

Chapter 2

From CSP to Game Semantics

Samson Abramsky

Abstract In this short essay, we describe in informal terms how game semantics can be seen to arise as a perturbation of process calculi such as CSP, by making an explicit distinction between the rôles of the System and the Environment. Drawing out the consequences of this distinction uncovers a wealth of mathematical structure, with Game intuitions entering in a natural and compelling fashion. This leads ultimately to the elaboration of mathematically well-structured and behaviourally expressive semantic universes for computation. These provide a basis for fully abstract models of a wide range of programming languages, and lead on to algorithmic methods, with applications to compositional model-checking and program analysis.

2.1 Introduction

Tony Hoare has been a major influence on me, as on so many of my generation of computer scientists. Moreover, he has always shown a keen interest in the work on game semantics by myself and others, and his remarks have often forced me to sharpen my thinking. In this short note, a mini-essay rather than a paper, I will try to present and motivate some key features of game semantics from the point of view of CSP and other process calculi. I will try to convey how we can see game semantics as arising from a small — but significant! — perturbation of the CSP paradigm. We add one, apparently minor, piece of additional structure, and from this much else follows.

2.2 Polarity

We shall assume some background in CSP [18, 32], or other process calculi such as CCS [27]. We know that these calculi are designed to express **communication**, or more generally **interaction**, between concurrent processes. This is achieved in each

S. Abramsky (✉)

Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, UK
e-mail: samson@comlab.ox.ac.uk

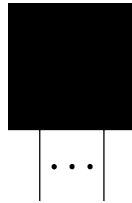
case by some form of synchronized communication which is built into the semantics of the parallel composition operation.

Although CSP allows for value-passing idioms, at a more fundamental level, these are “flattened out” or “compiled away” into pure atomic actions. Similar comments can be made for CCS. In the case of CCS, a basic distinction is made between actions and coactions (α and $\bar{\alpha}$), but this is purely formal, and its purpose is to support the particular form of binary “synchronization algebra” of CCS.¹

In principle, then, all actions in process calculus are at the same level; each action could, a priori, be performed by any process. In the language of logic, actions do not have (positive or negative) **polarities**; in the language of category theory, they do not have (co- or contra-) **variance**.

A basic starting point for game semantics is the recognition that introducing an explicit notion of **polarity of actions** provides a modest-looking fulcrum which we can use to lever up a great deal of structure. Moreover, this notion arises in a very direct fashion from basic intuitions about interactive systems.

The basic idea flows naturally from the observation that our universe of study in the arena of concurrent, distributed, mobile and pervasive processes is that of **open systems**. That is, we study systems which must be seen as embedded in some larger, and as yet not completely specified, system. (Just think of the Internet.) This means that a key part of identifying and delimiting any system we study is that we have a **boundary** between the system being considered, and the larger system which contains it. We call the system under consideration simply the **System**; and everything outside the boundary is the **Environment**. Actions can then be classified as performed by the **System** (under its control) or by the **Environment** (not under its control). The essence of the behaviour of the **System** is **how it interacts** with its **Environment** across this boundary.



To specify a **System** in these terms is to specify how it will react in the presence of an (unknown) **Environment**. Thus we must specify what the **System** will (or would) do given any possible behaviour by the **Environment**. This assumes a natural **nested conditional** or **hypothetical** form:

If the **Environment** initially does e_1 ,
then the **System** responds with s_1 ;

¹ The input–output distinction does occur in a more fundamental way in the π -calculus [28–30]. However, even here there are symmetric variants such as the fusion calculus [31], and the π -calculus does not seem to force a fundamental modification of the distinction we are making here.

If the Environment then does e_2 ,
 then the System responds with s_2 ;
 \vdots

This apparently minor manoeuvre, of expressing the System/Environment dichotomy directly in our classification of actions, releases much additional mathematical structure, as we shall now see.

2.3 Games

With the introduction of polarized moves, the **games** metaphor becomes natural, and indeed compelling. We think of the System and the Environment as the players in a two-person game. The hypothetical specification of the actions of a System in response to those of its Environment then exactly fits the game-theoretic notion of **strategy**; so we have a natural reading of processes in this polarized framework as strategies.

