Combining Proofs and Programs*

Stephanie Weirich[1]

1 **Department of Computer and Information Science, University of Pennsylvania Philadelphia, USA**
sweirich@cis.upenn.edu

Abstract

Programming languages based on dependent type theory promise two great advances: *flexibility* and *security*. With the type-level computation afforded by dependent types, algorithms can be more generic, as the type system can express flexible interfaces via programming. Likewise, type-level computation can also express data structure invariants, so that programs can be proved correct through type checking. Furthermore, despite these extensions, programmers already know everything. Via the Curry-Howard isomorphism, the language of type-level computation and the verification logic is the programming language itself.

There are two current approaches to the design of dependently-typed languages: Coq, Epigram, Agda, which grew out of the logics of proof assistants, require that all expressions terminate. These languages provide decidable type checking and strong correctness guarantees. In contrast, functional programming languages, like Haskell and Ωmega, have adapted the features dependent type theories, but retain a strict division between types and programs. These languages trade termination obligations for more limited correctness assurances.

In this talk, I present a work-in-progress overview of the Trellys project. Trellys is new core language, designed to provide a smooth path from functional programming to dependently-typed programming. Unlike traditional dependent type theories and functional languages, Trellys allows programmers to work with total and partial functions uniformly. The language itself is composed of two fragments that share a common syntax and overlapping semantics: a simple logical language that guarantees total correctness and an expressive call-by-value programming language that guarantees types safety but not termination.

Importantly, these two fragments interact. The logical fragment may soundly reason about effectful, partial functions. Program values may be used as evidence by the logic. We call this principle *freedom of speech*: whereas proofs themselves must terminate, they must be allowed to reason about any function a programmer might write. To retain consistency, the Trellys type system keeps track of where potentially non-terminating computations may appear, so that it can prevent them from being used as proofs.

1998 ACM Subject Classification F.3.3 Studies of Program Constructs (Type structure)

Keywords and phrases Dependent types, functional programming

Digital Object Identifier 10.4230/LIPIcs.RTA.2011.9

Category Invited Talk

Acknowledgements Trellys is a collaborative project between the University of Pennsylvania, the University of Iowa and Portland State University. This talk is based on joint work with Aaron Stump, Tim Sheard, Chris Casinghino, Vilhelm Sjöberg, Brent Yorgey, Harley D. Eades III, Garrin Kimmel, and Nathan Collins.

* This work was supported by the National Science Foundation (award 0910510)

22nd International Conference on Rewriting Techniques and Applications.
Editor: M. Schmidt-Schauß; pp. 9–9

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany