Formal Verification by Model Checking

Natasha Sharygina
Carnegie Mellon University

Outline

*Lecture 1:* Overview of Model Checking

*Lecture 2:* Complexity Reduction Techniques

*Lecture 3:* Software Model Checking

*Lecture 4:* State/Event-based software model checking

*Lecture 5:* Component Substitutability

*Lecture 6:* Model Checking Practicum (Student Reports on the Lab exercises)
Summary of the last lecture:
How does Spin work?

- We already saw:
  - The Algorithm
  - The Promela Language
- We need to see how the tool works.

High Level Organization

The Buchi automaton is turned into a Promela process and composed with the rest of the system.

The generated verifier is specific to the model and property we started with.
Command Line Tools

- Spin
  - Generates the Promela code for the LTL formula
    ~$ spin -f "[]<>p"
    - The proposition in the formula must correspond to the model declarations
  - Generates the C source code
    ~$ spin -a source.pro
    - The property must be included in the source

- Pan
  - Performs the verification
    - Has many compile time options to enable different features
    - Optimized for performance

- GUI for Spin
  Xspin
Simulator

• Spin can also be used as a simulator
  – Simulated the Promela program
• It is used as a simulator when a counterexample is generated
  – Steps through the trace
  – The trace itself is not “readable”
• Can be used for random and manually guided simulation as well

Comments

• DFS does not necessarily find the shortest counterexample
• There might be a very short counterexample but the verification might go out of memory
• If we don’t finish we might still have some sort of a result (coverage metrics)
Today’s Lecture

Advanced Techniques for Model Checking Software

(ComFoRT project)

Objectives

• C program and high-level designs verification
  – Sequential and concurrent

• Properties involve both data (states) and communication (events)
  – Specified as (State/Event) LTL formulas
  – Safety and Liveness

• Communication via shared actions
  – Synchronous communication
  – Asynchronous execution
Bluetooth L2CAP Spec

“When an L2CAP_ConnectRsp event is received in a W4_L2CAP_CONNECT_RSP state, an L2CAP process may send out an L2CA_ConnectInd event, disable the RTX timer, and move to state CONFIG.”

Spec involves both states and events

Labelled Kripke Structures

- Directed graph with labels on edges and states, \((S, \text{Init}, P, L, T, \Sigma, E)\)
  - Every state is labeled with a set of atomic propositions, \(P\), true in the state
  - Every LKS comes with an alphabet of actions, \(\Sigma\)

- State labeling function : \(L : S \rightarrow 2^P\)
- Transition labeling function : \(E : T \rightarrow (2^\Sigma \setminus \{\})\)
  - Assumption: LKSs are deadlock-free
    [see deadlock detection algorithm, MEMOCODE'04]
Traces and Languages

- **Trace**: infinite alternating sequence of states and actions
  - $(s_1 \ a \ s_2 \ b \ s_4 \ b \ s_1 \ldots)$

- **Language**: set of all traces
  - $L(M) = \{s_1 \ a \ (s_2 \ b + s_3 \ c) \ s_4 \ b\}^\omega$

---

Surge Protector: State/Event

State/Event model of the Surge Protector

(example is given for $m$: [0..2], $c$: [0..2])
Surge Protector : State Only

Kripke structure of the Surge Protector
(example is given for m: [0..2], c: [0..2])

State/Event LTL

Given LKS $M = (S, \text{Init}, P, L, T, \Sigma, E)$, and $p \in P$, $a \in \Sigma$,

$$\varphi ::= p \mid a \mid \neg \varphi \mid \varphi \& \varphi \mid X\varphi \mid G\varphi \mid F\varphi \mid \varphi U \varphi$$

$\pi = (s1 a1 s2 a2 \ldots)$ is a path and $\pi'$ is the suffix of $\pi$ starting at state $s_i$