In this way, we start to use the basic notions of game theory to structure our semantic universe, and we are well on the road to game semantics. But this is only the first step.

One immediate consequence of adopting the game perspective is that we have a natural “dynamic” take on **negation** and **duality**: as interchange of polarity (reversal of rôles in the game) rather than interchange of truth-values.

Referring back to our intuitive reading, this corresponds to the fact that our choice of point of view across the System–Environment boundary is conventional; we could as well take the Environment as the System under consideration, and vice versa. This ability to interchange between different points of view on the same interaction is characteristic of game-theoretic reasoning.

2.4 Determinism

We now proceed to elaborate further structural consequences of our basic move to polarization. We start in a low key, with a humble yet profoundly useful notion: that of **deterministic computation**. The intuitive notion of a deterministic program or computation is rather clear: **at each step, what the program does next is uniquely determined by the previous history of the computation**. The lack of an explicit distinction between the actions of the process and those of its environment means that the notion of deterministic computation in this sense in the standard setting of process calculi is a very limited one.²

² These calculi do have notions of “deterministic” and “confluent” process as part of their theory [27, 32], but these notions refer to the absence of **non-observable branching** in the system.

By contrast, in the polarized setting, the obvious notion of deterministic strategy – one for which the System actions are uniquely determined by the preceding Environment actions – easily and naturally cover the full sweep of deterministic computation. Because of polarization, one can distinguish between the Environment actions, which a strategy for System cannot control, and hence must be allowed to branch arbitrarily; and the System actions, which in the case of a deterministic strategy must be uniquely determined by what has gone before. In this way, a large portion of what would usually be encoded as non-determinism in the process calculus setting is simply referred to the Environment, while the strategy remains perfectly deterministic.

2.5 Interaction

As we have already mentioned, each process algebra has its own hand-crafted primitives and semantics for communication and synchronization. There are many plausible variations, and different styles suggest different models and process equivalences. This profusion of choices is something of an embarrassment – an instance of the “next 700” syndrome [4, 20].

By contrast, the polarization structure of games entails an **intrinsic notion of interaction**: namely the basic notion in extensive game theory of playing a strategy (for System) off against a counter-strategy (for the Environment) (or “evaluating a strategy profile”). The specification of a strategy for System, as we have already seen, allows for arbitrary branching at Environment moves, while specifying what the System then does in response. A counter-strategy works dually. Thus the strategy and counter-strategy “fit” together, just by the logic of polarization, without need of any synchronization algebra. Note in particular that in the case that the strategy and the counter-strategy are both deterministic, the result of this interaction is a **uniquely determined computation trace**.

This idea really becomes powerful when we combine it with **types**, to which we now turn.

2.6 Types

Process calculus is fundamentally untyped, although sometimes a rudimentary sorting by action alphabet is used. More elaborate type systems for process calculi have been proposed and studied, but they are a kind of super-structure, motivated more by specification and verification issues than by articulating the structure of the universe of processes.

They do not directly correspond to the sense in which the computation of a standard functional or imperative program is deterministic.

Once again the game metaphor shows its ability to reveal significant additional mathematical structure. We have already been led to a view of processes as strategies, but for which games? If we constrain the moves which can be made by a player (System or Environment) at each stage where it may perform an action, we are led to the usual notion of a game tree (game in extensive form), and we can see that games play the role of **types** in a very natural and intuitive fashion.

The real power of this step comes when we consider the possibility of building **compound types** expressing the behaviour of systems built hierarchically from sub-systems. These compound types will be interpreted as constructions on games, which build more complex games, e.g. by combining “game boards”. Indeed, it is by virtue of this step to compound games that we can see why it is plausible to restrict ourselves to two-person games. System and Environment may each actually comprehend a “team” of players; the important point is that we are drawing a boundary between the teams, and distinguishing the actions of one team from the other. A useful analogy is with the theory of functions. By using type constructions such as cartesian product, it suffices to develop a theory of one-place functions, rather than having to treat n -place functions as primitive for each n . Indeed, note that functions can be seen as special cases of (deterministic) strategies; where the System allows the Environment to branch on the input, and then produces the corresponding output.

