Retracing some paths in Process Algebra

Samson Abramsky
Laboratory for the Foundations of Computer Science
University of Edinburgh

1 Introduction

The very existence of the CONCUR conference bears witness to the fact that "concurrency theory" has developed into a subject unto itself, with substantially different emphases and techniques to those prominent elsewhere in the semantics of computation.

Whatever the past merits of this separate development, it seems timely to look for some convergence and unification. In addressing these issues, I have found it instructive to trace some of the received ideas in concurrency back to their origins in the early 1970's. In particular, I want to focus on a seminal paper by Robin Milner [Mil75][1], which led in a fairly direct line to his enormously influential work on CCS [Mil80, Mil89]. I will take (to the extreme) the liberty of of applying hindsight, and show how some different paths could have been taken, which, it can be argued, lead to a more unified approach to the semantics of computation, and moreover one which may be better suited to modelling today's concurrent, object-oriented languages, and the type systems and logics required to support such languages.

2 The semantic universe: transducers

Milner's starting point was the classical automata-theoretic notion of *transducers*, *i.e.* structures

$$(Q, X, Y, q_0, \delta)$$

where Q is a set of states, $q_0 \in Q$ the initial state, X the set of inputs, Y the set of outputs, and

$$\delta : Q \times X \rightharpoonup Y \times Q$$

[1]Similar ideas appeared independently in the work of Hans Bekić [Bek71].

is the transition function (here a partial function). If we supply a sequence of inputs $x_0, \ldots, x_k$ to such a transducer, we obtain the orbit

$$q_0 \xrightarrow{x_0} y_0, q_1 \xrightarrow{x_1} y_1, q_2 \xrightarrow{x_2} \cdots \xrightarrow{x_k} y_k, q_{k+1}$$

if $\delta(q_i, x_i) = y_i, q_{i+1}$, $0 \leq i \leq k$. This generalizes to non-deterministic transducers with transition function

$$\delta : Q \times X \longrightarrow \mathcal{P}(Y \times Q)$$

in an evident fashion.

The key idea in [Mil75] is to give a denotational semantics for concurrent programs as *processes*, which were taken to be extensional versions of transducers. There are two ingredients to this idea:

1. Instead of modelling programs by functions or relations, to model them by entities with more complex behaviours, taking account of the possible interactions between a program and its environment during the course of a computation.

   “The meaning of a program should express its history of access to resources which are not local to it.” [Mil75]

2. Instead of modelling concurrent programs by automata, with all the intensionality this entails, to look for a more extensional description of the *behaviours* of transducers.

To obtain this extensional view of transducers, consider the recursive definition

$$R = X \rightharpoonup Y \times R.$$

This defines a mathematical space of “resumptions” in which the states of transducers are “unfolded” into their observable behaviours. Milner solved equations such as this over a category of domains in [Mil75], but in fact it can be solved in a canonical fashion over **Set**—in modern terminology, the functor

$$T_{X,Y} : \mathbf{Set} \longrightarrow \mathbf{Set}$$

$$T_{X,Y}(S) = X \rightharpoonup Y \times S$$

has a final coalgebra $R \xrightarrow{\cong} T_{X,Y}(R)$. Indeed, Milner defined a notion $\sim$ of behavioural equivalence between transducers, and for any transducer

(Q, X, Y, q_0, δ) a map $h_\delta : Q \longrightarrow R$ which is in fact the final coalgebra homomorphism from the coalgebra

$$\hat{\delta} : Q \longrightarrow T_{X,Y}(Q)$$

to R (where $\hat{\delta}$ is the exponential transpose of δ), and proved that

$$(Q, X, Y, q_0, \delta) \sim (Q', X, Y, q'_0, \delta') \iff h_\delta(q_0) = h_{\delta'}(q'_0).$$

From a modern perspective, we can also make light of a technical problem which figured prominently in [Mil75], namely how to model non-determinism. Historically, this called forth Plotkin's work on powerdomains [Plo76], but for the specific application at hand, the equation

$$R = X \longrightarrow \mathcal{P}(Y \times R)$$

has a final coalgebra in the category of classes in Peter Aczel's non-well-founded set theory [Acz88], and if we are content to bound the cardinality of subsets by an inaccessible cardinable κ, then the equation

$$R = X \longrightarrow \mathcal{P}^{<\kappa}(Y \times R)$$

has a final coalgebra in **Set** [Bar93b]. Moreover, the equivalence induced by this model coincides with strong bisimulation [Acz88].

