Succinct Representation of Concurrent Trace Sets

Roopsha Samanta

Joint work with Ashutosh Gupta, Tom Henzinger, Arjun Radhakrishna and Thorsten Tarrach

2 December, 2014
Concurrent trace neighbourhoods
Concurrent trace neighbourhoods
Concurrent trace neighbourhoods
Concurrent trace neighbourhoods

Roopsha Samanta
Succinct Representation of Concurrent Trace Sets
Concurrent trace neighbourhoods
Concurrent trace neighbourhoods
Concurrent trace neighbourhoods

Roopsha Samanta
Succinct Representation of Concurrent Trace Sets
Concurrent trace neighbourhoods
Concurrent trace neighbourhoods
Concurrent trace neighbourhoods
Representation of trace neighbourhoods
Representation of trace neighbourhoods
Representation of trace neighbourhoods
Representation of trace neighbourhoods
Representation of trace neighbourhoods

\[ hb(B, z) \land hb(y, D) \]
Representation of trace neighbourhoods

\[ \text{or} \]

\[ \text{hb}(z, B) \lor \text{hb}(D, y) \]

Roopsha Samanta  
Succinct Representation of Concurrent Trace Sets
Representation of trace neighbourhoods

- **HB-formulas** - novel representation for concurrent trace sets
  - Can express arbitrary trace sets
  - Efficient computation of $\bigcup\{\text{trace sets}\}$
  - Succinct representations of good/bad trace neighbourhoods
  - Intuitively appealing
  - Can drive diverse concurrency applications . . .
Application: synchronization synthesis

\[ hb(z, B) \lor hb(D, y) \]
Application: synchronization synthesis

\[ \text{hb}(z,B) \lor \text{hb}(D,y) \]
\[ \text{Lock}([x:y],[B:D]) \]
Application: synchronization synthesis

\[ \text{hb}(z, B) \lor \text{hb}(D, y) \]

\[ \text{Lock}([x:y], [B:D]) \]
Application: synchronization synthesis

- Automated program completion
- Rewrite rules for synchronization synthesis
  - Identification of HB-formula patterns for sync. primitives
  - Mutex locks, barriers, shared-exclusive locks, wait-notify
Application: bug summarization

\[ hb(B, z) \land hb(y, D) \]
Application: bug summarization

\[ hb(B,z) \land hb(y,D) \text{ access}(y,v) \text{ access}(z,v) \text{ access}(B,v) \text{ access}(D,v) \]

\text{AtomicityViolation}([x:y],[B:D])
Application: bug summarization

- HB-formula as a counterexample summary
- Inference rules for more precise bug summaries
  - Identification of HB-formula and data access patterns for bugs
  - Data races, atomicity violations, two stage access bugs, define-use order violations
Application: CEGAR acceleration

Concrete program, $\mathcal{P}$

Predicate abstraction

Abstract program, $\mathcal{A}$

Model checking

$\mathcal{A}$ correct?

Yes

Declare $\mathcal{P}$ correct

No

Error trace in $\mathcal{A}$ feasible in $\mathcal{P}$?

Yes

Declare $\mathcal{P}$ incorrect

No

Theorem-prover/Constraint-solver

Refine $\mathcal{A}$ with new predicates to eliminate error trace
Application: CEGAR acceleration

Concrete program, $P$

Predicate abstraction

Abstract program, $A$

Model checking

$A$ correct?

Yes

Declare $P$ correct

No

Error trace in $A$ feasible in $P$?

Yes

Declare $P$ incorrect

No

Theorem-prover/Constraint-solver

Refine $A$ with new predicates to eliminate spurious error trace set
Application: CEGAR acceleration

- Representation of abstract, spurious, bad trace sets using HB-formulas
- Simultaneous learning of refinement predicates from multiple abstract cexs
Goal

Given a trace $\tau$ and a specification, generate:

- HB-formula $\phi_B$ representing $\text{nhood}(\tau)$, and,
- HB-formula $\phi_G$ representing $\text{nhood}(\tau)$. 
First attempt

Infeasible

Good $\phi_G$

Bad $\phi_B$
First attempt

- Formulate $\Phi$ [WKGG09]:
  - Quantifier-free first-order formula over vars & $hb$ constraints
  - Execution $\pi \in \text{nhood}(\tau)$ iff $\pi \approx$ model of $\Phi$

- To compute $\phi_B$:
  - Initially, $\phi_B = \text{false}$
  - In each step:
    - Obtain a model $\pi$ of $\Phi$ not subsumed by current $\phi_B$
    - Generalize $\pi$ into an HB-formula $\varphi$
    - $\phi_B = \phi_B \lor \varphi$