$\pi \vdash p$ iff $s_i$ is the first state of $\pi$ and $p \in L(s_i)$

$\pi \vdash a$ iff $a$ is the first action of $\pi$

$\pi \vdash \neg \varphi$ iff $\neg (\pi \vdash \varphi)$

$\pi \vdash X\varphi$ iff $\pi^2 \vdash \varphi$

$\pi \vdash \varphi, U \varphi_2$ iff there is some $i \geq t$ such that $\pi^i \vdash \varphi_2$

and for all $i \geq j \geq t$, $\pi_j \vdash \varphi_i$

$M \vdash \varphi$ iff, for every path $\pi \in L(M)$, $\pi \vdash \varphi$
Surge Spec: State/Event

\[
G ((c2 \to m=2) \land (c1 \to (m=1 \lor m=2)))
\]

Surge Spec: State Only

\[
G ((c=0 \lor c=2) \land X (c=1)) \to (m=1 \lor m=2) \land G ((c=0 \lor c=1) \land X (c=2)) \to m=2
\]
Surge Protector Verification

- Changes of current beyond threshold are disallowed

- State/Event Formula: $G ((c2 \rightarrow m=2) \& (c1 \rightarrow (m=1 \lor m=2)))$

- State Formula: $G (((c=0 \lor c=2) \& X (c=1)) \rightarrow (m=1 \lor m=2))$
  $\& G (((c=0 \lor c=1) \& X (c=2)) \rightarrow m=2)$

- Event Formula: $G (m0 \rightarrow ((\neg c1 W (m1 \lor m2))) \&$
  $G (m0 \rightarrow ((\neg c2 W m2)) \&$
  $G (m1 \rightarrow ((\neg c2 W m2))$

Automata-based Verification

*Given:* $M$ – LKS over $\Sigma$, $P$

*Philosophy:* $\varphi$ – SE/LTL formula

*How to check:* $M \models \varphi$

Possible Approach:
1. Convert $M$ into a conventional state-only Kripke structure
2. Convert $\varphi$ into a state-only LTL formula
3. Check whether $M \models \varphi$

*Inefficient!*

*What we do:*
1. Interpret $\varphi$ as an LTL formula over $\Sigma U P$
2. Compute $B \neg \varphi$ (using Wring [Somenzi, Bloem'00])
3. Construct $M \otimes B \neg \varphi$ (result is a Buchi automaton)
4. *Theorem:* We have $L (M \otimes B \neg \varphi) = \{\}$ iff $M \models \varphi$

*No extra cost in time or space!*
# Verification Results

<table>
<thead>
<tr>
<th>Current range</th>
<th>State Formula</th>
<th>Event Formula</th>
<th>State/Event Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>St</td>
<td>Tr</td>
<td>BC</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td>0.25</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>23</td>
<td>0.49</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>57</td>
<td>2.43</td>
</tr>
<tr>
<td>8</td>
<td>58</td>
<td>107</td>
<td>17.5</td>
</tr>
<tr>
<td>10</td>
<td>92</td>
<td>173</td>
<td>196</td>
</tr>
<tr>
<td>12</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>13</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Carnegie Mellon**
Parallel Composition

- Components **synchronize** on shared actions
  - proceed **independently** on local actions

- **Propositions** of components are **disjoint** (no shared variables)

---

Operational Semantics

\[
\begin{align*}
\text{M}_1 & \xrightarrow{a} \text{M}_1' & \text{M}_2 & \xrightarrow{a} \text{M}_2' \\
(\text{M}_1 \parallel \text{M}_2) & \xrightarrow{a} (\text{M}_1' \parallel \text{M}_2') \\
\end{align*}
\]

*Synchronization on Shared action*

\[
\begin{align*}
\text{M}_1 & \xrightarrow{a} \text{M}_1' & a & \notin \Sigma (\text{M}_2) \\
(\text{M}_1 \parallel \text{M}_2) & \xrightarrow{a} (\text{M}_1' \parallel \text{M}_2) \\
\end{align*}
\]

