Concurrent interaction games

Samson Abramsky

1 A question and a challenge

Our starting point will be a question and a challenge posed by Tony Hoare in [Hoare99]. The question:

> *Is game semantics inherently sequential?*

and the challenge:

> *If this is not the case, then it would be nice to obtain a top-down extensional characterization of sequentiality within a model that is non-sequential. …A full explanation of the concept of sequentiality must surely give equal weight to the alternative.*

As we will see, some recent work with Paul-André Melliès [AM99] is directly relevant to these issues.

2 Background: Why Games?

Firstly, though, we should provide some context for these questions, and why they are of interest in the first place.

As Information Technology has 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*. The "first-generation" models of computation gave an account of stand-alone programs computing functions from inputs to outputs. These 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.

Ultimately, it seems that *all* computation can be resolved into the interactions of large numbers of very simple agents: the complex behaviour of the overall system is an "emergent property" of these interactions.

The challenge has then been to find models of computation which take interaction as their basic ingredient. These models should combine sufficient expressive power to yield faithful descriptions of the phenomena arising from contemporary developments in IT, with sufficient mathematical structure and tractability to provide a basis for the formal analysis of such 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 "second-generation" models can build. Rather, the most basic task of such models is precisely to give a non-trivial analysis of these concepts. Much of the work on models of concurrency, notably the pioneering work on CSP [Hoa85] and CCS [Mil89], can be seen as addressing these issues.

Over the last decade, there has been a development of ideas around Linear Logic, Geometry of Interaction and Game Semantics which have made a distinctive contribution to these matters. In particular, Game Semantics, in which a system is seen as interacting with its environment by playing a game with it, and computational processes are modelled as strategies for playing such games, has proved to be a powerful and versatile tool. While somewhat akin to the way games are used to model interactions between rational agents in Economics, Game Semantics plays much more attention to the fine structure of the interactions themselves, rather than focussing mainly on the outcomes of the plays. Again, while Game Semantics draws on previous work on games in Logic, it pays much more attention to the mathematical structure of categories of games, and to issues of *compositionality* and *syntax-independence*—which are key to the analysis of interaction.

3 Some uses of games

Semantics of Programming Languages One of the basic tasks for a theory of computation is to give a precise, mathematically well-structured semantics for realistic programming languages. Programming languages are important because they embody the key conceptual tools of programming (e.g. procedures, higher-order functions, abstract data types, modules, objects, concurrent processes etc.). These have generally evolved in an empirical and pragmatic fashion, and they pose the challenge of finding a deeper mathematical analysis. This challenge can be quite severe: following the pioneering work of Christopher Strachey in the 1960's, progress towards getting accurate models of realistic programming languages has been slow. The new and powerful ideas coming from Game Semantics have changed the possibilities here significantly, to the point where fully abstract models of languages such as SCHEME and Core ML can be given. Moreover, the entire "space" of programming languages between the purely functional subsets and their extensions with non-functional features such as local state and control can be given a precise and highly structured account in terms of a lattice of sub-models determined by orthogonal constraints, and tightly related by factorization theorems. An overview of this work is given in the survey paper [AMcC99], and a brief summary in [Abr97].

Semantics of Proofs The traditional notion of completeness of logical systems focussed on characterizing provability in terms of classes of models. A new area of investigation, concerned with characterizing the space of *proofs* of a logic (which can be thought of as highly "regular" computational processes, to be modelled as strategies for playing certain games associated with the formulas of the logic), and leading to a novel notion of "full completeness", was opened up in [AJ92b], and has already generated considerable further work. A further development, extending the full completeness results to more expressive logics, is currently under way [AM99]. It is this more recent work, addressing some apparently very abstract foundational issues, which has opened some new perspectives on the issue of games and concurrency.

