle \& \langle \text{true} \rangle U_4 \| U_5 \| \text{ARB} \langle A_2 \rangle \\
\langle A_4 \rangle U_4 \langle A_3 \rangle \& \langle \text{true} \rangle U_5 \| \text{ARB} \langle A_4 \rangle \\
\langle A_5 \rangle U_5 \langle A_4 \rangle \& \langle \text{true} \rangle \text{ARB} \langle A_5 \rangle
\]

Result: $\langle \text{true} \rangle U_1 \| .. \| U_5 \| \text{ARB} \langle P \rangle$ holds

Compositional verification scaled to >5 users while monolithic verification ran out of memory
[SPIN’ 06]
symmetric rule

1. $\langle A_1 \rangle M_1 \langle P \rangle$
2. $\langle A_2 \rangle M_2 \langle P \rangle$
3. $P \models A_1 \land A_2$

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

1. $\langle A_1 \rangle M_1 \langle P \rangle$
2. $\langle A_2 \rangle M_2 \langle P \rangle$
3. $P \models A_1 \land A_2$

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

Unsound!
Not sound if $\rightarrow$ is interpreted as logical implication

Use induction over time steps:
- $P$ holds initially in $M$
- If assumption $A$ holds up to the $k$-th step in any trace of $M$, then
- Guarantee $P$ holds up to the $k+1$-th step in that trace, for all $k \geq 0$
symmetric rule [SAVCBS05]

1. $\langle A_1 \rangle M_1 \langle P \rangle$
2. $\langle A_2 \rangle M_2 \langle P \rangle$
3. $\mathcal{L} (\text{coA}_1 \parallel \text{coA}_2) \subseteq \mathcal{L} (P)$

$\langle \text{true} \rangle M_1 \parallel M_2 \langle P \rangle$

coA$_i$ = complement of A$_i$, for i=1,2
$\alpha P \subseteq \alpha M_1 \cup \alpha M_2$;  $\alpha A_i \subseteq (\alpha M_1 \cap \alpha M_2) \cup \alpha P$, for i =1,2

Common traces ruled out by assumptions satisfy P

Sound!
learning framework [SAVCBS05]

\[
\begin{align*}
\langle A_1 \rangle M_1 \langle P \rangle & \quad \text{false} \\
L^* \quad A_1 & \quad \text{true} \\
L(coA_1 || coA_2) & \subseteq L(P) \\
\text{counterex. analysis} & \quad \text{false} \\
P & \text{holds in } M_1 || M_2 \\
P & \text{violated in } M_1 || M_2
\end{align*}
\]
Similar to asymmetric rule
- Applied recursively to 3 components
- First and last component coincide
- Hence learning framework similar
Automated Compositional Verification part 2

Corina Păsăreanu
CMU Silicon Valley/ NASA Ames Research Center
Part I
- assume-guarantee reasoning
- computing assumptions
- learning assumptions
- multiple components
- different assume-guarantee rules

Part II
- alphabet refinement
- assume-guarantee abstraction refinement
- reasoning about code
- related work
- conclusion
Compositional verification

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

- Check $P$ on entire system: too many states!
- Use the natural decomposition of the system into its components to break-up the verification task
- Check components in isolation:
  - Typically a component is designed to satisfy its requirements in specific contexts / environments
- Assume-guarantee reasoning:
  - Introduces assumption $A$ representing $M_1$’s “context”
assume-guarantee reasoning

- Reason about triples:
  \[ \langle A \rangle M \langle P \rangle \]
  The formula 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

\[
\begin{align*}
1. & \quad \langle A \rangle M_1 \langle P \rangle \\
2. & \quad \langle true \rangle M_2 \langle A \rangle \\
\hline
\quad & \quad \langle true \rangle M_1 || M_2 \langle P \rangle
\end{align*}
\]

“discharge” the assumption

How do we come up with the assumption A?
(usually a difficult manual process)

**Solution**: synthesize A automatically
Given component M, property P, and the interface of M with its environment, generate the weakest environment assumption **WA** such that: \( \langle \text{WA} \rangle M \langle P \rangle \) holds

Weakest means that for all environments E:

\[
\langle \text{true} \rangle M \parallel E \langle P \rangle \iff \langle \text{true} \rangle E \langle \text{WA} \rangle
\]
STEP 1: composition, hiding, minimization

