Generic Instantiation and Tool Support

Thai Son Hoang

Institute of Information Security, ETH Zurich, Switzerland

Rodin Workshop 2013, Turku, Finland
11th June 2013

(joint work between ETH Zurich and Hitachi Ltd.)

Tackling the complexity of systems modelling

Abrial and Hallerstede (2007)

. . . modeling a large and complex computer system results in a large and complex model. . . . proofs will be more and more difficult to perform as models become inevitably larger and larger.

. . . We present three techniques, refinement, decomposition, and instantiation, that we consider indispensable for modeling large and complex systems.

Refinement, Decomposition, and Instantiation

Current Status

- **Refinement**: An integral part of Event-B
- **Decomposition**: Shared-event / shared-variable decomposition
- **Generic instantiation**:
  - R. Silva and M. Butler: Supporting Reuse of Event-B Developments through Generic Instantiation. (ICFEM 2009)
  - Ulyana Tikhonova at. al. (Rodin Workshop 2013) (this morning)

Generic Instantiation

\[
C_0 \text{ extends } C_1 \text{ extends } \ldots \text{ extends } C_n
\]

\[
\text{sees } \ldots \text{ sees } \ldots \text{ sees } \]

\[
M_0 \text{ refines } M_1 \text{ refines } \ldots \text{ refines } M_n
\]

- The development is parameterised by \( S \) and \( c \)
- Reuse model: Instantiating to \( S = E(T) \) and \( c = F(T, d) \)

\[
B(T, d) \Rightarrow A(E(T), F(T, d))
\]
### Instantiating Sets and Constants. An Example

**Generic context**

- **sets**: MESSAGE
- **constants**: maxsize
- **axioms**:
  - finite(MESSAGE)
  - maxsize $\in \mathbb{N}^*$

**Specific context**

- **sets**: ID, INFO
- **axioms**:
  - finite(ID)
  - finite(INFO)

- Instantiation: MESSAGE = ID $\times$ INFO, maxsize = 3
- To be proved:
  \[
  \text{finite}(ID) \land \text{finite}(INFO) \Rightarrow \text{finite}(ID \times INFO) \land 3 \in \mathbb{N}^*
  \]

---

### A Generic Instantiation Tool

![Diagram](diagram.png)

---

### The Usefulness of Generic Instantiation

The similarity between refinement and generic instantiation

#### The Usefulness of Refinement

The ability to perform abstraction with state variables.

#### The Usefulness of Generic Instantiation

The ability to perform abstraction with sets and constants

---

### Demo
Train Control Example (1)

Typical modelling style using variables

\[
\begin{align*}
\text{trains} & \subseteq \text{TRAIN\_ID} \\
\text{head} & \in \text{trains} \rightarrow \text{SECTION} \\
\text{rear} & \in \text{trains} \rightarrow \text{SECTION} \\
\text{area} & \in \text{trains} \rightarrow \mathcal{P}(\text{SECTION}) \\
\text{connection} & \in \text{trains} \rightarrow (\text{SECTION} \rightarrow \text{SECTION})
\end{align*}
\]

Train Control Example (2)

- Trains can be represented by some abstract data type (e.g., sequence of sections \(\text{SEQUENCE}\)).
- Operations with sequences:
  - Extend head: \(\text{extend} \in \text{SEQUENCE} \times \text{SECTION} \rightarrow \text{SEQUENCE}\)
  - Remove rear: \(\text{front} \in \text{SEQUENCE} \rightarrow \text{SEQUENCE}\)
  - Disjointness: \(s_1 \mapsto s_2 \in \text{disjoint}\)
  - Sub-sequence: \(s_1 \mapsto s_2 \in \text{subset}\)
- Properties of sequences

\[
\begin{align*}
\forall s_1, s_2, s_3.
\text{if } s_1 \mapsto s_2 \in \text{subset} \land s_2 \mapsto s_3 \in \text{disjoint} \Rightarrow s_1 \mapsto s_3 \in \text{disjoint} \\
\forall s_1, s_2, s_1 \mapsto s_2 \in \text{disjoint} \Rightarrow s_2 \mapsto s_1 \in \text{disjoint}
\end{align*}
\]

Summary

- Instantiation enables abstraction with models’ parameters.
- Context instantiation as a part of the development?

\[
\begin{align*}
C_0 & \text{ extends } C_1 & \text{ extends } C_n & \text{ instantiates } D_0 \\
& \text{ sees } & \text{ sees } & \text{ sees } & \text{ sees } \\
& \text{ sees } & \text{ sees } & \text{ sees } & \text{ sees } \\
M_0 & \text{ refines } M_1 & \text{ refines } M_n & \text{ refines } N_0
\end{align*}
\]