Generalization: partial satisfying assignments

$\phi_B$ exactly represents $\text{nhood}(\tau)$
First attempt

- Issues:
  - Inefficient in practice
  - Unclear how to generate $\phi_G$
Second attempt

Infeasible

Good

$\phi_G$

Bad

$\phi_B$
Second attempt
Second attempt
Second attempt

Formulate $\Phi$ [WKGG09]:
- Quantifier-free first-order formula over vars & $hb$ constraints
- Execution $\pi \in nhood(\tau)$ iff $\pi \approx$ model of $\Phi$

To compute $\phi_B$:
- Initially, $\phi_B = false$
- In each step:
  - Obtain a model $\pi$ of $\Phi$ not subsumed by current $\phi_B$
  - Generalize $\pi$ into an HB-formula $\varphi$
  - $\phi_B = \phi_B \lor \varphi$
- Generalization: *data-flow analysis* + *minimal unsat core*

$\phi_B$ soundly overapproximates $nhood(\tau)$
Second attempt

- **Formulate $\Phi$ [WKGG09]:**
  - Quantifier-free first-order formula over vars & $hb$ constraints
  - Execution $\pi \in \text{nhood}(\tau)$ iff $\pi \approx$ model of $\Phi$

- **To compute $\phi_B$:**
  - Initially, $\phi_B = false$
  - In each step:
    - Obtain a model $\pi$ of $\Phi$ not subsumed by current $\phi_B$
    - Generalize $\pi$ into an HB-formula $\varphi$
    - $\phi_B = \phi_B \lor \varphi$
  - Generalization: data-flow analysis + minimal unsat core

- **To compute $\phi_G$:** $\neg \phi_B$

$\phi_G$ soundly overapproximates the good neighbourhood of $\tau$
### Experiments using TARA for $\phi_B$ generation

<table>
<thead>
<tr>
<th>Name</th>
<th>#P/#I</th>
<th>#\pi/#Disj.</th>
<th>Iterations</th>
<th>Total time</th>
<th>Size of $\phi_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>reorder_2</td>
<td>2/3</td>
<td>2/2.0</td>
<td>1</td>
<td>18ms</td>
<td>1/2.0</td>
</tr>
<tr>
<td>define_use</td>
<td>2/4</td>
<td>2/2.0</td>
<td>1</td>
<td>15ms</td>
<td>1/2.0</td>
</tr>
<tr>
<td>em28xx</td>
<td>2/8</td>
<td>4/2.0</td>
<td>1</td>
<td>16ms</td>
<td>1/2.0</td>
</tr>
<tr>
<td>locks</td>
<td>3/8</td>
<td>10/1.6</td>
<td>12</td>
<td>27ms</td>
<td>12/5.5</td>
</tr>
<tr>
<td>2stage</td>
<td>2/8</td>
<td>5/1.4</td>
<td>8</td>
<td>26ms</td>
<td>8/3.8</td>
</tr>
<tr>
<td>drbd_receiver</td>
<td>2/9</td>
<td>5/1.6</td>
<td>40</td>
<td>42ms</td>
<td>40/3.9</td>
</tr>
<tr>
<td>md</td>
<td>3/11</td>
<td>4/1.8</td>
<td>40</td>
<td>76ms</td>
<td>40/6.1</td>
</tr>
<tr>
<td>lazy01</td>
<td>3/12</td>
<td>6/3.7</td>
<td>2</td>
<td>31ms</td>
<td>2/3.0</td>
</tr>
<tr>
<td>locks_hb</td>
<td>4/13</td>
<td>10/2.2</td>
<td>&gt;29.0k</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>lc_rc</td>
<td>4/14</td>
<td>8/2.0</td>
<td>4.6k</td>
<td>21.4s</td>
<td>4.6k/16.7</td>
</tr>
<tr>
<td>barrier_locks</td>
<td>3/18</td>
<td>17/2.6</td>
<td>10.6k</td>
<td>1.4min</td>
<td>10.6k/10.0</td>
</tr>
<tr>
<td>stateful01</td>
<td>3/19</td>
<td>10/3.4</td>
<td>2.3k</td>
<td>10.5s</td>
<td>2.3k/9.4</td>
</tr>
<tr>
<td>read_write_lock</td>
<td>4/22</td>
<td>16/3.4</td>
<td>9.2k</td>
<td>1.6min</td>
<td>9.2k/16.1</td>
</tr>
<tr>
<td>loop</td>
<td>2/38</td>
<td>14/2.7</td>
<td>2</td>
<td>38ms</td>
<td>2/3.0</td>
</tr>
<tr>
<td>fib_bench</td>
<td>3/39</td>
<td>24/3.6</td>
<td>&gt;20.5k</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>i2c_hid</td>
<td>2/42</td>
<td>26/4.5</td>
<td>&gt;23.4k</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>rt18169-1</td>
<td>7/71</td>
<td>22/2.7</td>
<td>&gt;20.4k</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>rt18169-2</td>
<td>7/116</td>
<td>41/2.3</td>
<td>&gt;7.3k</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>rt18169-5</td>
<td>7/134</td>
<td>48/3.1</td>
<td>&gt;5.5k</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>rt18169-4</td>
<td>7/142</td>
<td>48/3.0</td>
<td>&gt;8.4k</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>rt18169-6</td>
<td>7/144</td>
<td>52/2.9</td>
<td>&gt;8.1k</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>usb_serial-1</td>
<td>7/151</td>
<td>87/3.7</td>
<td>&gt;5.5k</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>usb_serial-2</td>
<td>7/163</td>
<td>93/3.6</td>
<td>&gt;4.4k</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>rt18169-3</td>
<td>8/174</td>
<td>61/3.6</td>
<td>&gt;4.2k</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>usb_serial-3</td>
<td>7/178</td>
<td>100/3.7</td>
<td>&gt;4.3k</td>
<td>TO</td>
<td>TO</td>
</tr>
</tbody>
</table>
Goal

