Semantics of Nondeterminism
Paul Blain Levy, University of Birmingham
September 10, 2004

1 Track Record

Paul Levy has held the position of Lecturer in the School of Computer Science, University of Birmingham, since 2002. Prior to this, he carried out research at Queen Mary, University of London (as a PhD student) and in Boston and Paris (as a postdoctoral research associate).

Much of his work has used denotational semantics and category theory to find and exploit connections between disparate areas of research in theoretical computer science, including various structures for computational effects [1, 11]. His discovery of the “call-by-push-value” (CBPV) language [5, 3], underlying previous call-by-value and call-by-name languages, led to fine-grained accounts of domains, continuations, game semantics, possible worlds and nondeterminism. It has proved useful both for analyzing existing semantics and for developing new ones, such as the model for general references in [7].

These applications of call-by-push-value were described in his wide-ranging PhD thesis [6]. In particular, he described the utility of CBPV for analyzing game semantics, one of the themes of this proposal. The two CBPV operations of forcing a thunk and returning a value correspond directly to the two kinds of move in game semantics, “question” and “answer”.

A radical simplification of the algebra [8, 2], using adjunction models, led to a book [9] published in “Semantic Structures in Computation” in 2004. The book describes CBPV semantics for nondeterminism using may-testing equivalence, and reveals the adjunction structure in Hyland and Ong’s well-bracketed game semantics, both relevant to this proposal.

He is currently working on some of the major open problems in the semantics of nondeterminism, such as the modelling of infinite trace equivalence [10], bisimilarity and ambiguous choice. He was recently awarded a New Lecturer grant by the Nuffield Foundation for this work.

Additional Expertise

The School of Computer Science at the University of Birmingham contains leading researchers in computational effects (Uday Reddy, Hayo Thielecke), domain theory (Martín Escardó, Achim Jung), categorical semantics (Eike Ritter, Steve Vickers), game semantics (Dan Ghica1), model checking (Dan Ghica, Marta Kwiatkowska) and temporal logic (Mark Ryan). Several of these (Escardó, Ghica, Jung, Kwiatkowska, Vickers) have worked specifically on semantics of nondeterminism, and the proposed research is already benefiting from their feedback.


The School is part of the ESPRIT Working Group entitled Applied Semantics II, which provides funding for collaboration across Europe, and an annual workshop for discussion of emerging research.

References


1 arriving in January 2001


2 Proposed Research

2.1 Background

Among the most powerful tools available for reasoning about programs - and even about entire programming languages - is denotational semantics. This is a way of describing the meaning of a programming language by associating to each program a denotation: some kind of mathematical object, such as a function, a relation or a strategy for a game. The denotation of a program is defined in terms of the denotation of its constituent parts, so we can use such a semantics to reason about programs in a modular and compositional way.

When we try to reason about a complex system, especially one that uses concurrent processing, we are usually faced with the situation that its behaviour depends on all kinds of complicating factors: the precise form of the scheduler, the behaviour of other processes, even details of the chip. To make the reasoning tractable, we want to ignore all this low-level detail, and so we idealize the system as a nondeterministic program, i.e. a program that has a range of possible behaviours and can make choices during execution.

The simplest form of nondeterminism is called erratic: for example $M \sqcap N$ (in CSP notation), which chooses a boolean and then executes $M$ or $N$ accordingly. More sophisticated is McCarthy's fair or ambiguous choice [33]: the expression $M \&\& N$ evaluates both $M$ and $N$, on a fair scheduler, and then returns whatever answer it gets first. Thus $\text{diverge} \sqcap (\text{return 3})$ might diverge, whereas $\text{diverge} \&\& \text{amb} (\text{return 3})$ cannot.

In a perfect denotational semantics, 2 programs would have the same denotation precisely when they are "equivalent". But when should programs be considered equivalent? This is a key question in semantics, and nondeterminism causes a proliferation of answers.

Besides the large amount of work on operational semantics and logics for nondeterminism, especially process calculi, there has also been much denotational research. We summarize some prominent strands.

Domains To adapt domain theory, in which recursion is interpreted as a least fixpoint, to nondeterministic languages, researchers developed various powerdomain constructions. The Hoare powerdomain represents may-testing assertions such as

$$\text{MyProg may insult the customer}$$  \hspace{1cm} (1)

that we generally wish to rule out (safety properties). The Smyth powerdomain represents must-testing assertions such as

$$\text{MyProg must greet the customer}$$  \hspace{1cm} (2)

that we generally wish to ensure (liveness properties). The Plotkin powerdomain models safety and liveness properties together. These constructions, and variants for countable nondeterminism, have been enormously influential, impacting, for example, on the FDR model checker for CSP.

