A game-theoretic, concurrent and fair model of the typed λ-calculus, with full recursion

Yiannis N. Moschovakis*

Department of Mathematics
University of California
Los Angeles, CA 90095-1555, USA

and Department of Mathematics
University of Athens
Panepistimioupolis, Athens, Greece

This paper, and the talk on which it is based, were strongly influenced by two, contradictory words of advice. First, there is Gian-Carlo Rota's eloquent injunction in [17] to "publish the same result often"; and so I will take some time to describe again and (I hope) motivate and explain better the *game-theoretic model of concurrency, with fair merge and full recursion* introduced in [7] and further studied in [9, 8, 10, 12]. Second, there is this young computer scientist friend of mine, who complaints about conferences in which "everyone presents a finished, polished paper on what they did the year before, so that the talks are stylized and do not lead to meaningful interaction among the participants"; and so I put off writing the paper until after the meeting, and I spent all my time up to it perfecting as best I could the new theorem I wanted to present. Still not quite what I would like to prove, this result adds products and function spaces to the constructions of [7, 9], which then yield a *concurrent model of the typed λ-calculus* which still accommodates fairness and full recursion. As it happened, game-theoretic semantics of higher-type languages were featured prominently in this conference, quite different from mine, to be sure, but, still, not entirely unrelated, and so my unconventional choice for structuring the talk and this paper made good sense in the end.

After a brief, somewhat loose, first formulation of the basic problem which motivated [7, 9] in the remainder of this introduction, I will review these two papers in Sections 1 and 2. About a third of this repeats material in [9] (the basic definitions), to keep this paper reasonably self-contained, while the rest comprises examples, explanations, commentary and some argumentation; it is best to read these sections in conjunction with Sections 1 – 4 and 6 – 8 of [9]. The new results on higher types are presented in Section 3, in a form which (I hope) will convey some part of their meaning even to those who will skim quickly (or even skip over) the first two sections.

* During the preparation of this paper, the author was partially supported by a Grant from the National Science Foundation.

A. The Basic Problem (roughly). Suppose L is a programming language interpreted on some domain[1] D, so that the denotation $[\![A]\!]$ of each *program* (closed term) A of L is some point in D, and each *n-ary program transformation* $A(v_1, \ldots, v_n)$ (open term with n free variables) is assigned some (countably) continuous function $f_A : D^n \to D$. If L^* is an extension of L by some *non-deterministic constructs*, for example *autonomous choice*, $A \vee B$, it is then natural (following Park [13]) to interpret the programs of L^* by non-empty subsets of D, members of

$$\mathcal{P} = \mathcal{P}(D) = \{x \subseteq D \mid x \neq \emptyset\}, \tag{1}$$

so that in particular $[\![A \vee B]\!] = [\![A]\!] \cup [\![B]\!]$. Set functions compose naturally, and so it is easy to find in $\mathcal{P}(D)$ denotations for programs which are explicitly defined from the givens, but how about recursive definitions? In other words, if we follow the classical, Scott-Strachey [18] fixpoint interpretation of recursion, how do we solve equations of the form

$$x = f(x) \qquad (f : \mathcal{P} \to \mathcal{P}),$$

and for what set functions $f : \mathcal{P} \to \mathcal{P}$ can we do this? The standard answer to this question is the theory of *powerdomains* [15, 19], but it does not solve the problem when we include the *fair merge* among the givens of L^*. What, precisely, this means and why I was interested in it, I will explain in Section 1.

1 Modeling interaction and concurrency with games

Consider the *Airlines Reservation System*, the classic example of a large, concurrent distributed system. There is a huge **state**, comprising (among other data) the flight schedules of hundreds of Airline companies; the locations of thousands of planes and their characteristics (company ownership, mechanical type and condition, location, etc.); millions of confirmed reservations and waiting lists, each with its own, complex profile; contractual agreements among Airline companies and directives issued by various boards which regulate the industry; and, of course, the weather. The state is so enormous and complex that it completely defies explicit description, and it is very rapidly (almost constantly) changing by the individual **acts** of a myriad **agents** who interact with it at separately (and randomly) chosen times—travel agents, aspiring travelers, schedule officers, pilots, Airline company directors, regulators, and (I suppose) nature, which controls the weather. We may assume that each of these agents (from his point of view) acts purposely and deterministically towards some specific goals, knowing only a small part of the state and unable to affect it very much; but the totality

[1] Throughout the paper, a "domain" is any countably complete partially ordered set (with a least element $\perp$), and a set $F \subseteq D$ is *countably closed* in the *Scott topology* on D if it is downward closed and contains the supremum of every countable, non-empty, directed set $X \subseteq F$. More structure is surely required for a robust "domain theory", but this is all I need here, and it is best to formulate the basic questions which concern me in the widest, possible context.

(and interrelation) of their actions induces an "evolution in time" of the system which appears to be non-deterministic, almost chaotic.

Can we make up a simple mathematical model which makes it possible to reason usefully about such a huge, amorphous state and the complex interactions of agents with it? This has been doubted, and so many attempts to model interactive systems avoid the concept of (global) "state" altogether in favor of private interaction among the agents, message passing, matching of paired acts, etc. Before we argue in favor of "state", let us make precise the simplest, most obvious attempt to model it mathematically.

1.1 State structures

A **state structure** is a quintuple

$$\mathcal{S} = (States, \iota, Acts, skip, exec), \tag{2}$$

where the following hold:

1. *States* is some set of *states* which contains the *initial state* ι.
2. *Acts* is some set of *atomic acts* which includes the *idle* act *skip*.
3. $exec : States \times Acts \to States$ is a binary operation of act execution, so that each act a induces a total transition function

$$s \mapsto sa = exec(s, a)$$

on the states. We assume that $s\ skip = s$, for every state s.

There may be acts, distinct from *skip* which never change the state, but are, nevertheless important, for example "*sound the alarm*". In the Airlines Reservation System the acts are supposed to be "book (if possible) a first-class seat on Delta 137, on August 24, 1997 for ...", "cancel Delta 137 on August 24, 1997", and the like.

A *history* is a finite sequence of acts $h = (a_1, \ldots, a_n)$, and it acts on states in the obvious way,

$$sh = (\cdots(sa_1)a_2)\cdots)a_n.$$

A state s' is *accessible* from a state s, if $s' = sh$ for some history h—i.e., if s' can be reached from s by a finite sequence of acts.

