Research Statement

Jason Reed

Research Agenda

Motivations

Mathematics succeeds when it builds bridges: a result connecting one approach (or one entire field of study) to another allows existing ideas to be transported across it, to be reused, reexamined, and applied to new problems. A venerable example is the Curry-Howard correspondence, the observation that type theorists, programming language designers, logicians, and category theorists are often studying the same objects under different names. It tells us in particular that we *ought not*, and *need not* be satisfied with programming languages that appear to us as a jumbled bag of features, for we can look to logic (and type theory, and category theory) for advice as to how to organize the features of our programming language in a more coherent way.

My underlying interest is in finding ways to tell a similarly satisfying story about the foundations of the wide (and growing) variety of nonstandard logics, for example modal logics expressing nuances of necessity, possibility, knowledge, belief, time, and location, or substructural logics capturing resources used linearly, strictly, affinely, or precisely n times, or according to tree-shaped bunched contexts, or in a particular order. I believe that we ought not and need not be satisfied with logics that appear to us as a jumbled bag of propositional connectives which perhaps, by apparent lucky coincidence, happen to satisfy a handful of convenient theorems. Rather we should be able to look to deeper organizing principles to guide us when introducing new logical ideas, and designing new logics.

The payoff is that creating new *languages* suited to the task at hand should become nearly as routine as, say, writing new *programs*, when we have better measures of their mathematical coherence and fitness. Just as type systems afford greater confidence to the working programmer that her partial program, while complex, still *makes sense*, improving our understanding of the principles underlying the design of a logical system itself allows us to more easily describe more complicated logics while remaining confident that we will be able to effectively use and reason about them.

When we can more freely invent new logical ideas, we can then obtain new types for functional programming, new mechanisms in logic programming languages, and new challenges for the design of algorithms for type checking, type inference, unification, and compilation. As far as applications are concerned my research to date has focused on introduction of new logical ideas to *logical frameworks*, tools for representation of formal systems and automated reasoning about them.

Future Directions

What makes a logic well-behaved?
How can the space of 'all possible logics' be characterized?

To be sure these are informal, underspecified questions. Nonetheless we seem to have some significant leads to pursue. The oldest and most basic are the principle of cut-elimination and identity that we find in Gentzen's relating of natural deduction to sequent calculus: to whatever extent logic is about a consequence relation, we at least expect it to be suitably transitive and reflexive, just as categories have composition and identities. One can find in Martin-Löf [ML96] a carefully articulated and comparatively modern statement of a *judgmental* standpoint concerning the role of propositions and propositional connectives, further constraining the design of logics. And finally we have the development in the late 80s and early 90s by Girard, Andreoli, and others, of properties that increasingly seem *sine qua non* criteria for a 'sensible' logic: in short that their propositions can be *polarized* [Gir91, Gir93] and that their proofs can be *focused* [And92].

My aim for a unifying account of substructural logics is by way of translations into simpler, non-substructural logics, that *preserve the focusing structure* of the logic. See [RP09] and below for the groundwork of this approach, but there is much work to be done here investigating other particular logics.

Judgmental modal concepts — such as the necessary and lax truth of Pfenning and Davies [PD01], and the exponential ! of linear logic — also seem to beg an explanation for the common themes of their definitions. Some preliminary ideas about decomposing several known modalities into a pair of connectives of opposite polarity (being the logical equivalent of adjoint functors from category theory) can be found in [Ree09].

What good is understanding logic?
What are its applications?
How can we effectively implement and reason about logical algorithms?

I am interested in type-checking, proof-checking, and proof-synthesis algorithms. A theme underlying all of these is the need for good unification algorithms, and aggressive approximations to undecidable unification problems.

One important role for unification is making logical frameworks more practical and pleasant to use. Type inference is not strictly necessary, as the user could in principle supply every type to every argument, but this is unthinkable in practice. Implementations of unification algorithms themselves are also one of the few places where an imperative style is considered necessary for performance, where trailing and undo mechanisms are then required to compensate for lack of purity, when we wish to pretend as if we had programmed functionally all along. Are there new language features or reasoning patterns to better account for the very limited form of state that unification typically uses?

Unification is also of course at the heart of logic programming. I suspect that many of the challenges posed by the prospect of logic programming with substructural logics might well be solved 'merely' (although it is still no small project) by appropriate unification algorithms over an algebraic equational theory suited to the substructural logic in question.

Completed Work

Substructural Logical Frameworks

The logical framework LF [HHP93] and the Twelf [PS99] implementation built on top of it are capable of advanced representation techniques (such as *higher-order abstract syntax* [PE89]) for encoding formal systems such as programming languages, and supports theorem-checking for inductive theorems represented constructively as logic programs.