Intensional semantics Metric space and presheaf semantics achieve great versatility at the cost of intensionality. Metric space semantics [1] counts the number of internal unwindings in a computation, whilst presheaf [20] and multi-set [12] semantics count the number of possible choices leading to the same result. Indeed, for nondeterministic dataflow, this is provably unavoidable [24].

Games The game semantics of [27] has been adapted to model nondeterminism, for may-testing and must-testing [22].

Bisimulation As well as may-testing and must-testing, we can consider branching-time properties such as

$$\text{MyProg may insult the customer and then be in a state where it will not insult again and may apologize.}$$  \hspace{1cm} (3)

which can be formalized in Hennessy-Milner logic [23]. Programs satisfying the same such properties are said to be bisimilar (or lower bisimilar, as divergence is ignored), and there are numerous variants of this used in the study of different calculi. A denotational model for bisimilarity is given in [14], using the Plotkin powerdomain. Bisimulation has also been represented in presheaf semantics using surjective open maps [29].

Concurrency Other kinds of models, such as event structures, Petri nets, pomsets, Mazurkiewicz traces and asynchronous transition systems have been used to model the conjunction of nondeterminism and concurrency. But, though these will be borne in mind, the main focus of this project is on nondeterminism alone, which, I hope to make clear, is quite hard enough.

2.2 Aims and Objectives

This project has two aims: to analyze the existing semantics in the light of my previous research on call-by-push-value and computational effects, and, more speculatively, to find new models that do not have certain limitations of the existing ones. Here is a summary of the present objectives, to be modified in the light of results.

1. Type theory and affineness
   (a) to formulate a type theory that refines both call-by-push-value and Affine HOPLA
   (b) to give lower set and presheaf semantics that decomposes existing models
   (c) to introduce input-request handling into this sequential language, in a way that generalizes exception handling.

2. Algebraic structure
(a) to give a categorical semantics for this type theory, of which the above-mentioned models would be instances
(b) to give continuation and stack semantics for input-request handling
(c) to explore the connections with linear continuation-passing transforms, and its models, that appear to exist.

3. Game semantics
   (a) to characterize storage-free nondeterministic strategies
   (b) to give game semantics for infinite traces
   (c) to apply this to reasoning about conditional liveness properties, related to Roscoe’s SBD model
   (d) to give game semantics for lower bisimilarity and convex bisimilarity.

4. Bisimilarity
   (a) to characterize when a set of formulas in Hennessy-Milner logic contains precisely those formulas satisfied by a program
   (b) to examine definability of functions
   (c) to identify which fixpoint of selfmaps are computed by recursion
   (d) to extend these to higher-order types
   (e) to adapt Howe’s method for proving congruence of bisimilarity to infinitary languages
   (f) based on all these studies, to give a CBPV model for bisimilarity
   (g) to adapt this work to convex bisimilarity

5. Must testing
   (a) to extend Broy’s semantics of ambiguity from first-order to CBPV
   (b) using this semantics, to attempt to prove/disprove the context lemma in this setting.

2.3 Analysis of Existing Semantics

Computational Effects

A computational effect is an imperative language feature such as input and output, storage, cell generation, divergence, nondeterminism, probabilistic choice, exceptions and control effects. A fruitful line of research has explored commonalities between these effects.

The proposal in [34] to use strong monads to organize the semantics of effects has been tremendously influential, including as special cases both the powerset and the 3 powerdomain constructions for nondeterminism. A structure for call-by-value languages that, in some respects, offers more flexibility is the Freyd category [11]. More recently, enriched Lawvere theories have been used, providing operators and equations to generate a strong monad of countable rank [40]—including bounded powerset and all 3 powerdomains—and providing techniques for modelling combinations of effects [28, 1].

Call-By-Push-Value

In [5], analysis of semantics of effects brought to light a kind of semantic machine code called call-by-push-value (CBPV). Based on a thoroughgoing separation of values and computations, this language provides fine-grained primitives from which both call-by-value and call-by-name are built. These primitives are astonishingly ubiquitous, turning up in denotational models using domains, algebras, continuations, possible worlds and games, as well as operational and machine semantics.

In CBPV, a value of type $UB$ is a thunk of a computation of type $B$, while a computation of type $FA$ returns a value of type $A$. By considering the stacks that arise when programs are run, it is shown in [8] that CBPV decomposes the strong monad of [34] into an adjunction between values and stacks; thus Moggi’s type $TA$ is expressed in CBPV as $UFA$.

