Template Semantics for Model-Based Notations

Jianwei Niu   Joanne Atlee   Nancy Day

WATFOM
University of Waterloo
Waterloo Formal Methods (WatForm)

The use of mathematics to model and reason about computer systems - usually for the purpose of ensuring that the system will be acceptable.

Jo Atlee (me)
Nancy Day
Richard Trefler
Mark Aagaard
John Thistle

Computer Science

Electrical and Computer Engineering
WatForm Research Programs

• Practical Formalisms: Mathematical languages that are usable by practitioners.

• Automated techniques for analyzing formal models

• Combining analysis techniques to verify larger or new types of problems

• Methodologies for constructing models to facilitate analysis (e.g., abstraction)

• Case Studies
Notation-specific Analyzers

Goal: Want to analyze specifications written in our favourite specification notation

Problems:
• Proprietary or custom specification notations
• Semantic gap between specification notation and analyzer’s computation model
• Specification notations have evolving or competing semantics
Various Approaches

Direct Translation: [Zave & Jackson, Mikk et al., Chan et al., Sreemani & Atlee, Avrunin & Corbett & Dillon]

Spec in Formal Notation M → Existing Model Checker (Notation X)
Spec in Formal Notation N → Existing Model Checker (Notation Y)
Spec in Formal Notation P → Existing Model Checker (Notation Z)
Various Approaches

Direct Translation: [Zave & Jackson, Mikk et al., Chan et al., Sreemani & Atlee, Avrunin & Corbett & Dillon]

Intermediate Notations: [Bensalem et al., Bosza et al.]
Various Approaches

Generate analyzers from notation’s semantics:
[Cleaveland & Sims, Dillon & Stirewalt, Pezzè & Young, Day & Joyce]

- Spec in Formal Notation M
- Semantics for Notation M
- Evaluation Engine
- Existing Model Checker (Notation X)

- Hard to write
- Incomplete (no data variables)
We propose a template-based approach to defining model-based notations:

- Semantics common to notations are pre-defined as parameterized predicates in the template.
- A notation’s distinct semantics are specified as parameters.

Specifics of notation $M$ given by parameters.

Common Semantics

Spec in Formal Notation $M$

Existing Model Checker (Notation $X$)
Today’s Talk

Template Semantics

• Template definitions
• Template parameters
• Step semantics
• Composition operators

Generating Analyzers from Template Definitions
Computation Model

- Hierarchical Transition Systems (HTS), with:
  - States (hierarchical)
  - Internal events
  - External events
  - Variables
  - Transitions
    
    <source, trigger, condition, assignments, gen_events, dest>

- No Concurrency: concurrency introduced when composing multiple HTSs
Semantics of HTS

- **Snapshot**: observable point in execution
- **Operational Semantics**: a relation over pairs of consecutive snapshots (steps)
  - **micro-steps**: execute a single transition
  - **macro-steps**: execute a sequence of micro-steps until a stable state is reached
Semantics of HTS

• **Snapshot:** observable point in execution

  - **Basic Elements**
    - CS = current states
    - IE = current internal events
    - AV = current variable values
    - O = generated external events
Semantics of HTS

• **Snapshot**: observable point in execution

<table>
<thead>
<tr>
<th>Basic Elements</th>
<th>Auxiliary Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS = current states</td>
<td>CSa = used to determine which transitions are enabled</td>
</tr>
<tr>
<td>IE = current internal events</td>
<td>IEa =</td>
</tr>
<tr>
<td>AV = current variable values</td>
<td>AVa =</td>
</tr>
<tr>
<td>O = generated external events</td>
<td>Ia =</td>
</tr>
</tbody>
</table>
Step Semantics of HTS

Common semantics

• **enabled transitions**: identifies which transitions that are enabled by the enabling states, events, and variable values

• **apply**: apply a transition’s effects to snapshot

• **init**: initialize snapshot at start of macro-step

• **micro-step**: selects an enabled transition and applies its actions (destination state, variable-value assignments) to the snapshot

• **macro-step**: sequence of micro-steps, terminating with a stable state (in which no transition is enabled)
Step Semantics of HTS

Common semantics
• enabled transitions
• apply
• init

Template parameters
• enabling states
• enabling events
• enabling variable values
• change state
• generate events
• change variable values
• initialize state info
• initialize event info
• initialize variable info
## Template Parameters

<table>
<thead>
<tr>
<th></th>
<th>INIT</th>
<th>NEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>en_states</td>
<td></td>
<td></td>
</tr>
<tr>
<td>en_trig</td>
<td></td>
<td></td>
</tr>
<tr>
<td>en_cond</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Enabling States**: how initialized at beginning of macro-step, how changes when a transition is taken, how used to enable a transition.

**Enabling States**
- CS
- IE
- AV
- O
- CSA
- IEA
- AVA
- Ia
- en_states
- en_trig
- en_cond
Which states can enable transitions?

