A (Small) Improvement of Event-B?

Stefan Hallerstede

Institut für Informatik, Universität Düsseldorf
Düsseldorf, Germany
halstefa@cs.uni-duesseldorf.de

Abstract. Event-B and the Rodin tool use a number of simple techniques that make the modelling method around them effective in practical applications. We present two of these techniques, anticipation and witnesses. It is interesting how a couple of very simple techniques are so important for the method to work. Finally we propose a small enhancement of Event-B that would extend the use of witnesses.

Keywords. Event-B, Formal Methods, Methodology, Proof

1 Introduction

We believe that the main purpose of modelling is reasoning, finding out why something works or why it does not. Reasoning should be formal in order to achieve a high degree of certainty about the corresponding claims we make. In the Event-B modelling method [1] reasoning is supported by formal proof. With each formal model we create a number of proof obligations is associated that, once discharged, establish certain properties of the model. Reasoning is not confined to carrying out a formal proof though. Whenever we fail to discharge some proof obligations, we modify the model and try to discharge the proof obligations of the modified model, and so on (Figure 1). This method of reasoning is supported by the Event-B formalism that has been designed to achieve a close correspondence between models and proof obligations. Event-B is intended for the modelling of complex systems. The large number of details of a complex system to be considered is introduced piecemeal by formal refinement [2]. We use refinement more as a technique to structure complex proofs, focusing less on preserving correctness along a sequence of models.

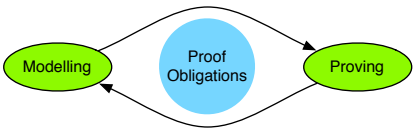

Fig. 1: Formal Reasoning in Event-B

During the evolution of Event-B [1,3,4,5,6,7,8,9,10,11] many decisions and developments have been made to make Event-B an effective practical modelling method. Many of those are as simple as effective. In this paper we discuss two of them, the use of anticipated events [6] and of refinement witnesses [10].

In Section 2 we briefly discuss anticipation of events and in Section 3 we discuss refinement witnesses and suggest a small improvement of Event-B: to use witnesses also for non-deterministic assignments.[1]

2 Anticipation

Anticipated events are described in [6] as a technique to couple events introduced during refinement with their variant and decouple them from variables. The approach solves the technical problem of finding a good ordering for a chain of refinements by relaxing the constraints on that order thus increasing the number of good orderings. Anticipated events can be used to avoid using lexicographic variants altogether, $(m \mapsto n) < (m' \mapsto n') \Leftrightarrow m < m' \vee (m = m' \wedge n < n')$. Say, m is the variant of a machine M and n the variant of some refinement N of M. For an anticipated event f of M we would show that it does not increase m, that is, $m < m'$. Ultimately, event f has to be refined by a convergent event f in machine N, say. In machine N we have to show that f decreases the variant n, that is, $n < n'$. Following this technique we implicitly construct a lexicographic variant $(m \mapsto n)$, similar to the one shown above.

Anticipated events have turned out to be a very useful concept for modelling beyond the original purpose. Below are three examples of their practical use.

Counter abstraction: specification of an abstract timer. An abstract model of timer needs only express that after an arbitrary finite amount of time it will raise an alarm. The counter by which it could be implemented is irrelevant.

invariants
$alarm \in BOOL$
$v \in 0 \mathinner{..} 1$
$v = 1 \Rightarrow alarm = TRUE$

event *INITIALISATION*
$alarm :\in BOOL$
$v := 0$

anticipated **event** *tick*
when $v = 0$
then $alarm :\in BOOL$

event *one*
when $alarm = TRUE$
then $v := 1$

In some refinement the timer can be implemented by a counter or in some other way that provides convergence.

Fewer variables: avoid introducing new variables. In Event-B new events must refine *skip*. This means that usually variables manipulated in a loop can not be variables from an abstract machine. The gcd algorithm, for instance, computes its result in variable q:

$q \longleftarrow getgcd(x, y)$
$p, q := x, y;$
do $p < q \rightarrow q := q - p$
$\square \ q < p \rightarrow p := p - q$
od

event *getgcd*
$q := gcd(x, y)$

anticipated **event** *loop2*
$q :\in \mathbb{N}_1$

[1] We do not present an introduction to Event-B. Introductions to Event-B can be found in the references mentioned above.