STEP 2: backward propagation of error along $\tau$ transitions

STEP 3: property extraction (subset construction & completion)

property true! (all environments)

property false! (all environments)

assumption
use an off-the-shelf learning algorithm to build assumption

- process is *iterative*
- assumptions generated by querying the system, gradually refined
- queries answered by model checking
- refinement based on counterexamples
learning assumptions

- Use $L^*$ to generate candidate assumptions
- $\alpha A = (\alpha M_1 \cup \alpha P) \cap \alpha M_2$

---

**Model Checking**

- **query**: string $s$
  - $\langle s \rangle M_1 \langle P \rangle$
- **false**
  - string $c \uparrow \alpha A$
  - $\langle A_i \rangle M_1 \langle P \rangle$
  - **false** + cex $c$
  - $\langle \text{true} \rangle M_2 \langle A_i \rangle$
  - **true**
- **false**
  - string $c \uparrow \alpha A$
  - $\langle c \uparrow \alpha A \rangle M_1 \langle P \rangle$
  - **false**

---

1. $\langle A \rangle M_1 \langle P \rangle$
2. $\langle \text{true} \rangle M_2 \langle A \rangle$

$\langle \text{true} \rangle M_1 \parallel M_2 \langle P \rangle$

- guaranteed to terminate
- reaches weakest assumption or terminates earlier
rule ASYM
- Assumption alphabet was fixed during learning
- $\alpha A = (\alpha M_1 \cup \alpha P) \cap \alpha M_2$

[SPIN’06]: A subset alphabet
- may be sufficient to prove the desired property
- may lead to smaller assumption
interface action d not relevant for the property

- e.g. may never appear on a path to error
- no need to include in alphabet assumption
- results in smaller assumption
How do we compute a good subset of the assumption alphabet?

Solution: iterative alphabet refinement

- Start with small alphabet
- Apply learning framework
- Add actions as necessary
- Discovered by analysis of counterexamples from model checking
1. Initialize $\Sigma$ to subset of alphabet $\alpha A = (\alpha M_1 \cup \alpha P) \cap \alpha M_2$

2. If learning with $\Sigma$ returns true, return true and go to 4. (END)

3. If learning returns false (with counterexample $c$), perform extended counterexample analysis on $c$.

   If $c$ is real, return false and go to 4. (END)

   If $c$ is spurious, add more actions from $\alpha A$ to $\Sigma$ and go to 2.

4. END
extended counterexample analysis

\[ \alpha A = (\alpha M_1 \cup \alpha P) \cap \alpha M_2 \]
\[ \Sigma \subseteq \alpha A \text{ is the current alphabet} \]

**Refiner:** compare \( c \uparrow \alpha A \) and \( t \uparrow \alpha A \)

Add actions to \( \Sigma \) and restart learning
let $A$ be assumption resulted from $L^*$ with full interface alphabet $\alpha A$

during alphabet refinement:

- assumptions $A', A''$ with alphabets $\alpha A' \subseteq \alpha A'' \subseteq \alpha A$
- are under-approximations, i.e.,

$$L(A') \subseteq L(A'') \subseteq L(A)$$

See also learning with optimal alphabet refinement
- developed independently by Chaki & Strichman 07
alphabet refinement

\[ \Sigma = \{ \text{out} \} \quad \alpha A = \{ \text{send, out, ack} \} \]

\[ \langle \text{true} \rangle \text{ Output } \langle A_i \rangle \rightarrow \text{false with } t = \langle \text{send, out} \rangle \]

\[ \langle t^{\uparrow} \Sigma \rangle \text{ Input } \langle P \rangle \rightarrow \text{false with counterex. } c = \langle \text{out} \rangle \]

\[ \langle t^{\uparrow} \alpha A \rangle \text{ Input } \langle P \rangle \rightarrow \text{true Not a real counterexample!} \]

compare \langle \text{out} \rangle \text{ with } \langle \text{send, out} \rangle \rightarrow \text{add “send” to } \Sigma \]
comparison with original learning

Assumption size

Time

thanks Mihaela Bobaru
comparison with non-compositional (memory)

thanks Mihaela Bobaru
Rule ASYM more effective than rules SYM and CIRC

Recursive version of ASYM the most effective
  – When reasoning about more than two components

Alphabet refinement improves learning based assume guarantee verification significantly

