Animating and Model Checking B Specifications with Higher-Order Recursive Functions*

Michael Leuschel and Jens Bendisposto
Institut für Informatik
Heinrich-Heine Universität Düsseldorf
Universitätsstr. 1, D-40225 Düsseldorf
leuschel@cs.uni-duesseldorf.de

The B-method [1] is a theory and methodology for formal development of computer systems. It is used in industry in a range of critical domains. In addition to the proof activities it is increasingly being realised that validation of the initial specification is important, as otherwise a correct implementation of an incorrect specification is being developed. This validation can come in the form of *animation*, e.g., to check that certain functionality is present in the specification. Another useful tool is *model checking*, whereby the specification can be systematically checked for certain temporal properties. In previous work [2], we have presented the ProB animator and model checker to support those activities.

In this work we present two important improvements upon previous work.

First, realistic specifications often contain complicated functions. Take the following excerpt of a specification (translated from a Z specification given to us by Anthony Hall):

```
removeDuplicates = {ss,rs | ss: seq(seq(PLACE)) & rs:seq(seq(PLACE))
    &    (ss=<> => rs=<>) &
         (card(ss)=1 => rs=ss) &
         (card(ss)>1 => ( #(s1,s2).(s1:seq(PLACE) & s1=first(ss)
                          & s2:seq(PLACE) & s2=ss(2) &
               (last(s1)= first(s2) =>
                  rs = front(s1) -> removeDuplicates(tail(ss)) )
            &  (last(s1)/=first(s2) =>
                  rs = s1 -> removeDuplicates(tail(ss)))   )))}
```

Animation and validation of such functions and specifications is very important, to ensure that the specified functions actually compute the correct output. However, such functions pose a major challenge to animation and model checking. Earlier versions of ProB required that the whole function *removeDuplicates* be explicitly computed, which is prohibitively expensive or impossible. The central idea of this new research is to compile such functions into symbolic closures

*This research is being carried out as part of the EU funded research projects: IST 511599 RODIN (Rigorous Open Development Environment for Complex Systems).

Dagstuhl Seminar Proceedings 06191
Rigorous Methods for Software Construction and Analysis
http://drops.dagstuhl.de/opus/volltexte/2006/640

which are only examined when the function is applied to some particular argument. In the case of recursive closures, these also need to be unrolled on demand. This enables ProB to successfully animate and model check a new class of specifications, where animation is especially important due to the involved nature of the specification.

The general schema for writing recursive functions that can be symbolically animated is as follows:

$$f = \{arg_1, arg_2, \ldots, argn, out \mid arg_1 \in T1 \wedge \ldots out \in T_{n+1} \wedge$$
$$(COND_1 => out = EXP1) \wedge$$
$$\ldots$$
$$(COND_N => out = EXP_N)\}$$

where $COND_i$ are mutually exclusive and cover the entire domain of the function, and where EXP_i can make reference to f. For example, the factorial function can be written (and then animated by ProB) as follows:

```
CONSTANTS   fact
PROPERTIES
 fact = {x,y| x:NAT & y:NAT &
   (x=0 => y=1) &
   (x>0 => (y=x*fact(x-1)))
 } &
 fact: NAT --> NAT
```

Second, it is important that complicated specifications can be animated in such a way that domain experts can easily validate whether the specification corresponds to their expectations. For this we have developed a generic Flash -based animation engine which allows to easily develop visualizations for a given specification. This generic Flash movie connects through a TCP Socket to the Server, that is integrated into a new Eclipse RCP based release of the ProB animator shown in Figure 1. The information is exchanged using XML-Fragments. One of the main advantages is that the animation can be shown to several users simultaneously; the only requirement is a web browser with installed Flash-Plugin. Our tool supports state-based animations, which use simple pictures to represent a specific state of the B-model, and transition-based animations consisting of picture sequences. To avoid the creation of many different animations the tool supports the composition the visualization of several separated parts. In addition to the graphical elements, gluing code is needed that maps the states and transitions of a machine to the graphical representation. For this we use the Java Beanshell, which allows to write interpreted Java code that can be modified at runtime. The tool provides a set of Java objects that represent the items in a Flash movie like animation clips or labels.