In the corresponding abstract Event-B machine we model the body of the loop by an anticipated event *loop2*, specifying that in some refinement *loop2* is implemented by a convergent event that may modify variable q.

Decomposition of a proof: partial correctness and termination. When introducing a loop in a development of a sequential program, we have to prove that the loop body preserves the invariant and decreases a variant. If we make the loop body an anticipated event, we can prove invariant preservation at one stage and termination at a later stage. This reduces the complexity at each stage and separates concerns of invariant preservation from termination.

3 Witnesses

Originally, witnesses have been introduced in Event-B in order to decompose proof obligations [12] but also the methodical benefits had been recognised [10]. The reasoning underlying the use of witnesses is very simple: Let v be the abstract variables, $I(v)$ the abstract invariant, w the concrete variables, $J(v, w)$ the gluing invariant, p the abstract event parameters, $G(p, v)$ the abstract event guard, $S(p, v, v')$ the abstract event actions before-after predicate, q be the concrete event parameter, $H(q, w)$ the concrete event guard, and $T(q, w, w')$ the concrete event actions before-after predicate. With

$$\begin{aligned} K(v, q, w, w') &\;\widehat{=}\; I(v) \wedge J(v, w) \wedge H(q, w) \wedge T(q, w, w') \\ L(p, v, w, w') &\;\widehat{=}\; G(p, v) \wedge S(p, v, v') \wedge J(v', w') \end{aligned}$$

the refinement proof obligation for the refinement of the abstract by the concrete event is thus:

$$K(v, q, w, w') \quad \Rightarrow \quad \exists\, p,\, v' \cdot L(p, v, w, w') \quad . \tag{1}$$

In the introduction we discussed the close correspondence between Event-B models and proof obligations. We see that the granularity of the correspondence could be improved greatly if (1) could be decomposed into three implications with conclusions $G(p, v)$, $S(p, v, v')$, and $J(v', w')$, for instance. Because of the existential quantification "$\exists\, p,\, v'$" this is not possible. In some step of the proof of (1) we would usually instantiate the bound identifiers p and v' by expressions r and u', subsequently, proving the conclusions $G(r, v)$, $S(r, v, u')$, and $J(u', w')$ separately. We can do this systematically for all refinements specifying *witnesses* $W(p, v, v', q, w, w')$ that serve to instantiate p and v'. Of course, we have to verify that the witnesses exist [2]

$$K(v, q, w, w') \quad \Rightarrow \quad \exists\, p,\, v' \cdot W(p, v, v', q, w, w') \quad . \tag{2}$$

Applying modus ponens in the premises of (1) and observing that p and v' do not occur free in (1) the following (3) implies (1)

$$K(v, q, w, w') \wedge W(p, v, v', q, w, w') \quad \Rightarrow \quad \exists\, p,\, v' \cdot L(p, v, w, w') \quad . \tag{3}$$

[2] This proof obligation is usually a simple consequence of the invariant and easily discharged.

Finally, we can instantiate "$p, v' := p, v'$" in the conclusion so that (4) implies (3), hence, also (1):

$$K(v, q, w, w') \wedge W(p, v, v', q, w, w') \quad \Rightarrow \quad L(p, v, w, w') \quad . \tag{4}$$

Implication (4) can now be decomposed and proved separately for each conjunct $G(p, v)$, $S(p, v, v')$, and $J(v', w')$ in the conclusion.[3] As an example of the use of witnesses we refine the abstract timer from the preceding section where witnesses are specified in the **with** clauses of the events:

invariants
$time \in \mathbb{N}$
$time = 0 \Rightarrow alarm = TRUE$
$time > 0 \Rightarrow alarm = FALSE$

convergent **event** *tick*
when $time > 0$
with $alarm' = bool(time' = 0)$
then $time := time - 1$

event *INITIALISATION*
with $alarm' = FALSE$
then
$time :\in \mathbb{N}_1$
$v := 0$

event *one*
when $time = 0$
then $v := 1$

More complicated examples of witnesses can be found, for instance, in [13]. Witnesses have turned out to be very valuable for explaining how refinement is achieved. They have also increased the potential of animation of Event-B models opening up an efficient possibility for refinement animation [14].

