Reasoning About CBV Functional Programs in Isabelle/HOL*

John Longley Randy Pollack
Edinburgh University, U.K.

We are developing a program logic for reasoning about Standard ML (SML) programs. Our work is implemented in Isabelle/HOL so we can experiment with machine checked proof. We have started with the functional core language, and plan to add exceptions, then references and I/O to the language.

While the full SML language has a formal definition by a big-step operational semantics, this does not support convenient reasoning about programs. We are developing a high level program logic that is useful for reasoning about features such as recursive functions and datatypes. The logic makes use of purely operational concepts, such as observational equivalence of program fragments. Thus the logic will be intelligible to programmers with no knowledge of the semantic underpinnings. On the other hand, the logic is justified by denotational models. This is achieved by using models that enjoy a tight fit with the programming language via the concept of *logical full abstraction* developed in [LP97]. See the last section of [Lon03] for an overview of the approach.

In contrast to some program logics [Stä03, HT95] we choose to use higher order logic (HOL) as a meta language in order to have a rich language for specification of object language programs. For example, we consider a program for insertion sort whose specification, involving being ordered and being a permutation, is expressed using HOL inductively defined relations. Such inductively defined specifications are more natural, easier to write, understand and reason about than equational specifications involving other program fragments.

Isabelle/HOL provides a ready made proof environment, with good proof automation and flexible user interaction, in which to experiment with our program logic. However there are serious weaknesses in our use of Isabelle/HOL as a meta theory, and the work reported here should be seen as only experimental preparation to building a dedicated SML proof tool.

Some aspects of the encoding There is a typeclass, `SML`, to contain programming language types, and a subclass for SML equality types. Types in other typeclasses retain their purely logical meaning in HOL. Logical equality in typeclass `SML` is intended to represent observational equivalence of programs: indistinguishability in any context.

We use a higher order syntax embedding of the programming language into Isabelle/HOL: an ML function type constructor, `->`, is distinguished from the logical HOL function type, `=>`. As usual, there are constants

```
lam :: "('a => 'b) => 'a->'b"            (binder "fn " 30)
APP :: "['a->'b,'a] => 'b"               (infixl "$" 55)
```

relating object and meta function types. (`$` is infix application for `SML` functions.) In these declarations, `'a` and `'b` are inferred to be in typeclass `SML`.

Isabelle/HOL typechecking over typeclass `SML` serves to typecheck programs. This is very convenient for both developing and using our tool, but not quite faithful to the SML definition, as HOL polymorphism is not the same as SML polymorphism.

We have polymorphic constants `Fix` and `bot`, which are definable in official SML. There is no judgement for being a value (since values may be observationally equivalent to non values), but a judgement for *definedness*, `dfd`, defined in terms of `bot`. Thus the CBV beta rule is given by an axiom

*Version of February 13, 2004 (1024)

```
beta_rule: "dfd x ==> (lam F) $ x = F x"
```

`Fix` is axiomatised to be the least fixpoint

```
dfd (C (Fix $ F)) ==> (All n. dfd (C (mlIter n bot F)))
```

where `C::('b->'c)=>'a` is any context, and `mlIter` is (roughly) iteration over HOL naturals.

Other built-in types are unit, booleans, sums and products. They each come with constructors, a case constant with computation rules, and a standard induction principle over `dfd` elements. We intend to introduce other datatypes uniformly, proving the appropriate rules from an initial algebra property (equivalently final coalgebra property). However, for experiment, we have introduced ML natural numbers and ML lists by axioms, and developed some programs over these types. For example, we proved a fast Fibonacci program equal to a naive Fibonacci program which is easily seen to satisfy the expected recursion equations. We have proved properties of list folding, e.g. Theorems 16, 17 and 18 from Paulson's textbook [Pau91]. We have specified polymorphic sorting abstractly, and proved insertion sort is correct.

Positive datatypes in general Define *observational limit* by

```
obsLim (y::'a) (x::nat=>'a) ==
      ALL (C::'a => UNIT). dfd (C y) = (EX (n::nat). dfd (C (x n)))
```

In our intended model, not every chain has an observational limit. We can prove that `Fix $ F` is an observational limit of finite iterations of `F`.

To reason in general about positive recursive datatypes we need one more axiom

```
(ALL (C::'a=>UNIT). C x = C y)  ==>  x = y
```

which can be seen as an extensionality axiom, saying that Leibniz equality (i.e. observational equivalence) implies extensional equality. From this we prove that observational limits are unique.

Now datatypes such as the SML type of *sequences*

```
datatype 'a seq = scons of 'a * (unit -> 'a seq)
```

can be defined and reasoned about. `seq` is characterised by a functor `T 'a = 'a * (unit -> 'a)`. We have proved the equivalence of several well known principles for this datatype:

- Its constructor, `FOLD = scons`, is an initial algebra.
- Its eliminator, `UNFOLD`, is a final coalgebra.
- `Fix $ (fn F. FOLD o (T $ F) o SUNFOLD) = Identity"`.
- Sequence extensionality: `(ALL n. nth $ n $ s = nth $ n $ t) ==> s = t`.
- Two streams are equal if they are related by some pre-equality relation [Pit94].

Taking any of these as axiomatic, we have proved that a program that removes all zeros from a sequence of naturals, returns a sequence containing no zeros.

References

[HT95] Steve Hill and Simon Thompson. Miranda in Isabelle. In Lawrence C. Paulson, editor, *Preceedings of the first Isabelle Users Workshop*, number 397 in University of Cambridge Computer Laboratory Technical Reports Series, pages 122–135, September 1995.

[Lon03] John Longley. Universal types and what they are good for. In GQ Zhang, J. Lawson, Y.M. Liu, and M.K. Luo, editors, *Domain Theory, Logic and Computation, Proc. 2nd Inter. Symp. on Domain Theory.* Kluwer, 2003.

[LP97] J. Longley and G. Plotkin. Logical full abstraction and PCF. In J. Ginzburg, editor, *Tbilisi Symposium on Language, Logic and Computation.* SiLLI/CSLI, 1997.

[Pau91] L. Paulson. *ML for the Working Programmer.* Cambridge Univ. Press, 1991.

[Pit94] A. M. Pitts. A co-induction principle for recursively defined domains. *Theoretical Computer Science*, 124:195–219, 1994.

[Stä03] R. F. Stärk. Axioms for strict and lazy functional programs. *Annals of Pure and Applied Logic*, 2003. To appear.