In defense of state. Does this mathematical abstraction make sense, and can it be useful, in view of the enormous complexity (even "fuzziness") of the state and the set of acts involved? Consider the somewhat analogous problem of modeling the behavior of systems of particles in classical mechanics. Here too, it is certainly impossible to specify explicitly the state, which theoretically records the position and momentum of some millions of particles, perhaps the molecules of a gas under some specified temperature and pressure distribution; and it is equally impossible to give a deterministic description of the "evolution in time" of the system. Still, the concept of "state" is not abandoned; it is assumed (as

a mathematical idealization, if you wish) that the system is in a specific state at each moment, and this assumption is used in the derivation of useful facts about the system—in this case laws which predict the evolution in time of some important, observable aspects of the state. In our case too, the usefulness of the model should come from how we choose to model *agents* and *how they interact with the state*, which should capture, help understand, and predict interesting, observable aspects of the system. We will do this with a game.

1.2 The game of interaction

With each state structure $\mathcal{S}$ as in (2), we associate the two-person *game of interaction* $\mathsf{G} = \mathsf{G}(\mathcal{S})$, in which Player II represents some *agent* and Player I represents *the world*, i.e., *all the other agents interacting with the state*. A *run* of G is played in stages, with I and II exchanging moves at the nth stage, so that they alter the *current state* g_n, initially set by $g_{-1} = \iota$. At stage n, I moves first some state s_n which must be accessible from g_{n-1}; II then responds with a pair (a_n, w_n) of an act a_n and an *indicator* $w_n = \partial$ or $w_n = 1$, and the next current stage is set, $g_n = s_n a_n$; if $w_n = 1$, the run ends, otherwise we proceed to the next stage $n+1$. We picture a run of G by the following diagram, which is read from left-to-right:

I:		s_0			s_1			s_2			s_3				$\cdots$
II:			(a_0, w_0)			(a_1, w_1)			(a_2, w_2)			(a_3, w_3)			
State:	ι	s_0		$s_0 a_0$	s_1		$s_1 a_1$	s_2		$s_2 a_2$	s_3		$s_3 a_3$		$\cdots$

1.3 Deterministic agents as partial strategies – behaviors

Games are useful because they make it possible to give and use succinct, intuitively clear definitions of complex objects whose rigorous definitions are cumbersome. For example, a (deterministic) *partial strategy* σ *for Player* II in the game of interaction G is some, specific method which (when it is defined) instructs him how to play—how to respond to arbitrary moves by I. Formally, it is a partial function

$$\sigma : \mathit{States}^* \rightharpoonup \mathit{Acts} \times \{\partial, 1\}$$

on finite sequences of states to pairs (a, w), which II can use to respond by $\sigma(s_0, \ldots, s_n)$ to I's successive moves $s_0, \ldots, s_n$; and it is *normalized* if it satisfies some natural conditions [9, 3.1] which insure that two members of

$$\mathsf{B} = \mathsf{B}(\mathcal{S}) = \text{the set of behaviors (normalized partial strategies)} \tag{3}$$

are equal (as partial functions) exactly when they induce the same *behavior* for II in all runs of the game $\mathsf{G}(\mathcal{S})$.

The set B of behaviors is a domain, under the natural partial ordering on partial functions, and *we use behaviors to model deterministic agents*. It is easy

to see how natural operations on agents can be defined by continuous functions on behaviors; *sequential execution*, for example, is defined ([9, 3.2]) by

$$\sigma\,;\tau = \text{play first by } \sigma\text{, and then (if } \sigma \text{ ends the game) by } \tau. \tag{4}$$

The role of behaviors in modeling deterministic interaction is explained in the first, two paragraphs of [9, 3.6] which should, ideally, be read at this point. In the remainder of this subsection I will attempt to flesh out the example which is only alluded to there.

Suppose Prof. M wants to book a flight from Los Angeles to Chicago. His first move is likely to be a call to some travel agency to get information: what flights are available, whether there is space on them, how much they cost, etc. Suppose there are three good flights, A, B and C, in the order of desirability in M's opinion. His second move is to request a reservation on A, which, however, is no longer available; so he calls again and requests a reservation on B, luckily still available, at which point M has finished his job. The run of the game G which represents this particular interaction of M with the Airlines Reservation System is

I:		s_0				s_1				s_2				s_3				s_4		
M:				(skip,∂)				(a,∂)				(skip,∂)				(b,∂)				$(\mathit{skip},1)$
State:	ι	s_0			s_0	s_1			s_1	s_2			s_2	s_3			s_3b	s_4			s_4

where a stands for the act of requesting a reservation on Flight A, and similarly for b and B. In his first move (skip,∂) M does not (really) act, he only consults the state to learn what is available; in his second move he requests a seat on Flight A; he then consults the state in his next move to find out that (unfortunately) no reservation for him is recorded in the current state s_2, which means that his previous attempt did not succeed; in his next move he succeeds, and so the new state s_3b records a seat for him on Flight B; and, finally, his last move executes no act, it is done only to consult the state and make sure that his previous act succeeded, at which time he ends the game.

It is important to note that, in this modeling, the agent M is not represented by the sequence of acts he executes but by the *strategy* he is following, which, in this simple case, amounts simply to consulting the system and then trying first for A, next for B and lastly for C; the same strategy would have instructed him to end the game with his third move, had he succeeded to get on A, or to prolong the game for (at least) two more moves had the second attempt also failed.

How come there was a seat on Flight A in state s_0 but none in state s_1? Perhaps because some other guy, Mr. R, got the last seat while M was trying to order his "choices": this accords with the rules of the game, by which s_1 is accessible from s_0, i.e., $s_1 = s_0a_1a_2\cdots a_n$ for some acts—one of them, apparently, a request by R for the last seat on A. *The states "moved" by Player* I (the "world") *are the results of all the other agents acting on the state in-between its interactions with* M.

Now consider the situation from the point of view of R, who has been trying desperately to put together a trip from Los Angeles to New York, with a stopover in Chicago. R is playing his own game of interaction with the Airlines Reservation System, and the move of his which frustrated M's first effort was, perhaps, his 22nd, in which he succeeded in changing state s'_{22} to $s'_{22}a'$ (which records his reservation), following a complex strategy, with much inquiring, requesting and getting turned down; and he is not through—he still does not have a good flight to New York. Clearly, M and R do not know each other, they are not in any sort of direct communication, and their separate interactions with the Airlines Reservation System are in no way synchronized. If we can speak of "the interaction between M and R", then the only significant elements of it are that they make the same, crucial request for a seat on Flight A, and that R made his request "before" M. This is recorded in the runs of their separate plays of the game G, specifically in the facts that $s'_{22}a'_{22}$ records a seat for R, while $s_2a_2 = s_2$ does not record a seat for M; and these two, different *perceptions* of the situation are all that this modeling captures of their conflict. It is hard to see what else there is to capture, or how to capture this much (as naturally) without a notion of state.