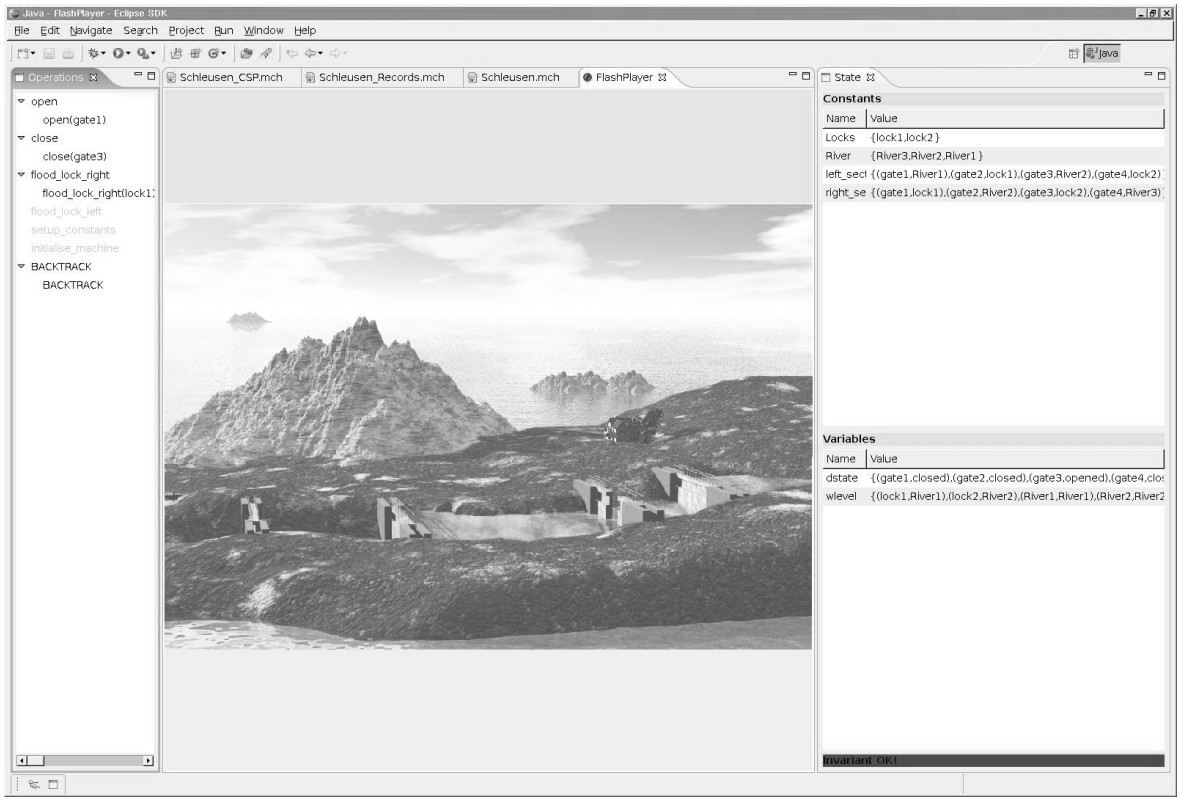

Figure 1: Eclipse Version of the ProB animator

Future work

Writing the gluing code is still an annoying task, therefore we will develop a toolkit for rapid development of the flash animations including a code generator for the gluing code. Another task is the option to control ProB from the animation to build user interface prototypes from B machines.

References

[1] J.-R. Abrial. *The B-Book*. Cambridge University Press, 1996.

[2] M. Leuschel and M. Butler. ProB: A model checker for B. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME 2003: Formal Methods*, LNCS 2805, pages 855–874. Springer-Verlag, 2003.