However, this is not central to our concerns here. Rather, we want to focus on three important choices in the path followed by Milner from this starting point:

- Type-free *vs.* typed
- Extrinsic *vs.* intrinsic interaction
- Names *vs.* information paths.

We want to examine the consequences of making different choices on these issues.

2.1 Typed vs. type-free

Rather than looking at a single type-free space of resumptions as above, and trying to invent some plausible operations on this space, we will focus instead on the *category* of resumptions, and try to identify the structure naturally present in this category.

The category $\mathcal{R}$ of resumptions (we will for simplicity confine ourselves to the deterministic resumptions) has as objects sets, and as morphisms

$$\mathcal{R}(X, Y) = X \rightharpoonup Y \times \mathcal{R}(X, Y)$$

i.e. the space of resumptions parameterized by the sets of "inputs" X and "outputs" Y. The composition of resumptions $f \in \mathcal{R}(X, Y)$ and $g \in \mathcal{R}(Y, Z)$ is defined (coinductively [Acz88]) by:

$$f; g(x) = \begin{cases} (z, f'; g') & f(x) = (y, f'),\ g(y) = (z, g') \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The identity resumption $\mathtt{id}_X \in \mathcal{R}(X, X)$ is defined by

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

We can picture this composition as sequential (or "series") composition of transducers.

We can define a monoidal structure on $\mathcal{R}$ by

$$X \otimes Y = X + Y \quad \text{(disjoint union of sets)}$$

and if $f \in \mathcal{R}(X, Y)$, $g \in \mathcal{R}(X', Y')$, $f \otimes g \in \mathcal{R}(X \otimes X', Y \otimes Y')$ is defined by:

$$f \otimes g(\mathtt{inl}(x)) = \begin{cases} (\mathtt{inl}(y), f' \otimes g), & f(x) = (y, f') \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$f \otimes g(\mathtt{inr}(x')) = \begin{cases} (\mathtt{inr}(y'), f \otimes g'), & g(x') = (y', g') \\ \text{undefined} & \text{otherwise.} \end{cases}$$

This is (asynchronous) parallel composition of transducers: at each stage, we respond to an input on the X "wire" according to f, with output appearing on the Y wire, and to an input on the X' wire according to g, with output appearing on the Y' wire.

The remaining definitions to make this into a symmetric monoidal structure on $\mathcal{R}$ are straightforward, and left to the reader. Note that the associativity and symmetry isomorphisms, like the identities, have just one state; they are "history-free".

Finally, there is a feedback operator: for each X, Y, U a function

$$\mathtt{Tr}^U_{X,Y} : \mathcal{R}(X \otimes U, Y \otimes U) \longrightarrow \mathcal{R}(X, Y)$$

defined by

$$\mathtt{Tr}^U_{X,Y}(f)(x) = \begin{cases} (y, f'), & \exists k.\, f(x) = (u_0, f_0), \\ & f_0(u_0) = (u_1, f_1), \\ & \vdots \\ & f_k(u_k) = (y, f') \\ \text{undefined} & \text{otherwise.} \end{cases}$$

One should picture a token entering at the X wire, circulating k times around the feedback loop at the U wire, and exiting at Y.

This feedback operator satisfies a number of algebraic properties (to simplify the statement of these properties, we elide associativity isomorphisms, *i.e.* we pretend that $\mathcal{R}$ is *strict* monoidal):

Naturality in X

$$\mathtt{Tr}^U_{X,Y}((g \otimes \mathtt{id}_U); f) = g; \mathtt{Tr}^U_{X',Y}(f)$$

where $f : X' \otimes U \longrightarrow Y \otimes U$, $g : X \longrightarrow X'$.

Naturality in Y

$$\mathtt{Tr}^U_{X,Y}(f; (g \otimes \mathtt{id}_U)) = \mathtt{Tr}^U_{X,Y'}(f); g$$

where $f : X \otimes U \longrightarrow Y' \otimes U$, $g : Y' \longrightarrow Y$.

Naturality in U

$$\mathtt{Tr}^U_{X,Y}(f; (\mathtt{id}_Y \otimes g)) = \mathtt{Tr}^{U'}_{X,Y}((\mathtt{id}_X \otimes g); f)$$

where $f : X \otimes U \longrightarrow Y \otimes U'$, $g : U' \longrightarrow U$.

Vanishing

$$\mathtt{Tr}^I_{X,Y}(f) = f$$

where $f : X \longrightarrow Y$, and

$$\mathtt{Tr}^{U \otimes V}_{X,Y}(f) = \mathtt{Tr}^U_{X,Y}(\mathtt{Tr}^V_{X \otimes U, Y \otimes U}(f))$$

where $f : X \otimes U \otimes V \longrightarrow Y \otimes U \otimes V$.

Superposing

$$\mathtt{Tr}^U_{X \otimes Z, Y \otimes W}((\mathtt{id}_X \otimes \mathtt{sym}_{Z,U}); (f \otimes g); (\mathtt{id}_Y \otimes \mathtt{sym}_{U,W})) = \mathtt{Tr}^U_{X,Y}(f) \otimes g$$

where $f : X \otimes U \longrightarrow Y \otimes U$, $g : Z \longrightarrow W$.

Yanking

$$\mathtt{Tr}^{X}_{X,X}(\mathtt{sym}_{X,X}) = \mathtt{id}_X.$$

This says that $\mathcal{R}$ is a *traced (symmetric) monoidal category* in the sense of [JSV95] (*cf.* also [Has96] for the symmetric and cartesian cases, and [BE93] for related axioms).

2.2 Intrinsic vs. extrinsic interaction: paths vs. names

Why this apparent digression into the structure of the category of resumptions? Our aim is to address the question of how to model *interaction between processes*, which is surely the key notion in concurrency theory, and arguably in the semantics of computation as a whole. Resumptions as they stand model a single process in terms of its potential interactions with its environment. To quote Robin Milner again:

> "A crucial feature is the ability to define the operation of *binding* together two processes (which may represent two cooperating programs, or a program and a memory, or a computer an an input/output device) to yield another process representing the composite of the two computing agents, with their mutual communications internalized." [Mil75]

The route Milner followed to define this binding was in terms of the use of "names" or "labels": in terms of resumptions, one modifies their defining equation to

$$R(X,Y) = X \rightharpoonup Y \times L \times R(X,Y)$$

where L is a set of labels, so that output is tagged with a label, which can then be used by some "routing combinator" to dispatch the output to its destination process. This led in a fairly direct line of descent to the action names α, β, γ of CCS [Mil80, Mil89], and the names of the π-calculus [MPW92] and action structures [MMP95]. Clearly a great deal has been achieved with this approach. Nevertheless, we wish to lodge some criticisms of it.

- interaction becomes extrinsic: we must add some additional structure, typically a "synchronization algebra" on the labels [Win83], which implicitly refers to some external agency for matching up labels and generating communication events, rather than finding the meaning of interaction in the structure we already have.

- interaction becomes ad hoc: because it is an "invented" additional structure, many possibilities arise, and it is hard to identify any as canonical.

- interaction becomes global: using names to match up communications implies some large space in which potential communications "swim", just as the use of references in imperative languages implies some global heap. Although the scope of names may be delimited, as in the π-calculus, the local character of particular interactions is not immediately apparent, and must be laboriously verified. This appears to account for many of the complications encountered in reasoning about concurrent object-oriented languages modelled in the π-calculus, as reported in [Jon93, Jon96].

We will now describe a construction which appears in [JSV95], and which can be seen as a general form of the "Geometry of Interaction" [Gir88], and also as a general but basic form of game semantics [Abr96b]. This construction applies to any traced monoidal category $\mathcal{C}$, *i.e.* to any calculus of boxes and wires closed under series and parallel composition and feedback, and builds a compact closed category $\mathcal{G}(\mathcal{C})$, into which $\mathcal{C}$ fully and faithfully embeds. (It is in fact the unit of a (bi)adjunction between the categories of traced monoidal and compact closed categories.) Its significance in the present context is that it gives a general way of introducing a symmetric notion of interaction which addresses the issues raised above:

- interaction is intrinsic: it is found from the basic idea that processes are modelled in terms of their interactions with their environment. Building in the distinction between "process" and "environment" at a fundamental level makes interaction inherent in the model, rather than something that needs to be added.

