On the Correctness of Model Transformations

Matthew Nizol
CSE 814, Fall 2014
Thursday, December 11, 2014
Agenda

- Context: Model-driven development
- Background on verification techniques
- Presentation of each technique
- Comparison of techniques
Context: What do we mean by “Model”?

- An *abstraction* of a software system
- Many types of models in software development:

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formal vs. informal</td>
<td>Alloy spec vs. whiteboard sketch</td>
</tr>
<tr>
<td>Static vs. dynamic</td>
<td>Class diagram vs. state chart</td>
</tr>
<tr>
<td>High- vs. low-level</td>
<td>Ontology vs. XML schema</td>
</tr>
<tr>
<td>Visual vs. textual</td>
<td>UML vs. program code</td>
</tr>
</tbody>
</table>
Context: Model-driven Development

- **Problem:**
  - Complex software hard to develop
  - Related artifacts hard to keep synchronized

- **Solution:**
  - Models increase the abstraction level of development
  - Transformations formalize relationships between models
Model Transformations

- Convert *source* model to *target* model
- Composed of transformation *rules*
- Rules may be *imperative* or *declarative*
- Example applications:

<table>
<thead>
<tr>
<th>Exogenous</th>
<th>Endogenous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code generation</td>
<td>Refactoring</td>
</tr>
<tr>
<td>Reverse engineering</td>
<td>Optimization</td>
</tr>
<tr>
<td>Language migration</td>
<td>Refinement</td>
</tr>
</tbody>
</table>
## Transformation Properties

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>Type correctness</td>
</tr>
<tr>
<td></td>
<td>Well-formedness</td>
</tr>
<tr>
<td>Semantics</td>
<td>Correspondence</td>
</tr>
<tr>
<td></td>
<td>Preservation</td>
</tr>
<tr>
<td>Execution of the transformation</td>
<td>Confluence</td>
</tr>
<tr>
<td></td>
<td>Termination</td>
</tr>
</tbody>
</table>
Verification Approaches

- Informal:
  - Testing
  - Inspection

- Formal:
  - Model Checking
  - Theorem Proving
  - Graph-theoretic
Verification Approaches

- Informal:
  - Testing
  - Inspection

- Formal:
  - Model Checking
  - Theorem Proving
  - Graph-theoretic

Model Transformation

Source → Model Transformation → Target

Direct

Indirect
Approach 1: Indirect, Model Checking

Figure adapted from Varro et al.
Example: Source Model

Sensor

- Sensing
  - Captured
    - Image Capture
    - Clear
    - Process Image
      - Clear
      - Obstacle
    - Compute Distance
      - Clear
      - Close Obstacle / Actuator.Turn

Actuator

- Moving
  - Turn
    - Resume / Sensor.Resume
  - Braking
  - Rotate Chassis
  - Complete
    - Compute Angle
    - Stopped
Example: Target Model (Petri Net)

Tool: pneditor.org
Example: Target Model (Petri Net)

Tool: pneditor.org
Example: Target Model (Petri Net)

Tool: pneditor.org
Example: Target Model (Petri Net)

Tool: pneditor.org
Example: Target Model (Petri Net)

Tool: pneditor.org
Example: Target Model (Petri Net)

Tool: pneditor.org
MODULE sensor(resumemsg)
VAR
  state : {imagecapture, processimage, computedist, waiting};
  transition : {captured, clear, obstacle, closeobs, resume};
ASSIGN
  init(state) := imagecapture;
  next(state) :=
    case
      state = imagecapture & transition = captured : processimage;
      ...
      state = waiting & transition = resume : imagecapture;
      TRUE : state;
    esac;
  next(transition) :=
    case
      state = imagecapture : captured;
      state = processimage : {clear, obstacle};
      state = computedist : {clear, closeobs};
      state = waiting & resumemsg : resume;
      TRUE : transition;
    esac;
Example: Source to NuSMV (Partial)

MODULE sensor(resumemsg)
VAR
    state : {imagecapture, processimage, computedist, waiting};
    transition : {captured, clear, obstacle, closeobs, resume};
ASSIGN
    init(state) := imagecapture;
    next(state) :=
        case
            state = imagecapture & transition = captured : processimage;
            ...
            state = waiting & transition = resume : imagecapture;
            TRUE : state;
        esac;
    next(transition) :=
        case
            state = imagecapture : captured;
            state = processimage : {clear, obstacle};
            state = computedist : {clear, closeobs};
            state = waiting & resumemsg : resume;
            TRUE : transition;
        esac;

Define states and transitions
MODULE sensor(resumemsg)

VAR
  state : {imagecapture, processimage, computedist, waiting};
  transition : {captured, clear, obstacle, closeobs, resume};

ASSIGN

  init(state) := imagecapture;

  next(state) :=
    case
      state = imagecapture & transition = captured : processimage;
      ...
      state = waiting & transition = resume : imagecapture;
      TRUE : state;
    esac;

  next(transition) :=
    case
      state = imagecapture : captured;
      state = processimage : {clear, obstacle};
      state = computedist : {clear, closeobs};
      state = waiting & resumemsg : resume;
      TRUE : transition;
    esac;
Example: Source to NuSMV (Partial)

```plaintext
MODULE sensor(resumemsg)
VAR
  state : {imagecapture, processimage, computedist, waiting};
  transition : {captured, clear, obstacle, closeobs, resume};
ASSIGN
  init(state) := imagecapture;

next(state) :=
  case
    state = imagecapture & transition = captured : processimage;
    ...
    state = waiting & transition = resume : imagecapture;
    TRUE : state;
  esac;

next(transition) :=
  case
    state = imagecapture
    state = processimage : {clear, obstacle};
    state = computedist : {clear, closeobs};
    state = waiting & resumemsg : resume;
    TRUE : transition;
  esac;
```