Options:  
- Current states
- Only current states at the beginning of the macro-step

\[ \text{en} = \exists \text{source } \in \text{CS} \]

\[
\text{CS} = \{A\}
\]

\[ \text{CS} \not\ni A \]
Which states can enable transitions?

Options:  – Current states
           – Only current states at the beginning of the macro-step

source ∈ CS

CS = \{A\}
Which states can enable transitions?

Options:
- **Current states**
- Only current states at the beginning of the macro-step
Which states can enable transitions?

Options:  

– **Current states**

– Only current states at the beginning of the macro-step

![Diagram]

- CS = \{A\}
- CS = \{B\}
- CS = \{C\}

source ∈ CS
Which states can enable transitions?

Options:

– Current states

– Only current states at the beginning of the macro-step

CS = \{A\}

B

CS = \{B\}

C

CS = \{C\}

D

source \notin CS
Which states can enable transitions?

Options:  – Current states
          – Only current states at the beginning of the macro-step

\[ \text{en\_states} \equiv \text{source} \in \text{CS} \cap \text{CSa} \]

\[ \text{CS} = \{A\} \]
\[ \text{CSa} = \{A\} \]
Which states can enable transitions?

Options:  – Current states
         – Only current states at the beginning of the macro-step

source $\in \text{CS} \cap \text{CSa}$

CS = \{A\}
CSa = \{A\}
Which states can enable transitions?

Options:  
- Current states
- Only current states at the beginning of the macro-step

source $\notin CS \cap CSa$

CS = \{A\}  
CSa = \{A\}

CS = \{B\}  
CSa = \{A\}
**Template Parameters**

- **EN** states and **IE** events:

<table>
<thead>
<tr>
<th></th>
<th>INIT</th>
<th>NEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ia</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**how initialized at beginning of macro-step**

**how changes when a transition is taken**

**Enabling Events**

- **en_states**
- **en_trig**
- **en_cond**

**how used to enable a transition**
Which events trigger transitions?

Options:  
- events generated since the beginning of the macro-step
- events generated in the last micro-step  
- events that haven’t been processed

\[ en_{\text{trig}} = \{ \text{trigger} \in I \cup IE \} \]
Which events trigger transitions?

Options:
- events generated since the beginning of the macro-step
- events generated in the last micro-step
- events that haven’t been processed

\[ \text{trigger} \in I \cup IE \]
Which events trigger transitions?

Options:  
- events generated since the beginning of the macro-step  
- events generated in the last micro-step  
- events that haven’t been processed

\[ I = \{x\} \quad I_E = \{y\} \quad \text{trigger} \in I \cup I_E \]
Which events trigger transitions?

Options:  
- events generated since the beginning of the macro-step  
- events generated in the last micro-step  
- events that haven’t been processed

\[ I = \{x\} \]
\[ IE = \{\} \]
\[ I = \{x\} \]
\[ IE = \{y\} \]
\[ I = \{x\} \]
\[ IE = \{y,z\} \]

\[ \text{trigger} \in I \cup IE \]
Which events trigger transitions?

Options:

– events generated since the beginning of the macro-step
– events generated in the last micro-step
– events that haven’t been processed
Which events trigger transitions?

Options:  
- events generated since the beginning of the macro-step  
- events generated in the last micro-step  
- events that haven't been processed  

\[
\begin{align*}
\text{en}_\text{trig} & \equiv \\
\text{trigger} & \in I \cup IE \setminus IEa
\end{align*}
\]

I={x}  
IE={}  
IEa={}
Which events trigger transitions?

Options:  
- events generated since the beginning of the macro-step
- events generated in the last micro-step
- events that haven’t been processed

\[ \text{trigger} \in I \cup IE \setminus IEa \]

```
I={x}
IE={}  
IEa={}  
```

![Diagram showing transitions between states A, B, C, and D with events x/y, y/z, and y/w]
Which events trigger transitions?

Options:

– events generated since the beginning of the macro-step
– events generated in the last micro-step
– events that haven’t been processed
Which events trigger transitions?

Options:

- events generated since the beginning of the macro-step
- events generated in the last micro-step
- events that haven’t been processed

\[ \text{trigger} \in I \cup IE \setminus IEa \]
## Template Parameters

<table>
<thead>
<tr>
<th></th>
<th>INIT</th>
<th>NEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>-</td>
<td>CS-source+dest</td>
</tr>
<tr>
<td>IE</td>
<td>{}</td>
<td>IE+gen_events</td>
</tr>
<tr>
<td>AV</td>
<td>-</td>
<td>AV ⊕ assignments</td>
</tr>
<tr>
<td>O</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Csa</td>
<td>CS</td>
<td>CSa</td>
</tr>
<tr>
<td>IEa</td>
<td>{}</td>
<td>IEa+trig</td>
</tr>
<tr>
<td>AVa</td>
<td>AV</td>
<td>AVa</td>
</tr>
<tr>
<td>Ia</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Enabling States

- **en_states**: source ∈ CSa
- **en_trig**: trigger ∈ (I ∪ IE) \ IEa
- **en_cond**: AVa ⊨ condition