- interaction is modelled as composition in the category $\mathcal{G}(\mathcal{C})$. Thus interaction is aligned with the computation-as-cut-elimination paradigm, and hence a unification of concurrency with other work in denotational semantics, type theory, categorical logic etc. becomes possible. See [AGN96a, Abr93, Abr95b] for a detailed discussion of this point.

- interaction is local. The dynamics of composition traces out "information paths", which are closely related to the types of the processes which interact. There is no appeal to a global mechanism for matching names. As we will see, this is general enough to model λ-calculus, state and concurrency, but, we believe, carries much more structure than the use of names to mediate interactions.

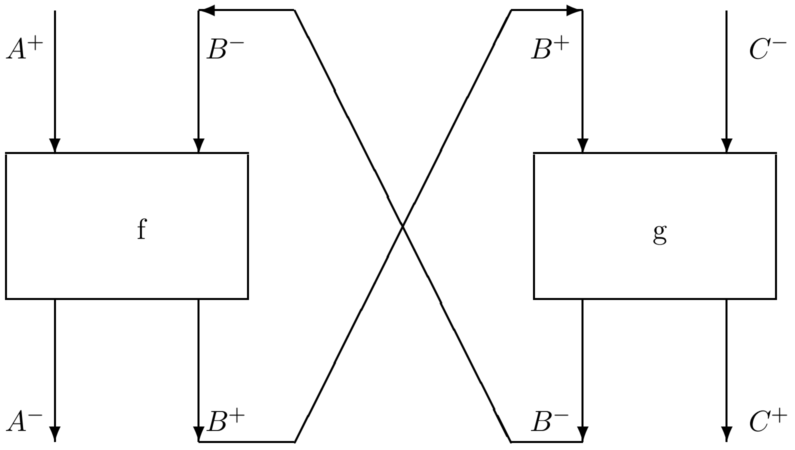
3 The $\mathcal{G}$ construction

Given a traced monoidal category $\mathcal{C}$, we define a new category $\mathcal{G}(\mathcal{C})$ as follows:

- The objects of $\mathcal{G}(\mathcal{C})$ are pairs (A^+, A^-) of objects of $\mathcal{C}$. The idea is that A^+ is the type of "moves by Player (the System)", while A^- is the type of "moves by Opponent (the Environment)".
- A morphism $f : (A^+, A^-) \longrightarrow (B^+, B^-)$ in $\mathcal{G}(\mathcal{C})$ is a morphism

$$f : A^+ \otimes B^- \longrightarrow A^- \otimes B^+$$

  in $\mathcal{C}$.
- Composition is defined by symmetric feedback (*cf.* [AJ94b, AJ94a]):



If $f : (A^+, A^-) \longrightarrow (B^+, B^-)$ and $g : (B^+, B^-) \longrightarrow (C^+, C^-)$ then $f; g : (A^+, A^-) \longrightarrow (C^+, C^-)$ is defined by

$$f; g = \mathtt{Tr}^{B^- \otimes B^+}_{A^+ \otimes C^-, A^- \otimes C^+}(\alpha; f \otimes g; \gamma)$$

where

$$\alpha : A^+ \otimes C^- \otimes B^- \otimes B^+ \xrightarrow{\cong} A^+ \otimes B^- \otimes B^+ \otimes C^-$$

and

$$\gamma : A^- \otimes B^+ \otimes B^- \otimes C^+ \xrightarrow{\cong} A^- \otimes C^+ \otimes B^- \otimes B^+$$

are the canonical isomorphisms defined using the symmetric monoidal structure. (Again, we have elided associativity isomorphisms.)