Define next state given a transition
Example: Source to NuSMV (Partial)

MODULE sensor(resumemsg)
VAR
    state : {imagecapture, processimage, computedist, waiting};
    transition : {captured, clear, obstacle, closeobs, resume};
ASSIGN
    init(state) := imagecapture;
    next(state) :=
        case
            state = imagecapture & transition = captured : processimage;
            ...
            state = waiting & transition = resume : imagecapture;
            TRUE : state;
        esac;
    next(transition) :=
        case
            state = imagecapture : captured;
            state = processimage : {clear, obstacle};
            state = computedist : {clear, closeobs};
            state = waiting & resumemsg : resume;
            TRUE : transition;
        esac;

Define legal transitions from a state
Example: Some properties we can prove

<table>
<thead>
<tr>
<th>UML</th>
<th>Petri Net</th>
</tr>
</thead>
<tbody>
<tr>
<td>G (s.state = closeobs -&gt; F a.state = turning)</td>
<td>G (closeobs -&gt; F turning)</td>
</tr>
<tr>
<td>G (a.state = turning -&gt; s.state = waiting)</td>
<td>G (turning -&gt; waiting)</td>
</tr>
<tr>
<td>G ((s.state = imagecapture</td>
<td>s.state = processimage) -&gt; a.state = moving)</td>
</tr>
</tbody>
</table>
Approach 2: Direct, Deductive Reasoning

- A model transformation can be represented as a series of rules in a control graph

- ADL (Assertion Description Language) permits reasoning on such transformations
Assertion Description Language (ADL)

- An ADL sentence is of the form:
  \(<\text{location}> : \langle\text{assertion}\rangle\)

- A location is relative to a node in the control graph, e.g. before(rule 1) or after(rule 1)

- An assertion has the form:
  \(<\text{operator}> \langle\text{pattern}\rangle\)
## ADL Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>None P</td>
<td>Pattern P is not in the model</td>
</tr>
<tr>
<td>Exists P</td>
<td>Pattern P is in the model</td>
</tr>
<tr>
<td>Any P1 → P2</td>
<td>If Pattern P1 is present, so is P2</td>
</tr>
<tr>
<td>ForOne P1 → P2</td>
<td>P2 replaces one instance of P1</td>
</tr>
<tr>
<td>ForEach P1 → P2</td>
<td>P2 replaces every instance of P1</td>
</tr>
<tr>
<td>Terminates</td>
<td>The rule terminates</td>
</tr>
</tbody>
</table>
Example: Flattening a model

Example adapted from Asztalos et al.
Example: Transformation rules
Example: Property and Precondition

- If a path exists before deleting the composite node, the path exists afterwards:
  - Before(rule 3): None P1
  - Before(rule 3): None P2

- Preconditions:
  - Before(rule 1): Any P1 → LHS1
  - Before(rule 1): Any P2 → LHS1
### Example: Proof

<table>
<thead>
<tr>
<th>Deduction</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Before(r1): Any P2 → LHS1</td>
<td>Precondition</td>
</tr>
<tr>
<td>2  After(r1): None LHS1</td>
<td>Application of rule 1</td>
</tr>
<tr>
<td>3  After(r1): None P2</td>
<td>(1), (2) + P2 is a subgraph of LHS1</td>
</tr>
<tr>
<td>4  After(r2): Exists P2 →</td>
<td>Rule 2 only removes composite edges</td>
</tr>
<tr>
<td>Before(r2): Exists P2</td>
<td></td>
</tr>
<tr>
<td>5  Before(r2): None P2 →</td>
<td>Contrapositive of (4)</td>
</tr>
<tr>
<td>After(r2): None P2</td>
<td></td>
</tr>
<tr>
<td>6  After(r2): None P2</td>
<td>Modus Ponens with (3), (5)</td>
</tr>
</tbody>
</table>

Note: Control graph is linear so Before(n) is equivalent to After(n)
## Discussion

<table>
<thead>
<tr>
<th></th>
<th>Indirect, Model Chk</th>
<th>Direct, Deductive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verification coverage</td>
<td>Single source model</td>
<td>All source models</td>
</tr>
<tr>
<td>Transformation considered...</td>
<td>Transient process</td>
<td>Persistent artifact</td>
</tr>
<tr>
<td>Generality</td>
<td>Everything a black box</td>
<td>Requires use of ADL, graph transformations</td>
</tr>
</tbody>
</table>
## Discussion

<table>
<thead>
<tr>
<th></th>
<th>Indirect, Model Chk</th>
<th>Direct, Deductive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expressivity</strong></td>
<td>Language of chosen tool</td>
<td>Pattern-based properties</td>
</tr>
<tr>
<td><strong>Effort</strong></td>
<td>Transformation to tool formalism and of properties</td>
<td>Manual proof required.</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>State explosion. Can be alleviated during translation to tool.</td>
<td>Unclear how theoretical deduction rules can be efficiently decided.</td>
</tr>
</tbody>
</table>
References


