Program certification with computational effects

J.-G. Dumas[*], D. Duval[*], **B. Ekici**[*], D. Pous[†]
[*]LJK, Université de Grenoble, France
[†]LIP, ENS Lyon, France
`{Jean-Guillaume.Dumas,Dominique.Duval,Burak.Ekici}@imag.fr`
`Damien.Pous@ens-lyon.fr`

8 octobre 2014

Dynamic evaluation is a paradigm in computer algebra which was introduced for computing with algebraic numbers. In linear algebra, for instance, dynamic evaluation can be used to apply programs which have been written for matrices with coefficients modulo some prime number to matrices with coefficients modulo some composite number. A way to implement dynamic evaluation in modern computing languages is to use the *exceptions* mechanism provided by the language. In this paper, we pesent a proof system for exceptions which involves both raising and handling, by extending Moggi's approach based on monads. Moreover, the core part of this proof system is dual to a proof system for the *state effect* in imperative languages, which relies on the categorical notion of comonad [Dumas :12 :duality]. Both proof systems are implemented in the Coq proof assistant, and they are combined in order to deal with both effects at the same time.

The *decorated logic* provides a rigorous formalism for proving properties of programs involving computational effects. To start with, let us describe the main features of the *decorated logic for exceptions.* Its syntax is given as follows, where T is any exception name.

$$\begin{array}{llcl}
\text{Types :} & t & ::= & A \mid B \mid \dots \mid t+t \mid \mathbb{0} \mid V_T \\
\text{Terms :} & f & ::= & id \mid f\circ f \mid [f|f] \mid inl \mid inr \mid [\,] \mid \mathtt{tag}_T \mid \mathtt{untag}_T \\
\text{Decorations :} & (d) & ::= & (0) \mid (1) \mid (2) \\
\text{Equations :} & e & ::= & f \equiv f \mid f \sim f
\end{array}$$

Here, $\mathbb{0}$ is the empty type while V_T represents the set of values which can be used as arguments for the exceptions with name T. Terms represent functions ; they are closed under composition and "copairs" (or case distinction), *inl* and *inr* represent the canonical inclusions into a coproduct (or disjoint union). The basic functions for dealing with exceptions are $\mathtt{tag}_T\colon V_T \to \mathbb{0}$ and $\mathtt{untag}_T\colon \mathbb{0} \to V_T$. A fundamental feature of the mechanism of exceptions is the distinction between *ordinary* (or *non-exceptional*) values and *exceptions.* While $\mathtt{tag}_T$ encapsulates its argument (which is an ordinary value) into an exception, $\mathtt{untag}_T$ is applied to an exception for recovering this argument. The usual `throw` and `try/catch` constructions are built from the more basic $\mathtt{tag}_T$ and $\mathtt{untag}_T$ operations [Dumas :14a :coqexc]. We use *decorations* on terms for expressing how they interact with the exceptions. If a term is *pure*, which means that it has nothing to do with exceptions, then it has decoration (0) ; in particular, $id^{(0)}$, $inl^{(0)}$ and $inr^{(0)}$ are pure. We decorate *throwers* with (1) and *catchers* with (2) ; clearly $\mathtt{tag}_T^{(1)}$ is a thrower while $\mathtt{untag}_T^{(2)}$ is a catcher. A thrower may throw exceptions and must propagate any given exception, while a catcher may recover from exceptions. Using decorations provides a new schema where term signatures are constructed without any occurrence of a "type of exceptions". Thus, signatures are kept close to the syntax. In addition, decorating terms gives us the flexibility to cope with more than one interpretation of the set of exceptions. This means that with such an approach, any proof in this decorated logic is valid for different implementations of the exceptions. Besides, we have two different kinds of equality between terms : two terms are *weakly equal* if they have the same behavior on ordinary values but may show differences on exceptions, and they are *strongly equal* if they have the same behavior on both ordinary values and exceptions. We respectively use $\sim$ and $\equiv$ symbols to denote weak and strong equalities.

This syntax is enriched with a set of *rules* that are decorated versions of the rules for *equational logic*. The *equivalence* rules ensure that both weak and strong equalities are equivalence relations. The *hierarchy* rules allow to consider any pure term as a thrower, any thrower as a catcher, and any weak equality as a strong one. The "copair" construction $[f, g]$ cannot be used when both f and g are catchers, since this would lead to a conflict when the argument is an exception. But $[f, g]$ can be used when only g is a catcher, it is the catcher $[f, g]^{(2)}$ which is characterized by the equations $[f, g] \circ \mathit{inl} \sim f$ and $[f, g] \circ \mathit{inr} \equiv g$. This means that exceptional arguments are treated by $[f, g]$ as they would be by g. The *substitution* rule for weak equations $f_1^{(2)} \sim f_2^{(2)} \implies f_1 \circ g \sim f_2 \circ g$ is valid *only* when the substituted term g is *pure*. The behaviour of the $\mathtt{untag}_T$ functions is given by the rules $\mathtt{untag}_T \circ \mathtt{tag}_T \sim id_T$ and $\mathtt{untag}_T \circ \mathtt{tag}_R \sim [\,]_R \circ \mathtt{tag}_T$ for all exception names $T \neq R$ (where $[\,]_R : \mathbb{0} \to R$ is the canonical embedding).

Such a formal system enables us to prove properties of programs involving exceptions. The decorated logic for states and the decorated logic for exceptions, which are mutually dual, are implemented in Coq [Dumas :14a :coqexc]. For instance, we have used these logics for proving the primitive properties of the state effect proposed in [Plotkin :02] and the dual properties of exceptions. To cope with programs including both states and exceptions at the same time, we have composed these Coq implementations, by merging the syntax and the rules. We have also translated the basic imperative programming language IMP in our library, as well as the language IMP_EXC made of IMP extended with exceptions. We have used this implementation to prove some properties of IMP and IMP_EXC programs. For instance, we have checked some simple properties of programs calculating the rank of a (2x2) matrix modulo a composite number using dynamic evaluation [Dumas :14a :coqexc].

We would like to be able to prove more general properties of algorithms for linear algebra using dynamic evaluation implemented through exceptions. For this purpose, we plan to implement Hoare logic for IMP_EXC in decorated terms. We also plan to study other effects (partiality, IO, non-determinism, ...) and to compose them in a systematic way.

Références

[Dumas :14a :coqexc] J.-G. Dumas, D. Duval, B. Ekici, and J.-C. Reynaud. Certified proofs in programs involving exceptions. CICM'14, Coimbra, Portugal, 2014.

[Dumas :14b :coqsts] J.-G. Dumas, D. Duval, B. Ekici, and D. Pous. Formal verification in Coq of program properties involving the global state effect. JFLA, 2014.

[Dumas :12 :duality] J.-G. Dumas, D. Duval, L. Fousse and J.-C. Reynaud. A duality between exceptions and states. Journal of Mathematical Structures in Computer Science 22, p. 719-722(2012).

[Plotkin :02] G.-D. Plotkin, J. Power. Notions of Computation Determine Monads. FoSSaCS 2002. Springer-Verlag Lecture Notes in Computer Science 2303, p.342-356.