Towards Verified Implementation of Event-B Models in Dafny

Mohammadsadegh Dalvandi, Michael Butler
University of Southampton

{md5g11,mjb}@ecs.soton.ac.uk

Ideally the development process of software systems with Event-B and Rodin platform should lead to implementation and executables. The Event-B language and Rodin in their basic form do not have any facility to support this. To bridge this gap, some research has been carried out and several plugins have been developed to provide facilities for the Rodin platform to make code generation possible. While the specification and modelling phase of the system can be proved by automatic and interactive provers in Event-B and Rodin, the generated code by most of the existing code generators is not verified. One way to tackle this issue is to verify the generated code with a static program verifier. A static program verifier proves the correctness of a code with regards to a well-defined formal specification. Using this approach to generate verifiable code from Event-B models requires the generation of not only the code itself but also verification assertions from the Event-B.

The focus of this research is on linking Event-B models and their implementation in Dafny [1]. Dafny is a programming language and program verifier based on the Z3 SMT-solver. Dafny has proved to be a powerful program verifier by verifying a number of challenging problems [2]. By providing formal specifications for a Dafny program, the verifier proves the correctness (or incorrectness) of the code with regards to its specifications. The specifications includes pre- and post-conditions, loop invariants, and variable framing. The language also supports specification-only updateable ghost variables and functions which can be used recursively. Ghost entities (ghost variables, functions and etc.) are only used for verification purposes and do not form part of the compiled code.

To identify the differences and similarities between Event-B models and Dafny specifications and programs, implementation of a map abstract data type was taken from [2] for a case study and modelled in Event-B. The Map ADT is implemented in Dafny by a linked-list and specified by two sequences: one for storing keys and the other for storing associated values. In Event-B, the map is modelled by an abstract model following by two successive refinements. In the abstract level the map is modelled simply by the use of a partial function where keys are in domain and values are in range. Sequences are introduced in first refinement and the linked-list is added in the second level.

To decrease the distance between Event-B and Dafny syntax, the standard Event-B may be extended by Theory Plug-in [3] and new data-types and operators may be defined. For modelling maps in Event-B, several new operators, inference rules and rewrite rules have been added to the existing theory of sequences. Without using the theory plugin there will be a huge syntactic gap between Event-B and Dafny.

Although Dafny does not have any special object invariant construct, this can be easily done by defining a validity function and using it as both pre- and post-conditions of every method. This can be seen equivalent to Event-B invariants. Therefore all invariants of the Event-B model can be placed within the body of the validity function and vice versa. The precondition of methods in the map implementations is only the validity function. Apart from the validity function, which must be placed in the post-condition of all methods, other post-conditions are needed to specify the desirable and exact behaviour of a method. This is essential because of Dafnys modular verification approach. The Dafny verifier only looks at the specification (pre- and post-conditions) of the other methods to understand their behaviour when it is verifying a method. Modelling the map data structure in Event-B shows that the required post-conditions can be inferred from guards and actions of each event. One or more events in an Event-B model may be equivalent to a method in a Dafny program. Related events can be translated to a single method in Dafny and each event will translate to a conditional statement within that method. Guards of each event will be conditions and actions will be consequence of that conditional statement so each event regarding to its guards will perform a possible action by the method.

The following code snippets show an Event-B event and its equivalent implementation in Dafny. The event models the way in which a new key and its value is added to the map. The Dafny code contains both implementation and specification which describes the desirable behaviour of the method:

```
Add1 ≙
        any k,v
        where
                @grd1: k∉ran(keys)
        then
                @act1: keys := seqPrepend(keys, k)
                @act2: values := seqPrepend(values, v)
        end
```

```
method Add(k: KEY, v: VALUE)
        ...
        ensures k !in old(keys) ==> keys == [k] + old(keys)
                && values == [v] + old(values);
  {
        ...
        if(k !in keys){
                keys := [k] + keys;
                values := [v] + values;
        }
        ...
  }
```

There are some important issues about linking Event-B models with Dafny implementations which should be addressed and are subject of this ongoing research. There is a considerable syntactic gap between Event-B and Dafny. Event-B language with the help of the Theory Plug-in can be extended. This very interesting feature can help to decrease the gap. Although this gives rise to the problem of how translating newly defined operators to Dafny implementations where they dont have an equivalent in Dafny. Another issue to be investigated is refinement. Answering the question that how and which refinement levels should be used for generating specification and code is important. Is there a specific guideline that should be followed during modelling phase in order to be able to generate specification and code from the Event-B models is another valid question. After answering these questions and proposing a comprehensive approach to generate Dafny specification and code from the models, developing a tool for automation of the code generation process should be planned.

References

[1] K. R. M. Leino, “Dafny: An automatic program verifier for functional correctness,” in *Logic for Programming, Artificial Intelligence, and Reasoning.* Springer, Conference Proceedings, pp. 348–370.

[2] K. R. M. Leino and R. Monahan, “Dafny meets the verification benchmarks challenge,” pp. 112–126, 2010.

[3] M. Butler and I. Maamria, “Practical theory extension in event-b,” in *Theories of Programming and Formal Methods.* Springer, 2013, pp. 67–81.