- The identities are given by the symmetry isomorphisms in $\mathcal{C}$:

$$\mathtt{id}_{(A^+,A^-)} = \mathtt{sym}_{A^+,A^-} : A^+ \otimes A^- \xrightarrow{\cong} A^- \otimes A^+.$$

There is an evident involutive duality on this category, given by

$$(A^+, A^-)^* = (A^-, A^+).$$

There is also a tensor structure, given by

$$(A^+, A^-) \otimes (B^+, B^-) = (A^+ \otimes B^+, A^- \otimes B^-).$$

$\mathcal{G}(\mathcal{C})$ is a compact-closed category [KL80], with internal homs given by

$$(A^+, A^-) \multimap (B^+, B^-) = (A^- \otimes B^+, A^+ \otimes B^-).$$

4 Examples

4.1 From resumptions to strategies

To interpret the category $\mathcal{G}(\mathcal{R})$, think of an object (X^+, X^-) as a rudimentary two-person game, in which X^+ is the set of moves for Player, and X^- the set of moves for Opponent. A resumption $f : X^- \longrightarrow X^+$ is then a *strategy* for Player. Note that we can represent such a strategy by its set of *plays*:

$$P(f) = \{x_1 y_1 \cdots x_k y_k \mid f(x_1) = (y_1, f_1), \ldots, f_{k-1}(x_k) = (y_k, f_k)\}.$$

One can then show that composition in $\mathcal{G}(\mathcal{R})$ is given by "parallel composition plus hiding" [Abr94, AJ94a, Abr96b]:

$$P(f; g) = \{s \upharpoonright X, Z \mid s \in P(f) || P(g)\}$$

$$S || T = \{s \in \mathcal{L}(X, Y, Z) \mid s \upharpoonright X, Y \in S \wedge s \upharpoonright Y, Z \in T\}$$

where $X = X^+ + X^-$, $Y = Y^+ + Y^-$, $Z = Z^+ + Z^-$, and

$$\mathcal{L}(S_1, S_2, S_3) = \{s \in (S_1 + S_2 + S_3)^* \mid s_i \in S_j \wedge s_{i+1} \in S_k \implies |j - k| \leq 1\}.$$

The identities are the "copycat" strategies as in [AJ94a, Abr96b]. We can then obtain the simple category of games described in [Abr96b] by applying a specification structure in the sense of [AGN96b] to $\mathcal{G}(\mathcal{R})$, in which the properties over (X^+, X^-) are the prefix-closed subsets of $(X^- X^+)^*$, *i.e.* the "safety properties" [AP93], which in this context are the game trees.

4.2 Some geometries of interaction

Suppose we begin with the simpler category **Pfn** of sets and partial functions (which is a lluf sub-category of $\mathcal{R}$). This is easily seen to be a sub-traced-monoidal category of $\mathcal{R}$, with tensor as disjoint union, and the trace given by a sum-of-paths formula (*cf.* [AM82]). That is, if

$$f : X + U \rightharpoonup Y + U$$

is a partial function, then

$$\mathtt{Tr}^{U}_{X,Y}(f) = \bigvee_{k\in\omega} f_k,$$

where $f_k(x)$ is defined and equal to y iff starting from x we perform exactly k iterations of the feedback loop around U before exiting at Y with result y:

$$f_k = \mathtt{inl}_{X,U}; (f; [0, \mathtt{inr}_{X,U}])^k; f; [\mathtt{id}_Y, 0]$$

where 0 is the everywhere undefined partial function. We can think of this sub-category of $\mathcal{R}$ as the "one-state resumptions", so that, applying the $\mathcal{G}$ construction to **Pfn** we get a category of history-free strategies [AJ94a].

As a minor variation, we could start with the category **PInj** of sets and partial injective maps. Then $\mathcal{G}(\mathbf{PInj})$ is essentially the original Geometry of Interaction construction of Girard, as explained in [AJ94a, AJM96]. In particular, the composition in $\mathcal{G}(\mathbf{PInj})$ corresponds exactly to the Execution Formula. This category can be lifted to the setting of Hilbert spaces by applying the free construction described in [Bar93a], which sends a set X to the Hilbert space $l_2(X)$ of square summable families $\{a_x \mid x \in X\}$.

As a final variation, we could start with **Rel**, the category of sets and relations. This yields a non-deterministic version of the Geometry of Interaction, which can be generalized via non-deterministic resumptions to a category of non-deterministic strategies. $\mathcal{G}(\mathbf{Rel})$ is the example mentioned at the end of [JSV95].

