Games in the Semantics of Programming Languages

Samson Abramsky
Department of Computer Science
University of Edinburgh

1 Introduction

Mathematical models of computation, from the λ-calculus to PRAM's, have played a key role in the development of software technology, from programming languages to the design and analysis of algorithms. The first generation of models of computation (*c.* 1950–1980) were *functional* in character; that is, they abstracted the behaviour of a program as the computation of a function. This was an appropriate abstraction in the era of batch processing, and provided a good basis for the formalization of the *correctness* of such programs with respect to their specifications, and of the *computational complexity* of algorithmic problems.

Meanwhile, Information Technology had progressed from batch processing through multi-tasking operating systems to distributed systems and on to today's Internet. The focus has correspondingly shifted, from stand-alone *programs* to *systems*, and from *computation* to *interaction*. Stand-alone programs computing functions are now seen as (very) special cases of a wider class of behaviours, in which the components of a complex system of concurrently executing agents interact with each other to achieve some global effect. On a "macro" scale, this description evidently fits today's distributed transactions across the Internet; but on a "micro" scale, it is equally true of how functional computations are ultimately realized, whether in software or in hardware.

The challenge has then been to find models of computation which take interaction as their basic ingredient, and which combine sufficient expressive power to yield faithful descriptions of the wealth of phenomena arising from contemporary developments in IT, with sufficient mathematical structure and tractability to provide a basis for the formal analysis of such systems. An additional, and no less important goal is to address seriously the issue of *doing the fundamental science of complex interacting systems.*

The fundamental character of this challenge is indicated by the following observation. The notions of *function*, *set* and *algorithm* were available "off-the-shelf" from mathematics and logic for use in computer science. By contrast, there is no adequate pre-existing theory of processes, interaction, information flow etc. on which these "second-generation" models can build. Rather, their most basic task is precisely to give a non-trivial analysis of these concepts.

Game Semantics

Game semantics has had a fairly lengthy history in logic, with pioneering work done by Lorenzen in the 1950's [Lor60]. The basic idea is to interpret a formula as a two-person game (played between the "Proponent" asserting the thesis, and the "Opponent" seeking to refute it), and a *proof* of the formula as a *winning strategy* for Proponent. Significant further contributions were made by Joyal [Joy77], who organized Conway's games into a category, and by Blass [Bla93], who gave a game semantics for Girard's Linear Logic.

In joint work with Radha Jagadeesan in 1992 [AJ92], a number of key ideas were introduced:

- A "dictionary" between the concepts of game semantics, and those of process models of computation, in which the Player or Proponent is interpreted as the *System* being modelled, and the Opponent as its *Environment*. The interaction between system and environment is modelled by the idea of a *play* of the game. The significance of game semantics as a computational model is then apparent; it models interaction in an intrinsic fashion. In fact game semantics can be seen as a key *refinement* of the process models which had been previously considered; the refinement being that the distinction between System and Environment is made explicit, and used as a key for eliciting much of the mathematical structure. This (apparently) simple step has proved to be almost "unreasonably effective".

- Based on this distinction, precise technical links were made between constructions in game semantics and process models. Most importantly, composition of strategies (roughly, "playing one strategy off against another") was formalized as "parallel composition plus hiding", following the process interpretation of Cut-elimination in Linear Logic previously given in my work on "Proofs as Processes" [Abr94]. This has proved to be an effective and tractable formalization, which has been a basic tool in the subsequent fine-grained analysis of models based on games.

- An important difference from the previous literature from Logic on game semantics was the insistence on giving a free-standing, thoroughly semantic, compositional account. This is not a trivial matter—Blass' game semantics had a non-associative composition, for example. This insistence on the semantic approach, and on analyzing the structure of categories of games, has been fundamental to later work.

- Using the above tools, a result on *Full Completeness* for the multiplicative fragment of Linear Logic was proved. More important than the result itself was the basic paradigm which was established. Traditional completeness theorems in logic have focussed on characterizing *provability* by validity in some model or class of models. The focus from the point of view of computation or proof theory is on the proofs themselves as mathematical objects, rather than on the mere fact of provability. The idea of full completeness is to have a model given by purely semantic means (i.e. independently of the syntax) such that every element of the model is the denotation of some proof (this can be stated in terms of the *fullness* of the functor from the free category based on the syntax, hence the name). This result made it plausible that game semantics could provide a powerful tool for addressing *full abstraction problems* for the semantics of programming languages. Full completeness is also of considerable interest for logic itself, and in fact this paper spawned a (still-growing) literature on full completeness results for the same or similar fragments of Linear Logic with respect to a variety of models [HO92, Loa94, BS96].