*Asynchronous Execution*
Case Studies

- **MicroC/OS-II**
  - Real-time OS for embedded applications
  - Widely used (cell phones, medical devices, routers, washing machines...)
  - 6000+ LOC
  - Verified locking discipline
    - Locks and Unlocks alternate and locks are eventually released
  - Found four bugs
    - Missing unlock and return (three known – one unknown)
## Results

<table>
<thead>
<tr>
<th>Name</th>
<th>St-B</th>
<th>Tr-B</th>
<th>St-Mdl</th>
<th>T-BA</th>
<th>T-Mdl</th>
<th>T-Ver</th>
<th>T-Tot</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS</td>
<td>25</td>
<td>47</td>
<td>7951</td>
<td>0.690</td>
<td>48.8</td>
<td>6.84</td>
<td>56.9</td>
<td>39.3</td>
</tr>
<tr>
<td>SE</td>
<td>20</td>
<td>45</td>
<td>4331</td>
<td>0.497</td>
<td>18.8</td>
<td>2.92</td>
<td>22.8</td>
<td>24.2</td>
</tr>
<tr>
<td>SS</td>
<td>25</td>
<td>47</td>
<td>7574</td>
<td>0.699</td>
<td>43.6</td>
<td>1.65</td>
<td>46.4</td>
<td>38.1</td>
</tr>
<tr>
<td>SE</td>
<td>18</td>
<td>40</td>
<td>3691</td>
<td>0.407</td>
<td>15.3</td>
<td>1.089</td>
<td>17.3</td>
<td>21.2</td>
</tr>
<tr>
<td>SS</td>
<td>25</td>
<td>47</td>
<td>24.8 M</td>
<td>0.874</td>
<td>X</td>
<td>X</td>
<td>851</td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>20</td>
<td>45</td>
<td>13.6M</td>
<td>0.655</td>
<td>33.1</td>
<td>2.17</td>
<td>2207</td>
<td>162</td>
</tr>
<tr>
<td>SS</td>
<td>25</td>
<td>47</td>
<td>32.6M</td>
<td>0.836</td>
<td>X</td>
<td>X</td>
<td>347</td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>18</td>
<td>40</td>
<td>15.9M</td>
<td>0.713</td>
<td>34.6</td>
<td>4149</td>
<td>4185</td>
<td>321</td>
</tr>
<tr>
<td>BUG</td>
<td>8</td>
<td>14</td>
<td>873</td>
<td>0.205</td>
<td>3.41</td>
<td>0.261</td>
<td>3.88</td>
<td>X</td>
</tr>
</tbody>
</table>

## Case Studies

- **IPC Module**
  - Deployed by a world leader in robotics engineering systems
  - 1500+ LOC
  - 4 components
  - Over 30 billion states after predicate abstraction

- Discovered **synchronization bug** in a matter of hours
  - Process can incorrectly block while writing to a queue
  - Undetected despite seven years of testing/industrial use
Case Studies

• Controller for a metal casting plant, used by Alcoa
  – ~30,000 LOC
  – Verified proper sequencing of various stages of casting:
    • Sequencing happens in prescribed order
    • The next stage eventually gets sequenced

• No bugs found… yet.

Key Contributions

• State/Event-based modeling and specification
• Efficient (direct) model checking algorithm
• CEGAR loop for software systems
  – Safety and liveness properties
• Compositional software verification
  – Component-wise abstractions
  – Component-wise counterexample validation
  – Component-wise refinements
Related Work

- Modal mu-calculus [Kozen 83]
- Doubly labeled transition systems [De Nicola and Vaandrager 95]
- CTL-based verification of doubly labeled transition systems [Gnesi et al. 96]
- State/event framework for three-valued logic verification [Huth et al. 01]
- SLAM [Ball et al. 00-…]
- BLAST [Henzinger et al. 02-…]

Deadlock Detection

- Deadlock for concurrent blocking message-passing programs

- Need for an automated procedure
Example

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

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

\[ M_1 \parallel M_2 \]

Deadlock

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

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

\[ M_1 \parallel M_2 \]