As a direct consequence of this idea of using structured types to build specifications of complex systems, we shall show how the primitive form of interaction intrinsically supported by games leads on to the fundamental notions of **composition** and **identity**.

2.7 Identity and Composition

2.7.1 Copy-Cat Strategies

Consider the following little fable, illustrated by Fig. 2.1. The idea is to rely on logic, rather than on any talent at Chess. We arrange to play two games of Chess with the grandmaster, say Gary Kasparov, once as White and once as Black. Moreover, we so arrange matters that we start with the game in which we play as Black. Kasparov makes his opening move; we respond by playing the **same** move in the **other** game – this makes sense, since we are playing as White there. Now Kasparov responds (as Black) to our move in that game; and we copy that response back in the first game. We simply proceed in this fashion, copying the moves that our opponent makes in one board to the other board. The net effect is that **we play the same game twice – once as White, and once as Black**. (We have essentially made Kasparov play against himself.) Thus, whoever wins that game, we can claim a win in one of our games against Kasparov! (Even if the game results in a stalemate, we have done as well as Kasparov over the two games – surely still a good result!) Of course, this idea has nothing particularly to do with Chess. It can be applied to any two-person



Fig. 2.1 How to beat a Grandmaster

game of a very general form. The use of Chess-boards to illustrate the discussion will continue, but this underlying generality should be kept in mind.

What are the salient features which can be extracted from this example?

A Dynamic Tautology There is a sense (which will shortly be made more precise) in which the copy-cat strategy can be seen as a **dynamic version** of the tautology $A \vee \neg A$. Note, indeed, that an essential condition for being able to play the copy-cat is that the roles of the two players are inter-changed on one board as compared to the other. Note also the disjunctive quality of the argument that we must win in one or other of the two games. But the copy-cat strategy is a **dynamic process**: a two-way channel which maintains the correlation between the plays in the two games.

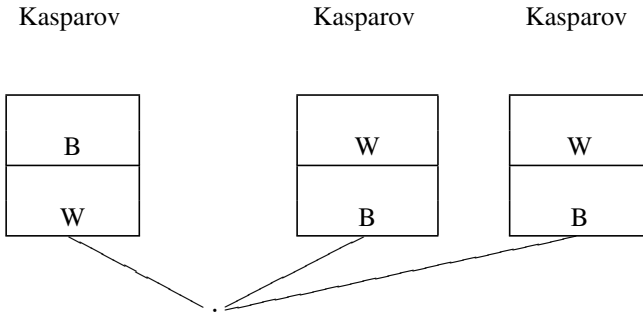
Conservation of Information Flow The copy-cat strategy does not **create** any information; it reacts to the environment in such a way that information is conserved. It ensures that exactly the same information flows out to the environment as flows in from it. Thus one gets a sense of logic appearing in the form of **conservation laws for information dynamics**.

The Power of Copying Another theme which appears here, and of which more will be seen later, concerns the surprising power of simple processes of copying information from one place to another. Indeed, as shall eventually be seen, such processes are **computationally universal**.

The Geometry of Information Flow From a dynamical point of view, the copy-cat strategy realizes a channel between the two game boards, by performing the **actions**

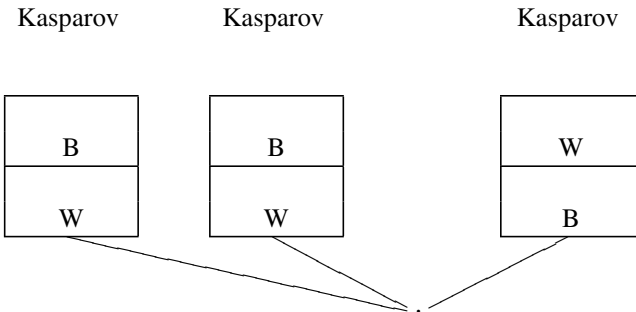
of copying moves. But there is also some implicit **geometry** here. Indeed, the very idea of two boards laid out side by side appeals to some basic underlying spatial structure. In these terms, the copy-cat channel can also be understood geometrically, as creating a graphical link between these two spatial locations. These two points of view are complementary, and link the logical perspective to powerful ideas arising in modern geometry and mathematical physics.