Full Abstraction for PCF

Motivated by the full completeness results, it became of compelling interest to re-examine perhaps the best-known "open problem" in the semantics of programming languages, namely the "Full Abstraction problem for PCF", using the new tools provided by game semantics.

PCF is a higher-order functional programming language; modulo issues of the parameter-passing strategies, it forms a fragment of any programming language with higher-order procedures (which includes any reasonably expressive object-oriented language). The aspect of the Full Abstraction problem I personally found most interesting was: to construct a syntax-independent model in which every element is the denotation of some program (note the analogy with full completeness, whose definition had in turn been motivated in part by this aspect of full abstraction). This is not how the problem was originally formulated, but by "general abstract nonsense", given such a model one can always quotient it to get a fully abstract model in the original sense. The seminal papers on PCF by Milner and Plotkin appeared in 1977 [Mil77, Plo77]; and although much important work was done on the full abstraction problem, it had not proved possible to construct a fully abstract model by syntax-independent means.

In 1993 fully abstract models for PCF based on game semantics were constructed, independently, by Abramsky, Jagadeesan and Malacaria [AJM96]; Hyland and Ong [HO96]; and Nickau. This was a striking illustration of the effectiveness of the tools provided by game semantics in making a fine-grained analysis of subtle computational notions which had eluded previous attempts. In effect, these models interpreted higher-order functional programs as certain interactive processes (i.e. strategies for certain games corresponding to the types of PCF). This was not new in itself (for example, Milner had given mappings of the λ-calculus into his π-calculus). What *was* new was the precise characterization of exactly which such processes were in the image of the interpretation, the characterization moreover being in terms of "local", compositional properties (the two key ones being "history-freeness" or"innocence", i.e. a constraint on the information available to a strategy at any given stage in a computation; and "well-bracketedness", expressing that the control flow followed a properly nested stack discipline. These properties are *local* in that they are properties either of single "runs" or "plays", or at most of two plays at a time.) But was this to prove an isolated result?

The semantic cube

Over the past three years, a substantial further development of the application of games to the semantics of programming languages has taken place. Much of this can be organized around a "semantic cube" first suggested in [AM97a], and later elaborated in my Marktoberdorf lectures and other work. Here we only discuss two dimensions—a diamond rather than a cube.

The idea is that a "space of programming languages" is articulated as a family of extended typed λ-calculi. This gives rise to a "syntactic cube", whose "origin" is a purely functional language; various "orthogonal" extensions are considered, such as local state (e.g. **ref**-types), or non-local control operators (e.g. **call-cc** or exceptions). The corresponding semantic cube has the highly constrained model used to give the fully abstract model for PCF (which extends to much richer functional languages [M96]), at the origin; while the various language extensions are mirrored by relaxation of the various (independent) semantic constraints on languages which were used to capture the purely functional model. In particular, local state is captured simply by allowing fully history-sensitive strategies, while non-local control is captured by relaxing the well-bracketing constraint.

A series of results [AM97a, AM97b, AM97c, Lai97, HY97] have shown that the correspondence between the syntactic and semantic cubes is indeed exact in the sense that the semantic models are fully abstract for the corresponding languages. A remarkable and unexpected feature of these results is that in each case definability for the model of the extended language is proved by using a technique of *factorization theorems*. That is, one shows that every strategy in the larger (less constrained) space factors by a constrained strategy through a particular, "generic" unconstrained strategy. In the case of local state, this will be the strategy interpreting **newref**, while for non-local control it will be **call-cc**. This can be seen as a kind of "semantic normal form", and it immediately yields definability for the model of the extended language by reduction to definability in the restricted language. Thus the original result for PCF proves to be the lynch-pin for a whole range of results for much richer languages.

Strachey's programme

To put these results in context, it is worth recalling the programme for a Mathematical Theory of Programming Languages mapped out by the pioneer Christopher Strachey in [MS76]. Strachey's specific goal in that work was to give a precise mathematical account of a wide-spectrum modern programming language, which would provide a rigorous basis for software engineering. A reasonable restatement of this programme in current terms might be: *to give a fully abstract model of Standard ML* (or better, for a meta-language adequate to describe Standard ML). This has been a goal which has seemed well out of reach for the 20 years following the appearance of Strachey's monograph; the methods needed to build a fully abstract model of even the functional subset of such a meta-language were lacking; while getting a fully abstract model e.g. for local state was seen as a difficult problem even modulo a fully abstract model for PCF.

This landscape has been transformed by the recent results on game semantics. We are now well on the way to completing the version of Strachey's programme stated above.