Deadlock \( \iff \) a reachable state cannot perform any actions at all
Deadlock and Composition

Deadlock

\[ M_1 || M_2 \]

No Deadlock

Deadlock and Composition

No Deadlock

\[ M_1 || M_2 \]

Deadlock
Iterative Refinement

Abstraction → Model → Verification
 System OK → Yes
 No → Counterexample

Abstraction Refinement → Improved Abstraction Guidance

Counterexample Valid?
 Spurious Counterexample → Yes

Conservative Abstraction

P

[2,3]

A

[4,5]

[6,7]
Conservative Abstraction

• Every trace of \( P \) is a trace of \( \mathcal{A} \)
  – Preserves safety properties: \( \mathcal{A} \models \phi \Rightarrow P \models \phi \)
  – \( \mathcal{A} \) over-approximates what \( P \) can do

• Some traces of \( \mathcal{A} \) may not be traces of \( P \)
  – May yield spurious counterexamples - \( \langle a, e \rangle \)

• Eliminated via abstraction refinement
  – Splitting some clusters in smaller ones
  – Refinement can be automated

Original Abstraction

\[
\begin{align*}
\mathcal{A} & : 1 \rightarrow [1] \rightarrow [2,3] \rightarrow [4,5] \rightarrow [6,7] \\
\mathcal{P} & : 1 \rightarrow [1] \rightarrow [2,3] \rightarrow [4,5] \rightarrow [6,7]
\end{align*}
\]
Formal Verification by Model Checking

Natasha Sharygina
Carnegie Mellon University

Outline

Lecture 1: Overview of Model Checking

Lecture 2: Complexity Reduction Techniques

Lecture 3: Software Model Checking

Lecture 4: State/Event-based software model checking

Lecture 5: Component Substitutability

Lecture 6: Model Checking Practicum (Student Reports on the Lab exercises)
Summary of the last lecture:
How does Spin work?

• We already saw:
  – The Algorithm
  – The Promela Language
• We need to see how we does the tool work.

High Level Organization

LTL formula
LTL Translator

Promela Model
Promela Parser

Buchi Automaton
Buchi Translator

Abstract Syntax Tree
Automata Generator

The Buchi automaton is turned into a Promela process and composed with the rest of the system.

The generated verifier is specific to the model and property we started with.
Command Line Tools

- Spin
  - Generates the Promela code for the LTL formula
    ~$ spin –f "[]<>p"
  - The proposition in the formula must correspond to the model declarations
  - Generates the C source code
    ~$ spin –a source.pro
  - The property must be included in the source

- Pan
  - Performs the verification
    - Has many compile time options to enable different features
    - Optimized for performance

- GUI for Spin
  Xspin
Simulator

- Spin can also be used as a simulator
  - Simulated the Promela program
- It is used as a simulator when a counterexample is generated
  - Steps through the trace
  - The trace itself is not “readable”
- Can be used for random and manually guided simulation as well

Comments

- DFS does not necessarily find the shortest counterexample
- There might be a very short counterexample but the verification might go out of memory
- If we don’t finish we might still have some sort of a result (coverage metrics)
Today’s Lecture

Advanced Techniques for Model Checking Software

(ComFoRT project)

Objectives

• C program and high-level designs verification
  – Sequential and concurrent

• Properties involve both data (states) and communication (events)
  – Specified as (State/Event) LTL formulas
  – Safety and Liveness

• Communication via shared actions
  – Synchronous communication
  – Asynchronous execution
Bluetooth L2CAP Spec

“When an L2CAP_ConnectRsp event is received in a W4_L2CAP_CONNECT_RSP state, an L2CAP process may send out an L2CA_ConnectInd event, disable the RTX timer, and move to state CONFIG.”