Further evidence that the copy-cat strategy embodies more substantial ideas than might at first be apparent, can be obtained by varying the scenario. Consider now the case where we play against Kasparov on **three boards**; one as Black, two as White.



Does the copy-cat strategy still work here? In fact, it can easily be seen that it does **not**. Suppose Kasparov makes an opening move m_1 in the left-hand board where he plays as White; we copy it to the board where we play as White; he responds with m_2 ; and we copy m_2 back to the board where Kasparov opened. So far, all has proceeded as in our original scenario. But now Kasparov has the option of playing a **different** opening move, m_3 say, in the rightmost board. We have no idea how to respond to this move; nor can we copy it anywhere, since the board where we play as White is already “in use”. This shows that these simple ideas already lead us naturally to the setting of a **resource-sensitive** logic, in which in particular the Contraction Rule, which can be expressed as $A \rightarrow A \wedge A$ (or equivalently as $\neg A \vee (A \wedge A)$) cannot be assumed to be valid.

What about the other obvious variation, where we play on two boards as White, and one as Black?

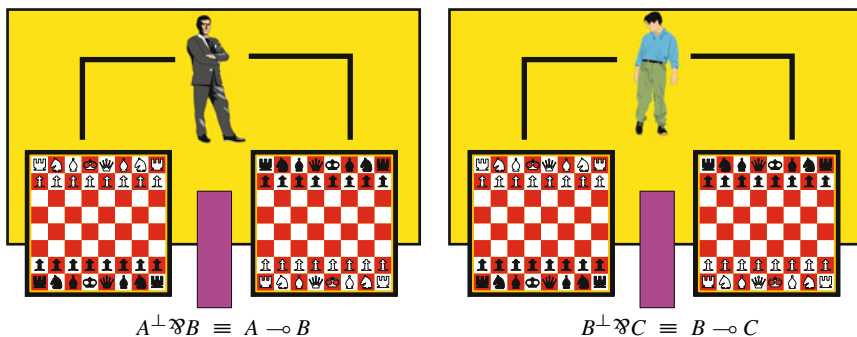


It seems that the copy-cat strategy **does** still work here, since we can simply ignore one of the boards where we play as White. However, a geometrical property of the original copy-cat strategy has been lost, namely a **connectedness** property, that information flows to every part of the system. This at least calls the corresponding logical principle of Weakening, which can be expressed as $A \wedge A \rightarrow A$, into question.

These remarks indicate that we are close to the realm of Linear Logic and its variants, and, mathematically, to the world of monoidal (rather than cartesian) categories.

2.7.2 Composition as Interaction

We also show how interaction can be explained in the same terms: Constructors create “potentials” for interaction; the operation of plugging modules together so that they can communicate with each other **releases** this potential into **actual computation**.



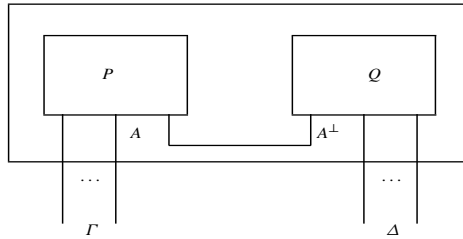
Here we see two separate sub-systems, each with a compound structure, expressed by the **logical types of their interfaces**. What these types tell us is that these systems are **composable**; in particular, the **output type** of the first system, namely B , matches the input type of the second system. Note that this “logical plug-compatibility” makes essential use of the duality, just as the copy-cat strategy did. What makes Gary (the player for the first system) a fit partner for interaction with Nigel (the player for the second system) is that they have **complementary views** of their locus of interaction, namely B . Gary will play in this type “positively”, as Player (he sees it as B), while Nigel will play “negatively”, as Opponent (he sees it as B^\perp). Thus each will become part of the environment of the other – part of the potential environment of each will be realized by the other, and hence part of the **potential** behaviour of each will become **actual** interaction.