1.4 Players and fairness

Having resolved to model deterministic agents by behaviors, we next model non-deterministic agents by non-empty sets of behaviors, points in

$$\mathcal{P} = \mathcal{P}(\mathsf{B}(\mathcal{S})) = \{x \mid x \subseteq \mathsf{B}(\mathcal{S}), x \neq \emptyset\}.$$

I call these objects **players**, thinking of any $x \in \mathcal{P}$ as able to exhibit any one of his behaviors, i.e., to play the game of interaction by any one of the partial strategies he "possesses". The precise, relevant definitions are given in [9, 4.1-4.2,4.6], to which the remainder of this subsection is a commentary.

To begin with, is there a need for non-deterministic agents, especially after our modeling made it clear that a good part of what is often viewed as "non-deterministic behavior" is really the "visible part" (the run) of some agent playing *with a deterministic behavior* the game of interaction against the state? In other words, is it necessary to make room in the model for the indecisive Prof. N., who can follow either one of two equally desirable strategies and resolves to choose among them only the last minute, by flipping a coin? Perhaps not, but it is useful (in fact necessary) to be able to *view any set of agents as a single agent*, and this leads us to players.

Consider, for example, the time-sharing computing system in some University, where the word is going around that "the logicians are hogging the system". There are five logicians, and maybe they don't like it, but the other faculty lump them together into a "group", a single agent—and, indeed, sometimes they act like a unit, requesting additional resources for their experiments, etc. Now what "behavior" is this "logic agent" exhibiting? It must be some function of the (assumed deterministic) behaviors of the individual logicians, somehow combining

all five of them, since they all need to be served; and, I think, the best candidate for it is their (assuredly non-deterministic) "fair merge".

Adapting Park [13, 14] to this behavior model, let a *fair merger* (or *scheduler*) be any infinite, binary sequence $\mu : \mathbb{N} \to \{0, 1\}$ which changes value infinitely often, and for any two players x and y, let

$$parkmerge(x, y) = \{\mu[\sigma, \tau] \mid \sigma \in x, \tau \in y, \mu \text{ a fair merger}\}, \tag{5}$$

where $\mu[\sigma, \tau]$ is the strategy by which (roughly) II plays by σ or τ at the nth stage accordingly as $\mu(n) = 0$ or $\mu(n) = 1$.[2] It seems natural to assume that, as a single agent, the group of logicians can exhibit any one of the behaviors in the (similarly defined) quite complex set $parkmerge(\ell_1, \ldots, \ell_5)$, where each $\ell_i = \{\sigma_i\}$ is the *deterministic* (singleton) *player* modeling a logician—at least when the "operating system" (or whoever) is using simple, state-independent "fair schedulers" to make sure that every user is served. The point is that the logicians are not allocating resources to themselves, and they have no idea how the runs of their interactive programs are "interleaved" by the system; this interleaving, in fact, may be done partly by hardware, in a truly non-deterministic fashion.

The main, conceptual argument against using fair constructs in modeling concurrency is that nobody calls a "fair merger" in his program, but this misses the point: if I want to *prove* that my deterministic, interactive program will succeed in whatever goals I was trying to achieve, *I may assume* that it will be given as many chances as it needs to interact with the state (it will not be "starved"), but *I may not assume anything more*, as I do not know the scheduler; and it is at this level of "specification", if you wish, that fair constructs are natural and necessary. It is hard to say this more eloquently that Park [13], I have already paraphrased him once in [9, 1.1], and I will resist trying once more to improve on his argument.[3]

1.5 Controlling agent behavior – interaction types and locality

By the formal definitions we have given, agents (players) are all-powerful—they can see the entire state each time before they make a move and they can execute every act. Nobody can do this much, and we need one more, basic notion to come down to a realistic model of "actual" agents. This is a notion of *type* which controls not a kind of "value" as in usual type theory (there are no values in

[2] The precise definitions are given in [9, 4.6] and [10], where I reserved the more natural term *fairmerge* for the more complex operation which uses *state-dependent, fair mergers*.

[3] Another reason for the banishment of fairness from the theory of interactive computation is the general belief that it cannot be done, which (I think) is widely held because the usual models are built on (brute) *continuity assumptions* and do not make room for it. (This is certainly the case with the arguments in the classic Dijkstra [4], against which Park [13] is arguing.) The present theory and (especially) the higher-type constructions in Section 3 of this paper should go some way towards removing this prejudice.

this simple model), but the kind of interactive behavior in which an agent can participate.

An **interaction type** in a state structure $\mathcal{S}$ is any set of behaviors a which is (Scott-)*closed*; a behavior σ is *of type* a if $\sigma \in \mathsf{a}$; and a player x is *of type* a if $x \subseteq \mathsf{a}$ (see [9, 4.3–4.5], where these sets are just called *types*). Interaction types are closed under intersection, and so each player has a least type,

$$\mathit{type}(x) = \cap\{\mathsf{a} \mid \mathsf{a} \text{ is a type and } x \subseteq \mathsf{a}\}.$$

This is hard to compute in specific cases, but it is often easy to see (or to insure, if we are specifying x) that a certain player x is of some simple type a, and this can give us a great deal of information about x's behaviors.

For example, for any set E of acts, define the *effect type*

$$\mathsf{eff}(E) = \{\sigma \mid (\forall s_0, \ldots, s_n, a, w)[\sigma(s_0, \ldots, s_n) = (a, w) \Longrightarrow a \in E \cup \{\mathit{skip}\}]\},$$

so that a player of type $\mathsf{eff}(E)$ can only execute (real) acts in E.[4] There is a similar, somewhat more complex notion of *dependence type* $\mathsf{dep}(E)$ which determines *the part of the state* which any $\sigma \in \mathsf{dep}(E)$ *can see*—basically, the part which has been built up from the initial state by acts in E, see [9, 4.5]. It is easy to define more complex types, for example such that if $\sigma \in \mathsf{a}$, then σ cannot execute an act b unless some other act c has already been executed twice—and then σ must execute d just after he executes b, etc.

In the example of the Airlines Reservation System, the effect types of Prof. M and Mr. R (and even the indecisive Prof. N.) are those determined by "reservation acts", and their dependence types code the part of the state which includes scheduled flights and recorded reservations. For a simpler and more important example, consider the following.

