Jakarta: a toolset for reasoning about JavaCard

G. Barthe G. Dufay M. Huisman S. Melo de Sousa

JavaCard is a dialect of Java that enables Java technology to run on new generation smart cards and other devices with limited memory. As JavaCard is becoming increasingly popular, there has been a strong interest, both from academics and industrials, to reason formally about the JavaCard platform.

We report on preliminary results with Jakarta, a toolset for specifying and reasoning about the JavaCard platform. The main ingredients of the toolset are (1) the Jakarta Specification Language (JSL), a front-end for producing highly readable executable specifications; (2) the Jakarta Transformation Kit (JTK), a program to manipulate and transform JSL specifications; (3) the Jakarta Prover Interface (JPI), a compiler that translates JSL specifications into proof assistants; (4) the Jakarta Automation Kit (JAK), a toolset to support reasoning about executable specifications within proof assistants.

The main goal of Jakarta is to provide *support for abstractions*. Through JTK, Jakarta offers interactive support to define a function $f' : \sigma' \to \sigma'$ from a previously defined function $f : \sigma \to \sigma$. The process, which operates at the level of rewrite rules, can be used if σ' is less specific than σ, in which case the process requires an abstraction function $\alpha : \sigma \to \sigma'$. In the context of reasoning about the JavaCard platform, abstraction is useful to construct *e.g.* the BCV from the JCVM.

To improve the support for abstractions, the Jakarta toolset has been designed with the following objectives.

- *Clarity of specifications*: Jakarta specifications are written in JSL, a front-end for producing executable specifications. The JSL language is a first-order, polymorphically typed system which features polymorphic datatype declarations and function definitions by conditional rewrite rules. The formalism is reasonably neutral and leads to specifications that are easy to read, extend and manipulate.

- *Tool independence*: the JSL specification language is minimal and JSL specifications are thus easy to embed into other formalisms. We have developed a compiler JPI that translates JSL specifications into an intermediate representation language JIR, and then from JIR to proof assistants, prototyping environments and programming languages. Due to the limited format of JSL rewrite rules, JSL specifications are close to their compiled counterpart, which ensures that what you specify (in JSL) is what you reason about (*e.g.* in Coq).

- *Proof automation*: the limited format of JSL makes it possible to develop customised tools to reason about JSL specifications. For example, JAK generates for every JSL function f an inversion principle, which is based on an analysis of JSL rewrite rules. Given a predicate ϕ, relating f's input to its output, an application of the inversion principle reduces the task of proving $\forall \vec{x} : \vec{\sigma}.\ \phi\ (\vec{x},\ f\ \vec{x})$ to several simpler subtasks. This has shown particularly useful in proving the correctness of the BCV.

A further pay-off of the independence of proof assistants is that JSL supports *partial functions*, such as the head of a list. While such partial functions cannot be compiled into the specification language of proof assistants, they can be transformed automatically via JTK into a specification based on total functions, which can be translated to proof assistants.