4.3 Stochastic interaction

As a more substantial variation of the above, consider the following category of *stochastic kernels* [Law62, Gir81]. Objects are structures $(X, \mathcal{M}(X))$, where $\mathcal{M}(X)$ is a σ-algebra of subsets of X. A morphism $f : X \longrightarrow Y$ is a function

$$f : X \times \mathcal{M}(Y) \longrightarrow [0, 1]$$

such that for each $x \in X$ $f(x, \cdot) : \mathcal{M}(Y) \longrightarrow [0,1]$ is a measure, and for each $M \in \mathcal{M}(Y)$, $f(\cdot, M) : X \longrightarrow [0,1]$ is a measurable function. One can think of stochastic kernels as "probabilistic transition functions". Note that we do not require that each $f(x, \cdot)$ is a probability measure, *i.e.* that $f(x, Y) = 1$, since we wish to allow for "partial" transition functions.

Composition is by integration: if $f : X \to Y$ and $g : Y \to Z$, then

$$f; g(x, M) = \int_Y g(\cdot, M) df(x, \cdot).$$

Identities are given by point measures:

$$\mathtt{id}_X(x, M) = \begin{cases} 1, & x \in M \\ 0, & x \notin M. \end{cases}$$

Tensor product is given by disjoint union; note that $\mathcal{M}(X + Y) \cong \mathcal{M}(X) \times \mathcal{M}(Y)$.

Feedback is given by a sum-over-paths formula. Given $f : X \otimes U \longrightarrow Y \otimes U$, and $x \in X$, we define for each $k \in \omega$ a measure μ_k on $\mathcal{M}(U)$ which gives the probability that we will end up in M starting from x after exactly k traversals of the feedback loop:

$$\mu_0(M) = f(\mathtt{inl}(x), (\varnothing, M))$$

$$\mu_{k+1}(M) = \int_U f(\mathtt{inr}(\cdot), (\varnothing, M)) d\mu_k.$$

The probability that we will end up in $M \in \mathcal{M}(Y)$ starting from x after exactly k iterations of the feedback loop is given by:

$$f_0(x, M) = f(\mathtt{inl}(x), (M, \varnothing))$$

$$f_{k+1}(x, M) = \int_U f(\mathtt{inr}(\cdot), (M, \varnothing)) d\mu_k.$$

Finally, the trace is defined by summing over all paths:

$$\mathtt{Tr}^U_{X,Y}(f)(x, M) = \Sigma_{k \in \omega} f_k(x, M).$$

4.4 From particles to waves: the "New Foundations" version of Geometry of Interaction

All the above models can be thought of as dynamical systems in which an information "token" or "particle" traces some path around a network. This particulate interpretation of diagrams of boxes and wires is supported by the "additive" (disjoint union) interpretation of the tensor. It is also possible to give an interpretation in which an information "wave" travels through the network; formally, this will be supported by a "multiplicative" (cartesian product) interpretation of the tensor.

Specifically, we can define a traced monoidal structure on the category **Cpo** of cpo's and continuous functions, in which the tensor is given by the cartesian product, and feedback by the least fixpoint operator: that is, if $f : D \times A \longrightarrow E \times A$, then

$$\mathtt{Tr}^{A}_{D,E}(f) = \lambda d : D.\, f(d, \mathbf{Y}(f(d, \cdot); \mathtt{snd})); \mathtt{fst}.$$

The category $\mathcal{G}(\mathbf{Cpo})$ is then exactly the category $\mathcal{GI}(\mathcal{C})$ described in [AJ94b].

A sub-category of this category will consist of dataflow networks, built up from objects which are domains of streams. The symmetric feedback operator giving the composition in $\mathcal{G}(\mathbf{Cpo})$ has been used in this context [SDW96, GS96], *inter alia* in developing assumption/commitment style proof rules for dataflow networks.

4.5 The continuous case?

One final "example" should be mentioned, although we have not as yet succeeded in working out the details. The operations of series and parallel composition and feedback are standard in continuous-time control systems, electronic circuits and analogue computation. In particular, feedback is interpreted by solving a differential equation. There should then presumably be a traced monoidal category $\mathcal{C}$ of manifolds and smooth maps, for which $\mathcal{G}(\mathcal{C})$ would give an "infinitesimal" model of interaction. Such a category might be relevant to the study of hybrid systems [PS95].

