To arrive where we started: experience of mechanizing binding

Tom Ridge
University of Cambridge
Thomas.Ridge@cl.cam.ac.uk http://www.cl.cam.ac.uk/~tjr22

We discuss experience gained from several case studies involving binding. Our main goal is a representation and libraries which enable fast mechanization of metatheory. Our secondary goal, which is not universally shared, is that mechanized proofs should be as close as possible to informal proofs.

The mechanized case studies are: type soundness for MiniML; type soundness for TAPL fragments; a verified theorem prover for first-order logic; an investigation into transferring results between binding representations using isomorphisms; Craig's interpolation theorem; various POPLmark solutions; generalised term models; operational reasoning for Caml programs. The verified prover, and the operational reasoning for Caml, are both published research papers. Most of the material can be found at the author's homepage.

The case studies illustrate the move from naive approaches (variables as strings, manual alpha conversion), to simple approaches (de Bruijn), to more sophisticated approaches (variants on locally nameless and Gordon's approach), to generality (a universal datatype for terms with binding, with libraries of supporting lemmas). We believe our conclusion is surprising.

We first make the point that most theorem provers provide the basic support for mechanizing metatheory. The experience of the author, together with Gilles Peskine and Scott Owens, on POPLmark and TAPL proofs in Isabelle, HOL and Coq, is that differences in foundations (e.g. constructive v. classical, simple types v. dependent types) do not make much difference to mechanization. However, strong automation can be crucial. Especially, HOL's first-order automation is extremely powerful, which avoids the need, common in other systems, to write custom tactics.

A naive variables-as-strings, capturing-substitution approach can work well in practice. Close attention to the language used in definitions can reveal a lot about the properties that are required (and not required) of a representation during proof, particularly whether a naive approach is workable. For example, since ML programs never substitute open terms under binders, no alpha-renaming is required, moreover type schemes are compared up to subsumption, never up to equality. We therefore mechanized the syntax and semantics of MiniML using a naive approach at both type and term level. We applied the same technique to mechanize metatheory for substantial fragments of TAPL, and operational reasoning for Caml programs. Peskine and Owens have formalized operational semantics and metatheory for OCaml in this style. However, sometimes one wants more from the representation.

Our mechanization of a verified theorem prover used de Bruijn notation. Whilst the representation is fixed, the language is relatively flexible: $[s/x](\mathit{Lam}\ t) = \mathit{Lam}([\uparrow s/\uparrow x]t)$, but from there one can choose to generalise in a number of ways (tradition involves a `lift` primitive which, whilst sufficient, is not necessary, and is arguably over complicated). Regardless of this choice, de Bruijn is an extremely effective representation. We mechanised POPLmark 1a using de Bruijn, and note that the most successful POPLmark solutions also use de Bruijn. On the other hand, informal proofs use a language of named terms.

What is the cost of moving to a named representation? Isomorphisms provide an easy way to lift results from de Bruijn to named representations. Since the isomorphisms are (almost) trivial, the overhead of named *terms* is (almost) nothing. For *derivations*, going under a binder in de Bruijn always creates "room at the bottom" for a new free variable. The isomorphic image of this operation in named representations is extremely unnatural, and one prefers to choose fresh variables. In short, for terms there is nothing to choose between representations. For derivations there is a clear difference.

We failed to heed this evidence, and pursued a named approach with alpha equivalence as equality, broadly similar to Gordon's approach, and the recently popular locally nameless (but with named abstraction). The underlying model is de Bruijn, but the language is that of named terms. We mechanized standard results from proof theory, including Craig's interpolation theorem, along these lines. For the POPLmark challenge, we mechanized 1a and 2a using this approach. However, each mechanization follows the same outline, but with different term datatypes. Proving the same properties over and over for each mechanization is tiresome.

To capture commonalities, we abstracted from the case studies by declaring a universal type of terms-with-binding, together with a library of useful theorems. For example, POPLmark term $\bigwedge X <: S.\ T$ is represented as `Node Forall [S,Bind X T]`. The library was developed in HOL, and ported to Coq by Robert Atkey. The library incorporates variables (with an associated type, e.g. term or type variables for POPLmark), swapping, substitutions (from variables to variables, of a term for a variable), and so on. Using these general library lemmas in a particular mechanization is easy. One defines the particular terms one is interested in, using the universal datatype, then rewrites library lemmas, using these definitions, to obtain the results in the particular setting. To exercise the library, we rewrote our POPLmark 1a and 2a solutions.

However, the gain in treating alpha as equality is mitigated by the difficulty of performing case analysis over terms: in a naive approach $\mathit{Lam}\ x\ t = \mathit{Lam}\ x'\ t'$ iff $(x,t) = (x',t')$, however, if alpha equivalence is equality, we know only that $x' \notin \mathit{fv}\ \mathit{Lam}\ x\ t \wedge x \notin \mathit{fv}\ \mathit{Lam}\ x'\ t' \wedge \mathit{swap}\ x\ x'\ t = t'$. Moreover, the alpha varying properties of terms are also shared by derivations, so that it seems sensible to generalise the library to derivations with binding. However, derivations are less regular than terms, and capturing this commonality appears difficult without using a naive approach.

Therefore we conclude that a naive approach, with names as strings and explicit handling of alpha equality, is preferable. By building a library of lemmas for a general datatype of terms-and-derivations-with-binding using this representation we hope to mitigate the work this representation requires, but capture binding uniformly at the term and derivation levels. To demonstrate this approach, we are currently mechanizing POPLmark in this style.

We shall not cease from exploration
And the end of all our exploring
Will be to arrive where we started
And know the place for the first time.