Synthesis of Multicore Protocols
Challenge Problem

Milo Martin
University of Pennsylvania
Synthesis of Multicore Protocols
Breakout Session

- Five presentations
  - "Multicore Protocol Design with Concolic Snippets"
    Milo Martin <milom@cis.upenn.edu>
  - "Mining Environment Assumptions for Synthesis"
    Sanjit Seshia <sseshia@eecs.berkeley.edu>
  - "Architectural & Application Mapping Challenges for Multicores"
    Alberto Sangiovanni Vincentelli <alberto@eecs.berkeley.edu>
  - "Sythesis of Parameters and Policies for Systems-On-Chip"
    Susmit Jha <jha@eecs.berkeley.edu>
  - "Synthesizing the Alternating Bit Protocol - A Challenge Problem in Decentralized Controller Synthesis"
    Stavros Tripakis <stavros.tripakis@gmail.com>

- Interspersed discussion
Multicore Protocols

- Design challenging due to asynchronous model of communication

- Examples:
  - Cache coherence protocols
  - Network-on-chip (NoC) & system-on-chip (SoC) protocols

- Successful application domain for model checking
  - Murphi, SMV: Hardware cache coherence protocols
  - Industrial adoption (Intel, IBM, ...)

- On-chip heterogeneity and SoCs of increasing importance
Protocol Model: Communicating Processes

Finite set of processes coordinating by exchanging messages over network links
Processes as Extended Finite-State Machines

Control States + Variables
  Sharers: Set of process IDs
  Acks: int

Transitions labeled with input events & guarded updates

- **Trigger:** Msg on channel ReqMsg
- **Guard:** Msg.Type = GetM
- **Update:** Add Sender to Sharers
  - Acks--
  - Send Data to Sender
  - Send Invalidate to others

- How can we help the designer figure out “correct” guards and updates for each transition?
Iterative Design Flow
Q1 & Q5: Designer interaction & specifications

Designer provides: (1) high-level requirements (2) states & variables (3) concrete & symbolic scenarios
Protocol Specification using “Concolic” Scenarios

- Approach: allow for “Concolic” snippets
- Can be a mix of concrete and symbolic values
- “Concolic” coined by Sen et al in the context of testing of programs using both concrete and symbolic inputs
Concolic Snippet Example 1

Process Directory
Transition
  From Shared
  To Busy
Input channel: ReqMsg
Output channels: RespMsg, InvMsg

Guard:
  ReqMsg.Type == GetM & ReqMsg.Sender == 1 & Sharers == {2,3}

Update:
  RespMsg.Acks = 2;
  RespMsg.Destination = 1;
  InvMsg.Destination = {2,3}

Values of all variables can be concrete
Concolic Snippet Example 2

Process Directory
Transition
  From Shared
  To Busy
Input channel: ReqMsg
Output channels: RespMsg, InvMsg

Guard:
  ReqMsg.Type == GetM & ReqMsg.Sender == 1

Update:
  RespMsg.Acks = Size(Sharers);
  RespMsg.Destination = 1;
  InvMsg.Destination = Sharers

Same snippet can mix concrete and symbolic values
Concolic Snippet Example 3

Process Directory
Transition
From Shared
To Busy
Input channel: ReqMsg
Output channels: RespMsg, InvMsg

Guard:
ReqMsg.Type == GetM

Update:
RespMsg.Acks = Size(Sharers);
RespMsg.Destination = ReqMsg.Sender;
InvMsg.Destination = Sharers - ReqMsg.Sender

Values of all variables can be symbolic:
Classical EFSM description maps directly to such snippets
Artifact: implementable extended finite state machine
Q2: Tool feedback

Tool feedback: counterexamples from model checker (deadlock or invariant violation)
Designer-in-the-loop Flow

- **Inputs:**
  - Variable types and corresponding expression grammar
  - For each process,
    1. Control states of EFSM
    2. List of all variables, input/output messages
    3. Set of concolic snippets
  - High-level requirements (invariants and temporal logic formulas)

- **Solution strategy:**
  - For each EFSM transition, generate possible expressions for guards and updates, one by one, until you find a choice that is consistent with all the corresponding snippets
  - Check if resulting protocol meets all requirements, using a model checker (Murphi) and if not, report a counter-example
Iterative Design