4 Concurrent Games: the very idea

Traditional forms of game semantics which have appeared in logic and computer science have been *sequential* in format: a play of the game is formalized as a sequence of moves. The key feature of this sequential format is the existence of a *global schedule* (or *polarization*): in each (finite) position, it is (exactly) one player's turn to move[1]. This sequential format turns out to have important limitative consequences when we wish to use game semantics to model programs or proofs:

- There is a modelling limitation. Sequential games can be used to model sequential computation, but do not yield models of parallel computation in a natural way.
- There is also a mathematical limitation. Despite the evident inherent duality in games (interchange the rôles of the two players), sequentiality is an obstacle to modelling logics in a classical format, such as (Classical) Linear Logic. In fact, sequential games have only yielded satisfactory models of *fragments* of Linear Logic: the Multiplicative fragment in [AJ92b], the Multiplicative-Exponential fragment in [BDER97], and the *negative fragment* in most other work. This has been sufficient to model λ-calculus-based programming languages, but is inadequate as a general account. We will illustrate this problem with respect to Andreas Blass's pioneering work on game semantics below.

We will solve these problems by making a radical departure from *all* the formal versions of "games in extensive form" used to date in Logic and Computer Sci-

[1] All the games considered in this paper, and in the relevant literature to date, are two-person games.

ence of which we are aware. We shall introduce a "true concurrency" version of games in which the global polarization is abandoned. *Local* decisions are still polarized, in the sense that they are for one player or the other to make, but globally the two players act in a distributed, asynchronous fashion. At any given time, *both* may be active in different parts of the "game board". Moreover, these concurrent games are a strict generalization of the usual sequential games.

Remarkably, this generalization and apparent complication can be formalized in a simple and robust way, arguably *more* elegant and mathematically tractable than the current formalizations of sequential games. In fact, the key ideas of the formalization were present in Abramsky and Jagadeesan's "New Foundations for the Geometry of Interaction" [AJ92a]. What was missing in that work was the game-theoretic interpretation of the mathematics, which in turn suggests a more intuitive presentation using closure operators.

5 The mathematical model

As a convenient point of departure, we begin with *Blass games* [Bla92]. These have the form

$$\prod_{i\in I} A_i \quad \text{or} \quad \sum_{i\in I} A_i$$

where each A_i is (co)inductively a Blass game. The idea is that in $\prod_{i\in I} A_i$, Opponent starts by playing some $i \in I$, and play then proceeds as in A_i. $\sum_{i\in I} A_i$ is defined dually, with Player making the initial move $i \in I$. Thus these games are trees; at each stage, it is (exactly) one player's turn to move—the "global schedule". Play proceeds as a sequence of moves tracing a path through the tree. This commitment to a sequential format forces an interleaving interpretation of the multiplicatives (analogous to the Expansion Theorem of CCS), which leads inexorably to the failure of composition of strategies to be associative, as shown in [AJ92b].

We begin our development of concurrent games with the idea that game trees can be viewed as partial orders, in which $x \leq y$ means that the position y can be reached from the position x by playing some additional moves. This is a natural "information ordering" as in Domain theory [AJ94]. If we add "limit points" corresponding to the infinite branches in the game tree, we obtain a complete partial order D.

Viewed in these terms, the construction of sums and products of games as in Blass games can be described as *lifted sums* as far as the underlying domains of positions are concerned:

$$D_{\sum_{i\in I} A_i} = D_{\prod_{i\in I} A_i} = (\sum_{i\in I} D_{A_i})_\perp.$$

We shall represent strategies as functions on these domains of positions: $f : D \to D$, where $f(x)$ is the position obtained from x by extending it with whatever moves the strategy makes in that position. It is then immediate that

$f(x) \sqsupseteq x$. Moreover, those positions where f has no moves to make (e.g. because "it is not its turn") are exactly the *fixpoints* of f. In the usual way, we require computationally reasonable strategies to be monotonic and continuous. Finally, as a useful normalizing condition, we require strategies to be *idempotent*: $f^2 = f$.