Because nondeterminism is a computational effect, one would expect the CBPV primitives to arise in its semantics. And so they do: not only can algebra models be obtained from the powerset and powerdomain monads (as from any strong monad), but both the powerset model and the lower powerdomain model contain a much simpler CBPV structure [5, 9].

HOPLA [36] is a call-by-name higher-order process language, equipped with may-testing and presheaf semantics, that was developed from a linear logic model with exponential. The type of a process describes the possible computation paths that it can perform. It is used in [35] to encode several concurrent calculi, including CCS, CCS with process passing, and mobile ambients with public names.

The may-testing models of CBPV and of HOPLA are closely related, and preliminary research indicates a semantics-preserving translation from HOPLA to CBPV with nondeterminism and recursion. In particular, the HOPLA type $!P$, which is used to express prefixing in concurrent calculi, is expressed in CBPV as $FUP$. This points to a significant unification of quite disparate lines of research, with benefits in both directions. In particular, the work on presheaf semantics for HOPLA suggests similar semantics of CBPV (up to isomorphism) while a finer-grained analysis of HOPLA is obtained.
Output-Input and Affineness

Suppose we have a countable set of input-request messages, such as “Enter your name”, each with an associated countable set of permitted inputs (in this case, the set of strings). This kind of “output-input” is familiar to beginning programmers. It is easy to give a monad for this effect, generalizing the exception, output and input monads [34]: a simple exception has empty input set, and a plain output message has singleton input set.

But when we try to generalize exception handling to output-input, we need to ensure that an input-request message is not handled more than once, as it does not make computational sense to rewind a process. In other words, handling must be affine, and a type theory is required to express that.

Such a type theory, “Affine HOPLA”, a variant of HOPLA, is provided in [37] for a somewhat similar purpose: to translate a language with output-input into a functional language in a way that preserves may-testing equivalence. But this study has been dogged by a lack of operational understanding.

It is therefore proposed to formulate a synthesis of CBPV with Affine HOPLA, retaining may-testing and presheaf semantics, which would give a clear picture of the relationship between the various constructs, and bring to bear the strong operational character of CBPV. Preliminary research suggests that only one function connective will be required.

A categorical semantics is also required for this language, based on both the adjunction semantics of CBPV [8] and the various semantics of linear and affine logic, e.g. [32]. I would collaborate on this task with Eike Ritter.

This question is related to the broader one of relationships between CBPV and linear or affine type theories. Firstly, some CBPV semantics arise from linear models, and the foremost example is the may-testing semantics mentioned above, whose linear structure is described in [26]. Secondly, there appear to be links with the “linearly used continuation” transforms in [16], which could be exploited in the manner of [30] to give game semantics for this Affine CBPV.

Game Semantics and Powerdomains

It remains open [21] how to identify the nondeterministic strategies that are “innocent” (i.e. can be implemented without using storage). I propose to attack this using powerdomains: for example, for may-testing, such a strategy could be represented as a down-closed set of finite innocent functions.

2.4 Development of New Models

Limitations of Existing Semantics

In both domain and game models, recursion is interpreted as the least fixpoint of a selfmap. These cpo-enriched semantics are extremely suitable for modelling may-testing, where it describes an important computational fact: any observation made, in finite time, of the output of a function or process can depend on only finitely many observations of the input.

But for must-testing this reasoning is problematic, because a liveness property cannot be observed in finite time, and a computational process cannot in general be expected to be monotonic (let alone continuous). Typically, this arises with fair scheduling. For example, McCarthy’s amb is non-monotonic with respect to both Smyth and Plotkin powerdomains, including the variants for unbounded nondeterminism.

Furthermore, whilst cpo-enriched semantics can model absolute liveness properties, such as (2), they cannot model conditional liveness properties, such as

If MyProg insults the customer, then it must apologize

Explicitly, let $P$ be a program with two possible behaviours: (i) insult, then apologize, then diverge; (ii) diverge. Let $Q$ be a program that has both of these possible behaviours, but might also (iii) insult, then diverge. Clearly $P$ satisfies (4) but $Q$ does not. Yet it is shown in [39] that every cpo-enriched model must equate $P$ and $Q$. That includes, for example, all the standard models of CSP in [41].

