Proofs as Processes

Samson Abramsky

April 14, 1994

The main purpose of this short paper is to serve as an introduction to the following paper, "On the π-calculus and Linear Logic", by Gianluigi Bellin and Philip Scott. The circumstances from which it arises are as follows. In June 1991 I gave a lecture on "Proofs as Processes" at the symposium held at Tel-Aviv University to celebrate Boris Trakhtenbrot's 70'th birthday [BT91]. Material from this lecture also appeared in lectures subsequently given at the International Category Theory Meeting in Montreal (June 1991) and the London Mathematical Society Symposium on Category Theory in Computer Science in Durham (July 1991). The material was also presented in my Tutorial Lectures on Linear Logic given at the International Logic Programming Symposium in San Diego in October 1991. The lecture notes for these talks, particularly the Tutorial Lecture, were quite widely circulated; however, I did not write up a paper for publication, for reasons shortly to be explained.

My point of departure in this work was the "Propositions as Types" paradigm (encompassing such notions as "Curry-Howard Isomorphism", Realizability, Functional Interpretation and BHK semantics) familiar from proof theory and typed functional programming (see *e.g.* [GLT89]). In this paradigm, we have the correspondences

Formulas	Types
Proofs	(Functional) Programs
Normalisation of Proofs	Computation

Thus a familiar proof rule such as Implication Elimination is equated to the type inference rule for function application:

$$\frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

Type checking in functional programming is employed to constrain the use of functional application, guaranteeing "compatibility" of function and arguments, and hence good behaviour of well-typed programs (*e.g.,* Strong Normalisation [GLT89]). Type checking is probably one of the most successful applications of "formal methods" to date.

My programme was to transfer the Propositions as Types paradigm to concurrency, so that concurrent processes, rather than functional programs, become the computational counterparts of proofs. This was the intention behind the "Proofs as Processes" slogan. The motivation was twofold:

- To lay the foundation for a useful discipline of typed concurrent programming.
- To provide the basis for a more structural view of concurrency, comparable to what we have for functional languages; and indeed to integrate the two paradigms into a single, unified theory.

The material on “Proofs as Processes” presented in the above mentioned lectures was a first substantive step towards this programme. The idea was to seek to turn the hints and speculations in Girard’s work in Linear Logic [Gir87] about its applicability to concurrent computation into a fully realised, detailed connection. The key element of the approach had already appeared in my earlier paper on “Computational Interpretations of Linear Logic” [Abr93a]. The question considered there was : how to give a computational interpretation of the *duality* in Classical Linear Logic. The Cut rule of Intuitionistic Logic (or indeed of Intuitionistic Linear Logic)

$$\frac{\Gamma \vdash A \quad A, \Delta \vdash B}{\Gamma, \Delta \vdash B}$$

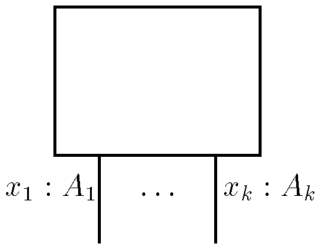
has an asymmetric form: the Cut formula appears as the conclusion—the output —in the first premise, and as a hypothesis—an input—in the second. The standard computational interpretation models this by function composition, the basic example of a non-commutative operation. By contrast, the Cut rule of Classical Linear Logic

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, A^{\perp}}{\vdash \Gamma, \Delta}$$

is completely symmetric with respect to its premises, bearing in mind that we have $A = A^{\perp\perp}$, and also the Exchange rule. This symmetry requires that it be interpreted by a commutative operation. As already suggested in [Abr93a], the appropriate operation is *parallel composition*, which appears in one form or another in process calculi such as CCS or CSP [Mil89, Hoa85].