A small improvement: witnesses for non-deterministic choices. We have shown how witnesses are used to obtain proof obligations that are easier to handle than (1). In practice, we observe however, that the refinement condition $K(v, q, w, w') \wedge W(p, v, v', q, w, w') \Rightarrow S(p, v, v')$ that we have obtained from the decomposition often has itself a form similar to (1).

In order to be able to use witnesses for non-deterministic assignments, too, in the absence of explicit support, non-deterministic assignments can be modelled by means of guards [15,14] and distinguished by a labelling convention; for instance, guards are labelled grd$n$ and non-deterministic assignments chc$n$. However, this complicates support for certain proof obligations, in particular, deadlock freedom. (Relying on a labelling convention for proof obligation generation does not appear reliable.) The best solution would seem to treat non-deterministic assignments the same way as guards. An additional benefit would be that the need for primed identifiers in Event-B models would disappear, too, simplifying further the existing concept of witnesses described above.

4 Conclusion

The modelling method Event-B is effective in practice because of a number of simple techniques that have been incorporated into it. They usually originated as

[3] Note, that we have not renamed any identifiers in the conclusion which would have obscured the correspondence with model.

a solution to some technical problem but then proved to be useful in a much wider context. We believe that it is important to take note of such developments. If we keep record of the small improvements, we can mark the methodical progress we make and we avoid losing the knowledge about why Event-B works.

References

1. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2009) To appear.
2. Back, R.J.: Refinement Calculus II: Parallel and Reactive Programs. In deBakker, J.W., deRoever, W.P., Rozenberg, G., eds.: Stepwise Refinement of Distributed Systems. Volume 430 of Lecture Notes in Computer Science., Springer (1989) 67–93
3. Abrial, J.R., Butler, M., Hallerstede, S., Voisin, L.: An open extensible tool environment for Event-B. In Liu, Z., He, J., eds.: ICFEM 2006. Volume 4260., Springer (2006) 588–605
4. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: An Open Toolset for Modelling and Reasoning in Event-B. Software Tools for Technology Transfer (2009) To appear.
5. Abrial, J.R., Cansell, D.: Click'n'Prove: Interactive Proofs within Set Theory. In: Theorem Proving in Higher Order Logics. Volume 2758 of LNCS. (2003) 1–24
6. Abrial, J.R., Cansell, D., Méry, D.: Refinement and Reachability in EventB. In Treharne, H., King, S., Henson, M., Schneider, S., eds.: ZB 2005. Volume 3455 of LNCS. (2005) 222–241
7. Abrial, J.R., Hallerstede, S.: Refinement, Decomposition and Instantiation of Discrete Models: Application to Event-B. Fundamentae Informatica **77** (2007) 1–28
8. Abrial, J.R., Mussat, L.: Introducing dynamic constraints in B. In Bert, D., ed.: B'98 : The 2nd International B Conference. Volume 1393 of LNCS., Springer (1998) 83–128
9. Abrial, J.R., Mussat, L.: On using conditional definitions in formal theories. In Bert, D., Bowen, J., Henson, M., Robinson, K., eds.: ZB 2002. Volume 2272 of LNCS. (2002) 242–269
10. Hallerstede, S.: Justifications for the Event-B Modelling Notation. In Julliand, J., Kouchnarenko, O., eds.: B 2007. Volume 4355 of LNCS., Springer (2007) 49–63
11. Métayer, C., Jean-Raymond-Abrial, Vosin, L.: Event-B Language. Technical report, ETH Zürich (2005)
12. Hallerstede, S.: The Event-B Proof Obligation Generator. Technical report, ETH Zürich (2005)
13. Hallerstede, S.: Proving Quicksort correct in Event-B. In Boiten, E., Derrick, J., eds.: Refine 2009. ENTCS (2009)
14. Hallerstede, S., Leuschel, M., Plagge, D.: Refinement-Animation for Event-B — Towards a Method of Validation. In: ABZ 2010. LNCS, Springer (2007) 14 pages. To appear.
15. Colley, J.: Private communication (2009)