To understand this, consider f applied to $f(x)$. The only moves made in $f(x)$ which were not already made in x are those made by f itself: $f(x)$ contains no more information supplied by the Opponent (i.e. the environment) than x did. Hence anything f decides to do at $f(x)$ it should have already been able to decide to do at x, and we require that $f(f(x)) = f(x)$. Of course, this allows several moves to be made in a block by a player. This possibility already exists in Blass games, e.g. Opponent must move twice initially in

$$(A \times B) \times (C \times D).$$

An important point is that strategies may not be well-defined at all positions. In general there are some positions that can never be reached by following that strategy. To mesh with the requirement that strategies are increasing functions, we adjoin a top element to the domain of positions D, writing this as $D^\top$. We represent f being undefined at x by $f(x) = \top$.

In summary, strategies (for either player) are represented as continuous closure operators on $D^\top$, which under modest assumptions on D (bounded completeness) is a complete lattice. We can completely specify a game as a structure (D, S, S^*), where D is the domain of positions, S is the set of legal strategies for Player, and S^* is the set of legal counter-strategies, i.e. strategies for Opponent.

This strictly generalizes Blass games: for such games, D is the domain of finite and infinite sequences under the prefix ordering corresponding to the paths through the game tree, and the conventions about who is to play are formalized by saying that for all $x \in D$, either $\sigma(x) = x$ for all $\sigma \in S$ (it is Opponent's turn to move), or $\tau(x) = x$ for all $\tau \in S^*$ (it is Player's turn). However, this is a very special case of our general setting; and we will overcome the problems with Blass games precisely by allowing situations in which *both* players can move.

To do this, we shall interpret the game boards for the multiplicatives differently to Blass: by a true concurrency rather than an interleaving representation. In our setting, this is simply a matter of defining

$$D_{A \otimes B} = D_{A \invamp B} = D_A \times D_B,$$

the cartesian product of domains.

How should the Linear Logic connectives be interpreted as acting on sets of strategies? We define $S_{A^\perp} = S^*_A$, $S^*_{A^\perp} = S_A$, which corresponds to interchanging the rôles of Player and Opponent. Clearly then $A^{\perp\perp} = A$. For products, we define

$$S_{\prod_{i \in I} A_i} = \{\langle \sigma_i | i \in I \rangle \mid \forall i \in I, \sigma_i \in S_{A_i}\}$$

where

$$\begin{array}{rcl}\langle \sigma_i | i \in I \rangle(\bot) & = & \bot \\ \langle \sigma_i | i \in I \rangle(\mathtt{in}_i(x)) & = & \mathtt{in}_i(\sigma_i(x)).\end{array}$$

This corresponds exactly to the idea that Player must first wait for Opponent to choose an $i \in I$, and then plays according to some strategy for A_i.

$$S^*_{\prod_{i \in I} A_i} = \{\mathtt{in}_i(\sigma) \mid i \in I, \sigma \in S^*_{A_i}\}$$

where

$$\begin{array}{rcl}\mathtt{in}_i(\sigma)(\bot) & = & \mathtt{in}_i(\sigma(\bot)) \\ \mathtt{in}_i(\sigma)(\mathtt{in}_i(x)) & = & \mathtt{in}_i(\sigma(x)) \\ \mathtt{in}_i(\sigma)(\mathtt{in}_j(x)) & = & \top, \ \ (i \neq j).\end{array}$$

Again, this corresponds to the idea that a strategy for Opponent will firstly choose some i, then play according to a strategy for Opponent in A_i. Note that the last case above covers "unreachable states" for $\mathtt{in}_i(\sigma)$. We can then define sums by De Morgan duality: $\sum_{i \in I} A_i = (\prod_{i \in I} A_i^{\bot})^{\bot}$.

For the strategies for tensor, we define

$$S_{A \otimes B} = \{\sigma \times \tau \mid \sigma \in S_A, \tau \in S_B\}$$

where $\sigma \times \tau(x, y) = (\sigma(x), \tau(y))$. (This is really smash product with respect to $\top$; if either $\sigma(x) = \top$ or $\tau(y) = \top$, then the result is $\top$.) This exactly captures the idea of informational independence between Player's actions in A and in B (cf. [AJ92b]). How Player moves in A depends only on the information available in A, and similarly for B.