My PhD thesis work began with the attempt to find a corresponding methodology of inductive theorem-checking for the *linear* logical framework LLF developed by Cervesato and Pfenning [CP02]. With type-constructors taken from the proofs-as-types interpretation of linear logic [Gir87], LLF supports encodings that manipulate stateful update in a modular and elegant fashion. For instance, the encoding of a programming language with updateable reference cells becomes significantly simpler.

However, the strategies familiar from experience using Twelf for representing the statement of inductive theorems about such systems do not easily extend to the linear case: there are complications with the way that the linearity interacts with dependent types.

The solution [Ree07] was to tease apart linear hypotheses into ordinary variables *as such* on the one hand, and on the other hand, abstract identifiers of linear resources, akin to the role of the abstract identifiers of Kripke worlds in hybrid modal logics [ABM01, BdP03, BdP06]. These identifiers are moreover first-class entities in the language, subject to an algebraic discipline (i.e. that they form a commutative monoid) that reflects the expected structural properties of the linear context, and critically allows the logical framework user to refer explicitly to bundles of resources (in other words, linear contexts) when stating theorems.

Generalizing this work also leads to a much more powerful analysis of the role of structured contexts in substructural logics. In joint work with Pfenning [RP09], we describe a similar construction that isolates the algebraic behavior of the context — in a fashion related to Belnap's *display logic* [Bel82] — and works uniformly as a constructive resource semantics for linear, ordered and bunched logic, simply by varying the algebra involved.

Algorithms for Mechanized Reasoning

Unification, solving equations for unknowns that are syntactic expressions, is a pervasive problem in automated reasoning. Higher-order unification , in which the unknowns can be *functions*, is undecidable, but has subsets that admit practical algorithms [DHK95], which are used frequently in practice.

However, their theory turns out to be surprisingly subtle. The version of unification implemented in Twelf, for instance, is not terminating — despite the sketch of a proof of its termination in the literature [DHKP96]! One rewrite rule in an algorithm that works by progressively simplifying a set of equations my sometimes subtly *un*simplify a remote part of the problem. One could simply remove the offending rewrite, but at the cost of solving only a smaller subset of all higher-order unification problems. I proposed and prove correct a modified rewrite step to replace it [Ree08], yielding a new, terminating constraint simplification algorithm for a dynamic pattern fragment of higher-order unification in a dependent type system.

Proof irrelevance is a property of some components of data structures that some sort of witness — a proof — of a property is required, but *which* one it is does not matter. The concept is connected to certain modal logics studied by Pfenning [Pfe01], and Awodey and Bauer [AB01]. The ability to mark certain parts of large certificates (for instance in a proof-carrying code [NL97] setting) as proof irrelevant enables optimizations that elide those parts, asking the recipient of the certificate to reconstruct them — perhaps proofs of easily (but tediously) re-provable decidable properties. Proof irrelevance guarantees that the possibility that the recipient reconstructs a *different* proof will not affect the validity of the rest of the proof.

I investigated the extension of the type theory LF [Ree03] and the higher-order unification algorithm used on it to include proof-irrelevance features [Ree02a, Ree02b]. My presentation *en passant* fixed a minor bug in the original work of Pfenning [Pfe01], in which the Validity lemma did not actually hold, due to some confusion about the meaning of proof irrelevance at the kind level (as opposed to the type level). The modifications to the type system and associated algorithms were comparatively simple, a testament to the modularity of the underlying ideas.

Classical Logic vs. Constructive Logic

It is well-known [Gri90, Par92, OS97, Wad03] that classical logic can be interpreted as a programming language with control operators such as call/cc. The embedding of intuitionistic logic into classical modal logic means that intuitionistic logic itself — which ordinarily corresponds to a plain λ-calculus — can be regarded instead as a programming language that *does* have control operators, but which are subject to essentially *modal* restriction to avoid provability of classical tautologies such as double-negation elimination.

Frank Pfenning and I described [RP07] this intuitionistically compatible form of call/cc in a labelled deduction style [Gab90] and gave a novel constructive proof for its soundness, whose algorithmic content is a non-deterministic translation of programs that eliminates uses of intuitionistic call/cc and is compatible with dependent types. The proof has been formally verified on the propositional fragment in the Twelf meta-logical framework.

The fact that the *intuitionistic* call/cc is compatible with dependent types is significant, because we have thereby avoided Herbelin's counterexample [Her05] which demonstrates that unbridled use of classical logic causes proofs to collapse. I feel this is an excellent object lesson in the importance of constructive reasoning in language design.

References

[AB01] Steve Awodey and Andrej Bauer. Propositions as [Types]. Technical report, Institut Mittag-Leffler, 2001.

[ABM01] C. Areces, P. Blackburn, and M. Marx. Hybrid logics: Characterization, interpolation and complexity. *Journal of Symbolic Logic*, 66(3):977–1010, 2001.