The use of cpo-enrichment to model bisimulation in [14] causes a similar problem. On non-divergent terms it models bisimilarity, but on general terms, as explained in the paper, it models a different relation called partial bisimulation. Here is an example of it, adapted from [38]. Let $R$ be a program with 2 possible behaviours: (i) insult then diverge; (ii) insult, then either insult again or apologize. Let $S$ be a program that has both of these possible behaviours, but might also (iii) insult, then either apologize or diverge. These two programs are equated in Abramsky’s domain, because they are mutually partial bisimilar, even though they are not actually bisimilar, because $S$ satisfies (3) whereas $R$ does not.

These examples illustrate a general trend in the semantics of nondeterminism. Researchers have investigated the mathematical structures that are available (such as cpo-enrichment) and seen what operators and equivalences they can model—erratic choice, absolute liveness, partial bisimilarity, as well as the very intensional equivalences of metric space and presheaf semantics. This is surely reasonable, but, meanwhile, other operators and equivalences that are computationally natural—ambiguous choice, conditional liveness, bisimilarity—have been largely neglected in denota-
tional semantics, because the technology just does not exist.

Our aim in the second part of the project is to develop this technology, with the ultimate goal of forming new models of CBPV that represent these operators and equivalences. However, because so little is understood about them, we have to “map the landscape” in an incremental way. Thus, even limited results or counterexamples relating to low-order types may represent substantial progress.

Infinite Trace Equivalence

Consider a language with an instruction “print tick”. A program can have 3 possible behaviours: (a) print finitely many ticks, then stop, (b) print finitely many ticks, then diverge, (c) print infinitely many ticks. Possibly the most obvious equivalence to study is “infinite traces”: two programs are equivalent if their range of possible behaviours are the same. Clearly a model for this equivalence will allow us to reason about conditional liveness. The semantics of [17, 41, 42] use infinite traces, but do not model this equivalence (Brookes’ model counts unwindings).

My preliminary research [10] has indicated a serious problem in finding such a model, independently discovered by Roscoe [42]. We define commands $M$ and $N$ each of which use an identifier $x$ that represents a command. Let $M$ be the program that can either (i) print an arbitrary number of ticks, then stop, (ii) print an arbitrary number of ticks, then diverge, or (iii) execute $x$. Let $N$ be the program that has all these behaviours, but can also (iv) print a tick, then execute $x$. (It is possible to write these programs using only binary erratic choice.) Then both $M$ and $N$ can print infinitely many ticks precisely if $x$ can, and they can both exhibit all the other behaviours regardless of $x$. So, for any given $x$, the commands $M$ and $N$ are infinite trace equivalent.

But suppose we write $\mu x. M$ for the the command $M$ with $x$ recursively bound to $M$. Then $\mu x. M$ and $\mu x. N$ are different: the latter can print infinitely many ticks, whereas the former cannot. (In more abstract terms, they represent different fixpoints of the same selfmap on the powerset of infinite traces.) Consequently, a model for infinite trace equivalence cannot be well-pointed in the manner of domain theoretic semantics; there is no “right” way to compute fixpoints represented by recursion.

By contrast, in game semantics (which is not well-pointed), $M$ and $N$ would have different denotations, because a strategy would explicitly state when $x$ is executed. The techniques developed in game semantics allow us to model not just recursion, but higher-order types and type recursion, without any need for order-enrichment. However, in the absence of enrichment, new methods for proving adequacy (correctness) have to be found; preliminary research suggests that comparison of the game model to a more intensional one will accomplish this.

Bisimilarity

An important advance [13] was the application of the bisimilarity concept—traditionally used in the imperative setting—to functional languages. This makes it possible to apply the theory of computational effects in functional languages.

For sequential languages, whether imperative or functional, bisimilarity is known to be a congruence [25, 31]. Though difficulties have been found in adapting this to infinitary terms (e.g. Böhm trees) [35, 38], my preliminary research suggests a solution using a variation on the known techniques.

Congruence rules out the kind of pathology mentioned for infinite traces. Informally, suppose $M$ and $N$ are both terms of type $A$ in context $x : A$. Suppose that, for each value of $x$, the term $M$ and $N$ have the same behaviour, meaning that $M$ and $N$ represent the same selfmap on the set represented by $A$. The congruence theorem tells us that $\mu x. M$ and $\mu x. N$ represent the same fixpoint of that selfmap. So if we are given only the selfmap, and know that it is definable by at least one term, it has a “right” fixpoint—but what is it? How can we compute it?

The usual answer, viz. the least, or greatest, fixpoint in some partial order, does not seem to work; no known order gives bisimilarity. So some other structure is required that allows us to compute the fixpoint, and to show that it is the right one (via an adequacy theorem).