In order to define the counter-strategies for the Tensor (and hence the strategies for Par and Linear implication, and eventually the morphisms in the category of concurrent games), we introduce the most important feature of our formalization: the elegant treatment it affords of composition of strategies. Suppose firstly that $\sigma \in S$ and $\tau \in S^*$ in a game (D, S, S^*). How do we play σ off against τ? We define

$$\begin{array}{rcl}\langle \sigma | \tau \rangle & = & \mathbf{Y}(\sigma \circ \tau) = \bigvee_{k \in \omega} (\sigma \circ \tau)^k(\bot) \\ & = & \bigvee_{k \in \omega} (\tau \circ \sigma)^k(\bot) = \mathbf{Y}(\tau \circ \sigma).\end{array}$$

The fact that these two least fixpoints coincide follows easily from the fact that σ and τ are continuous closures; in fact, this is a special case of the construction of the *join* of two closure operators. Thus $\langle \sigma | \tau \rangle \in D$ is the position we reach as a result of playing σ against τ. The equality of the two formulas above also shows that this is independent of all questions about "who starts".

Now given closure operators σ on $(D \times E)^{\top}$ and τ on $(E \times F)^{\top}$, we want to "compose" them to get a closure $\sigma; \tau$ on $(D \times F)^{\top}$. We define this as follows:

$$\sigma; \tau(x, z) = (\pi_1 \circ \sigma(x, y), \pi_2 \circ \tau(y, z))$$

where

$$y = \langle \pi_2 \circ \sigma(x, -) | \pi_1 \circ \tau(-, z) \rangle.$$

That is, given input in D and F, we play σ and τ off against each other in E relative to this input, and obtain their external response taking into account their interaction with each other.

In particular, if σ is a closure on $(D \times E)^\top$, it induces an "action" taking closures on $D^\top$ to closures on $E^\top$, $\alpha \mapsto \alpha; \sigma$, and a "coaction" taking closures on $E^\top$ back to closures on $D^\top$, $\beta \mapsto \sigma; \beta$. E.g. $\sigma; \beta(x) = \pi_1 \circ \sigma(x, y)$, where $y = \langle \pi_2 \circ \sigma(x, -) | \beta \rangle$.

Now we can define

$$\begin{aligned} S^*_{A \otimes B} = \{\sigma : (D_A \times D_B)^\top \mid \quad & \forall \alpha \in S_A.\, \alpha; \sigma \in S^*_B \\ \wedge \quad & \forall \beta \in S_B.\, \sigma; \beta \in S^*_A\}. \end{aligned}$$

Again, by De Morgan duality,

$$A \parr B = (A^\perp \otimes B^\perp)^\perp, \quad A \multimap B = A^\perp \parr B.$$

In particular,

$$\begin{aligned} S_{A \multimap B} = \{\sigma : (D_A \times D_B)^\top \mid \quad & \forall \alpha \in S_A.\, \alpha; \sigma \in S_B \\ \wedge \quad & \forall \beta \in S^*_B.\, \sigma; \beta \in S^*_A\}. \end{aligned}$$

This is a symmetric, "classical" version of the familiar logical relations condition: σ's action carries Player strategies on A to Player strategies on B, and its coaction carries counter-strategies on B to counter-strategies on A. We round out our account by defining the units, which all have the one-point poset as their underlying domain. For the tensor unit $\mathbf{1}$, $S_\mathbf{1} = \{(\perp \mapsto \perp)\}$, while $S^*_\mathbf{1} = \{(\perp \mapsto \perp), (\perp \mapsto \top)\}$. For the unit for Plus (i.e. the initial object) we have $S_\mathbf{0} = \emptyset$, $S^*_\mathbf{0} = \{(\perp \mapsto \top)\}$. The exponentials can also be defined in our model (the basic ideas are along the lines of [AJ92a]), but for lack of space we will refrain from doing so. The reader will really understand our model by checking that composition is associative, and seeing how the problems with Blass games simply do not arise in our setting.

6 A more concrete presentation

The foregoing mathematical development may well have seemed quite abstract, and hard to relate to one's intuitions. Let us start again, and give an essentially equivalent, but more concrete presentation.