This leads to a dynamical interpretation of the fundamental operation of **composition**, in mathematical terms:

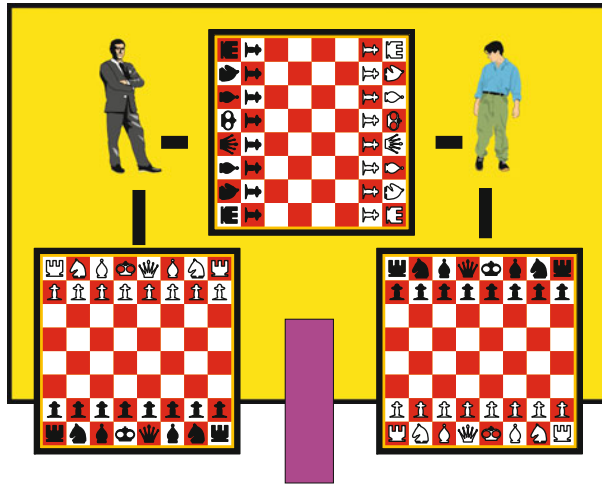
$$\frac{A \xrightarrow{\text{Gary}} B \xrightarrow{\text{Nigel}} C}{A \xrightarrow{\text{Gary; Nigel}} C}$$

or of the **Cut rule**, in logical terms:

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



Composition as Interaction



The picture here shows the new system formed by plugging together the two sub-systems. The “external interface” to the environment now shows just the left-hand board A as input, and the right-hand board C as output. The Cut formula B is hidden from the environment, and becomes the locus of interaction inside the black box of the system. Suppose that the Environment makes some move m in C .

This is visible only to Nigel, who as a strategy for $B \multimap C$ has a response. Suppose this response m_1 is in B . This is a move by Nigel as Player in B^\perp , hence appears to Gary as a move by Opponent in B . Gary as a strategy for $A \multimap B$ has a response m_2 to this move. If this response is again in B , Nigel sees it as a response by the environment to his move, and will have a response again; and so on. Thus there is a sequence of moves m_1, \dots, m_k in B , ping-ponging back and forth between Nigel and Gary. If, eventually, Nigel responds to Gary's last move by playing in C , or Gary responds to Nigel's last move by playing in A , then this provides the response of the **composed strategy** Gary; Nigel to the original move m . Indeed, all that is visible to the Environment is that it played m , and eventually some response appeared, in A or C .³

Summarizing, the two strategies are played off against each other in the shared part of their interfaces (where one plays as White and the other as Black – corresponding to matching of logical polarities), leaving a residual interface to the environment. Note the “duality” between this operation and the copy-cat (Identity and Cut): the copy-cat strategy makes **the same thing happen in two different places**, while composition makes **two different things (with opposite polarities, hence non-conflicting) happen in the same place**.

2.8 Categories: the “Objective Structure” of Interaction

Putting together the various ingredients we have developed from the basic idea of polarization, we now have a mathematical universe in which we have:

- Games as **types or objects**
- Strategies as **morphisms**
- Composition of strategies by playing them off against each other in the common subgame
- Copy-cat strategies as identities for this composition

This means that the games and strategies naturally organize themselves into a **category**. This immediately moves us onto a higher plane of mathematical organization. Recall that once we have fixed a category, any constructions defined by universal properties therein are already fixed up to unique isomorphism. For example, if a category is cartesian closed, and hence provides a model of higher-order computation, this is uniquely determined by the bare structure of arrows under composition in the category. This example is highly relevant, since the successful application of game semantics to providing fully abstract models for a wide range of λ -calculus based

³ It is also easy to show that if Nigel and Gary are both playing **winning strategies**, meaning that they always have a response to the Environment's actions, and that the infinite plays which may arise from following these strategies satisfy some given **liveness specifications**, then the composed strategy will again be a winning strategy, with respect to a liveness specification defined compositionally in a natural fashion from the given ones. See [1] for details and a proof of this.

programming languages rests upon the construction of suitable cartesian closed categories of games and strategies. The mathematical structure present in a category is objectively **there**, it is not a matter of one of “700 choices”.