Recently, in joint work with Kohei Honda and Guy McCusker, we have extended the analysis of local state to general reference types, resulting in a fully abstract model for a call-by-value λ-calculus with general **ref**-types. This was the major remaining step towards an analysis of Core ML. Combining this with the impressive recent progress made by my student Jim Laird in analyzing the "control axis" of the semantic cube, the goal of obtaining a fully abstract model for Core ML is now well in sight. Given the structure of factorization theorems with respect to sub-models accompanying these results, it should be possible to give a rather comprehensive and highly articulated account of the entire "space" of programming languages between the purely functional fragment and full Core ML.

The analysis of general reference types is of considerable interest in itself. It carries much of the same force as "mobility" in the context of Milner's π-calculus. Of particular interest is that the model is quite different in character from those previously proposed for references or for names as in the π-calculus. References are analyzed in terms of information flow; a reference "is" a strategy for connecting up readers with writers. The ramifications of this remain to be explored.

References

[AJM96] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. Submitted for publication, 1996.

[AM97a] S. Abramsky and G. McCusker. Linearity, Sharing and State. In *Algol-like languages*, ed. P. O'Hearn and R. D. Tennent, 317–348, Birkhauser 1997.

[AM97b] S. Abramsky and G. McCusker. Full Abstraction for Idealized Algol with Passive Expressions. Submitted for publication, 1997.

[AM97c] S. Abramsky and G. McCusker. Call-by-value games. In *Proceedings of CSL '97*, 1997.

[AJ94] S. Abramsky and R. Jagadeesan. New Foundations for the Geometry of Interaction. *Information and Computation*, 111(1), (1994), 53–119.

[AJ92] S. Abramsky and R. Jagadeesan. Games and Full Completeness for Multiplicative Linear Logic", *Journal of Symbolic Logic*, (1994), vol. 59, no. 2, 543–574.

[Abr94] S. Abramsky. Proofs as Processes. *J. Theoretical Computer Science* 135, (1994), 5–9.

[Abr96] S. Abramsky. Retracing some paths in process algebra. In *CONCUR '96: Concurrency Theory, 7th International Conference.* U. Montanari and V. Sassone, eds. (Springer-Verlag) 1996, 1–17.

[BHA85] G. Burn, C. Hankin and S. Abramsky. Strictness Analysis for Higher Order Functions. *Science of Computer Programming* 7, (1986), 249–278.

[AGN97a] S. Abramsky, S. Gay and R. Nagarajan. Interaction Categories and the Foundations of Typed Concurrent Programming. In *Proceedings of the 1994 Marktoberdorf Summer Sxhool on Deductive Program Design*, M. Broy, ed. (Springer-Verlag) 1996, 35–113.

[AGN97b] S. Abramsky, S. Gay and R. Nagarajan. Specification Structures and Propositions-as-Types for Concurrency. In *Logics for Concurrency: Structure vs. Automata—Proceedings of the VIIIth Banff Higher Order Workshop*, G. Birtwistle and F. Moller, eds. (Springer-Verlag) 1996, 5–40.

[Abr97] S. Abramsky. Axioms for Full Abstraction and Full Completeness. Submitted for publication, 1996.

[AR97] S. Abramsky and J. Rosicky. **SProc** as a final coalgebra. Submitted for publication, 1995.

[Bla93] A. Blass. A Game Semantics for Linear Logic. Annals of Pure and Applied Logic 56, 1993.

[BS96] R. Blute and P. J. Scott. Linear Lauchli Semantics. Annals of Pure and Applied Logic, 1996.

[HY97] K. Honda and N. Yoshida. Game-theoretic analysis of call-by-value computation. In *Proceedings of ICALP '97.*

[HO92] M. Hyland and C.-H. L. Ong. Fair Games and Full Completeness for Multiplicative Linear Logic without the MIX rule. Unpublished Manuscript, 1992.

[HO96] M. Hyland and C.H. L. Ong. On full abstraction for PCF. Submitted for publication, 1996.

[Joy77] A. Joyal. Remarques sur la théorie des jeux a deux personnes. Gazette des sci. math. du Quebec, 1,4 1977.

[Lai97] J. Laird. Full abstraction for functional languages with control. In *Proceedings of LiCS '97*.

[Loa94] R. Loader. Linear Logic, Totality and Full Completeness. In *Proceedings of LiCS '94*.

[Lor60] P. Lorenzen. Logik und Agon. In *Atti Congr. Internat. di Filosofia*, vol. 4, Sansoni, Firenze, 1960.

[M96] G. McCusker. Games and Full Abstraction for a functional meta-language with recursive types. Ph.D. thesis, University of London, 1996.

[MS76] R. Milne and C. Strachey. A Theory of Programming Language Semantics. Chapman and Hall, 1976.

[Mil77] R. Milner. Fully Abstract Models for Typed λ-calculi. *Theoretical Computer Science* 4:1–22, 1977.

[Plo77] G. Plotkin. LCF Considered as a Programming Language. *Theoretical Computer Science* 5:223–255, 1977.