An **integer variable** over a state structure $\mathcal{S}$ is a pair

$$V = (\mathit{write}_V, \mathit{read}_V),$$

where $\mathit{write}_V : \mathbb{N} \to \mathit{Acts}$ is a function assigning a distinct act to each integer, $\mathit{read}_V : \mathit{States} \rightharpoonup \mathbb{N}$ is a partial function, and *for every state s, every integer n and every history h which does not include any $\mathit{write}_V(m)$ acts*,

$$\mathit{read}_V(s\,\mathit{write}_V(n)h) = n;$$

in short, if someone "writes" n to V, then any "read" after that will recover n, as long as no subsequent "write to V" act has been executed. We let

$$\mathit{Acts}_V = \{\mathit{write}_V(n) \mid n \in \mathbb{N}\}$$

be the set of write-acts of V, and we call V **local** to two players x and y *relative to a set of players* $\mathcal{P}_0$, if for every $z \in \mathcal{P}_0$ other than x and y,

$$\mathit{type}(z) \subseteq \mathsf{eff}(\mathit{Acts} \setminus \mathit{Acts}_V) \cap \mathsf{dep}(\mathit{Acts} \setminus \mathit{Acts}_V),$$

[4] The act *skip* is not automatically included in every effect type in [9, 4.4], to give a finer modeling, which is not, however, very useful.

i.e., if no agent in $\mathcal{P}_0$ other than x or y can either write to V or read the integer value stored in V.

Similar definitions can be given for *stacks*, *buffers*, *private channels* between agents, etc. These are toy examples, to be sure, and much more sophisticated definitions must be given for real systems, but, I hope that they suggest how, in principle, this could be done on this modeling of interaction.

1.6 Review of the model

It is proposed that a concurrent, interactive system can be usefully modeled by a state structure $\mathcal{S}$ and a set $\mathcal{P}_\mathcal{S}$ of players in $\mathcal{P}(\mathsf{B}(\mathcal{S}))$, the non-empty sets of behaviors over $\mathcal{S}$. The members of $\mathcal{P}_\mathcal{S}$ represent the agents who are interacting with $\mathcal{S}$ (and indirectly with each other), each of them (separately and concurrently) playing the game $\mathsf{G}(\mathcal{S})$ with one of his behaviors $\sigma \in x$ against all the others. Any group of players can be combined into a single player by a *fairmerge* operation, suitably defined by analogy with (5). Each player x has a (least) interaction type, $\mathit{type}(x)$, and we can infer useful facts about the behaviors of x from the knowledge that $\mathit{type}(x) \subseteq \mathsf{a}$, for any type a.

Finally, the "time evolution" of the system can proceed along any sequence of states $s_0, s_1, s_2, \ldots$, where

$$s_0 = \sigma(\iota), \quad s_{n+1} = \sigma(s_0, \ldots, s_n),$$

and σ is any behavior in $\mathit{fairmerge}\{x \mid x \in \mathcal{P}_\mathcal{S}\}$, *the fair merge of all the agents.*

2 Recursion and intensionality

If agents (or **processes**)[5] are modeled by *players* over some state structure $\mathcal{S}$, how do we represent *process transformations*, and how can we define natural operations of *composition* and *recursive definition* on these transformations? As it happens, this is primarily a mathematical problem, which does not depend on our taking the posets of behaviors $\mathsf{B}(\mathcal{S})$ as basic domains, and so we will formulate it abstractly—keeping the case $D = \mathsf{B}(\mathcal{S})$ in mind as a rich source of examples.

So, for this section, fix some domain D, and let $\mathcal{P} = \mathcal{P}(D)$ be the set of *players over* D as in (1). The aim is to give a brief review of the results in [9, 10, 12] and to set the stage for the new results in the next section.

2.1 Implementable player functions

According to Park (and others), an (abstract) "implementation" of a function $f : \mathcal{P} \to \mathcal{P}$ is any continuous $F : D \to D$ such that

$$d \in x \Longrightarrow F(d) \in f(x) \qquad (x \in \mathcal{P}),$$

[5] The two terms "agent" and "process" have been given many precise definitions, reflecting the many theories in concurrency modeling. I use them synonymously and only informally, in comments, resorting to the rigorous term "player" in the precise formulation of definitions and results.

and f is "implementable" if there exists a ("full") set $\mathcal{F}$ of implementations which determines all its values, i.e.,

$$f(x) = \{F(d) \mid d \in x, F \in \mathcal{F}\} \qquad (x \in \mathcal{P}).$$

This is a natural but technically imperfect notion, not closed under substitution, e.g., it does not cover the unary "execute twice" operation

$$t(x) = x\,;x = \{\sigma\,;\tau \mid \sigma, \tau \in x\},$$

which is fully implemented by the single, *binary* implementation

$$F(\sigma, \tau) = \sigma\,;\tau.$$

It is not hard to see that, to get a robust notion, we need not only n-ary implementations, for every n, but *infinitary* ones, and so we arrive at the following, precise definitions [9, 7.1], [12, Section 3].

A (unary) **polyfunction** on D is any continuous function

$$F : D^{\mathbb{N}} \to D \tag{6}$$

(where $D^{\mathbb{N}}$ is the set of infinite sequences from D), and it is an (abstract) *implementation* of a function $f : \mathcal{P} \to \mathcal{P}$ if

$$X : \mathbb{N} \to x \Longrightarrow F(X) \in f(x) \qquad (x \in \mathcal{P}); \tag{7}$$

we call $f : \mathcal{P} \to \mathcal{P}$ **implementable** if all its values are computed by some (full) set of implementations $\mathcal{F}$, i.e.,

$$f(x) = \{F(X) \mid X : \mathbb{N} \to x, F \in \mathcal{F}\} \qquad (x, y \in \mathcal{P}). \tag{8}$$

The definition extends naturally to n-ary implementable functions $f : \mathcal{P}(D)^n \to \mathcal{P}(D)$, using n-ary polyfunctions $F : (D^{\mathbb{N}})^n \to D$ for implementations.

2.2 The problem of concurrent recursion

The implementable functions (of all arities) on $\mathcal{P}(D)$ give a rich class of process transformations, which includes union $x \cup y$, and also sequential execution $x\,;y$ and the *parkmerge* (5) (along with all reasonable, strict merge operations) when $D = \mathsf{B}(\mathcal{S})$. It is obviously closed under *composition*, since (for example, easily)

$$f(g(x)) = \{F(\lambda(i)G_i(X)) \mid F \in \mathcal{F}, G_i \in \mathcal{G}\}$$

when $\mathcal{F}$ and $\mathcal{G}$ implement fully f and g respectively. There is a problem with recursion, however, not only *how to define it* on the implementable functions, but also *how to justify any proposed definition*, since it will no longer be the familiar, least-fixed-point recursion of domain theory. The idea is to look for a definition of recursion which (at the least) *satisfies the same equational laws as least-fixed-point recursion*, and so we turn to a consideration of these laws next.