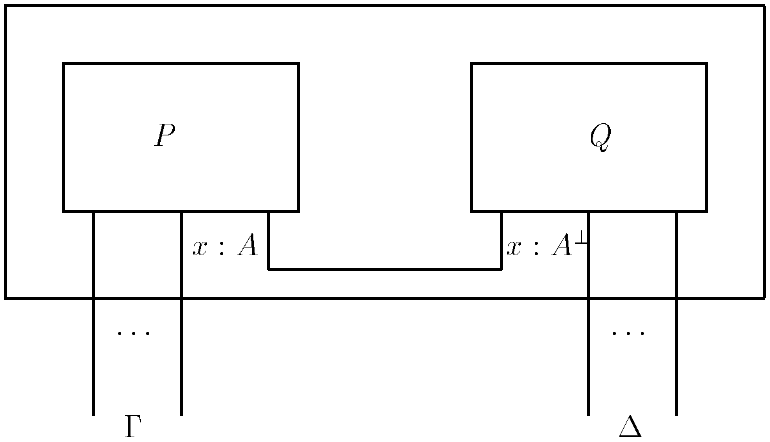
Two additional insights set the scene for the “Proofs as Processes” work:

1. A reading of one-sided sequents $\vdash \Gamma$, where $\Gamma \equiv A_1, \ldots, A_n$, as *interface specifications* for concurrent processes,

   where the A_i constrain how the interface ports labelled $x_1, \ldots, x_n$ can be plugged into corresponding ports in the environment of the process.

   Thus we see the constraint embodied in the cut rule, that A can only be cut with $A^{\perp}$, as a “plug-compatiblity condition” constraining the setting up of interactions between processes, just as the Implication Elimination rule restricts functional application, *i.e.,* interactions between functions and their arguments.



2. A more precise reading of Cut as "Parallel Composition + Hiding/Restriction" *i.e.,* as $(P|Q)\backslash x$ in CCS, or $(P||Q)/x$ in CSP, where x is the channel formed by plugging together the interfaces of P and Q. The dynamics of Cut, which logically appears as the process of Cut Elimination, then corresponds to the interaction of P and Q via this channel—in CCS terms, to the τ-computation, in CSP to the hidden actions. Termination of Cut-Elimination corresponds to *finiteness* of interaction: in CCS terminology to convergence (no infinite τ-computations), in CSP terminology to no "infinite chattering"[1].

With this background, I then gave a translation from proofs in Linear Logic into Milner's π-calculus (in the original version described in [MPW92]), and outlined the results relating the computational behaviour of the proofs under Cut Elimination to that of the processes under τ-computation. This material is essentially what is presented in Section 3 of the Bellin-Scott paper. The differences are as described by the authors: they use the synchronous version of the π-calculus developed by Robin Milner, partly in response to my translation of Linear Logic and the issues it gave rise to; and they replace the bidirectional buffers I had used by unidirectional ones. The justification of this latter step is the beautiful treatment of flow of information in Section 5 of the Bellin-Scott paper, which builds on as yet unpublished results of Bellin and Van der Wiele. I should stress at this point that everything *after* section 3 in the Bellin-Scott paper is their own original work, and goes much more deeply into Linear Logic proof theory than I had done.

My lectures also contained other material which is not described in the Bellin-Scott paper. In particular the combinators that arise in the translation are the syntax of what I called "Linear Realizability Algebras" (LRA's). The idea was to capture the minimal structure needed to give a (type-free) realizability interpretation of proofs in CLL, in much the same way that partial combinatory algebras provide the basis for Intuitionistic realizability. π-calculus is one example of an LRA; others have subsequently been provided, *e.g.,* in Concurrent Constraint Programming by Radha Jagadeesan and Prakash Panangaden (unpublished), and by myself and Jagadeesan for the Geometry of Interaction [AJ92]. Moreover, the proof of Strong Normalization for CLL can be abstracted into a powerful general result for LRA's, somewhat analogous to the "Fundamental Theorem of Logical Relations" [Sta82, Plo80]. A fairly detailed treatment of LRA's is given in [AJ94], the full version of [AJ92].