Learning based assume guarantee reasoning
  – Can incur significant time penalties
  – Not always better than non-compositional (monolithic) verification
  – Sometimes, significantly better in terms of memory
abstraction
Instead of learning $A$, build $A$ as an over-approximating abstraction of $M_2$

Why?
- Use “more” information from $M_1$ and $M_2$
- Nondeterministic assumptions can be exponentially smaller than deterministic ones

[CAV’08]
**Existential abstraction**
- Maps concrete states in $M_2$ to abstract states in $A$
- Add abstract transition in $A$ if exists “corresponding” concrete transition in $M_2$
  - $L(M_2^{\uparrow \alpha A}) \subseteq L(A)$

**Premise 2:** $\langle true \rangle M_2 \langle A \rangle$ holds by construction
Variant of CEGAR with differences:
- use counterexample from $M_1$ to refine abstraction of $M_2$
- $A$ keeps information only about the interface (abstracts away the internals)

Assumptions and number of iterations bounded by $|M_2|$
example: AGAR results

Input:

Output:

A₁:

Check: \langle A₁ \rangle M₁ \langle P \rangle
Abstract cex.:
{0,1,2}, out, {0,1,2}
example: AGAR results

Input:

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>in</td>
</tr>
<tr>
<td>1</td>
<td>send</td>
</tr>
<tr>
<td>2</td>
<td>ack</td>
</tr>
</tbody>
</table>

Order_{err}:

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>in</td>
</tr>
<tr>
<td>1</td>
<td>out</td>
</tr>
<tr>
<td>1</td>
<td>out</td>
</tr>
<tr>
<td>1</td>
<td>in</td>
</tr>
</tbody>
</table>

Output':

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>send</td>
</tr>
<tr>
<td>1</td>
<td>out</td>
</tr>
<tr>
<td>1</td>
<td>ack</td>
</tr>
</tbody>
</table>

\( A_1: \)

<table>
<thead>
<tr>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>ack</td>
</tr>
<tr>
<td>send</td>
</tr>
<tr>
<td>out</td>
</tr>
</tbody>
</table>

\( A_2: \)

<table>
<thead>
<tr>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>ack</td>
</tr>
<tr>
<td>{0,2}</td>
</tr>
<tr>
<td>send</td>
</tr>
<tr>
<td>{1}</td>
</tr>
</tbody>
</table>

Check: \( \langle A_1 \rangle M_1 \langle P \rangle \)
Abstract cex.:
\{0,1,2\}, out, \{0,1,2\}
example: learning results

Input:

Order_{err}:

Output':

A_1:

A_2:

A_3:

A_4:
### Table 1. Comparison of AGAR and learning for 2 components, with and without alphabet refinement.

<table>
<thead>
<tr>
<th>Case</th>
<th>$k$</th>
<th>AGAR</th>
<th>Learning</th>
<th>AGAR</th>
<th>Learning</th>
<th>Sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mem.</td>
<td>Time</td>
<td>Mem.</td>
<td>Time</td>
<td>$M_1$</td>
</tr>
<tr>
<td>Gas Station</td>
<td>3</td>
<td>16</td>
<td>4.11 3.33</td>
<td>177</td>
<td>42.83 –</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>19</td>
<td>37.43 23.12</td>
<td>195</td>
<td>100.17 –</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>22</td>
<td>359.53 278.63</td>
<td>45</td>
<td>206.61 –</td>
<td>5</td>
</tr>
<tr>
<td>Chiron,</td>
<td>2</td>
<td>10</td>
<td>1.30 0.92</td>
<td>9</td>
<td>1.30 1.69</td>
<td>10</td>
</tr>
<tr>
<td>Property 2</td>
<td>3</td>
<td>36</td>
<td>2.59 5.94</td>
<td>21</td>
<td>5.59 7.08</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>160</td>
<td>8.71 152.34</td>
<td>39</td>
<td>27.1 32.05</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>55.14 –</td>
<td>111</td>
<td>569.23 676.02</td>
<td>3</td>
</tr>
<tr>
<td>Chiron,</td>
<td>2</td>
<td>4</td>
<td>1.07 0.50</td>
<td>9</td>
<td>1.14 1.57</td>
<td>4</td>
</tr>
<tr>
<td>Property 3</td>
<td>3</td>
<td>8</td>
<td>1.84 1.60</td>
<td>25 n jmj</td>
<td>4.45 7.72</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>16</td>
<td>4.01 18.75</td>
<td>45</td>
<td>25.49 36.33</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>52.53 –</td>
<td>122</td>
<td>134.21 271.30</td>
<td>1</td>
</tr>
<tr>
<td>MER</td>
<td>2</td>
<td>34</td>
<td>1.42 11.38</td>
<td>40</td>
<td>6.75 9.89</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>67</td>
<td>8.10 247.73</td>
<td>335</td>
<td>133.34 –</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>58</td>
<td>341.49 –</td>
<td>38</td>
<td>377.21 –</td>
<td>9</td>
</tr>
<tr>
<td>Rover Exec</td>
<td>2</td>
<td>10</td>
<td>4.07 1.80</td>
<td>11</td>
<td>2.70 2.35</td>
<td>3</td>
</tr>
</tbody>
</table>
AGAR vs learning

