Assume-Guarantee Reasoning for Deadlock

Sagar Chaki, Nishant Sinha
November 15, 2006
Overview

We present a framework that uses **learning** and **automated Assume-Guarantee (AG)** reasoning to detect **deadlocks**

- Concurrent systems with blocking message-passing communication
- Develop a notion of regular failure languages
- Propose a new kind of Failure Automata that accept such languages
- Develop an algorithm $L^F$ to learn deterministic FA that accept an unknown regular failure language
- Use $L^F$ to learn appropriate assumptions for deadlock detection
- Present experimental results
Finite LTS

\[ M = (Q, I, \Sigma, T) \]

- \( Q \equiv \) non-empty set of states
- \( I \in Q \equiv \) initial state
- \( \Sigma \equiv \) set of actions \( \equiv \) alphabet
- \( T \subseteq Q \times \Sigma \times Q \equiv \) transition relation

\[ \Sigma(M) = \{a, b, c, d, e, f\} \]
Components **handshake** (synchronize) over shared actions

- Else proceed independently (asynchronously)
- CSP semantics

Composition of $M_1 \& M_2 \equiv M_1 || M_2$

- State of $M_1 || M_2$ is of the form $(s_1,s_2)$ where $s_i$ is a state of $M_i$
Example

\[ M_1 \quad \Sigma = \{a, b, c\} \]

\[ M_2 \quad \Sigma = \{a, b', c\} \]

\[ M_1 \parallel M_2 \]
Deadlock

\[ M_1 \Sigma = \{a,b,b',c\} \]

\[ M_2 \Sigma = \{a,b,b',c\} \]

\[ 1,1' \xrightarrow{a} 2,2' \]

\[ M_1 \parallel M_2 \]

Deadlock
Deadlock and Composition
Deadlock and Composition

M₁

M₁ || M₂

M₁
Failures & Failure Languages

Trace $\in \Sigma^* = \text{sequence of actions}$

Refusal $\subseteq \Sigma = \text{set of actions}$

Failure $\in \Sigma^* \times 2^\Sigma = \text{a trace, followed by a refusal}$

\[
\Sigma = \{a,b\}
\]

\[
L(M_1) = \{ \lambda, \{b\}, a, \{a\}, ab, \{a\}, abb, \{a\}, \ldots \}
\]

\text{Downward closed}

\[
\Sigma = \{a,b,c\}
\]

\[
L(M_2) = \{ \lambda, \{b\}, c, \{b\}, cc, \{b\}, \ldots, a, \{a,c\}, ab, \{a,b,c\} \}
\]

\text{Deadlock}
AG Rule for Deadlock

Consider the following (idea for a) non-circular proof rule

\[ M_1 \parallel A \text{ does not deadlock} \]
\[ M_2 \leq A \]
\[ M_1 \parallel M_2 \text{ does not deadlock} \]  

AG-NC

We are interested in the largest A that satisfies the first premise.

- Under what conditions is such a language uniquely defined?
- What kind of automata accept such languages?
- Can we learn such automata efficiently?
Downward Closed Failure Languages

A failure language $L$ is downward closed if

$$\forall t \in \Sigma^*, \forall R, R' \in 2^\Sigma, (t,R) \in L \land R' \subseteq R \Rightarrow (t,R') \in L$$

There is always an unique maximal downward closed $A$ that satisfies the first premise of AG-NC.

Clearly, languages accepted by LTSs are downward closed.

However, the class of languages accepted by LTSs is simply too restricted.

We need automata with more general accepting conditions.
**Failure Automata (FLA)**

\[ A = (Q, I, \Sigma, T, F, \mu) \]

- \( Q, I, \Sigma, T \) defined as for LTSs
- \( F \subseteq Q \) is a set of final or accepting states
- \( \mu \) maps accepting sets to maximal refusal sets

\[ L(A_1) = L(M) \]

\[ L(A_2) = \text{maximal } A \text{ for } M \]
Some Results

A failure language is regular iff it is accepted by some FLA

- Deterministic FLA have the same accepting power as FLA in general
- Every regular failure language is accepted by a unique minimal DFLA

The maximal language satisfying premise #1 is unique and regular

- Hence accepted by an unique minimal DFLA

Deadlock can be expressed as a regular failure language containment problem: M does not deadlock iff \( L(M) \subseteq \text{No-DL} \) where \( \text{No-DL} = (\Sigma^* \times 2^\Sigma) - (\Sigma^* \times \{\Sigma\}) \) is the set of all non-deadlocking failures

\[
\begin{align*}
L(M_1 || A) & \subseteq \text{No-DL} \\
L(M_2) & \subseteq L(A) \\
L(M_1 || M_2) & \subseteq \text{No-DL}
\end{align*}
\]

AG-NC

Sound and Complete
Next Steps

We develop a learning algorithm $L^F$ that can learn any unknown regular failure language $U$

$L^F$ uses a minimally adequate teacher (MAT) that can answer two kinds of queries

- **Membership**: Given a failure $f$, does $f$ belong to $U$?
- **Candidate**: Given a DFLA $C$, is $L(C) = U$? If not, the MAT also returns a counterexample failure in the symmetric difference of $L(C)$ and $U$

We use $L^F$ to learn the maximal $A$

- MAT will be implemented via model checking

In case of a deadlock we return a counterexample witness
The Algorithm $L^F$

Maintains an observation table whose rows are labeled with traces and columns with failures. Iteratively does the following:

1) Build the table using membership queries

2) Constructs a candidate DFLA $C$ from the table and makes a candidate query with $C$

3) If candidate query succeeds, returns $C$ as the final answer

4) If candidate query fails, uses the counterexample to construct a new failure $f$ and adds $f$ to the columns of the table. Repeats from Step 1.