[1]It should be mentioned that Thierry Coquand has independently explored some similar ideas in his work on the computational content of Classical Logic [Coq92]

Back to chronology. In July 1991, I was contacted by Robin Milner. At the Marktoberdorf summer school which had just finished, Bob Constable (who had been to the Trakhtenbrot symposium) had mentioned my translation of Linear Logic to the π-calculus. Robin had insisted on not being told the details, and worked out his own translation. Now he wanted to compare notes. Our respective translations crossed over in the post. They were close enough to being the same to instil confidence in the naturalness of the translation. The significant differences were

1. Milner used unidirectional buffers, as already discussed.

2. He used parallel composition instead of non-determinism for the additives and exponentials. This latter point is discussed in Section 6 of the Bellin-Scott paper, and is related there to slicings of proof nets.

The discussions we had at this time apparently provided some stimulus towards Milner's development of a synchronous version of the π-calculus, used by Bellin and Scott; this in turn led to Milner's subsequent extensive development of his theory of Action Structures [Mil93a, Mil93b].

Phil Scott arrived in Edinburgh in October 1991 for his sabbatical year. He and Gianluigi Bellin, initially as an "exercise", looked at my lecture notes on "Proofs as Processes"; the end result is the paper appearing in this issue, which contains a wealth of new insights and results. It seems fair to say that they have been more concerned with what the π-calculus can tell us about the computational fine structure of Linear Logic proof theory, rather than what Linear Logic can tell us about concurrency.

This brings me back to the "Proofs as Processes" programme, and its current status. My reason for not having published the material in my lectures is that I felt I had not really achieved my principal objective, to develop a Propositions-as-Types paradigm for concurrency. To achieve this convincingly, I needed to show how a process calculus, sufficiently expressive to allow a reasonable range of concurrent programming examples to be handled, could be exhibited as the computational correlate of a proof system, in analogy to the situation with typed functional languages. The translation of Linear Logic proofs into π-calculus, by contrast, exhibits meanings of proofs as certain, very special process terms. In short, what was needed was a demonstration of "Processes as Proofs"; or, switching to a more semantic mode of description, and thinking in terms of categories rather than syntactically formulated proof systems, of "Processes as Morphisms". Such a demonstration has now been provided, in my opinion, by the subsequent work on Interaction Categories [Abr93b, Abr94]. I still feel that the "Proofs as Processes" work was an essential step along the way, and I am particularly pleased that it has also provided some stimulus to the work of Bellin and Scott.

References

[Abr93a] S. Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993. Revised version of Imperial College Technical Report DoC 90/20, October 1990.

[Abr93b] S. Abramsky. Interaction categories (extended abstract). In G. L. Burn, S. J. Gay, and M. D. Ryan, editors, *Theory and Formal Methods 1993*, Workshops in Computing, pages 57–70. Springer-Verlag, 1993.

[Abr94] S. Abramsky. Interaction categories and communicating sequential processes. In A. W. Roscoe, editor, *A Classical Mind: Essays in honour of C. A. R. Hoare*, pages 1–16. Prentice Hall International, 1994.

[AJ92] S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. In *Proceedings of the Seventh Symposium on Logic In Computer Science*, pages 211–222. Computer Society Press of the IEEE, June 1992.

[AJ94] S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. *Information and Computation*, 1994. In press.

[BT91] V. Breazu-Tannen. Report on international symposium on theoretical computer science. *Bulletin of the European Association of Theoretical Computer Science*, pages 295–297, 1991.

[Coq92] T. Coquand. A semantics of evidence for clasical arithmetic. Preliminary version, 1992.

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

[GLT89] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil93a] R. Milner. Action calculi I. In *Proceedings of MFCS '93*, volume 711 of *Lecture Notes in Computer Science*, pages 105–121. Springer-Verlag, 1993.

[Mil93b] R. Milner. An action structure for the synchronous pi-calculus. In *Proceedings FCT '93*, volume 710 of *Lecture Notes in Computer Science*, pages 87–105. Springer-Verlag, 1993.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100:1–77, 1992.

[Plo80] G. D. Plotkin. Lambda-definability in the full type hierarchy. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 363–373. Academic Press, 1980.

[Sta82] Rick Statman. Completeness, invariance and λ-definability. *J. Symbolic Logic*, 47:17–26, 1982.