5 Consequences

We shall, very briefly, sketch some further developments from this point.

5.1 Correctness issues

We can associate correctness properties with the rudimentary types of $\mathcal{G}(\mathcal{C})$, in the setting of specification structures [AGN96b]. Types can then carry strong correctness information, and the type inference rule for composition

$$\frac{f : A \rightarrow B \quad g : B \rightarrow C}{f; g : A \rightarrow C}$$

becomes a compositional proof rule for process interaction. See [Abr93, Abr95b, AGN96a, AGN96b] for further discussion and applications.

We shall mention some particular cases for the examples described above.

Resumptions In this case, we can get the structure of games as safety properties, and of winning strategies as liveness properties, as described in [AJ94a, Abr96b]. In particular, the fact that winning strategies are closed under composition corresponds to a guarantee that there is no "infinite chattering" [Hoa85] in interaction.

Geometry of Interaction In this case, we can focus on nilpotency as a semantic analogue of normalization, as in [Gir88], or instead proceed as in the previous example, as in [AJ94a], where a Full Completeness Theorem for Multiplicative Linear Logic is obtained.

5.2 Modelling types and functions

The divide between concurrency theory and denotational semantics, type theory and categorical logic is bridged in our approach, since the categories we construct, or derivatives thereof, have the right structure to model typed, higher-order programming languages. The key point is that we are now modelling functions as processes, and function application as a particular form of process interaction, as advocated in [Mil92], but in a highly structured, syntax-free and compositional fashion.

Moreover, the quality of these process models of functional computation is high: the models based on games yielded the first syntax-independent constructions of fully abstract models for PCF [AJM96, HO96], and this has been followed by a number of further results [AM95, McC96b, McC96a]. The degree of mathematical structure in these models is also witnessed by the axiomatic treatment of full abstraction it has been possible to extract from them [Abr96a].

5.3 State and concurrency

It has also proved possible to give a game semantics for Idealized Algol [Abr95a], which is a clean integration of higher-order functional programming with imperative features and block structure [Rey81, Ten94]. Again, this has led to the first syntax-independent construction of a fully abstract model [AM96]. The treatment of local variables is process-based, following the line of [Mil75, Mil80, Red96]; but with the the right mathematical tools now available, a more definitive treatment can be given, as confirmed by the results on full abstraction.

This model of Idealized Algol extends smoothly to incorporate concurrency [Abr95a]. It remains to be seen how accurate the model of the concurrent language is, but the situation looks quite promising: moreover, Idealized Parallel Algol is rich enough to represent rather directly many of the features of today's concurrent object-oriented languages.

References

[Abr93] S. Abramsky. Interaction categories (extended abstract). In *Theory and Formal Methods '93*, Workshops in Computer Science, pages 57–70. Springer-Verlag, 1993.

[Abr94] S. Abramsky. Proofs as processes. *Theoretical Computer Science*, 135:5–9, 1994.

[Abr95a] S. Abramsky. A game semantics for Idealized Parallel Algol. Unpublished lecture, 1995.

[Abr95b] 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 1–15. Prentice Hall International, 1995.

[Abr96a] S. Abramsky. Axioms for full abstraction and full completeness. Submitted for publication, 1996.

[Abr96b] S. Abramsky. Semantics of interaction. In *Proceedings of 1995 CLiCS Summer School, Isaac Newton Institute.* Cambridge University Press, 1996. To appear.

[Acz88] P. Aczel. *Non-well-founded sets.* CSLI, 1988.

[AGN96a] S. Abramsky, S. Gay, and R. Nagarajan. Interaction categories and the foundations of typed concurrent programming. In *Deductive program design: Proceedings of the 1994 Marktoberdorf International Summer School.* Springer-Verlag, 1996. To appear.

[AGN96b] S. Abramsky, S. Gay, and R. Nagarajan. Specification structures and propositions-as-types for concurrency. In *Logics for Concurrency: Structure vs. Automata*, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[AJ94a] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59(2):543–574, 1994.