The new $f$ added ensures that the number of rows will increase strictly in the next iteration. Number of rows cannot exceed the number of states of the minimal DFLA accepting $U$. Hence $L^F$ always terminates and moreover, uses polynomial amount of resources.
Overall Deadlock Detection Procedure

Model Checking

\[ M_1 \not\models \text{No-DL} \]

\[ M_2 \models A \]

Learning with \( L^* \)

Membership queries are answered via simulation

\[ M_1 \times M_2 \not\models \text{no-DL} \]
Experimental Setup

Implemented AG-NC as well as the following circular rule:

\[
L(M_1 \parallel A_1) \subseteq \text{No-DL} \quad L(M_2 \parallel A_2) \subseteq \text{No-DL} \\
W(A_1) \parallel W(A_2) \subseteq \text{No-DL} \\
\hline
L(M_1 \parallel M_2) \subseteq \text{No-DL}
\]

Experimented with benchmarks derived from Linux device drivers and Inter-Process Communication library (synchronizing via locks) and Dining Philosophers

2.4 GHz machine with 2 GB RAM limit and 1 hour timeout
## Experimental Results: No Deadlock

<table>
<thead>
<tr>
<th>Exp</th>
<th>Loc</th>
<th>Comp</th>
<th>Non-Circ</th>
<th>Circular</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>T</td>
<td>M</td>
</tr>
<tr>
<td>MC</td>
<td>7272</td>
<td>2</td>
<td>308</td>
<td>903</td>
</tr>
<tr>
<td>MC</td>
<td>7272</td>
<td>4</td>
<td>*</td>
<td>1453</td>
</tr>
<tr>
<td>Ide</td>
<td>18905</td>
<td>3</td>
<td>338</td>
<td>50</td>
</tr>
<tr>
<td>Ide</td>
<td>18905</td>
<td>5</td>
<td>*</td>
<td>84</td>
</tr>
<tr>
<td>Synclink</td>
<td>17262</td>
<td>4</td>
<td>1547</td>
<td>19</td>
</tr>
<tr>
<td>Synclink</td>
<td>17262</td>
<td>6</td>
<td>*</td>
<td>27</td>
</tr>
<tr>
<td>Mxser</td>
<td>15717</td>
<td>3</td>
<td>2079</td>
<td>140</td>
</tr>
<tr>
<td>Mxser</td>
<td>15717</td>
<td>5</td>
<td>-</td>
<td>179</td>
</tr>
<tr>
<td>Tg3</td>
<td>36774</td>
<td>3</td>
<td>1568</td>
<td>118</td>
</tr>
<tr>
<td>Tg3</td>
<td>36774</td>
<td>6</td>
<td>-</td>
<td>157</td>
</tr>
<tr>
<td>IPC</td>
<td>818</td>
<td>3</td>
<td>703</td>
<td>338</td>
</tr>
<tr>
<td>DP</td>
<td>82</td>
<td>6</td>
<td>100</td>
<td>330</td>
</tr>
<tr>
<td>DP</td>
<td>109</td>
<td>8</td>
<td>1551</td>
<td>565</td>
</tr>
</tbody>
</table>

1 hour timeout; 2 GB memory limit; * = out of resource; - = no data
## Experimental Results: Deadlock

<table>
<thead>
<tr>
<th>Exp</th>
<th>Loc</th>
<th>Comp</th>
<th>Non-Circ</th>
<th>Circular</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>T</td>
<td>M</td>
</tr>
<tr>
<td>MC</td>
<td>7272</td>
<td>2</td>
<td>386</td>
<td>980</td>
</tr>
<tr>
<td>MC</td>
<td>7272</td>
<td>4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Ide</td>
<td>18905</td>
<td>3</td>
<td>*</td>
<td>80</td>
</tr>
<tr>
<td>Ide</td>
<td>18905</td>
<td>5</td>
<td>*</td>
<td>89</td>
</tr>
<tr>
<td>Synclink</td>
<td>17262</td>
<td>4</td>
<td>127</td>
<td>181</td>
</tr>
<tr>
<td>Synclink</td>
<td>17262</td>
<td>6</td>
<td>1188</td>
<td>*</td>
</tr>
<tr>
<td>Mxser</td>
<td>15717</td>
<td>3</td>
<td>657</td>
<td>364</td>
</tr>
<tr>
<td>Mxser</td>
<td>15717</td>
<td>5</td>
<td>3368</td>
<td>*</td>
</tr>
<tr>
<td>Tg3</td>
<td>36774</td>
<td>3</td>
<td>486</td>
<td>393</td>
</tr>
<tr>
<td>Tg3</td>
<td>36774</td>
<td>6</td>
<td>*</td>
<td>-</td>
</tr>
<tr>
<td>IPC</td>
<td>818</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DP</td>
<td>82</td>
<td>6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DP</td>
<td>109</td>
<td>8</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

1 hour timeout; 2 GB memory limit; * = out of resource; - = no data
Related Work

Use of learning for automated AG reasoning proposed originally by Cobleigh et al. [TACAS’03] for safety properties

Since been extended to simulation [CAV’05] and the use of symbolic techniques [CAV’05]

Brookes and Roscoe investigate failure semantics and its use for deadlock detection.

Assume-Guarantee reasoning is a rich area, but limited automation

Iterative abstraction-refinement has also been used in the context of compositional deadlock detection [MEMOCODE’03]
Questions?

chaki@sei.cmu.edu
nishants@cs.cmu.edu