Given a trace $\tau$ and a specification, synthesize synchronization to eliminate $\text{nhood}(\tau)$. 

Basic idea

- Use $\phi_G$ (in CNF)
- Identify HB-formula patterns for various synchronization primitives
- Formulate rewrite rules
- Repeatedly rewrite patterns into synchronization primitives
- Obtain CNF formula over synchronization primitives
- Pick a set $S$ of synchronization primitives, one from each conjunct
Examples

\[
\begin{align*}
\text{hb}(T_1[\ell_1], T_2[\ell_2]) & \lor \psi \\
\text{WaitNotify}(T_2[\ell_2], T_1[\ell_1]) & \lor \psi
\end{align*}
\]

\text{ADD\_WAIT\_NOTIFY}
Examples

\[ hb(T_1[\ell_1], T_2[\ell_2]) \lor hb(T_2[\ell'_2], T_1[\ell_1]) \lor \psi \quad \ell_1 \leq \ell'_1 \quad \ell_2 \leq \ell'_2 \]

\[
Lk(T_1[\ell_1 : \ell'_1], T_2[\ell_2 : \ell'_2]) \lor \psi
\]

ADD.LOCK
Examples

\[
\begin{align*}
(hb(T_1[\ell_1 - 1], T_2[\ell_2]) \lor \psi) & \land (hb(T_2[\ell_2 - 1], T_1[\ell_1]) \lor \psi) \\
\text{Barrier}(T_1[\ell_1], T_2[\ell_2]) & \lor \psi
\end{align*}
\]

ADD.BARRIER
Examples

Additional rewrite rules for:
- Shared exclusive locks
- Multithreaded locks
- Multithreaded barriers
- Merging locks (to avoid deadlocks)
Soundness of rewrite rules

Given a trace $\tau$ of a concurrent program $P$, let $P^S$ be obtained by inserting synchronization primitives from $S$. Let $\pi \in \text{nhood}(\tau)$ be a deadlock-free execution of $P^S$. Then, $\pi$ is not bad.
## Experiments

<table>
<thead>
<tr>
<th>Name</th>
<th>#L</th>
<th>#B</th>
<th>#WN</th>
</tr>
</thead>
<tbody>
<tr>
<td>reorder_2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>define_use</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>em28xx</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>locks</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2stage</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>drbd_receiver</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>md</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>lazy01</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>locks_hb</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>lc_rc</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>barrier_locks</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>stateful01</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>read_write_lock</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>#L</th>
<th>#B</th>
<th>#WN</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fib_bench</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>i2c_hid</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>rtl8169-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>rtl8169-2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>rtl8169-5</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>rtl8169-4</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>rtl8169-6</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>usb_serial-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>usb_serial-2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>rtl8169-3</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>usb_serial-3</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
A method and a tool TARA for succinct representations of sound overapproximations of $\text{nhood}(\tau)$ and $\text{nhood}(\tau)$

Three successful case studies using TARA

- Synchronization synthesis
- Bug summarization
- CEGAR acceleration

Other applications?
Thank you.