In fact, categories of games and strategies have fascinating mathematical structure. They give rise to:

- Constructions of **free categories with structure** of various kinds.
- **Full completeness** results characterizing the “space of proofs” for various logical systems.
- There are even connections with **geometry**, e.g. Temperley-Lieb and other diagram algebras [5].

2.9 Developments: The Game Semantics Landscape

Over the past 15 years, there has been an extensive development of Game Semantics in Computer Science.⁴ One major area of application has been to the semantics of programming languages, where it has led to major progress in the construction of **fully abstract models** for programming languages embodying a wide range of computational effects, and starting with the first semantic construction of a fully abstract model for PCF, thus addressing a famous open problem in the field. It has been possible to give crisp characterizations of the “shapes” of computations carried out within certain **programming disciplines**: including purely functional programming [2, 19], stateful programming [9], general references [7, 33], programming with non-local jumps and exceptions [21, 22], non-determinism [16], probability [11], concurrency [15], names [6], and more. In many cases, game semantics have yielded the first, and often still the only, semantic construction of a fully abstract model for the language in question.

There has also been a parallel line of development of giving **full completeness** results for a range of logics and type theories, characterizing the “space of proofs” for a logic in terms of informatic or geometric constraints which pick out those processes which are proofs for that logic [8, 10, 12, 13, 25]. This enables a new look at such issues as the boundaries between classical and constructive logic, or the fine structure of polymorphism and second-order quantification.

More recently, there has been an algorithmic turn, and some striking applications to verification and program analysis [3, 6, 14, 24].

⁴ A key quality of this form of game semantics, as compared to earlier work in the logical literature, such as the Game-Theoretical Semantics of Hintikka [17] and the Dialogical game semantics of Lorenzen and his school [26], is its **syntax-independence and compositionality**. Here compositionality refers, crucially, to the level of **strategies** as well as merely to the games.

2.10 Concluding Remarks

It should be emphasized that game semantics cannot be said to **subsume** CSP or other process calculi. Indeed, by imposing more structure, it makes it harder to achieve the same breadth of expressive power. Game semantics has been extended to concurrent languages with some success [15, 23], but the treatment to date is far from comprehensive. Moreover, in the course of modelling concurrent programming languages, some of the familiar issues of a proliferation of models and equivalences tend to recur, albeit in a considerably reduced form, as the setting is much more constrained.

Nevertheless, it seems fair to say that game semantics has found a fruitful path, combining the mathematical structure of denotational semantics with much of the behavioural expressiveness of process calculi. We hope to have conveyed something of how this trend in semantics arises naturally as a refinement of the CSP and process calculus point of view. It is a tribute to Tony Hoare's vision that many of his insights persist in this new guise, and combine gracefully with other structures, seemingly of quite a different character.

Acknowledgements My thanks to Bill Roscoe and Paul Levy for their comments on an earlier version of this paper, which led to several clarifications. The remaining obscurities and inaccuracies are entirely my responsibility.

References

1. Abramsky, S.: Semantics of interaction: an introduction to game semantics. In: Dybjer, P., Pitts, A. (eds.), *Proceedings of the 1996 CLICS Summer School*, Isaac Newton Institute, pp. 1–31. Cambridge University Press, Cambridge (1997).
2. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF. *Inform. Comput.* **163**, 409–470 (2000).
3. Abramsky, S.: Algorithmic game semantics: a tutorial introduction. In: *Proof System-Reliability*, pp. 21–47. Kluwer (2002).
4. Abramsky, S.: What are the fundamental structures of concurrency?: We still don't know! *Electr. Notes Theor. Comput. Sci. (ENTCS)* **162**, 37–41 (2006).
5. Abramsky, S.: Temperley-Lieb algebra: from knot theory to logic and computation via quantum mechanics. In: Chen, G., Kauffman, L., Lomonaco, S., (eds.), *Mathematics of Quantum Computation and Quantum Technology*, pp. 415–458. Taylor & Francis, New York (2007).
6. Abramsky, S., Ghica, D.R., Murawski, A.S., Stark, I.D.B., Ong, C.-H.L.: Nominal games and full abstraction for the nu-calculus. In: *Proceedings LICS 150–159* (2004).
7. Abramsky, S., Honda, K., McCusker, G.: A fully abstract game semantics for general references. In: *Proceedings LiCS 334–344* (1998).
8. Abramsky, S., Jagadeesan, R.: Games and full completeness for multiplicative linear logic. *J. Symbolic Logic* **59**, 543–574 (1994).
9. Abramsky, S., McCusker, G.: Linearity, sharing and state. In: O'Hearn, P., Tennent, R.D. (eds.) *Algol-Like Languages*, pp. 317–348. Birkhauser, Basel (1997).
10. Abramsky, S., Mellies, P.-A.: Concurrent games and full completeness. In: *Proceedings LiCS 431–442* (1999).
11. Danos, V., Harmer, R.: Probabilistic game semantics. *ACM Trans. Comput. Log.* **3**(3), 359–382 (2002).