**INIT**
- how initialized at beginning of macro-step

**NEXT**
- how changes when a transition is taken

**Enabling Events**

**Enabling Variables**
Composition Operators

Compose HTSs or collections of HTSs:
Composition Operators

• Constrain when components can take a step

• Share snapshot information:
  – communication of events
  – consistent values among shared variables
**Example: Parallel Composition**

**Case 1:**
Both components are enabled and execute simultaneously

**Case 2:**
One component is enabled and executes in isolation
Example: Interrupt Composition

Case 1:
One component has control; it executes

Case 2:
Control transfers to/from one component to the other
Composition Operators

• Steps are macro-steps as defined by template

• Components share information by updating snapshot elements with data from other components
  – events, shared variables, auxiliary snapshot elements
  – snapshots updated as defined by template-parameters
Interrupt Composition

\[ N_{\text{micro}}^{\text{interr}}((s\tilde{s}_1, s\tilde{s}_2), (s\tilde{s}_1', s\tilde{s}_2'), (\tilde{\tau}_1, \tilde{\tau}_2)) \leq_{\text{interr}} = \]

\[ \exists \tilde{\tau}_1, \tilde{\tau}. \left[ \begin{array}{l}
\land \quad s\tilde{s}_1.CS \neq \emptyset \land N_{\text{micro}}^1(s\tilde{s}_1, \tilde{iss}_1, \tilde{\tau}) \land \text{higher-\text{pri}}(\tilde{\tau}, \text{pri}(\text{enabled_trans}(s\tilde{s}_1, T_{\text{interr}}))) \\
\land \quad N_{\text{micro}}^1(s\tilde{s}_1, s\tilde{s}_1', \tilde{\tau}_1) \land \quad \tilde{s}_2' = \text{update}(s\tilde{s}_2, \tilde{\tau}_1) \\
\land \quad \tilde{\tau}_2 = \emptyset \land \text{higher-\text{pri}}(\tilde{\tau}, \tilde{\tau})
\end{array} \right] \quad (\text{* component 1 steps *})
\]

\[ \forall \tau. \left[ \begin{array}{l}
\land \quad \tau \in \text{pri}(\text{enabled_trans}(s\tilde{s}_1, T_{\text{interr}})) \land (\forall \tilde{iss}_1, \tilde{\tau}. N_{\text{micro}}^1(s\tilde{s}_1, \tilde{iss}_1, \tilde{\tau}) \implies \text{higher-\text{pri}}(\{\tau\}, \tilde{\tau}))
\end{array} \right] \quad (\text{* transition to component 2 *})
\]

\[ \exists \tau. \left[ \begin{array}{l}
\land \quad s\tilde{s}_1 = \text{update}(s\tilde{s}_1, \tau) \quad |_{CS} \land \quad \tilde{\tau}_1 = \emptyset \land \quad \tilde{\tau}_2 = \emptyset \\
\land \quad s\tilde{s}_2' = \text{update}(s\tilde{s}_2, \tau) \quad |_{\text{ent-comp(\tau)}}^{CSh} |_{n\_states\_his(s\tilde{s}_2, \tau)}^{EE_h} |_{n\_ext-ev\_his(s\tilde{s}_1, \tau)}
\end{array} \right] \quad (\text{* symmetric cases of two above replaced 1 with 2 and 2 with 1 *})
\]

Figure 5: Semantics of interrupt semantics for micro-steps
Composition Operators

- parallel
- interleaving
- sequence
- choice

- synchronization:
  - environmental
  - rendezvous

- interrupt
Today’s Talk

Template Semantics

- Template definitions
- Template parameters
- Step semantics
- Composition operators

Generating Analyzers from Template Definitions
Metro

Idea: To generate model compilers from notations’ semantics.

A model compiler compiles a specification into a more primitive representation, according to the notation’s computation model.
Metro

An instantiated template is a model compiler. It defines a notation’s semantics in terms of allowable execution steps.
Metro

When the template’s definitions are applied to a specification, it generates a transition-relation representation that is suitable for analysis.
Summary

We have developed a template approach to defining the operational semantics of model-based notations

• The result is a succinct method of describing the semantics of a specification notation

• Makes it easier to understand and to compare notations

• Makes it easier (possible) to compile specifications in a representation that is more suitable for automated analysis
Current Status

• We have defined the generic template definitions 
  *enabled-transition, apply, init, micro-step, macro-step*

• We have defined template parameters and 
  composition operators for several popular notations 
  *various statecharts variants, RSML, SCR, SDL88, Petri-Nets, process algebras*

• We have implemented a vertical slice of the Metro 
  model-compiler generator to handle Basic State 
  Transitions 
  *enabled-transition, apply, macro-step, interleaving composition,*
Future Work

• Continuing the implementation of Metro model-compiler generator

• Applying Metro to more sophisticated notations
  – Z?
  – Abstract State Machines?
  – Paderborn’s semantics for UML statecharts?

• Handling multi-notation specifications