The simplicity of the congruence proof is most encouraging; it suggests that the computation of the fixpoint is equally simple. There is thus a strong chance that the extensional model we seek actually exists.

To search for it, we will need several lines of attack. Our basic approach is to begin by exploring low-order types, and in particular “positive” types, built up using thunks, but taking no arguments besides ground values.

1. One possibility may be that a program should denote some kind of Hennessy-Milner formulas. To this end, we will investigate the question: when does a set of Hennessy-Milner formulas contain precisely those assertions satisfied by some program?
2. We can build a (non-extensional) game semantics for bisimilarity, and then collapse this to an extensional model, exploiting the congruence theorem on Böhm trees (which represent strategies). Though somewhat artificial, this model might help us to find a more natural one.
3. For positive types, we can say that a program should denote (say) a set of sets of sets. We can then ask: what functions between these things are definable by a program? For low-order types that
are not positive, we do not know a priori what sort of thing a program should denote, but as we move up the type hierarchy, we can use our definability results at one order to formulate the definability question at the next.

4. Once we know what functions are definable, and, in particular, what selfmaps, we can try to work out with examples how fixpoints are computed by recursion. For first-order functions (where bisimilarity coincides with may-testing), this is trivial, but general results for other types would help to indicate what structure is required in our denotational model.

5. We will analyze the congruence proof to see if we can learn anything about how fixpoints are computed.

The finer relation of convex bisimilarity is obtained by allowing branching-time properties to mention the possibility of divergence. This is known to be a congruence without amb [31]; in the presence of amb, it is a longstanding open problem, currently being studied in the π-calculus setting [19]. It makes sense to study questions about convex bisimilarity only after studying the same questions for bisimilarity.

Must-Testing And Ambiguous Choice

In the presence of amb, it is currently impossible to model may-and-must testing (actually the same as must-testing equivalence [31]). And there is a more basic problem: it is not known whether the context lemma, which states that any non-closed terms equivalent in every environment are equivalent, is true in this setting. Resolving this would provide a major advance. For a refutation would rule out an extensional semantics (or, at least, a fully abstract one), whereas a proof may well suggest a denotational model.

At least at first order, the context lemma is true, and the model is clear. To interpret recursion, take a least fixpoint to obtain convergence behaviours, and then a greatest fixpoint to obtain divergence behaviours. This is done in [18], and our task is to adapt it to higher-order types, again by a study of definability and fixpoints.

Recent Development

A very recent and exciting development, indicating the timeliness of the proposed research, is the model of [42], which interprets recursion by a new kind of fixpoint. This appears to be related both to the infinite trace game model, and to the model of ambiguous choice in [18]. Clarifying these relationships may well lead to a broader framework.

2.5 Relevance to Beneficiaries

The unification of various strands of research, particularly HOPLA and computational effects, that is proposed here, promises to benefit both groups of researchers. Each group has developed structures the other was not aware of, such as presheaf semantics and call-by-push-value.

Game semantics has already shown its power for model-checking programs [15], and the development of infinite trace game semantics should make it possible to check conditional liveness properties of software.

Bisimilarity and Hennessy-Milner properties have been widely used in the study of systems, yet reasoning remains at an operational level and is difficult. Compositional reasoning and verification would be greatly aided by a denotational semantics.

Fair schedulers are a commonplace in computer systems, but there is little technology for reasoning about even the simplest fair operator—McCarthy’s ambiguous choice. A denotational semantics for ambiguity would be an advance towards general reasoning about fairness.

2.6 Dissemination

The results of this research will be disseminated in the usual way, through conferences such as Concur, MFPS, LiCS, FoSSaCS, CSL, ICALP, and journals such as Information and Computation, TCS and LMCS, as well as on the Web.

2.7 Justification of Resources

Clearly there is a large amount of research proposed here, and given my teaching and administrative responsibilities, I will not be able to work full-time on it. I am therefore requesting funding for a full-time Research Associate to work on this project. I would advertise on appropriate mailing lists (Concurrency, Types, Categories, APPSEM, etc.) to find a researcher with appropriate expertise, such as concurrency, effects, operational reasoning, categorical semantics or games. And whatever the RA’s current expertise, it would be greatly broadened by working on this project.

I also request travel expenses for conferences and research visits for both me and the RA, in order to develop the research, to disseminate it, and to improve it through the criticism of colleagues.

References


3 Diagrammatic Workplan

Each task has been allocated a year, indicated in parentheses.