A Report on PRiME Code Generation Activities

Mohammadsadegh Dalvandi, Asieh Salehi Fathabadi and Michael Butler

{md5g11, asf08r, mjb}@ecs.soton.ac.uk
University of Southampton

In this abstract, we briefly present our experience in generating code from Event-B models of run-time management software for multi-core embedded platforms as part of the PRiME project[1]. We discuss the current limitation in Event-B Code Generation (CG) and outline our plan for a new CG tool.

1 RTM Code Generation

Development of RTM software can require laborious manual adjustment across different hardware platforms due to their different architectural characteristics. For instance, the same Q-Learning RTM [4] algorithm needs to be adjusted for different ARM-based platforms like Cortex A7 and A15, because each of the platforms support different sets of frequencies. To tackle this, we used Event-B to develop platform-independent models of the RTM software and extended the Tasking Event-B CG plug-in [3] to facilitate generation of platform-dependent code from the abstract Event-B model. The extension to the Tasking Event-B plug-in involves the introduction of *expanding* guards. Expanding guards direct the CG to generate an arbitrary number of branches in the generated code based on a single event and an expanding parameter. Using this approach, the CG can instantiate the platform independent model for an specific target platform and generate the code with very minimal modification to the Event-B model. We used this approach for generating platform-specific code for Cortex A7 and A15 platforms from a single Event-B model of the Q-Learning RTM algorithm.

2 Limitations

Our experiments with the Tasking Event-B CG plug-in revealed a number of limitations of the tool. We had to make a number of modifications to the tool to be able to generate the desired code. Here we outline some of the limitations and issues that we have identified:

- **Program Structure**: One of the most important limitations of the current CG plug-in is its restricted facilities for defining the program structure. The tasking body (i.e. algorithmic structure of the model) does not allow introduction of nested constructs like nested branches.

[1]PRiME: Power-efficient, Reliable, Many-core Embedded systems , www.prime-project.org

- **Dependencies**: The CG tool is highly dependent on other Rodin plug-ins, e.g. Theory Plug-in for translation rules. While it is a good practice to use other available plug-ins for common functionalities, it makes it difficult to maintain the tool.
- **Modularisation**: While the CG tool supports modularisation based on a fixed number of concurrent tasks, it does not support modularisation based on procedural abstraction.
- **Concrete Program Structure**: The current tool allows the introduction of the program structure at the concrete level only. This relies, more than anything else, on the intuition of the developer and their understanding of the algorithm.

3 A New Code Generation Tool

Our experience with the Tasking Event-B CG tool shows that there is a need for designing and implementing improvements to address the aforementioned limitations. The tool should be self-sufficient in terms of its basic operations as much as possible to avoid extension and maintenance complications in later stages. We have started developing a CG tool with having the following features in mind:

- **Algorithmic Refinement**: The new CG tool will provide facilities for introduction of program structure from the abstract level and stepwise refinement of the structure towards a concrete level based on our previous work [1].
- **Verifiable Code**: In addition to executable code, the tool will generate code contracts (assertions) for code-level verification [2].
- **Modular Programming**: The tool will enable the user to define independent procedures (methods/functions) and generate modular code based on procedural abstraction.

References

[1] Mohammadsadegh Dalvandi, Michael Butler, and Abdolbaghi Rezazadeh. Derivation of algorithmic control structures in Event-B refinement. *Science of Computer Programming*, 148(Supplement C):49 – 65, 2017. Special issue on Automated Verification of Critical Systems (AVoCS 2015).

[2] Mohammadsadegh Dalvandi, Michael J. Butler, and Abdolbaghi Rezazadeh. Transforming Event-B models to Dafny contracts. *ECEASST*, 72, 2015.

[3] Andrew Edmunds and Michael Butler. Tasking Event-B: An extension to Event-B for generating concurrent code. Event Dates: 2nd April 2011, February 2011.

[4] Asieh Salehi Fathabadi, Michael J. Butler, Sheng Yang, Luis Alfonso Maeda-Nunez, James Bantock, Bashir M. Al-Hashimi, and Geoff V. Merrett. A model-based framework for software portability and verification in embedded power management systems. *Journal of Systems Architecture*, 82:12 – 23, 2018.