Our notion of concurrent games can be thought of as played between two "teams" of players, distributed around the "game board", in an aynchronous fashion.

The basic action in a game is *making a move*, which is chosen from among several possible alternatives. The information content of the move is the *choice* or decision as to which move to play.

We fix a set C of "cells" as labels for the various choices or decisions that have to be made during the course of a play of the game, and a set V of "values", enumerating the possible choices. So the "event" $\{(c, v)\}$ means that alternative v was chosen at cell c. A state of the game board—a position of the game—is then represented by a partial function from cells to values

$$\{(c_1, v_1), \ldots, (c_k, v_k)\}$$

(so there is *no* information about the order in which decisions were taken).

We can take $C = V = \mathbb{N}$ for a "universal" choice of game board, into which any other (subject to countability) can be embedded. The positions are then partial functions on natural numbers, which under inclusion form one of the basic examples of a *domain* [AJ94]. Moreover, as play progresses, and more moves are made, the position *increases* in the natural information ordering:

$$\{(c_1, v_1), (c_2, v_2)\} \xrightarrow{(c_3, v_3)} \{(c_1, v_1), (c_2, v_2), (c_3, v_3)\}.$$

Call this universal game board $\mathbb{G}$.

A *strategy* is a function

$$f : \mathbb{G} \longrightarrow \mathbb{G}.$$

For the reasons already explained, this function should be monotonic and continuous, increasing ($x \subseteq f(x)$), and idempotent ($f(f(x)) = f(x)$).

What about unreachable positions? We need an "error element" $\top$ for situations when the strategy is applied to a position which could never have been reached if we had been following that strategy. To preserve the increasingness property, this error element must be top in the information ordering. So a strategy is a continuous closure operator on the complete lattice $\mathbb{G}^\top$ obtained by adjoining a top element to $\mathbb{G}$. The set of all such closure operators

$$S = \mathsf{Cl}(\mathbb{G}^\top)$$

is our universal space of strategies.

Various sub-classes of S are important. In particular, there are constraints on strategies which correspond to important computational properties. One of these is sequentiality, and hence represents our answer to the challenge we began with.

A closure operator $\sigma \in S$ has an *output function* $f : \mathbb{G} \to \mathbb{G}$ if

$$\forall x \in \mathsf{dom}(\sigma).\, \sigma(x) = f(x) \vee x$$

where

$$\mathsf{dom}(\sigma) = \{x \in \mathbb{G} \mid \sigma(x) \neq \top\}.$$

We recall the domain-theoretic conditions of *stability* and *sequentiality* [AC98]. These make sense because our underlying domain $\mathbb{G}$ is simple and concrete—it is in fact a "concrete domain" in the technical sense of Kahn and Plotkin

[KP78]. Indeed, because it is simply a (countable) product of flat domains (in fact, $\mathbb{G} = \mathbb{N}_{\perp}{}^{\omega}$), we can use Vuillemin's original definition of sequentiality [Vui73]: a function $f : \mathbb{G} \to \mathbb{G}$ is sequential if:

$$\forall c \in C. \forall x \in \mathbb{G}. f(x)(c)\uparrow \;\; \Rightarrow \;\; \forall y \supseteq x. f(y)(c)\uparrow$$

$$\vee \;\; (\exists c'. x(c')\uparrow \wedge \forall y \supseteq x. f(y)(c)\downarrow \;\Rightarrow\; y(c')\downarrow).$$

There says that there is some particular cell in the input which *must* get filled before we can fill the given output cell. Classically, this condition excludes the parallel-or function, Berry's "Gustave function", etc. [Vui73, KP78, Cur86].

We take the "constraint" or "healthiness condition" of having a sequential output function as picking out those strategies which are to be regarded as computing in a deterministic fashion, and hence sequentially implementable. In support of this contention, we have the fact that, when we make our universe of sequential strategies into a model for a sequential functional language such as PCF (the most elegant way of doing this uses realizability ideas [Abr99]), then it is fully abstract, and even universal if we restrict to effective strategies. Moreover, by Loader's results [Loa96], this is about as good a characterization of sequentiality as we are likely to get.