Spec involves both states and events

Labelled Kripke Structures

- Directed graph with labels on edges and states, $(S, \text{Init}, P, L, T, \Sigma, E)$
  - Every state is labeled with a set of atomic propositions, $P$, true in the state
  - Every LKS comes with an alphabet of actions, $\Sigma$
- State labeling function: $L : S \rightarrow 2^P$
- Transition labeling function: $E : T \rightarrow (2^\Sigma \setminus \{\})$
  - Assumption: LKSs are deadlock-free
    [see deadlock detection algorithm, MEMOCODE'04]
Traces and Languages

- **Trace**: infinite alternating sequence of states and actions
  - \((s_1 \ a \ s_2 \ b \ s_4 \ b \ s_1\ldots)\)

- **Language**: set of all traces
  - \(L(M) = \{s_1 \ a \ (s_2 \ b + s_3 \ c) \ s_4 \ b\}^\omega\)

Surge Protector: State/Event

State/Event model of the Surge Protector
(example is given for \(m: [0..2], c: [0..2]\))
Surge Protector: State Only

Kripke structure of the Surge Protector
(example is given for m: [0..2], c: [0..2])

State/Event LTL

Given LKS $M = (S, \text{Init}, P, L, T, \Sigma, E)$, and $p \in P$, $a \in \Sigma$,

$\phi ::= p \mid a \mid \sim \phi \mid \phi \& \phi \mid X\phi \mid G\phi \mid F\phi \mid \phi U \phi$

$\pi = (s_1 a_1 s_2 a_2 \ldots)$ is a path and $\pi'$ is the suffix of $\pi$ starting at state $s_i$

$\pi \models p$ iff $s_1$ is the first state of $\pi$ and $p \in L(s_1)$
$\pi \models a$ iff $a$ is the first action of $\pi$
$\pi \models \sim \phi$ iff $\sim (\pi \models \phi)$
$\pi \models X\phi$ iff $\pi^2 \models \phi$
$\pi \models \phi U \phi_2$ iff there is some $i \geq t$ such that $\pi^i \models \phi_2$
and for all $i \geq j \geq t - 1$, $\pi^j \models \phi$,

$M \models \phi$ iff, for every path $\pi \in L(M)$, $\pi \models \phi$
Surge Spec : State/Event

\[ G \left( (c_2 \rightarrow m=2) \& (c_1 \rightarrow (m=1 \lor m=2)) \right) \]

Surge Spec : State Only

\[ G \left( ((c = 0 \lor c = 2) \& X (c = 1)) \rightarrow (m = 1 \lor m = 2) \right) \& \\
G \left( ((c = 0 \lor c = 1) \& X (c = 2)) \rightarrow m = 2 \right) \]
Surge Protector Verification

- Changes of current beyond threshold are disallowed

- State/Event Formula: $G((c2 \rightarrow m=2) \land (c1 \rightarrow (m=1 \lor m=2)))$

- State Formula: $G(((c=0 \lor c=2) \land X(c=1)) \rightarrow (m=1 \lor m=2))$
  \[\land G(((c=0 \lor c=1) \land X(c=2)) \rightarrow m=2)\]

- Event Formula: $G(m0 \rightarrow ((\neg c1 \lor (m1 \lor m2)) \land G(m0 \rightarrow ((\neg c2 \lor m2)) \land G(m1 \rightarrow ((\neg c2 \lor m2)))$

Automata-based Verification

*Given:* $M$ — LKS over $\Sigma$, $P$

$\varphi$ — SE/LTL formula

*How to check:* $M \vDash \varphi$

*Possible Approach:*
1. Convert $M$ into a conventional state-only Kripke structure
2. Convert $\varphi$ into a state-only LTL formula
3. Check whether $M \vDash \varphi$

*Inefficient!*

*What we do:*
1. Interpret $\varphi$ as an LTL formula over $\Sigma \cup P$
2. Compute $B\neg\varphi$ (using Wring [Somenzi, Bloem’00])
3. Construct $M \otimes B\neg\varphi$ (result is a Buchi automaton)
4. *Theorem:* We have $L(M \otimes B\neg\varphi) = \{}$ iff $M \vDash \varphi$