12. Blute, R., Hamano, M., Scott, P.J.: Softness of hypercoherences and MALL full completeness. *Ann. Pure Appl. Logic* **131**(1–3), 1–63 (2005).
13. Devarajan, H., Hughes, D., Plotkin, G., Pratt, V.: Full completeness of the multiplicative linear logic of Chu spaces. In: *Proceedings LiCS* 234–242 (1999).
14. Ghica, D.R., McCusker, G.: Reasoning about idealized algol using regular languages. In: *Proceedings ICALP'00*, pp. 103–116 (2000). LNCS 1853.
15. Ghica, D.R., Murawski, A.S.: Angelic semantics of fine-grained concurrency. In: *Proceedings FOSSACS'04*, pp. 211–225 (2004). LNCS 2987.
16. Harmer, R., McCusker, G.: A fully abstract game semantics for finite nondeterminism. In: *Proceedings LiCS* (1999).
17. Hintikka, J., Sandu, G.: Game-theoretical semantics, in van Benthem and ter Meulen. *Handbook of Logic and Language*. Elsevier, Amsterdam (1996).
18. Hoare, C.A.R.: *Communicating Sequential Processes*. Prentice Hall, New Jersey (1985).
19. Hyland, J.M.E., Ong, C.-H.L.: On full abstraction for PCF. *Inform. Comput.* **163**, 285–408, (2000).
20. Landin, P.J.: The next 700 programming languages. *Commun. ACM (CACM)* **9**(3), 157–166 (1966).
21. Laird, J.: Full abstraction for functional languages with control. Extended abstract. In: *Proceedings LICS* (1997).
22. Laird, J.: A fully abstract games semantics of local exceptions. Extended abstract. In: *Proceedings LICS* (2001).
23. Laird, J.: Game semantics for higher-order concurrency. In: *Proceedings FSTTCS 2006*, Springer LNCS Vol. 4337 pp. 417–428 (2006).
24. Legay, A., Murawski, A.S., Ouaknine, J., Worrell, J.: On automated verification of probabilistic programs. *TACAS* 173–187 (2008).
25. Loader, R.: *Models of lambda calculi and linear logic*. D.Phil. thesis, Oxford University (1994).
26. Lorenzen, P.: Ein dialogisches Konstruktivitätskriterium. In: *Infinitistic Methods*, 193–200 (1961).
27. Milner, R.: *Communication and Concurrency*. Prentice Hall, New Jersey (1989).
28. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I *Inf. Comput.* **100**(1), 1–40 (1992).
29. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, II *Inf. Comput.* **100**(1), 41–77 (1992).
30. Milner, R.: *Communicating and Mobile Systems: The Pi Calculus*. Cambridge University Press, Cambridge (1999).
31. Parrow, J., Victor, B.: The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes *LICS* 1998: 176–185.
32. Roscoe, A.W.: *The Theory and Practice of Concurrency*. Prentice Hall (1997).
33. Tzevelekos, N.: Full abstraction for nominal general references. In: *Proceedings LICS* pp. 399–410 (2007).

Reflections on the Work of C.A.R. Hoare

Jones, C.B.; Roscoe, A.W.; Wood, K.R. (Eds.)

2010, XII, 430 p., Hardcover

ISBN: 978-1-84882-911-4