[And92] J. M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[BdP03] T. Braüner and V. de Paiva. Towards constructive hybrid logic. *Elec. Proc. of Methods for Modalities*, 3, 2003.

[BdP06] Torben Braüner and Valeria de Paiva. Intuitionistic hybrid logic. To appear., 2006.

[Bel82] Nuel Belnap. Display logic. *Journal of philosophical logic*, 11:375–417, 1982.

[CP02] Iliano Cervesato and Frank Pfenning. A linear logical framework. *Inf. Comput.*, 179(1):19–75, 2002.

[DHK95] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Higher-order unification via explicit substitutions. In D. Kozen, editor, *Proceedings of the Tenth Annual Symposium on Logic in Computer Science*, pages 366–374, San Diego, California, June 1995. IEEE Computer Society Press.

[DHKP96] Gilles Dowek, Thérèse Hardin, Claude Kirchner, and Frank Pfenning. Unification via explicit substitutions: The case of higher-order patterns. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 259–273, Bonn, Germany, September 1996. MIT Press.

[Gab90] Dov Gabbay. Labelled deductive systems. Technical Report 90-22, University of Munich, 1990.

[Gir87] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[Gir91] Jean-Yves Girard. A new constructive logic: classical logic. *Mathematical structures in Computer Science*, 1:255–296, 1991.

[Gir93] Jean-Yves Girard. On the unity of logic. *Annals of Pure and Applied Logic*, 59:201–217, 1993.

[Gri90] Timothy Griffin. A formulae-as-types notion of control. In *Conference Record of the 17th Annual Symposium on Principles of Programming Languages (POPL'90)*, pages 47–58, San Francisco, California, January 1990. ACM Press.

[Her05] Hugo Herbelin. On the degeneracy of sigma-types in presence of computational classical logic. In *TLCA*, pages 209–220, 2005.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[ML96] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.

[NL97] George C. Necula and Peter Lee. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Langauges (POPL '97)*, pages 106–119, Paris, January 1997.

[OS97] Luke Ong and Charles Stewart. A Curry-Howard foundation for functional computation with control. In *Conference Record of the 24th Annual Symposium on Principles of Programming Languages (POPL'97)*, pages 215–227, Paris, France, January 1997. ACM Press.

[Par92] Michel Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *Logic Programming and Automated Reasoning*, pages 190–201. Springer LNCS 624, 1992.

[PD01] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, 2001.

[PE89] Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *Proceedings of the ACM SIGPLAN '88 Symposium on Language Design and Implementation*, pages 199–208, Atlanta, Georgia, June 1989.

[Pfe01] Frank Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In J. Halpern, editor, *Proceedings of the 16th Annual Symposium on Logic in Computer Science (LICS'01)*, pages 221–230, Boston, Massachusetts, June 2001. IEEE Computer Society Press.

[PS99] Frank Pfenning and Carsten Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, July 1999. Springer-Verlag LNAI 1632.

[Ree02a] Jason Reed. Higher-order pattern unification and proof irrelevance. Appears in TPHOLs 2002 Track B proceedings, NASA tech report CP-2002-211736, 2002.

[Ree02b] Jason Reed. Proof irrelevance and strict definitions in a logical framework. Technical Report CMU-CS-02-153, Carnegie Mellon University, 2002.

[Ree03] Jason Reed. Extending higher-order unification to support proof irrelevance. In David A. Basin and Burkhart Wolff, editors, *Theorem Proving in Higher Order Logics, 16th International Conference, TPHOLs 2003, Rome, Italy, September 8-12, 2003, Proceedings*, volume 2758 of *Lecture Notes in Computer Science*. Springer, 2003.

[Ree07] Jason Reed. Hybridizing a logical framework. In P. Blackburn, T. Bolander, T. Braner, V. de Paiva, and J. Villadsen, editors, *Proceedings of the International Workshop on Hybrid Logic (HyLo 2006)*, 2007.

[Ree08] Jason Reed. Higher order constraint simplification in a dependent type theory. Submitted. Available at http://www.cs.cmu.edu/∼jcreed/papers/csl08-hocs.pdf, 2008.

[Ree09] Jason Reed. A judgmental deconstruction of modal logic. Draft manuscript, 2009.

[RP07] Jason Reed and Frank Pfenning. Intuitionistic letcc via labelled deduction. In C. Areces and S. Demri, editors, *Proceedings of the 5th Workshop on Methods for Modalities (M4M5)*, 2007.

[RP09] Jason Reed and Frank Pfenning. A constructive approach to the resource semantics of substructural logics. Submitted. Available at http://www.cs.cmu.edu/∼jcreed/papers/rp-substruct.pdf, 2009.

[Wad03] Philip Wadler. Call-by-value is dual to call-by-name. In *ICFP '03: Proceedings of the eighth ACM SIGPLAN international conference on Functional programming*, pages 189–201, New York, NY, USA, 2003. ACM Press.