Combining Reverse-Engineering with Model-Based Testing

Neil Walkinshaw

Department of Computer Science
University of Sheffield

March 2010
Motivation

State-based models are useful

- Documentation
- Automated validation and verification
  - Model-checking
  - Inspections
  - Testing

Often omitted in practice

- Costly to develop and maintain by hand
- Become increasingly inaccurate as software evolves
  - New requirements, fixes, improvements, ...
Motivation

State-based models are useful
- Documentation
- Automated validation and verification
  - Model-checking
  - Inspections
  - Testing

Often omitted in practice
- Costly to develop and maintain by hand
- Become increasingly inaccurate as software evolves
  - New requirements, fixes, improvements, ...
Conventional Reverse Engineering Approaches

- manually record program traces
- model inference
- final model
Conventional Reverse Engineering Approaches

Cook and Wolf
ACM TOSEM1998

Lo et al.
WCRE 2006

Ammons et al.
POPL 2002

Lorenzoli et al.
ICSE 2008

manually record program traces

model inference final model

Biermann, Feldman
IEEE Transactions on Computers1972

Reiss and Renieris,
ICSE 2002

Quante and Koschke,
WCRE 2007
Conventional Reverse Engineering Approaches

Which software features need to be exercised?  In what context?
What are the necessary inputs?  How many traces are necessary?

manually record program traces  model inference  final model
Conventional Reverse Engineering Approaches

Which software features need to be exercised?  In what context?
What are the necessary inputs?  How many traces are necessary?

manually record program traces  model inference  final model

Inference from positive traces alone is undecidable (Gold 1967)
Model-Based Testing

- manually supply specification
- model-based tester
- implementation
- passed tests
- failed tests
Model-Based Testing

What model is needed to ensure detection of all faults?

manually supply specification

model-based tester

implementation

passed tests

failed tests
Combining Reverse-Engineering and MBT

- Manually record program traces
- Model inference
- Final model
- Manually supply specification
- Model-based tester
- Passed tests
- Failed tests
- Implementation
Combining Reverse-Engineering and MBT
Combining Reverse-Engineering and MBT
Combining Reverse-Engineering and MBT
Combining Reverse-Engineering and MBT

- Model inference
  - Passed tests
  - Failed tests

- Model-based tester
- Hypothesis model

Neil Walkinshaw
Combining Reverse-Engineering with Model-Based Testing
Combining Reverse-Engineering and MBT

- Passed tests
- Failed tests
- Model-based tester
- Model inference
- Hypothesis model
- Markov
- Neural Nets
- k-tails
- Blue-Fringe
- RPNI
- Co-evolutionary approaches
Combining Reverse-Engineering and MBT

- Model inference hypothesis model
- Passed tests
- Failed tests
- Markov
- Neural Nets
- k-tails
- Blue-Fringe
- RPNI
- Co-evolutionary approaches

Model-based tester

W / WP-Method
UIO
TT
Random...

Hypothesis model

Neil Walkinshaw
Reverse-Engineering State Machine of Erlang TCP/IP Stack

**Context**
- EU ProTest project - Property-based testing of Erlang systems
  - Widely used in telecoms / finance / networking
    - Ericsson, BT, T-Mobile, Facebook, Amazon, etc.
  - Concurrent, distributed, functional
  - **Often implemented in terms of state-machines**
  - QuickCheck - Popular random MBT framework for Erlang

**Erlang TCP QuickCheck Model**
- QuickCheck framework - produces and reads network packets
- Abstract State Machine
  - Data state, set of functions on this state
  - State machine showing order in which functions can happen
Reverse-Engineering State Machine of Erlang TCP/IP Stack

Context

- EU ProTest project - Property-based testing of Erlang systems
  - Widely used in telecoms / finance / networking
    - Ericsson, BT, T-Mobile, Facebook, Amazon, etc.
  - Concurrent, distributed, functional
  - **Often implemented in terms of state-machines**
  - QuickCheck - Popular random MBT framework for Erlang

Erlang TCP QuickCheck Model

- QuickCheck framework - produces and reads network packets
- Abstract State Machine
  - Data state, set of functions on this state
  - State machine showing order in which functions can happen
Target and Initial Machines

Neil Walkinshaw

Combining Reverse-Engineering with Model-Based Testing
Results

- Several functions could take minutes to execute
- Process was limited to 474 iterations (9 hours)
- A total of 1085 tests were synthesised and executed
  - Because of the random testing, lots of these were duplicates
- 611 passed and 474 failed
Results
Overview of current work and open problems

1. How do we evaluate inferred state machines?
2. Reverse-Engineering data functions (generating true Abstract State Machines)
3. Automated trace abstraction
4. Finding better inference algorithms
Evaluating State Machine Accuracy

Comparing Behaviour

- Compare overlap between sequences in target and inferred machine
  - What strategy to generate these samples of sequences?
  - What metrics to measure the overlap?