2.3 The standard recursive identities

For each set τ of formal function symbols (the *signature*), each with an assigned non-negative arity, the terms of the *formal* (equational) *language of recursive equations*[6] $\mathrm{FL}\mu_0(\tau)$ are defined inductively by the clauses:

$$E :\equiv x \mid f(E_1, \ldots, E_n) \mid \mu x[x = E]$$

where x is any variable and f is any n-ary function symbol. The idea is to interpret $f(E_1, \ldots, E_n)$ by application, as usual, and to understand the last *recursion construct* as providing a "canonical" fixed point of the function of x defined by E. In the **standard models** of $\mathrm{FL}\mu_0$, the variables range over some fixed domain D; the function symbols are interpreted by continuous functions on D; and $\mu x[x = f(x)]$ is the least fixed point of f. A term identity $E = F$ is **standard** if *it is valid in every standard model*, e.g., the *fixpoint rule*

$$f(\mu x[x = f(x, y)], y) = \mu x[x = f(x, y)].$$

The standard $\mathrm{FL}\mu_0$ identities are very well understood; they form a decidable class, and they can be simply axiomatized,[7] which makes it easy to determine whether they hold of any proposed interpretation of the recursion construct. It is natural to insist that a precise definition of concurrent recursion should satisfy them, and, of course, it should also yield the expected fixed points in simple, specific examples—but then there is no such definition, because of the following, basic result of Whitney:

B. The Whitney obstruction. *If D has an infinite, increasing sequence converging to a maximal element, then it is not possible to interpret the function symbols of* $\mathrm{FL}\mu_0$ *by arbitrary implementable functions on $\mathcal{P}(D)$, and define $\mu x[x = f(x)]$ so that* (with normal application) *all standard identities hold, and in addition:*

(1) *If $f(x) = \{F(d) \mid d \in x\}$ with a continuous $F : D \to D$, then* $\mu x[x = f(x)] = \{\mu d[d = F(d)]\}$.
(2) $\mu x[x = x \cup c] \subseteq c \cup \{\perp\}$, *for every* $c \in \mathcal{P}(D)$.
(3) $\mu x[x = f(x) \cup g(x)] \supseteq \mu x[x = f(x)]$.

The first of these conditions says that $\mu x[x = f(x)]$) extends faithfully to $\mathcal{P}(D)$ the "least-fixpoint" operation on D, and the last two express simple, weak features of our understanding of a "canonical fixpoint operation"—Whitney calls them $\cup$-*normality*.

[6] This language is equivalent in expressive power with FLR_0 (called $\mathcal{L}$ in [9, 2.2]), which takes *mutual recursion* as a primitive and is used and studied extensively in [12, 5].

[7] The decidability of the class of standard identities is an old result, probably due to Courcelle, Kahn and Vuillemin [1], and an explicit axiomatization was probably first given (in categorical form) in Bloom-Esik [2]. See [5] for a careful study of them and additional references, and also [11], for an extension of the decidability and axiomatization results to the full, Formal Language of Recursion FLR of [6] which allows *functional recursion*.

Results of this type had been known for some time, e.g., the so-called "Brock-Ackerman anomalies" in [3]. Most involved some specific, seemingly natural interpretation of non-deterministic recursion, which was then shown to have some peculiar property—typically that it did not satisfy some standard identity or that some minimal, natural fairness condition was not satisfied, although this language was not used. The Whitney obstruction, however, is quite definitive, and it appears to kill any possibility of developing a natural theory of computation which allows both full recursion and fairness (used almost imperceptibly in the proof of B, to apply (2) with a player c which is not closed).

And yet, the model in [7, 9] claimed to do just what the Whitney obstruction prohibits (and much more), so what is the catch?

2.4 Implemented (intensional) player functions

The idea is to replace the (usual, extensional) "implementable functions" on $\mathcal{P}(D)$ by the intensional "implemented functions", which are *identified* (as computation procedures) *with specific sets of implementations*. As always with intensional notions, some care must be taken to define "identity" correctly, and so the precise definition is a bit technical; I will reproduce it here for the sake of completeness, but it will remain somewhat opaque without a careful reading of [9, Section 7].[8]

For each $X : \mathbb{N} \to D$ and each function $\pi : \mathbb{N} \to \mathbb{N}$, let

$$X^\pi(t) = X(\pi(t)), \tag{9}$$

and for any two (unary, for simplicity) polyfunctions F and G, let

$$F \lesssim G \iff (\exists \pi : \mathbb{N} \to \mathbb{N})(\forall X : \mathbb{N} \to D)[F(X) = G(X^\pi)]; \tag{10}$$

if $F \lesssim G$, we say that F is **reducible** to G.

A **unary, implemented player function** (**ipf**) on a domain D,

$$f : \mathcal{P}(D) \rightsquigarrow \mathcal{P}(D),$$

is any (non-empty) set f of unary polyfunctions on D which is closed under reducibility, and it determines (computes) the implementable function $\overline{f}$,

$$\overline{f}(x) = \{F(X) \mid X : \mathbb{N} \to x, F \in f\}.$$

We call the members of f its (natural, given) *implementations*, and, as a rule, they do not exhaust the set of implementations of $\overline{f}$—and so we often have $\overline{f} = \overline{g}$ with $f \neq g$.[9] Implemented player functions of n arguments are defined similarly.

[8] This is the notion from the "coarse" modeling of [9], not the finer one in [12]. I have not been able to establish the results in the next section for the finer model.

[9] On the flat domain $\mathbb{N}_\perp = \mathbb{N} \cup \{\perp\}$, for example, the polyfunction

$$G(X) = \text{if } (X(0) = 0) \text{ then } X(1) \text{ else } X(1)$$

is a peculiar implementation of the identity function, which is not reducible to the natural implementation I in (11).

Each set $\mathcal{G}$ of polyfunctions **generates** an ipf, namely

$$[\mathcal{G}] = \{F \mid F \lesssim G, \text{ for some } G \in \mathcal{G}\}.$$

The **implemented identity** on $\mathcal{P}(D)$ is the ipf $\imath = [\{I\}]$ generated by the single polyfunction

$$I(X) = X(0), \tag{11}$$

and its natural implementations are (easily) all polyfunctions of the form

$$I_k(X) = X(k) \qquad (k \in \mathbb{N}).$$

The **composition**

$$f(x) = g(h_1(x), h_2(x))$$

of given ipfs g, h_1 and h_2 (as an example), is generated by all polyfunctions of the form

$$F(X) = G(\lambda(i)H_{1,i}(X), \lambda(i)H_{2,i}(X)), \tag{12}$$

where G, $H_{1,i}$ and $H_{2,i}$ are, respectively, natural implementations of g, h_1 and h_2. It is easy to verify that this set is, in fact, closed under reducibility, and that

$$\overline{f}(x) = \overline{g}(\overline{h}_1(x), \overline{h}_2(x)).$$

I will not attempt to repeat here the crucial definition of **ipf recursion**, which requires both some technical preparation and several examples, to make sense; it is given in [9, Section 8], and again in [20, 12], for a finer model. Some care is also needed to make precise the notion of *intensional model* for FLμ_0, and this, too, is done in the same papers, although some will prefer the categorical version in Bloom-Esik [2]. When all this is said and done correctly, we have the following, basic result from [9, 12]:

C. Main Old Theorem. *For each, countably complete poset D, the set $\mathcal{P}(D)$ of players with the implemented player functions, ipf composition and ipf recursion, form an intensional model of* FLμ_0 *which validates all the standard identities, and also Whitney's conditions* (1) – (3).