Let us give a first indication of how we can use our simple universal (type-free) space of strategies to recover the same structure we displayed in a typed version in the previous section.

We split the "address space" by fixing

$$C = C_1 \uplus C_2$$

(C_1, C_2 infinite). Note that

$$\mathbb{G} = V_{\perp}^{C} = V_{\perp}^{C_1+C_2} \cong V_{\perp}^{C_1} \times V_{\perp}^{C_2} \cong \mathbb{G} \times \mathbb{G}.$$

So we can regard any $\sigma \in S$ as

$$\sigma : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G} \times \mathbb{G}.$$

The most basic example of a strategy is the *asynchronous copy-cat*:

$$\mathtt{id}(x, y) = (x \vee y, x \vee y).$$

The corresponding output function is the twist map:

$$\mathtt{tw}(x, y) = (y, x)$$

since

$$(x \vee y, x \vee y) = (x, y) \vee (y, x).$$

7 The process model

This may still seem too abstract. We can also describe the model in operational terms with a little process calculus.

We define a syntax of terms by the BNF

$$P \ ::= \ c?x \to P \mid c!e \mid P \| Q \mid 0 \mid \top \mid (\nu c)P \mid X \mid \mathsf{rec}\, X.P \mid \ldots$$

We define a structural congruence on these terms in the style of the π-calculus [Mil99]. The main novel feature is that $\top$ acts as a zero, and we have

$$c!v_1 \parallel c!v_2 \equiv \top \quad (v_1 \neq v_2).$$

We can then define a reduction semantics, again in the style of the π-calculus, but with one key difference: we have the reduction

$$c!v \parallel (c?x \to P) \quad \longrightarrow \quad c!v \parallel P[v/x]$$

where in CCS or the π-calculus we would have the rule

$$c!v \parallel (c?x \to P) \quad \longrightarrow \quad P[v/x].$$

This property of *persistence* of outputs (which corresponds to the information increasingness in the domain-theoretic presentation) is more akin to concurrent constraint programming [Sar93].

We can connect this operational view very directly to our view of strategies as closure operators by defining a denotational semantics for terms of our process calculus as closure operators, and proving a correspondence between this denotational semantics and the operational semantics. We can then see the process calculus terms as providing a syntax for our semantic objects, and moreover making the computational behaviour of strategies explicit via the associated operational semantics.

We give a couple of clauses for this denotational semantics.

$$[\![c?x \to P]\!](a) = \begin{cases} a, & a(c)\uparrow \\ [\![P[v/x]]\!](a), & a(c) = v \end{cases}$$

$$[\![c!v]\!](a) = \begin{cases} a \cup \{(c, v)\} & \text{if consistent} \\ \top & \text{otherwise} \end{cases}$$

Parallel composition is interpreted as the join of closure operators; note the connection to the composition operators defined in Section 5.

$$[\![P \| Q]\!] = \bigvee_{k \in \omega} ([\![P]\!] \circ [\![Q]\!])^k = \bigvee_{k \in \omega} ([\![Q]\!] \circ [\![P]\!])^k.$$

Applied to any a, this gives the least common fixpoint of P and Q above a. Note that it doesn't matter who starts!

Using this calculus, we can construct the copy-cat strategy described previously as an abstract closure operator in more explicit terms as a process.

(We use the splitting of the address space introduced in the previous section. For each cell c, we think of it occuring once in the "left part" of the address space, C_1, as $\mathtt{l}(c)$, and once in the right part C_2, as $\mathtt{r}(c)$). Now we can define the "copy-cat process" as:

$$\mathtt{id} \stackrel{\Delta}{=} \|_{c \in C} \; (\mathtt{l}(c)?x \rightarrow \mathtt{r}(c)!x \parallel \mathtt{r}(c)?x \rightarrow \mathtt{l}(c)!x)$$

We can think of this process as a big team of players, one for each matched pair of cells. This process copies the contents of either element of the pair into the other as soon as it is filled. Of course, if the pair of cells have incompatible contents, then this will fail. However, if we play this copy-cat strategy against any other strategy following the type constraints of Linear Logic, then this kind of failure can never happen.

