Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 249 (2009) 377–405
www.elsevier.com/locate/entcs

An Algebraic Account of References in Game Semantics

Paul-André Melliès[1] Nicolas Tabareau[2]

Laboratoire Preuves Programmes Systèmes
CNRS & Université Paris Diderot

Abstract

We study the algebraic structure of a programming language with higher-order store, in the style of ML references. Instead of working directly on the operational semantics of the language, we consider its fully abstract game semantics defined by Abramsky, Honda and McCusker one decade ago. This alternative description of the language is nice and conceptual, except on one significant point: the interactive behavior of the higher-order memory cell is reflected in the model by a strategy `cell` whose definition remains slightly enigmatic. The purpose of our work is precisely to clarify this point, by providing a neat algebraic definition of the strategy. This conceptual reconstruction of the memory cell is based on the idea that a general reference behaves essentially as a linear feedback (or trace operator) in an ambient category of Conway games and strategies. This analysis leads to a purely axiomatic proof of soundness of the model, based on a natural refinement of the replication modality of tensor logic.

Keywords: game semantics, general references, tensor logic, replication modality, compact-closed categories, trace operator, memory access.

1 Introduction

Game semantics as an idealized compilation

The purpose of game semantics is to interpret programs as interactive strategies playing against their environment. It is nice to think that game semantics provides in this way an idealized compilation scheme, where the programs written in a high-level language are translated as strategies expressed in the low-level language of automata theory. This line of thought motivated already Berry and Curien to define their model of the purely functional language PCF based on concrete data structures and sequential algorithms – an early precursor of game semantics [7]. From that point of view, it is important to investigate what features of (high-level) programming languages can be compiled in game semantics, and what expressive power of the

[1] Email: mellies@pps.jussieu.fr
[2] Email: tabareau@pps.jussieu.fr

doi:10.1016/j.entcs.2009.07.099

target (low-level) language is necessary in order to perform the translation. A typical question is:

> Does one need to equip the target language (seen as a game or as an automaton) with a notion of state, in order to compile a programming language admitting itself a notion of state?

In their seminal work on ground-type references in Idealized Algol, Abramsky and McCusker answered to this question in the most unexpected way, see [4]. In order to understand properly the unexpected nature of their answer, one needs to start from the beginning – that is, the interactive description of the purely functional programs discovered by Hyland and Ong a few years earlier. They established that the functional programs written in the language PCF translate precisely as *innocent* and *well-bracketed* strategies – where innocence and well-bracketedness are conditions on strategies formulated in the framework of arena games [10].

Idealized Algol is an imperative extension of the language PCF with ground-type memory cells, where boolean and integer values can be stored. In their work, Abramsky and McCusker established that the imperative programs written in Idealized Algol translate precisely as *visible* and *well-bracketed* strategies – where the notion of visibility is formulated in the framework of arena games, and weakens the notion of innocence defined by Hyland and Ong.

This result is remarkable because it demonstrates that one does not need to refine the target language of functional languages (arena games) in order to translate imperative programs: it it is sufficient to relax the original constraint (innocence) into a weaker one (visibility.) In particular, the framework of arena games does not carry any explicit notion of state. This "stateless approach" is inspired by the "object-based" semantics of Idealized Algol developed in [24]. One main benefit of the approach is to clarify in what sense a given programming feature is more expressive than another one. In particular, the same type in PCF and in Idealized Algol is interpreted as the same arena game in each interpretation: the only difference between the two languages is the class of strategies playing on the arena game.

General references and game semantics

Abramsky, Honda and McCusker carried on in this line and established a similar result for a call-by-value language with higher-order store, in the style of ML general references. Recall that general references can store not only values of ground types (i.e. integers, booleans) but also those of higher types (procedures, higher-order functions, or references themselves). They provide a useful and powerful programming construct: the ability to write, to read and to transmit higher-order information through the store is essential not only for efficiency but also for clarity of the program structures involving states. Thus, the definition of a fully abstract model characterizing the programs written in this paradigm was an important achievement of game semantics. The interested reader will find in [22] a recent application of the model to an investigation of aspect-oriented programming.

