Steven E. Ganz

Encapsulation of State with Monad Transformers

We relate the type-and-effect system of Tofte and Talpin and monadic systems while handling the permissive notion of encapsulation and various other features of their system. In particular, we translate from an object language that allows interaction between computations at different regions to a language based on the state monad transformer. The languages' syntactic classes are indexed respectively by sequences of region indicators and by the depth within evaluation constructs. The fundamental difference between a monadic programming language and a more traditional programming language with encapsulated effects is that the monadic language uses relative addressing of regions while the traditional language uses absolute addressing of regions. This relative addressing is inherent in the `return` and `run` constructs and there's no need for region identifiers to indicate a region for an effectful operation. Even memory addresses are free of region names, so that the representation is truly modular. This approach carries a restriction against upward (inward) references that gives a sequence-of-trees structure to the regions of a program, and thus to the store of such a program with state effects. We also define effect-annotated monad transformers and make use of them in presenting a type system that uses tree-structured store types and is equally free of region identifiers. We show this type system to be sound with respect to the operational behavior. Semantics-preserving translations from a direct-style effect system are provided. The reduction semantics and typing judgments are also indexed, giving them a precise characterization in terms of the syntactic classes. Under our restriction against upward references, the tree structure of monadic programs carries information not available to direct-style programs with a linear store — namely that certain regions aren't accessible. This difference cannot be made up by the translation, so the earlier one translates to monadic style, the more efficient the resulting program. We demonstrate support for various extensions, including recursion, polymorphism, eager deallocation of regions that departs from the stack regime, and subtyping to allow live pointers to be viewed as dangling ones. Finally, we briefly consider region-polymorphic functions and sketch an incremental implementation supporting computational reflection.

Daniel P. Friedman, Ph.D.

Christopher T. Haynes, Ph.D.

David Charles McCarty, Ph.D.

Paul W. Purdom, Jr., Ph.D.