TRANSIT: Synthesis of Distributed Protocols

Rajeev Alur, Jyotirmoy V. Deshmukh, Sela Mador-Haim, Milo M. K. Martin, Arun Raghavan, Abhishek Udupa
University of Pennsylvania

ExCAPE Annual PI Meeting
June 10-11, 2013

Goal: Assist Programmers of Distributed Protocols

- Published informal descriptions only specify common-case behaviors
- Corner-cases arising out of concurrency and asynchrony often unspecified
- Traditional specification methodologies using EFSMs require global reasoning
- Tedious case-by-case analysis in writing the specification
- Fixing counterexamples also requires global reasoning about all EFSMs
- Can ExCAPE techniques simplify the design process?

- Allow programmer to use multiple formats
- Protocol Description = Skeleton + Symbolic Constraints + Concrete Examples
- Make the process of specifying protocols less tedious and more intuitive!

Case Studies

- TRANSIT successfully used to design two textbook protocols by inexperienced programmers
- Complexity: ~ 7 Million States
- Manual effort: ~ tens of hours
- Synthesis completed in less than a second
- < 2 concrete examples per EFSM transition
- Used by an experienced protocol designer for the industrial strength SGI Origin protocol
  - Initial specification obtained from informal flows described in the paper
  - Counterexamples fixed by adding concrete snippets describing the correct behavior
  - Intermixing symbolic and concrete snippets found to be useful

Specifying Protocols with TRANSIT

- Specify common case behavior symbolically
  - From informal descriptions
  - From programmer intuition
  - Symbolic specifications need not be complete!
  - Tricky, corner-case behavior specified in terms of concrete execution fragments
  - Possibly in response to specific counterexamples returned by model checker
  - With concrete fixes to counterexamples programmer only reasons locally

TRANSIT language:

- Communication architecture (processes/channels)
- EFSM template (process states/variables)
- Partial EFSM description as Guarded commands
- Concolic (Concrete + Symbolic) Snippets

Protocol Completion using Synthesis

- Consistent expressions synthesized for guards and state variable updates
- Counter-Example Guided Inductive Synthesis
  - Candidate expressions generated by optimized search
  - Consistency checked using SMT queries

<table>
<thead>
<tr>
<th># of Expressions</th>
<th>Size of Randomly Generated Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>1000</td>
<td>4</td>
</tr>
<tr>
<td>10000</td>
<td>5</td>
</tr>
<tr>
<td>100000</td>
<td>6</td>
</tr>
<tr>
<td>1000000</td>
<td>7</td>
</tr>
</tbody>
</table>

The TRANSIT language:

- Protocol Descriptions:
  - Process $\langle \text{Dir} \rangle$
    - $\text{Sharers} : \text{Set}$
    - $\text{Owner} : \text{PID}$
- Protocol Skeleton
- Symbolic Snippets
- Concrete Examples
- Model Checker ($\mathcal{M}_\mu$)
- Candidate Protocol

Counterexample Guided Inductive Synthesis

- Consistency checked using SMT queries
- Optimized search for candidate expressions
- Candidate expressions synthesized for guards and state variable updates
- Counterexamples fixed by adding concrete snippets describing the correct behavior
- Intermixing symbolic and concrete snippets found to be useful