Developing on their previous work, Abramsky, Honda and McCusker established

that the imperative programs written with general references define precisely the *thread-independent* and *well-bracketed* strategies of arena game. So, just as visibility weakens innocence and characterizes programs written with ground-type references, thread-independence weakens visibility and characterizes programs written with general references, see [2] for details.

This nice sequence of characterization theorems may be summarized in the following table:

purely functional programs	$\simeq$	innocent and well-bracketed strategies
imperative programs with ground-type references	$\simeq$	visible and well-bracketed strategies
imperative programs with general references	$\simeq$	thread-independent and well-bracketed strategies

where we are careful to indicate with the sign $\simeq$ that the definability result holds in all rigor for the compact (that is, finite) strategies of a given class.

Investigating the algebraic structure of general references

The purpose of this paper is to uncover a series of elementary algebraic structures hidden in the arena game model of general references designed by Abramsky, Honda and McCusker. Each ingredient is simple and conceptually clear, and we review them in turn:

- first of all, one needs to relax linear logic into tensor logic in order to study arena games, and to define the right kind of replication modality,
- then, the most primitive notion of general reference – called *linear memory* cell – may be constructed by *feedback* using a trace operator,
- here, one recalls that there exists a canonical notion of trace operator in any compact-closed category,
- so, we will work inside a compact-closed category of Conway games and well-bracketed strategies,
- linear memory cells are much too primitive: hence, we will need the *replication modality* ! of tensor logic in order to replicate them an infinite number of time, and a notion of *memory access* in order to bundle this infinite number of linear memory cells into the familiar notion of memory cell for general references.

Put all together, these five ingredients induce a purely axiomatic proof of soundness for the model of general references defined by Abramsky, Honda and McCusker. We review below these ingredients in turn.

Tensor logic and replication modalities in game semantics

Tensor logic was introduced in [19] in order to regenerate the fruitful connection between linear logic and game semantics. Tensor logic is simply obtained by relaxing the involutive negation of linear logic into a general notion of *tensorial negation*. The shortest way to describe the logic is to start from its categorical semantics. A dialogue category is defined as a symmetric monoidal category $\mathcal{C}$ equipped with an exponentiable object $\bot$, called the *tensorial pole* of the dialogue category. Recall that an object $\bot$ is exponentiable when every functor

$$A \quad \mapsto \quad \mathcal{C}(A \otimes B, \bot) \quad : \quad \mathcal{C}^{op} \quad \to \quad Set$$

is representable by an object $B \multimap \bot$ and a bijection

$$\phi_{A,B} \quad : \quad \mathcal{C}(A \otimes B, \bot) \quad \cong \quad \mathcal{C}(A, B \multimap \bot)$$

natural in A. The tensorial negation of the dialogue category $\mathcal{C}$ is then defined as the functor

$$\neg \quad : \quad \mathcal{C} \quad \longrightarrow \quad \mathcal{C}^{op}$$

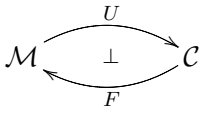
induced by the tensorial pole

$$\neg A \quad = \quad A \multimap \bot.$$

Every such tensorial negation induces a self-adjunction, whose monad is called the *linear continuation* monad of the dialogue category. Note that the dialogue category is $*$-autonomous precisely when the unit of the monad

$$\eta_A \quad : \quad A \quad \longrightarrow \quad \neg\neg A$$

is an isomorphism. One main reason for relaxing linear logic into tensor logic is that this enables to define a general notion of *replication modality* for categories of games and strategies. Indeed, such a category of games and strategies typically defines a dialogue category $\mathcal{C}$. A replication modality is then defined as a monoidal adjunction between the dialogue category $\mathcal{C}$ and a cartesian category $\mathcal{M}$



this generalizing the usual notion of a replication modality in a model of linear logic, see [6]. This notion of replication modality on tensor logic will become the key ingredient of our algebraic account of general references. The reason is that an explicit description of resources is necessary in order to describe how the memory cell manages the various copies induced by the sequence of write and read operations.