From Shared To Busy
Input channel: ReqMsg
Output channels: RespMsg, InvMsg

Guard:
    ReqMsg.Sender = 1 & ReqMsg.Type = GetM & Sharers = {2, 3}

Update:
    RespMsg.Acks = 2;
    InvMsg.Destination = {2, 3}

Based on this single example, synthesis tool computes update:

    RespMsg.Acks = Size(Sharers);
    InvMsg.Destination = Sharers

But this is incorrect and protocol deadlocks
Iterative Design Continued

Designer adds another concrete example
(corresponds to case when Sender is a Sharer)

From Shared To Busy
Input channel: ReqMsg
Output channels: RespMsg, InvMsg
  Guard:
    ReqMsg.Sender = 1 & ReqMsg.Type = GetM & Sharers ={1,2}
  Update:
    RespMsg.Acks = 1;
    InvMsg.Destination = {2}

Based on two snippets, synthesis tool computes update:

RespMsg.Acks = Size(Sharers + ReqMsg.Sender) - 1;
InvMsg.Destination = Sharers - ReqMsg.Sender
Q6 & Q7: Synthesis Strategy

Synthesis strategy: expression enumeration & SMT
Expression Enumeration

- **Goal:** Generate expression for a single variable in update for a single EFSM transition (also for guards and fields in output msgs)

- **Empirical observation:** typical expression size is small (max was size 13)

- **Current prototype:**
  - Supports types: Int, Set, Bool, PID, Address, Value, EnumTypes
  - Typical operations: Boolean connectives, arithmetic, set operations, if-then-else, ...

- **Enumeration with redundant expression optimization**
  - Using concrete examples, generate all “indistinguishable” expressions of size 1
  - Two expressions are “indistinguishable” if they generate same output in all examples
    - Subsumes commutativity/associativity (x+y vs y+x), composition (~(~x) vs x), constant propagation, etc.
  - Recursively build expressions of size N from smaller expressions
  - Repeat until expression that satisfies all examples is found

- **Candidate expression is then checked on symbolic examples via SMT**
  - Any counterexample is added to set of concrete examples, repeat enumeration above
Q3: User Studies

- Starting point: cache coherence protocols described in
  *A Primer on memory Consistency and Cache Coherence*
  Sorin, Hill, Wood, 2011

- Translated EFSMs for 2 protocols into (mostly concrete) snippets

- Case studies: Can a user with no prior experience in designing cache coherence protocol build a correct protocol from textbook description?
  - Case study A: Version of MSI protocol with non-blocking directory progress
  - Case study B: Augmenting MSI protocol with E state to obtain MESI
## Experience Report

<table>
<thead>
<tr>
<th></th>
<th>Case study A</th>
<th>Case study B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snippets used in first version</td>
<td>19</td>
<td>96</td>
</tr>
<tr>
<td>Time to implement first version</td>
<td>2 hrs</td>
<td>6 hrs</td>
</tr>
<tr>
<td>Snippets used in final version</td>
<td>86</td>
<td>108</td>
</tr>
<tr>
<td>Number of iterations</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>Total manual effort</td>
<td>6 hrs</td>
<td>13 hrs</td>
</tr>
<tr>
<td>Number of counterexamples examined</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Synthesis time in last iteration</td>
<td>52 min</td>
<td>15 min</td>
</tr>
<tr>
<td>Number of update expressions synthesized</td>
<td>175</td>
<td>260</td>
</tr>
<tr>
<td>Number of guard expressions synthesized</td>
<td>80</td>
<td>74</td>
</tr>
<tr>
<td>States explored in final protocol</td>
<td>7.7 million</td>
<td>1.5 million</td>
</tr>
</tbody>
</table>
Recap: Multicore Protocol Synthesis

Q1 & Q5: Designer interaction & specifications
- Specification: familiar communicating EFSMs, but transitions can be described using a mix of concrete and symbolic examples
- Designer provides: (1) high-level requirements, (2) states & variables, (3) concrete & symbolic scenarios

Q2: Tool feedback
- Counter example from model checker

Q3: User studies
- Case studies of tool usage

Q4: Artifacts synthesized
- Implementable extended finite state machine

Q6 & Q7: Synthesis strategy
- Synthesize expression from concrete/symbolic examples
- Iterative expression enumeration with SMT-based checking in the loop

Look at other multicore hardware protocols
- SoC & NoC protocols, keeping heterogeneity in mind