Example Results for TCP Model

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall (Sensitivity)</th>
<th>F</th>
<th>Specificity</th>
<th>BCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0.9</td>
<td>0.175</td>
<td>0.29</td>
<td>0.91</td>
<td>0.54</td>
</tr>
<tr>
<td>W-Method</td>
<td>0.16</td>
<td>0.935</td>
<td>0.286</td>
<td>0.39</td>
<td>0.66</td>
</tr>
</tbody>
</table>
Evaluating State Machine Accuracy

Comparing Structure

- Compare overlap between actual states and transitions
  - How do you identify matching states?

LTSDiff

- Have developed an algorithm to compare state transition structures
- Computes a ‘diff’ of two transition structures
  - identifies which states have been added / removed
- Compares the similarity of every pair of states in terms of its surrounding transitions
  - Computed as linear differential equations
Combining Model-Inference with Model-Based Testing

Current work and open problems

Evaluating Accuracy of Inferred Models
Accounting for data
Trace Abstraction
Finding better model inference algorithms

Evaluating and Comparing State Machines
Evaluating and Comparing State Machines

- initialise
- connect
- login
- changedirectory
- logout
- listfiles
- logout
-,listnames
- logout
- change directory
- logout
- appendfile
- storefile
- listfiles
- logout
- listfiles
- logout
- change directory
- logout
- delete
- logout
- makedir
- changedirectory
- storefile
- listfiles
- logout
- login
- changedirectory
- logout
- logout
- logout
- rename
- listfiles
- connect
- login
- logout
- logout
- appendfile
- setfiletype
- logout
- logout
- appendfile
- storefile
- setfiletype
- logout
- logout
- storefile
- setfiletype
Beyond labelled transition systems

- System behaviour cannot just be defined by sequences of symbols
- Usually dependent on underlying data constraints
- What are the data constraints that underpin each label?

Current (attempted) solutions

- Ensure that symbols imply data constraints
- Add data constraints by symbolic execution (Damas 2005)
- Augment labels with data constraints
  - Incorporate Daikon with state machine inference (Mariani 2008, Shahbaz 2008)
- Supply temporal (LTL) constraints
What about the data?

Beyond labelled transition systems

- System behaviour cannot just be defined by sequences of symbols
- Usually dependent on underlying data constraints
- What are the data constraints that underpin each label?

Current (attempted) solutions

- Ensure that symbols imply data constraints
- Add data constraints by symbolic execution (Damas 2005)
- Augment labels with data constraints
  - Incorporate Daikon with state machine inference (Mariani 2008, Shahbaz 2008)
- Supply temporal (LTL) constraints
Establishing the link between the model and the implementation

- Assumption: Straightforward to trace the major functions of interest in a system
  - Straightforward at a low level, or for systems with well-defined interfaces (Erlang processes, single objects in an OO system, etc.)
- JHotDraw trace: generate three canvasses, draw five figures on each canvas
  - 161,087 method calls (thousands of different methods)
  - How do we work out the alphabet of the machine we are trying to infer?
Finding better inference algorithms

STAMINA (State Machine Inference Approaches) Competition

- Training sets for randomly generated “software-like” state machines
- Varying difficulty (larger alphabets, sparse training sets)
- Cannot be solved by current baseline algorithm (BlueFringe)

Competition Details

- Prize: £700 (currently 782 Euros)
- Journal Special Issue (Empirical Software Engineering)
- http://stamina.chefbe.net/
Acknowledgements

Collaborators, Projects
- Kirill Bogdanov
- John Derrick
- Phil McMinn
- Bernard Lambeau
- Christophe Damas
- Pierre Dupont

Supported by
- EU ProTest project
- EPSRC REGI and STAMINA grants