(Extended Abstract - Submitted to the 1st Informal ACM SIGPLAN Workshop on Mechanizing Metatheory, WMM 2006)

Mechanized Reasoning for Binding Constructs in Typed Assembly Language Using Coq

Nadeem Abdul Hamid

Berry College, Mount Berry, GA 30149-5014, USA

nhamid@berry.edu

1. Problem and Motivation

Mechanized reasoning about programming languages and type systems is becoming increasingly important for the development of certified code frameworks. For instance, in order to realize the safety and security potential of proof-carrying code (PCC) [3] the development of formal, machine-checkable proofs is a necessity. Much of the difficulty and research surrounding PCC involves the generation of large, complex proofs in a user-friendly, automated way. Several approaches in this respect [2, 1] rely on encoding a typed assembly language (TAL) and mechanically proving its safety properties.

This work describes some aspects of the author's experience with the mechanical encoding of TAL for several prototype systems developed in the Yale FLINT group's PCC project. In particular, an approach to encoding TAL binding constructs is presented in which a first-order representation using de Bruijn indices is used for bound type variables, while free variables are represented by meta-level variables. This method allows for encoding and reasoning about the semantics of TAL without the need for maintaining explicit variable contexts, in a logic that otherwise does not support the use of higher order abstract syntax (HOAS).

2. Related Work

Current approaches to formalizing binders fall into one of a few categories: concrete, first-order representations; specialized logics that include a notion of "freshness"; or the use of higher-order abstract syntax (HOAS) [4]. One of the most popular concrete methods of representing binders is the use of de Bruijn's nameless dummies. de Bruijn indices are very direct and simple, however they complicate the entire development of proofs because the statements of theorems require extra clauses involving shifting of terms and contexts. The approach presented here avoids much of this problem.

A much different approach to handling issues of binding is that of HOAS, in which object language variables are represented by variables of the metalanguage or logic. While this approach works very elegantly with frameworks such as Twelf, it has obstacles in the context of a theory of inductive definitions, such as CiC.

Currently, the most similar approach to the one presented in this work is the "locally nameless" approach [5]. In this representation, there are two classes of variables: de Bruijn indices for bound variables, and an infinite type of names for representing free variables. Our work may be viewed as extending and integrating this first-order representation with HOAS in the context of CiC/Coq.

3. Results and Contributions

Over the course of several years, a progression of TAL variants have been formalized for various prototype PCC systems, the most ambitious being a variant supporting region-based memory management, with a type system based on the capability calculus [6], supporting polymorphism over several different kinds of constructors. Here, I present a simple variant of TAL to illustrate the approach used to encode binding constructs in these various flavors.

An approach is presented using a standard, first-order de Bruijn encoding of variables in closed terms. However, whenever such terms are opened up, or the bound variables are entered into a judgment context, the resulting free variables are then represented by variables of the metalogic. This avoids cluttering up almost all of the theorem statements with any explicit reasoning about variable indices or contexts. We do still need to define the substitution function explicitly, but in fact never have to reason about it for the soundness proofs (i.e. there is no need to prove any substitution, weakening, exchange lemmas or the like, commonly found in first-order developments).

An additional interesting feature of the de Bruijn representation here is the use of dependent types to encode the number of free variables in a term, providing a measure of partial correctness for definitions like substitution. While the use of such an encoding may be a common exercise to experienced users of proof assistants such as Coq and LEGO, we were not able to find any published description of such an encoding when we embarked on our own developments.

References

[1] K. Crary. Toward a foundational typed assembly language. In *Proceedings 30th ACM Symposium on Principles of Programming Languages*, pages 198–211. ACM Press, Jan. 2003.

[2] N. A. Hamid, Z. Shao, V. Trifonov, S. Monnier, and Z. Ni. A syntactic approach to foundational proof carrying-code. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science (LICS'02)*, pages 89–100. IEEE Computer Society, July 2002.

[3] G. C. Necula. Proof-carrying code. In *Proceedings 24th ACM Symposium on Principles of Programming Languages*, pages 106–119. ACM Press, Jan. 1997.

[4] F. Pfenning and C. Elliot. Higher-order abstract syntax. In *Proceedings ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation*, pages 199–208. ACM Press, 1988.

[5] R. Pollack. Reasoning about languages with binding. (Talk), 2005.

[6] D. Walker, K. Crary, and G. Morrisett. Typed memory management via static capabilities. *ACM Trans. Prog. Lang. Syst.*, 22(4):701–771, 2000.