Proposal for ESSLI course: Propositions as Types

Philip Wadler
University of Edinburgh

15 March 2015

a. Personal information

Philip Wadler
University of Edinburgh
Informatics Forum 5.31
10 Crichton Street
Edinburgh EH8 9AB
UNITED KINGDOM
wadler@inf.ed.ac.uk
http://homepages.inf.ed.ac.uk/wadler/

b. General proposal information

Title: Propositions as Types
Category: Foundational/intermediate

c. Contents information

Abstract

The principle of Propositions as Types describes a fundamental connection between logic and computation which views

propositions as *types*,
proofs as *programs*, and
normalisation of proofs as *evaluation of programs*.

The proposed course is intended to begin at the foundations and introduce students to a few intermediate or advanced topics. It is suitable for students in both logic and computing, presuming no previous knowledge of either, though familiarity with logic and computing will be helpful.

Motivation and description

[Taken from the introduction to Wadler [2015].]

Powerful insights arise from linking two fields of study previously thought separate. Examples include Descartes's coordinates, which links geometry to algebra, Planck's Quantum Theory, which links particles to waves, and Shannon's Information Theory, which links thermodynamics to communication. Such a synthesis is offered by the principle of Propositions as Types, which links logic to computation. At first sight it appears to be a simple coincidence—almost a pun—but it turns out to be remarkably robust, inspiring the design of theorem provers and programming languages, and continuing to influence the forefronts of computing.

Propositions as Types is a notion with many names and many origins. It is closely related to the BHK Interpretation, a view of logic developed by the intuitionists Brouwer, Heyting, and Kolmogorov in the 1930s. It is often referred to as the Curry-Howard Isomorphism, referring to a correspondence observed by Curry in 1958 and refined by Howard in 1969 (though not published until 1980, in a Festschrift dedicated to Curry). Others draw attention to significant contributions from de Bruijn's Automath and Martin-Löf's Type Theory in the 1970s. Many variant names appear in the literature, including Formulae as Types, Curry-Howard-de Bruijn Correspondence, Brouwer's Dictum, and others.

Propositions as Types is a notion with depth. It describes a correspondence between a given logic and a given programming language, for instance, between Gentzen's intuitionistic natural deduction and Church's simply-typed lambda calculus. At the surface, it says that for each proposition in the logic there is a corresponding type in the programming language—and vice versa. Thus we have

propositions as *types*.

But it goes deeper, in that for each proof of a given proposition, there is a program of the corresponding type—and vice versa. Thus we also have

proofs as *programs*.

And it goes deeper still, in that for each way to normalise a proof there is a corresponding way to evaluate a program—and vice versa. Thus we further have

normalisation of proofs as *evaluation of programs*.

Hence, we have not merely a shallow bijection between propositions and types, but a true isomorphism preserving the deep structure of proofs and programs, normalisation and evaluation.

Propositions as Types is a notion with breadth. It applies to a range of logics including propositional, predicate, second-order, intuitionistic, classical, modal, and linear. It underpins the foundations of functional programming, explaining features including functions, products, sums, parametric polymorphism, data abstraction, continuations, linear types, and session types. It has inspired theorem provers and programming languages including Agda, Automath, Coq, Epigram, $F^{\#}$, $F^{\star}$, Haskell, LF, ML, NuPRL, Scala, Singularity, and Trellys. Applications include CompCert, a certified compiler for the C programming language verified in Coq, a computer-checked proof of the four-colour theorem also verified in Coq, parts of the Ensemble distributed system verified in NuPRL, and ten thousand lines of browser plug-ins verified in $F^{\star}$.

Propositions as Types is a notion with mystery. Why should it be the case that intuitionistic natural deduction, as developed by Gentzen in the 1930s, and simply-typed lambda calculus, as developed by Church around the same time for an unrelated purpose, should be discovered forty years later to be essentially identical? And why should it be the case that the same correspondence arises again and again? The logician Girard and the computer scientist Reynolds independently developed the same calculus, now dubbed Girard-Reynolds. The logician Hindley and the computer scientist Milner independently developed the same type system, now dubbed Hindley-Milner. Curry-Howard is a double-barrelled name that ensures the existence of other double-barrelled names. Those of us that design and use programming languages may often feel they are arbitrary, but Propositions as Types assures us some aspects of programming are absolute.

Outline

- Introduction to propositions as types, including the history of the subject. (Based on Wadler [2015].)
- Detailed description of the correspondence for implication/function, conjunction/product, truth/unit, disjunction/sum, and false/zero. (Based on Girard et al. [1989].)
- Application of the correspondence to second order logic and polymorphic lambda calculus. (Based on Wadler [2007].)
- Application of the correspondence to classical logic and sequent calculus. (Based on Wadler [2003].)
- Application of the correspondence to linear logic and session types. (Based on Wadler [2012, 2014].)

Although texts by myself will be main texts of the course, there is much related work cited by those texts that we will also review: Gentzen [1935], Church [1940], Girard [1972], Reynolds [1983], Griffin [1990], Parigot [1992], Curien and Herbelin [2000], Girard [1987], Caires and Pfenning [2010], to name but a few.

Expected level and prerequisites

The course is intended to be suitable for those without previous experience, and so in that sense is foundational, but will touch on further developments in the area and so in that sense is intermediate or even advanced.

References

References

Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, pages 222–236, 2010.

Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68, June 1940.

Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *International Conference on Functional Programming (ICFP)*, pages 233–243, 2000.

Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39(2–3):176–210, 405–431, 1935. Reprinted in Szabo [1969].

Jean-Yves Girard. *Interprétation functionelle et élimination des coupures dans l'arithmétique d'ordre supérieure*. PhD thesis, Université Paris VII, 1972.

Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proof and Types*. Cambridge University Press, 1989.

Timothy Griffin. A formulae-as-types notion of control. In *Principles of Programming Languages (POPL)*, pages 47–58. ACM, January 1990.

Michel Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *Logic programming and automated reasoning*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer-Verlag, 1992.

John C. Reynolds. Types, abstraction and parametric polymorphism. In *IFIP Congress*, pages 513–523. North Holland, 1983.

M. E. Szabo, editor. *The collected papers of Gerhard Gentzen*. North Holland, 1969.

Philip Wadler. Call-by-value is dual to call-by-name. In *International Conference on Functional Programming (ICFP)*, pages 189–201. ACM, 2003.

Philip Wadler. The Girard-Reynolds isomorphism (second edition). *Theoretical Computer Science*, 1(1–3):201–226, 2007.

Philip Wadler. Propositions as sessions. In *International Conference on Functional Programming (ICFP)*, pages 273–286. ACM, 2012.

Philip Wadler. Propositions as sessions. *Journal of Functional Programming*, 24:384–418, 2014.

Philip Wadler. Propositions as types. *Communications of the ACM*, 2015. To appear.

d. Practical information

I know of no relevant immediately preceding meetings and events, or of external sources of funding.