*No extra cost in time or space!*
**Verification Results**

<table>
<thead>
<tr>
<th>Current range</th>
<th>State Formula</th>
<th>Event Formula</th>
<th>State/Event Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>St Tr BC MC</td>
<td>St Tr BC MC</td>
<td>St Tr BC MC</td>
</tr>
<tr>
<td>2</td>
<td>4 5 0.25 0.383</td>
<td>6 10 0.245 0.32</td>
<td>3 4 0.184 0.252</td>
</tr>
<tr>
<td>4</td>
<td>14 23 0.49 1.141</td>
<td>20 41 1.597 1.77</td>
<td>5 8 0.243 0.391</td>
</tr>
<tr>
<td>6</td>
<td>32 57 2.43 4.818</td>
<td>42 92 12.08 12.66</td>
<td>7 12 0.614 0.962</td>
</tr>
<tr>
<td>8</td>
<td>58 107 17.5 24.60</td>
<td>72 163 372.8 374.17</td>
<td>9 16 2.622 3.133</td>
</tr>
<tr>
<td>10</td>
<td>92 173 196 214.0</td>
<td>X X X X X X X X</td>
<td>11 20 33.56 34.5</td>
</tr>
<tr>
<td>12</td>
<td>X X X X X X X X</td>
<td>X X X X X X X X</td>
<td>13 24 534.9 536.4</td>
</tr>
<tr>
<td>13</td>
<td>X X X X X X X X</td>
<td>X X X X X X X X</td>
<td>X X X X X X X X</td>
</tr>
</tbody>
</table>
Parallel Composition

- Components **synchronize** on shared actions
  - proceed **independently** on local actions

- **Propositions** of components are **disjoint** (no shared variables)

Operational Semantics

\[
\begin{align*}
M_1 \xrightarrow{a} M_1' & \quad M_2 \xrightarrow{a} M_2' \\
(M_1 \parallel M_2) \xrightarrow{a} (M_1' \parallel M_2')
\end{align*}
\]

- **Synchronization on Shared action**

\[
\begin{align*}
M_1 \xrightarrow{a} M_1' & \quad a \not\in \Sigma (M_2) \\
(M_1 \parallel M_2) \xrightarrow{a} (M_1' \parallel M_2)
\end{align*}
\]

- **Asynchronous Execution**
Compositional Verification

- C Program
  - Abstraction
  - Guidance
- Predicate Abstraction
- Model
- SE-LTL Verification
  - Yes
    - System OK
  - No
    - Counterexample
    - Guide Example
    - Abstraction
    - Refinement
- Improved Abstraction
  - Guidance
- Abstraction Refinement
- No
  - Counterexample
  - Valid?
    - Yes
    - Spurious
    - Counterexample

Case Studies

- MicroC/OS-II
  - Real-time OS for embedded applications
  - Widely used (cell phones, medical devices, routers, washing machines...)
  - 6000+ LOC
  - Verified locking discipline
    - Locks and Unlocks alternate and locks are eventually released
  - Found four bugs
    - Missing unlock and return (three known – one unknown)
Results

<table>
<thead>
<tr>
<th>Name</th>
<th>St-B</th>
<th>Tr-B</th>
<th>St-Mdl</th>
<th>T-BA</th>
<th>T-Mdl</th>
<th>T-Ver</th>
<th>T-Tot</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS</td>
<td>25</td>
<td>47</td>
<td>7951</td>
<td>0.690</td>
<td>48.8</td>
<td>6.84</td>
<td>56.9</td>
<td>39.3</td>
</tr>
<tr>
<td>SE</td>
<td>20</td>
<td>45</td>
<td>4331</td>
<td>0.497</td>
<td>18.8</td>
<td>2.92</td>
<td>22.8</td>
<td>24.2</td>
</tr>
<tr>
<td>SS</td>
<td>25</td>
<td>47</td>
<td>7574</td>
<td>0.699</td>
<td>43.6</td>
<td>1.65</td>
<td>46.4</td>
<td>38.1</td>
</tr>
<tr>
<td>SE</td>
<td>18</td>
<td>40</td>
<td>3691</td>
<td>0.407</td>
<td>15.3</td>
<td>1.089</td>
<td>17.3</td>
<td>21.2</td>
</tr>
<tr>
<td>SS</td>
<td>25</td>
<td>47</td>
<td>24.8M</td>
<td>0.874</td>
<td>65.6</td>
<td>X</td>
<td>X</td>
<td>851</td>
</tr>
<tr>
<td>SE</td>
<td>20</td>
<td>45</td>
<td>13.6M</td>
<td>0.655</td>
<td>33.1</td>
<td>2.17</td>
<td>2207</td>
<td>162</td>
</tr>
<tr>
<td>SS</td>
<td>25</td>
<td>47</td>
<td>32.6M</td>
<td>0.836</td>
<td>66.0</td>
<td>X</td>
<td>X</td>
<td>347</td>
</tr>
<tr>
<td>SE</td>
<td>18</td>
<td>40</td>
<td>15.9M</td>
<td>0.713</td>
<td>34.6</td>
<td>4149</td>
<td>4185</td>
<td>321</td>
</tr>
<tr>
<td>BUG</td>
<td>8</td>
<td>14</td>
<td>873</td>
<td>0.205</td>
<td>3.41</td>
<td>0.261</td>
<td>3.88</td>
<td>X</td>
</tr>
</tbody>
</table>

Case Studies

- **IPC Module**
  - Deployed by a world leader in robotics engineering systems
  - 1500+ LOC
  - 4 components
  - Over 30 billion states after predicate abstraction

- Discovered **synchronization bug** in a matter of hours
  - Process can incorrectly block while writing to a queue
  - Undetected despite seven years of testing/industrial use
Case Studies

- Controller for a metal casting plant, used by Alcoa
  - ~30,000 LOC
  - Verified proper sequencing of various stages of casting:
    - Sequencing happens in prescribed order
    - The next stage eventually gets sequenced

- No bugs found… yet.

Key Contributions

- State/Event-based modeling and specification
- Efficient (direct) model checking algorithm
- CEGAR loop for software systems
  - Safety and liveness properties
- Compositional software verification
  - Component-wise abstractions
  - Component-wise counterexample validation
  - Component-wise refinements
Related Work

• Modal mu-calculus [Kozen 83]
• Doubly labeled transition systems [De Nicola and Vaandrager 95]
• CTL-based verification of doubly labeled transition systems [Gnesi et al. 96]
• State/event framework for three-valued logic verification [Huth et al. 01]
• SLAM [Ball et al. 00-…]
• BLAST [Henzinger et al. 02-…]

Deadlock Detection

• Deadlock for concurrent blocking message-passing programs

• Need for an automated procedure
Example

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

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

\[ M_1 \parallel M_2 \]

Deadlock

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

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

\[ M_1 \parallel M_2 \]

Deadlock \iff a reachable state cannot perform any actions at all
Deadlock and Composition

\[ M_1 \parallel M_2 \]

Deadlock

No Deadlock

Deadlock and Composition

\[ M_1 \parallel M_2 \]

No Deadlock

Deadlock
Iterative Refinement

Conservative Abstraction

- System Abstraction Guidance
- Abstraction
- Improved Abstraction Guidance
- Abstraction Refinement
- Counterexample
- Counterexample Valid?
- Model
- Verification
- Yes
- System OK
- No
- Counterexample
- Spurious Counterexample
- Yes

Conservative Abstraction

\[ \mathcal{P} \]

\[ \mathcal{A} \]
Conservative Abstraction

• Every trace of \( P \) is a trace of \( A \)
  – Preserves safety properties: \( A \models \phi \Rightarrow P \models \phi \)
  – \( A \) over-approximates what \( P \) can do

• Some traces of \( A \) may not be traces of \( P \)
  – May yield spurious counterexamples - \( \{a, e\} \)

• Eliminated via abstraction refinement
  – Splitting some clusters in smaller ones
  – Refinement can be automated