It is important to add that in the natural examples, on specific domains, ipf recursion yields the natural fixed points, those which we would expect from implementation considerations, cf., [9, 7.8].

3 Multidomains

The standard example of a multidomain, and the only one we have constructed up to this point, is the triple

$$\mathbf{\Pi}(D) = (\mathcal{P}(D), D, \Sigma_0), \tag{13}$$

where D is any domain and Σ_0 is the semigroup $\mathbb{N}^{\mathbb{N}}$ of number-theoretic functions, with composition, acting on $D^{\mathbb{N}}$ by

$$X \mapsto X^{\pi} = \lambda(t)X(\pi(t)). \tag{14}$$

This singling-out of Σ_0 (why not just speak of $\mathcal{P}(D)$, as we have until now?) suggests what is involved in generalizing the notion: to define products and function spaces so that we get an (intensional) model of the typed λ-calculus starting with the $\mathbf{\Pi}(D)$'s, we need to isolate the properties of Σ_0 which are used in the proofs of the Main Old Theorem and make that "structure" part of the definition of multidomain; the more complex multidomains we will construct will utilize a correspondingly more complex structure. With the definition of structure at hand, the formulation of the main result in the next subsection can be understood independently of its proof, on which I will comment briefly in Subsection 3.2.

An **action semigroup** on a domain D is a semigroup Σ, together with a continuous mapping $X \mapsto X^{\pi}$ $(X : \mathbb{N} \to D)$ for each $\pi \in \Sigma$, so that the following hold:

1. $\Big(\lambda(t)\bot\Big)^{\pi} = \lambda(t)\bot$, for every $\pi \in \Sigma$.
2. The **trivial** action semigroup Σ_0 is embedded in Σ, with its standard action in (14).
3. For all $X : \mathbb{N} \to D$ and $\pi, \rho \in \Sigma$, $(X^{\pi})^{\rho} = X^{\pi\rho}$.
4. For each sequence $\pi_0, \pi_1, \ldots$ of objects in Σ, there is a single $\pi \in \Sigma$, such that
$$X^{\pi}(t) = X^{\pi_t}(0) \qquad (X : \mathbb{N} \to D, t \in \mathbb{N}).$$

This last is a sort of "completeness property", trivially satisfied by Σ_0 with $\pi(t) = \pi_t(0)$.

A player $x \in \mathcal{P}(D)$ is **invariant** under Σ, if, for every $X : \mathbb{N} \to x$ and every $\pi \in \Sigma$, $X^{\pi} : \mathbb{N} \to x$. (Every player is invariant under Σ_0.)

3.1 The cartesian closed category of multidomains

In reading the theorem, notice that Parts (1) – (5) are true if we replace "multidomain" by "domain" throughout, and they express the simplest facts of domain theory; and so these basic results extend to the richer structures of the category $\mathbb{M}$, in which we can model concurrency and fairness, by Part (6).

D. Main New Theorem. *There is a category* $\mathbb{M}$ *of* **multidomains**, *with additional structure and properties as in* **Parts (1) - (6)**.

Part (1) The objects. *Each object of* $\mathbb{M}$ *is a triple* $\mathbf{\Pi} = (\Pi, D, \Sigma)$, *where* D *is a domain,* Σ *is an action semigroup on* D, *and*

$$\Pi \subseteq \mathcal{P}(D) = \{x \mid x \subseteq D, x \neq \emptyset\}$$

is a set of Σ-**invariant players** *on* D*, with* $\{\bot\} \in \mathit{\Pi}$*, satisfying a certain completeness condition.* We call $\mathbf{\Pi}$ a *multidomain over* the base domain D.

Each domain D *is represented in* $\mathbb{M}$ *by the deterministic multidomain of singletons*

$$\mathbf{D} = (\{\{d\} \mid d \in D\}, D, \Sigma_0),$$

over itself and with the trivial action semigroup Σ_0*. In particular, when* $I = \{\bot\}$*,* $\mathbf{I}$ *is a terminal object in* $\mathbb{M}$*.*

Part (2) The morphisms. *Each morphism* $f : \mathbf{\Pi}_1 \rightsquigarrow \mathbf{\Pi}_2$ *of* $\mathbb{M}$ *projects to* (or *implements*) *a set mapping* $\overline{f} : \mathit{\Pi}_1 \to \mathit{\Pi}_2$*, so that composition is respected,* $\overline{gf} = \overline{g} \circ \overline{f}$*; the identity* $\imath : \mathbf{\Pi} \rightsquigarrow \mathbf{\Pi}$ *implements the identity function,* $\overline{\imath}(x) = x$*. For the deterministic multidomains, the projection operation is a bijection of* $\mathrm{Hom}(\mathbf{D}_1, \mathbf{D}_2)$ *with the domain* $\mathrm{Cont}(D_1 \to D_2)$ *of continuous functions on* D_1 *to* D_2*.*

The morphisms of $\mathbb{M}$ are called *implemented player functions*, and their projections *implementable player functions.*

Part (3) Products. *For each indexed family* $\{\mathbf{\Pi}_i\}_{i \in I}$ *of multidomains, there is a multidomain product*

$$\mathbf{\Pi}^* = \mathrm{Prod}_{i \in I} \mathbf{\Pi}_i = (\mathit{\Pi}^*, \mathrm{Prod}_{i \in I} D_i, \Sigma^*)$$

over the product of the base domains D_i*, such that*:

(a) $\mathit{\Pi}^* = \{\mathrm{Prod}_{i \in I} x_i \mid (\forall i \in I)[x_i \in \mathit{\Pi}_i]\}$, i.e., the players in the product $\mathbf{\Pi}^*$ are the cartesian products of the players in the factors.

(b) *For each* $j \in I$*, there is a projection morphism* $p_j : \mathbf{\Pi}^* \rightsquigarrow \mathbf{\Pi}_j$*, which implements the projection function* $\overline{p}_j(x) = x_j$*.*

(c) *For each family* $\{f_i\}_{i \in I}$ *of morphisms,* $f_i : \mathbf{\Pi} \rightsquigarrow \mathbf{\Pi}_i$*, there is a unique morphism*

$$f = \langle f_i \rangle_{i \in I} : \mathbf{\Pi} \rightsquigarrow \mathbf{\Pi}^*,$$

such that, for each $i \in I$*,* $p_i f = f_i$*.*

In the case of two factors, we write $\mathbf{\Pi}_1 \times \mathbf{\Pi}_2$, $\langle f_1, f_2 \rangle$, etc.

