Augmenting formal development with use case reasoning

(abstract)

Alexei Iliasov

Newcastle University, UK

State-based methods for correct-by-construction software development rely on a combination of safety constraints and refinement obligations to demonstrate design correctness. One prominent challenge, especially in an industrial setting, is ensuring that a design is adequate: requirements compliant and fit for purpose. The paper presents a technique for augmenting state-based, refinement-driven formal developments with reasoning about use case scenarios; in particular, it discusses a way for the derivation of formal verification conditions from a high-level, diagrammatic language of use cases, and the methodological role of use cases in a formal modelling process.

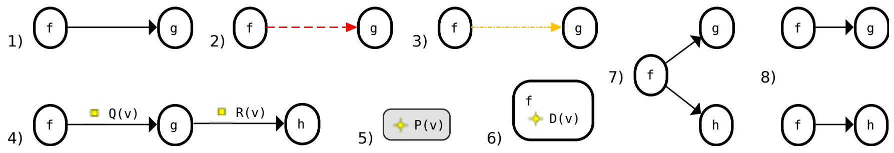
The approach to use case reasoning is based on our previous work on a graphical notation for expressing event ordering constraints [2, 1]. The extensions is realised as a plug in to the Event-B modelling tool set - the Rodin Platform [3] - and smoothly integrates into the Event-B modelling process. It provides a modelling environment for working with graph-like diagrams describing event ordering properties. In the simplest case, a node of such graph is an event of the associated Event-B machine; an edge is a statement about the relative properties of the connected nodes/events. There are three main edge kinds - **ena**, **dis** and **fis** - defined as relations over Event-B events.

$$\begin{aligned}
U &= \{f \mapsto g \mid \varnothing \subset f \subseteq S \times S \wedge \varnothing \subset g \subseteq S \times S\} \\
\mathbf{ena} &= \{f \mapsto g \mid f \mapsto g \in U \wedge \mathrm{ran}(f) \subseteq \mathrm{dom}(g)\} \\
\mathbf{dis} &= \{f \mapsto g \mid f \mapsto g \in U \wedge \mathrm{ran}(f) \cap \mathrm{dom}(g) = \varnothing\} \\
\mathbf{fis} &= \{f \mapsto g \mid f \mapsto g \in U \wedge \mathrm{ran}(f) \cap \mathrm{dom}(g) \neq \varnothing\}
\end{aligned}$$

where $f \subseteq S \times S$ is a relational model of an Event-B event (we treat an event as a next-state relation). These definitions are converted into *consistency* proof obligations. For instance, if in a use case graph there appears an **ena** edge connecting events b and h one would have to prove the following theorem (see [2] for a justification).

$$\forall v, v', p_b \cdot I(v) \wedge G_b(p_b, v) \wedge R_b(p_b, v, v') \Rightarrow \exists p_h \cdot G_h(p_h, v') \qquad (1)$$

A use case diagram is only defined in an association with one Event-B model, it does not exist on its own. The use case plug in automatically generates all the relevant proof obligations. A change in a diagram or its Event-B model leads to the re-computation of all affected proof obligations. These proof obligations are dealt with, like all other proof obligation types, by a combination of automated provers and interactive proof. Like in the proofs of model consistency



1) f **ena** g
2) f **dis** g
3) f **fis** g
4) f **ena** $\mathrm{q}.g \wedge \mathrm{q}.g$ **ena** $\mathrm{r}.h$
$\mathrm{q} = \{v \mid Q(v)\}, \mathrm{r} = \{v \mid R(v)\}$

5) $skip(C), C = \{v \mid P(v)\}$
6) $f(C), C = \{v \mid D(v)\}$
7) f **ena** $g \vee f$ **ena** h
8) f **ena** $g \wedge f$ **ena** h

Fig. 1. A summary of the core use case notation and its interpretation.

and refinement, the feedback from an undischarged use case proof obligation may often be interpreted as a suggestion of a diagram change such as an additional assumptions or assertion - predicate annotations on graph edges that propagate properties along the graph structure. The example in the next section demonstrates how such annotations enable the proof of a non-trivial property.

The use case tool offers a rich visual notation. The basic element of a diagram is an event, visually depicted as a node (in Figure 1, f and g represent events). Event definition (its parameters, guard and action) is imported from the associated Event-B model. One special case of node is skip event, denoted by a grey node colour (Figure 1, 5). Event relations **ena**, **dis**, **fis** are represented by edges connecting nodes ((Figure 1, 1-3)). Depending on how a diagram is drawn, edges are said to be in *and* or *or* relation (Figure 1, 7-8). New events are derived from model events by strengthening their guards (a case of symmetric assumption and assertion) (Figure 1, 6). Edges may be annotated with constraining predicates inducing assertion and assumption derived events (Figure 1, 4). Not shown on Figure 1 are nodes for the initialisation event start (circle), implicit deadlock event stop (filled circle) and nodes for container elements such as loop (used in the coming example). To avoid visual clutter, the repeating parts of a diagram may be declared separately as diagram *aspects*[2].

References

1. Alexei Iliasov. Augmenting Event-B Specifications with Control Flow Information. In *NODES 2010*, May 2010.
2. Alexei Iliasov. Use case scenarios as verification conditions: Event-B/Flow approach. In *Proceedings of 3rd International Workshop on Software Engineering for Resilient Systems*, Septembre 2011.
3. The RODIN platform. Online at http://rodin-b-sharp.sourceforge.net/.