Original Abstraction
Refined Abstraction

Deadlock: Problem

- Deadlock is not preserved by abstraction
Deadlock Detection: Insight

- Deadlock ⇔ a reachable state cannot perform any actions at all
  - Deadlock depends on the set of actions that a reachable state cannot perform

- In order to preserve deadlock A must over-approximate not just what P can do but also what P refuses

Refusal & Deadlock

- \( \text{Ref}(s) = \text{set of actions } s \text{ cannot perform} \)

- \( M \) deadlocks iff there is a reachable state \( s \) such that \( \text{Ref}(s) = \Sigma \)
  - Denote by \( D\text{Lock}(M) \)

- \( \text{Ref([s_1.. s_n])} = \text{Ref}(s_1) \cap .. \cap \text{Ref}(s_n) \)
Abstract Refusal

\[ \mathcal{AR}(\[s_1 \ldots s_n\]) = \mathcal{Ref}(s_1) \cup \ldots \cup \mathcal{Ref}(s_n) \]

\[ \mathcal{AR}(\[M_1 \ldots M_n\]) = \mathcal{AR}(\[M_1\]) \cup \ldots \cup \mathcal{AR}(\[M_n\]) \]

Abstract Deadlock

\( M \) abstractly deadlocks iff there is a reachable state \( s \) such that \( \mathcal{AR}(s) = \Sigma \)

- Denote by \( \mathcal{ADLock}(M) \)

\[ \neg \mathcal{ADLock}([M_1] \parallel \ldots \parallel [M_n]) \Rightarrow \neg \mathcal{DLock}(M_1 \parallel \ldots \parallel M_n) \]
Iterative Deadlock Detection

Counterexample to Abstract Deadlock

Counterexample Validation
Refinement

Counterexample Validation

Another spurious counterexample
Refinement

Counterexample Validation

Real Deadlock Detected
Case Studies

- **MicroC/OS-II**
  - Real-time OS for embedded applications
  - Widely used (cell phones, medical devices, routers, washing machines...)
  - 6000+ LOC
  - Verified locking discipline
    - Locks and Unlocks alternate and locks are eventually released
  - Found four bugs
    - Missing unlock and return (one bug - deadlock)
Case Studies

- **ABB IPC Module**
  - Deployed by a world leader in robotics
  - 15000+ LOC
  - 4 components
  - Over 30 billion states after predicate abstraction

- Discovered synchronization bug in a matter of hours
  - Process can incorrectly block while writing to a queue
  - Undetected despite seven years of testing/industrial use

Results

<table>
<thead>
<tr>
<th>Name</th>
<th>Plain</th>
<th>IterDeadlock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>St</td>
<td>T</td>
</tr>
<tr>
<td>ABB</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>SSL</td>
<td>25731</td>
<td>44</td>
</tr>
<tr>
<td>µCD-3</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>µCN-6</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>DPN-6</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>DPD-10</td>
<td>38268</td>
<td>87.6</td>
</tr>
</tbody>
</table>

* indicates out of time limit (1500s)
Ongoing and Future Work

- Shared memory
- Assume-Guarantee reasoning
- Industrial size examples
- Symbolic implementation
- Branching-time state/event logic (completed)

Lab Assignment

- Spit into groups of 4-5 people
- Design, implementation and verification of the current surge protector
  - In PROMELA/SPIN
  - In ComFoRT
- Comparative validation
- Presentations on March 31, 2005
Lab Assignment (2)

• Questions about ComFoRT
  – Natasha Sharygina: nys@sei.cmu.edu - theory
  – Sagar Chaki: chaki@sei.cmu.edu – tool support

Collaboration Opportunities

• Research and development projects on verification of software (ComFoRT project)

• As part of the PACC (Predictable Assembly from Certifiable Components) project at the SEI

• Joint work with Prof. Ed Clarke
Collaboration Opportunities

• Independent studies

• M.S. and Ph.D. Research (jointly with your current advisors)

• Internships

If interested contact me and we can discuss options