Part (4) Function spaces. *For any two multidomains* $\mathbf{\Pi}_1 = (\mathit{\Pi}_1, D_1, \Sigma_1)$ *and* $\mathbf{\Pi}_2 = (\mathit{\Pi}_2, D_2, \Sigma_2)$*, there is a function multidomain*

$$(\mathbf{\Pi}_1 \Rightarrow \mathbf{\Pi}_2) = (\mathit{\Pi}^*, D^*, \Sigma^*),$$

so that the following hold:

(a) $D^* = \mathrm{Cont}({D_1}^{\mathbb{N}} \to D_2)$ *is the domain of continuous functions from* ${D_1}^{\mathbb{N}}$ *to* D_2*.*

(b) $\mathit{\Pi}^* = \mathrm{Hom}(\mathbf{\Pi}_1, \mathbf{\Pi}_2)$. (This makes sense, because every morphism $f : \mathbf{\Pi}_1 \rightsquigarrow \mathbf{\Pi}_2$ is, literally, a subset of D^*.)

(c) *There is morphism* $ap : \mathbf{\Pi}^* \times \mathbf{\Pi}_1 \rightsquigarrow \mathbf{\Pi}_2$ *which "implements" the application function, in the sense that*

$$\overline{ap}(f, x) = \overline{f}(x) \qquad (f \in \mathit{\Pi}^*, x \in \mathit{\Pi}_1).$$

(d) *For each* $\varphi : \mathbf{\Pi}_1 \times \mathbf{\Pi}_2 \rightsquigarrow \mathbf{\Pi}$*, there is a morphism* $\varphi^* : \mathbf{\Pi}_1 \rightsquigarrow (\mathbf{\Pi}_2 \Rightarrow \mathbf{\Pi})$ *such that*

$$ap\langle \varphi^* p_1, p_2 \rangle = \varphi; \tag{15}$$

in particular,

$$\overline{\overline{\varphi^*}(x)}(y) = \overline{\varphi}(x, y) \qquad (x \in \mathit{\Pi}_1, y \in \mathit{\Pi}_2).$$

It is possible to introduce a more logician-friendly "intensional" terminology and notation, with variables, in which the basic identity (15) takes the form

$$\Big(\lambda(y)\varphi(x, y)\Big)(y) = \varphi(x, y) \quad \text{(intensionally, for } x \in \mathbf{\Pi}_1, y \in \mathbf{\Pi}_2); \tag{16}$$

it is not feasible to do this here, and, in any case, those familiar with category theory would rather see (15). The next, main new result of the paper is also stated in categorical terms, with the (vague, absent the precise definitions) intensional version added on as a remark:

(e) *With the product operation in* **Part (3)** *and this function space operation, the category* $\mathbb{M}$ *is cartesian closed*, i.e., it is an intensional model of the typed λ-calculus.

Part (5) Recursion. *For each multidomain* $\mathbf{\Pi} = (\mathit{\Pi}, D, \mathit{\Sigma})$*, there is an intensional model of the language* FLμ_0 (of Subsection 2.3)[10] *in which the variables range over* $\mathit{\Pi}$*; the function symbols are interpreted by arbitrary morphisms* $f : \mathbf{\Pi}^n \rightsquigarrow \mathbf{\Pi}$*; term substitution is defined using morphism multiplication and the projection morphisms of* $\mathbb{M}$*; and all the standard identities hold.*

Part (6) The full multidomain. *For each domain* D*, the triple* $\mathbf{\Pi}(D)$ *in* (13) *is a multidomain over* D*, with the trivial action semigroup and with the following, completeness property*: *for every set* $\mathcal{F}$ *of continuous functions on* D *to* D*, there is some* $f : \mathbf{\Pi}(D) \rightsquigarrow \mathbf{\Pi}(D)$ *which implements the set function*

$$\overline{f}(x) = \{F(d) \mid d \in x, F \in \mathcal{F}\}.$$

Much more is true of the full multidomains, as detailed in Sections 1 and 2.

[10] Bloom and Esik [2] formulate this notion in categorical language, but it is quite complex and (perhaps) not generally known; in most papers on the typed λ-calculus, e.g., [16], it is only assumed that fixed points exist, which is a weaker claim.

3.2 Some remarks on the proof

The key object which needs to be understood for the proof is the structure $(\mathcal{P}(D), D, \Sigma)$, with an arbitrary domain D and action semigroup Σ, which is not exactly a multidomain (because not all its members are invariant under Σ), but near enough. An *implemented player function*

$$f : (\mathcal{P}(D_1), D_1, \Sigma_1) \rightsquigarrow (\mathcal{P}(D_2), D_2, \Sigma_2)$$

is a set f of unary polyfunctions on D_1 to D_2, as in (6) (but from one domain to another), closed under reducibility, as in (10), with π in Σ_1, and also satisfying the following "completeness" condition which involves Σ_2: *if $F_0, F_1, \ldots$ is any sequence of polyfunctions in f and $\rho \in \Sigma_2$, then the polyfunction*

$$F(X) = \Big(\lambda(i)F_i(X)\Big)^{\rho}(0)$$

is also in f. It is not difficult to extend the theory of ipf composition and recursion to these more general, intensional objects.

A **multiposet** is any triple

$$\mathbf{\Pi} = (\mathit{\Pi}, D, \Sigma),$$

where $\mathit{\Pi} \subseteq \mathcal{P}(D)$ is a set of Σ-invariant players, and an ipf

$$f : \mathbf{\Pi}_1 \rightsquigarrow \mathbf{\Pi}_2$$

is as above, with the additional stipulations that $\overline{f} : \mathit{\Pi}_1 \to \mathit{\Pi}_2$ (as a set function), and

$$[F \in f \;\&\; (\forall X : \mathbb{N} \to x \in \mathit{\Pi}_1)[F(X) = G(X)]] \Longrightarrow G \in f.$$

A multiposet $\mathbf{\Pi} = (\mathit{\Pi}, D, \Sigma)$ is **recursively closed** if every system of ipf equations on $\mathbf{\Pi}$ to $\mathbf{\Pi}$ has its solutions in $\mathit{\Pi}$; and, finally, a **multidomain** is any multiposet $\mathbf{\Pi}$ which is **fully complete**, i.e., such that every function space $(\mathbf{\Lambda} \Rightarrow \mathbf{\Pi})$ is recursively closed. These are the objects and the morphisms of the category $\mathbb{M}$. The most difficult part of the proof is the construction of the action semigroup for the function space—and it is primarily for this part (and, less significantly, for the product construction) that we need to introduce the additional complexity of these structures. It is not possible to discuss this argument here.

