Towards a Coq Library for Programming Languages Metatheory with Concrete Names

Aaron Stump
Washington University in St. Louis

The goal of this work in progress is to build a library in Coq to support the development of programming languages (PL) metatheories. The library accounts for commonalities in PL metatheories in two ways. First, we make use of the Coq module system to define a datatype of abstract terms, parameterized by a signature module and a names module. The names module must provide a countably infinite set of names with decidable equality. The signature module must provide a Coq Set of operators, together with three arity functions telling, for each operator, 1) how many variables are bound by uses of the operator, 2) how many subterms such uses have which are not in the scope of those variable ("non-governed" subterms), and 3) how many subterms such uses have which are in the scope of those variables ("governed" subterms). Abstract terms are then either uses of variables or compound expressions built from an operator, an annotation of some operator-determined type, a list of non-bound subterms, a list of names bound by the expression, and a list of bound subterms (annotations will also be supported). This notion of abstract terms covers a large range of PL syntaxes. Using Coq's dependent types, the constructor for compound terms insists that the lists of non-governed subterms, bound variables, and governed subterms are of the lengths specified by the signature. The library may then define syntactic notions (indeed, the following currently are defined) like the set of free variables of a term, the alpha-canonization from d of a term (where consecutive bound variables starting with the d'th are used on every path from the root of the term), substitution of one term for a variable in another, etc. once and for all on abstract terms. Appropriate lemmas can also be proved once and for all about those notions. Particular PL developments may then import those lemmas for their syntaxes.

The second way in which the library under development accounts for commonalities in metatheory is by providing an abstract means for defining contextual functions on terms. These are functions defined by recursion on the structure of terms, making use of a context for free variables. The functions mentioned above for the first part of the library are all of this kind. The library enables definition of such functions as *contextual term interpretations* (CTIs). Useful lemmas may be proved abstractly for any CTI, including permutation, contraction, and weakening of the context, and a property equating the interpretation of certain alpha-canonizations of a term with the interpretation of the original term. Any function which can be defined as a contextual term interpretation inherits these lemmas for free. The essential idea of CTI is the following. A domain of interpretation A is given. Contexts are ordered lists of pairs of names and objects from A. Variables are interpreted by looking up their first occurrence in the context. A user-specified function interprets all variables not found in the context. Operators are interpreted as functions taking in the interpretations of the "non-bound" subterms, as well as a function from interpretations for the bound variables to interpretations for the "bound" subterms, and returning an interpretation for the application of the operator. Terms are then interpreted homomorphically. The library supports this interface by building up explicit contexts in cooperation with the user-provided interpretations.

The current milestone is to prove type preservation for a call-by-value simply typed lambda calculus with two kinds of λ-abstractions: one where execution does not pass into the body of the abstraction, and one where it does. A small-step evaluation relation and a simple type checker are both defined as CTIs. Since CTIs are called only with interpretations of terms, it turns out that evaluation most naturally gives the resulting term in alpha-canonical form. Since evaluation is defined as a CTI, we can use a substitution lemma proved for arbitrary CTIs in the library. While evaluation and simple type checking are currently implemented, the exact formulation and application of the substitution lemma are (as of 8/31/2006) still under development. It is anticipated that a first release of the above described CTI library will be made available by the date of the workshop (9/21/2006) at `http://cl.cse.wustl.edu/cti-lib/`.

Acknowledgements: The author gratefully acknowledges undergraduates Aayush Munjal and Michael Zeller for implementation of weakening and contraction, respectively; and the NSF under contract CCF-0448275.