[AJ94b] S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. *Information and Computation*, 111(1):53–119, 1994. Conference version appeared in LiCS '92.

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

[AM82] M. A. Arbib and E. G. Manes. The pattern-of-calls expansion is the canonical fixpoint for recursive definitions. *Journal of the ACM*, 29(2):577–602, 1982.

[AM95] S. Abramsky and G. McCusker. Games and full abstraction for the lazy λ-calculus. In *Tenth Annual Symposium on Logic in Computer Science*, pages 234–243, 1995.

[AM96] S. Abramsky and G. McCusker. Full abstraction for Idealized Algol. To appear, 1996.

[AP93] M. Abadi and G. Plotkin. A logical view of composition and refinement. *Theoretical Computer Science*, 114(1):3–30, 1993.

[Bar93a] M. Barr. Algebraically compact functors. Technical report, 1993.

[Bar93b] M. Barr. Terminal coalgebras for endofunctors on sets. Technical Report, 1993.

[BE93] S. Bloom and Z. Esik. *Iteration Theories*. Springer-Verlag, 1993.

[Bek71] H. Bekić. Towards a mathematical theory of processes. Technical Report TR25.125, IBM Laboratory, Vienna, 1971.

[Gir81] M. Giry. A categorical approach to probability theory. In *Categorical Aspects of Topology and Analysis*, volume 915 of *Lecture Notes in Mathematics*. Springer-Verlag, 1981.

[Gir88] J.-Y. Girard. Geometry of interaction I: interpretation of System F. In R. Ferro, editor, *Logic Colloquium '88*, pages 221–260. North Holland, 1988.

[GS96] R. Grosu and K. Stølen. A model for mobile point-to-point dataflow networks without channel sharing. Technical report, 1996.

[Has96] M. Hasegawa. Traced computational models. Technical report, 1996.

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

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

[Jon93] C. B. Jones. Process-algebraic foundations for an object-based design notation. Technical Report UMCS-93-10-1, University of Manchester, 1993.

[Jon96] C. B. Jones. Some practical problems and their influence on semantics. In Hanne Riis Nielson, editor, *Programming Languages and Systems—ESOP '96*, volume 1058 of *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag, 1996.

[JSV95] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. Technical report, 1995.

[KL80] G. M. Kelly and M. Laplaza. Coherence for compact closed categories. *Journal of Pure and Applied Algebra*, 19:193–213, 1980.

[Law62] F. W. Lawvere. The category of probabilistic mappings. Unpublished manuscript, 1962.

[McC96a] G. McCusker. *Games and Full Abstraction for a functional metalanguage with recursive types.* PhD thesis, Imperial College, University of London, 1996. to appear.

[McC96b] G. McCusker. Games and full abstraction for FPC. In *International Symposium on Logic in Computer Science*, 1996.

[Mil75] R. Milner. Processes: a mathematical model of computing agents. In *Logic Colloquium '73*, pages 157–173. North Holland, 1975.

[Mil80] R. Milner. *A Calculus of Communicating Systems.* Springer-Verlag, 1980.

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

[Mil92] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–142, 1992.

[MMP95] A. Mifsud, R. Milner, and J. Power. Control structures. In *Tenth Annual Symposium on Logic in Computer Science*, pages 188–198, 1995.

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

[Plo76] G. Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–487, 1976.

[PS95] A. Pnueli and J. Sifakis, editors. *Special issue on hybrid systems*, 1995. Theoretical Computer Science vol. 138 no. 1.

[Red96] U. Reddy. Global state considered unncessary: an object-based semantics for Algol. *Lisp and Functional Programming*, 1996.

[Rey81] J. C. Reynolds. The essence of Algol. In J. W. de Bakker and J. C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. North Holland, 1981.

[SDW96] K. Stølen, F. Dederichs, and R. Weber. Assumption/commitment rules for networks of asynchronously communicating agents. *Formal Aspects of Computing*, 1996.

[Ten94] R. D. Tennent. Denotational semantics. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 169–322. Oxford University Press, 1994.

[Win83] G. Winskel. Synchronization trees. In *Automata, Languages and Programming: 10th International Colloquium*, pages 695–711. Springer-Verlag, 1983.