3.3 Conclusions and further work

I think the results of this section support the proposition that the theory of non-deterministic, interactive computation can be developed pretty much like that of deterministic computation, in a setting broad enough to accommodate both fairness and full recursion; not conclusively, to be sure, but they make a start. Among the many problems left open here, I would single out the following five as most obvious.

(1) The problem of completeness. The notion of "full completeness" I have used for the proof is too strong, and seems to correspond to (full) *directed completeness* for posets, rather than *countable completeness*. Now, all interesting countably complete posets are, in fact, directed complete, but the analog of this does not seem to be true for multiposets. In particular, I cannot prove fully complete (and put in $\mathbb{M}$) the (trivially structured) multiposets

$$c\mathcal{A}(\mathcal{D}) = \{F[\mathbb{C}] \mid F : \mathbb{C} \to D \text{ is continuous}\} \quad (\mathbb{C} = \{0,1\}^{\mathbb{N}}),$$
$$\mathcal{A}(\mathcal{D}) = \{F[\mathcal{N}] \mid \mathcal{F} : \mathcal{N} \to \mathcal{D} \text{ is continuous}\} \quad (\mathcal{N} = \mathbb{N}^{\mathbb{N}})$$

of **compact analytic** and **analytic** subsets of some D, which are especially important: $c\mathcal{A}(\mathcal{D})$ is (essentially) the Plotkin powerdomain [15] on D (for profinite D) by the main result in [12], and $\mathcal{A}(\mathcal{D})$ (for suitable D) is the simplest multiposet in which Park's fairness can be modeled. (These multidomains are carefully analyzed in [8, 20].) All the "relevant" recursive equations can be solved in these structures, they are (in some sense) "sufficiently complete", and we can make this precise and work from there, but not in a very elegant way. So the problem is: *Are the multiposets $c\mathcal{A}(\mathcal{D})$ and $\mathcal{A}(\mathcal{D})$ fully complete, and if not* (which is the more likely)*, is there a good theory which explains their different properties of "partial completeness"?*

(2) *Can we solve "multidomain equations", much in the way that we solve domain equations?*

(3) *Is there a way to define the "full multidomain" $\mathbf{\Pi}(\mathbf{\Pi}_1)$ over an arbitrary multidomain $\mathbf{\Pi}_1$, so that the category is closed under "power"?* (We now have $\mathbf{\Pi}(\mathbf{\Pi}_1)$ only when $\mathbf{\Pi}_1 = \mathbf{D}$ is, essentially, a domain.)

(4) *Can we prove that ipf recursion satisfies not just the* $\mathrm{FL}\mu_0$ *identities, but all the identities in the language of the typed λ-calculus with fixed points which are valid under the standard, domain interpretations?* (The problem of axiomatization of this class of identities is open, and its solution would solve some open, difficult complexity problems, but this question can be approached in other ways.)

(5) *Is it possible to extend to the higher types the finer model of ipf recursion in* [12]*?*

These are all technical problems, and I don't think that they will necessarily require radically new methods of proof for their solution.

References

1. G. Kahn B. Courcelle and J. Vuillemin. Algorithmes d'equivalence et de reduction a des expressions minimales dans une classe d'equations recursives simples. In J. Loeckx, editor, *Automata, Languages, and Programming, 2nd Colloquium*, volume 14 of *Lecture Notes in Computer Science*. Springer Verlag, 1974.
2. Stephen L. Bloom and Zoltan Ésik. *Iteration theories: the equational logic of iterative processes*. EATCS Monographs in Theoretical Computer Science. Springer-Verlag, Berlin, 1993.

3. J. D. Brock and W. B. Ackerman. Scenarios: a model of non-determinate computation. In J. Diaz and I. Ramos, editors, *Formalization of programming concepts*, volume 107 of *Lecture Notes in Computer Science*, pages 252–259. Springer Verlag, 1981.
4. E. W. Dijkstra. *A discipline of programming*. Prentice-Hall, 1976.
5. A. J. C. Hurkens, Monica McArthur, Yiannis N. Moschovakis, Lawrence Moss, and Glen T. Whitney. The logic of recursive equations. To appear in the Journal of Symbolic Logic.
6. Yiannis N. Moschovakis. The formal language of recursion. *The Journal of Symbolic Logic*, 54:1216–1252, 1989.
7. Yiannis N. Moschovakis. A game-theoretic modeling of concurrency, Extended abstract. In *Proceedings of the fourth annual symposium on Logic in Computer Science*, pages 154–183. IEEE Computer Society Press, 1989.
8. Yiannis N. Moschovakis. Computable processes, Extended abstract. In *Proceedings of the 1990 POPL meeting in San Francisco*. Association for Computing Machinery, 1990.
9. Yiannis N. Moschovakis. A model of concurrency with fair merge and full recursion. *Information and Computation*, 93:114–171, 1991.
10. Yiannis N. Moschovakis. Computable concurrent processes. *Theoretical Computer Science*, 139:243–273, 1995.
11. Yiannis N. Moschovakis. The logic of functional recursion. In M. L. Dalla Chiara et. al, editor, *Logic and scientific method*, pages 179–207. Kluwer Academic Publishers, 1997.
12. Yiannis N. Moschovakis and Glen T. Whitney. Powerdomains, powerstructures and fairness. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic*, number 933 in Lecture Notes in Computer Science, pages 382–396, Berlin, 1995. Springer-Verlag.
13. David Park. On the semantics of fair parallelism. In *Proceedings of the Copenhagen Winter School*, volume 104 of *Lecture Notes in Computer Science*, pages 504–526. Springer Verlag, 1980.
14. David Park. The "fairness" problem and nondeterministic computing networks. In *Foundations of Computer Science IV*, pages 33–162, Amsterdam, 1983. Mathematisch Centrum.
15. G. D. Plotkin. A powerdomain construction. *SIAM Journal of Computation*, 5:452–487, 1976.
16. G. D. Plotkin. LCF as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.
17. Gian-Carlo Rota. Ten lessons I wish I had been taught. *Notices of the American Mathematical Society*, 44:22–25, 1997.
18. D. S. Scott and C. Strachey. Towards a mathematical semantics for computer languages. In J. Fox, editor, *Proceedings of the Symposium on computers and automata*, pages 19–46, New York, 1971. Polytechnic Institute of Brooklyn Press.
19. M. B. Smyth. Modeling concurrency with partial orders. *Journal of Computing System Science*, 16:23–36, 1978.
20. Glen T. Whitney. *Recursion structures for non-determinism and concurrency*. PhD thesis, UCLA, 1994.