- **No alphabet refinement:**
  - 14 cases, better in 9 (assumption size), 12 (memory consumption), 10 (running time)

- **With alphabet refinement:**
  - 15 cases: better in 5, 7, 6, respectively

- **Problem: unbalanced decompositions**
  - Learning exercises more first component
  - AGAR dominated by second component
Balanced decompositions

- No alphabet refinement:
  - 9 cases, better in 8 (assumption size), 9 (memory consumption), 9 (running time)

- With alphabet refinement:
  - 12 cases, better in 7, 7, and 6, respectively
Does $M_1 \parallel M_2$ satisfy $P$? Model check; **build** assumption $A$

Does $C_1 \parallel C_2$ satisfy $P$? Model check; **use** assumption $A$

[ICSE’ 2004] – good results but may not scale

**Solution:** replace model checking with testing! [IET Software 2009]
Given component C compute weakest assumption (WA):

- **safe**: accept NO illegal sequence of calls
- **permissive**: accept ALL legal sequences of calls

C is not a model but an actual (infinite-state) implementation
may and must abstraction

- software component \( C \) may be \textit{infinite state}
- apply predicate abstraction
- \textit{may abstraction} produces a finite over-approximation
- \textit{must abstraction} produces a finite under-approximation
assumption generation for infinite-state components

- \( L_{\text{safe}}(C) = L_{\text{err}}(C) \)

  Interface A is safe: \( L(A) \subseteq L_{\text{safe}}(C) \)
  Interface A is permissive: \( L_{\text{safe}}(C) \subseteq L(A) \)

An interface A permissive w.r.t. C's must abstraction and safe w.r.t. C's may abstraction is safe and permissive for C.
assumption generation for infinite-state components

An interface $A$ permissive w.r.t. $C$'s $\textbf{must}$ abstraction and safe w.r.t. $C$'s $\textbf{may}$ abstraction is safe and permissive for $C$. 
assumption generation for infinite-state components

- conceptually simple
- if concrete component is deterministic, so is the must abstraction
- can use learning for building WA
- expensive learning restarts; need of tighter integration of abstraction refinement with L*
- LearnReuse method [CAV’08]
- ARMC model checker: Java2SDK library classes, OpenSSL, NASA CEV model
interface generation in Java PathFinder

- [FASE’09]
- uses L* with heuristic for permissiveness check
- applies to JPF’s state-chart extension
- recent work combines learning with symbolic execution [SAS’12]

![Diagram of Java PathFinder (JPF)]
compositional verification for probabilistic systems

- labeled transition systems with both probabilistic and non-deterministic behavior
- verification of strong simulation conformance
- counterexamples are probabilistic tree structures
- assume-guarantee abstraction refinement [CAV’12]
- learning from probabilistic tree samples [LICS’12]
Learning Assumptions for Compositional Verification, J. M. Cobleigh, D. Giannakopoulou, C. S. Pasareanu, TACAS’03.
Towards a Compositional SPIN, C. S. Pasareanu, D. Giannakopoulou, SPIN’06.
interface automata:

interfaces:

learning for AG reasoning:

learning for separating automata:

assume-guarantee reasoning for probabilistic systems: many papers from Oxford University

what about shared-memory communication, thread-modular reasoning, Owicki-Gries method?
conclusion

- techniques for automatic assumption generation and compositional verification
  - finite state models and safety properties
  - learning and abstraction
- data abstraction to deal with software implementations
- promising in practice – when interfaces are small

future

- discovering good system decompositions
- parallelization for increased scalability
- beyond safety: liveness, timed and probabilistic reasoning
- run-time analysis
- make it practical!
thank you!