References

[AJ92a] S. Abramsky, R. Jagadeesan, New foundations for the geometry of interaction, *Information and Computation,* 111(1):53-119, 1994. Conference version appeared in LiCS '92.

[AJ92b] S. Abramsky and R. Jagadeesan. Games and Full Completeness for Multiplicative Linear Logic, *Journal of Symbolic Logic*, (1994), vol. 59, no. 2, 543–574. Conference version appeared in FSTTCS '92.

[AJ94] S. Abramsky and A. Jung. Domain Theory. *Handbook of Logic in Computer Science. Volume III: Semantic Structures*, pages 1–168. S. Abramsky, D. M. Gabbay and T. S. E. Maibaum, editors. Oxford University Press 1994.

[Abr97] S. Abramsky. Games in the Semantics of Programming Languages. *Proceedings of the 11th Amsterdam Colloquium.* P. Dekker, M. Stokhof and Y. Venema, eds. ILLC, University of Amsterdam, 1997, 1–6.

[AMcC99] S. Abramsky and G. McCusker. Game Semantics. *Computational Logic.* U. Berger and H. Schwichtenberg, eds. Springer-Verlag, 1999, 1–56.

[Abr99] S. Abramsky. Process Realizability. *Electronic Notes in Theoretical Computer Science.* Special issue for the Realizability Workshop, Trento, 1999.

[AM98] S. Abramsky and P.-A. Melliès. Concurrent games and full completeness. Draft paper, 37 pages, 1998.

[AM99] S. Abramsky and P.-A. Melliès. Concurrent games and full completeness. *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*, 431–44, 1999.

[AC98] R. Amadio, P.-L. Curien, *Domains and Lambda-calculi.* Cambridge University Press, 1998.

[BDER97] P. Baillot and V. Danos and T. Ehrhard and L. Regnier. Believe it or not, AJM's games is a Model of Classical Linear Logic. *Proceedings of LiCS '97*, pp. 68–75, 1997.

[BFSS88] E. S. Bainbridge and P. J. Freyd and A. Scedrov and P. J. Scott. Functorial Polymorphism. *Theoretical Computer Science* 70,1:35–64, 1988.

[Bla92] A. Blass, A game semantics for linear logic, *Annals of Pure and Applied Logic,* 56:183-220, North-Holland, 1992.

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

[Cur86] P.-L. Curien. Categorical combinators, sequential algorithms and functional programming. Pitman 1986. Revised edition, Birkhauser 1993.

[DJS97] V. Danos and J.-B. Joinet and H. Schellinx. A new deconstructive logic: Linear Logic. *Journal of Symbolic Logic,* 1997.

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

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

[Hoare99] C. A. R. Hoare. Personal communication, March 1999.

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

[KP78] G. Kahn and G. Plotkin. Concrete Domains. *Theoretical Computer Science*, 121:187–277, 1993. Appeared as TR IRIA-Laboria 336 in 1978.

[Loa94a] R. Loader. Linear Logic, Totality and Full Completeness. In *Proceedings of LiCS '94.*

[Loa94b] R. Loader, *Models of lambda calculi and linear logic: structural, equational and proof-theoretic characterisations*, PhD Thesis, Oxford, 1994.

[Loa96] R. Loader. Finitary PCF is not decidable. Submitted for publication, 1996.

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

[Mil99] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus.* Cambridge University Press, 1999.

[NPW81] M. Nielsen and G. Plotkin and G. Winskel. Petri nets, event structures and domains, part 1. *Theoretical Computer Science* 13:85–108, 1981.

[Sar93] V. Saraswat. *Concurrent Constraint Programming.* MIT Press, 1993.

[Vui73] J. Vuillemin. *Proof Techniques for Recursive programs.* Ph.D. thesis, Stanford University, 1973.