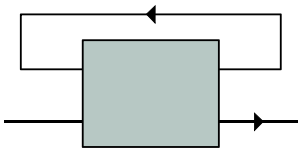
References as feedback in traced monoidal categories

Traced monoidal categories were introduced by Joyal, Street and Verity as a uniform account for several constructions dealing with *cyclic behaviours* in mathematics

and computer science – most notably braid closure in knot theory, trace operators in linear algebra, and feedback in computer science, see [12] for details. Recall that a *traced symmetric monoidal category* is a symmetric monoidal category $(\mathcal{C}, \otimes, I, c)$ equipped with a natural family of functions, called a *trace operator*,

$$\mathrm{Tr}^X_{A,B} \ : \ \mathcal{C}(X \otimes A, X \otimes B) \ \to \ \mathcal{C}(A, B)$$

which is nicely depicted in string diagrams



which satisfies a series of coherence axioms reflecting topological properties of the diagrammatic notation.

As far as we know, Milner was the first one to observe that linear feedback could be used in order to interpret *restriction* in process calculi. Milner describes this basic construction in the framework of action calculi, where he calls *reflection* the linear feedback operator. This feedback operator enables him to describe the restriction operator of arity

$$\nu \quad : \quad \epsilon \to p$$

as the trace of the diagonal

$$\delta_p \quad : \quad p \to p \otimes p$$

along the arity p, see [20].

Of course, we are interested in general references rather than channels, but the mechanism transforming a global variable into a local one works in a very similar way. In the usual monadic approach to effects, one interprets a programming language in a category by distinguishing between the objects A describing a value, and the objects TA describing computation of type A, where T is a given monad reflecting the particular notion of effect considered. In the case of references, the relevant monad is the state monad $X \mapsto S \multimap (S \otimes X)$ which enables to interpret a program P of type $A \to B$ as a morphism

$$A \quad \longrightarrow \quad S \multimap (S \otimes B)$$

taking a value of type A as input and producing a computation of type $S \multimap (S \otimes B)$ as output. It follows from the definition of linear implication in a symmetric monoidal category that this morphism may be seen alternatively as a morphism

$$f \quad : \quad S \otimes A \quad \longrightarrow \quad S \otimes B$$

This reformulation enables to see the monadic interpretation as the description of an input/output system with a store accessible by the user. Now, imagine that the trace of the morphism f along the object S is defined. In that case, we obtain a

morphism

$$g \quad : \quad A \quad \longrightarrow \quad B$$

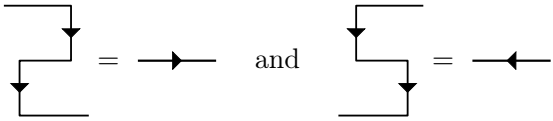
which behaves exactly like the original program P in which the state S is not global anymore, but local. This is precisely the analogous to the restriction (or localization) of Milner, where the channels are replaced by memory addresses.

Feedback in compact closed categories

We want to reconstruct the notion of general reference from the existence of a linear feedback, or trace operator, in a symmetric monoidal category of interest. One most natural way to obtain such a trace operator is to start from a compact closed category.

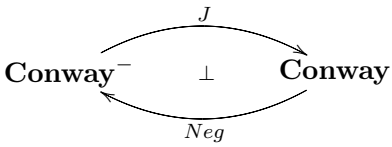
Recall that a *compact closed category* [13] is a symmetric monoidal category $\mathcal{C}$ in which every object A has a dual object. This means that for every object A, there exists an associated object A^* and two morphisms called *unit* $\eta_A : I \to A^* \otimes A$ and *counit* $\varepsilon_A : A \otimes A^* \to I$, depicted in the language of string diagrams as

satisfying the two zigzag identities



Every such compact closed category is equipped with a canonical trace operator defined in the expected diagrammatic way.

So the second ingredient of our reconstruction of general references will be to work in a compact closed category. More precisely, we will consider the full subcategory $\mathbf{Conway}^-$ of games starting by an Opponent move inside a compact closed category **Conway** of Conway games. This subcategory is symmetric monoidal closed and inherits a trace operator from the ambient compact closed category. In fact, the category $\mathbf{Conway}^-$ inherits its closed structure from an adjunction



where Neg removes all the initial Player's moves in the general Conway game. It is worth mentioning that early discussions with Masahito Hasegawa on this specific example enabled him to clarify the general categorical situation: he showed indeed that every traced symmetric monoidal *closed* category embeds as a full co-reflective subcategory in a compact closed category – typically defined by the Int construction, see [9].

Conway games and well-bracketed strategies

Since we are interested in defining a compact closed category of games and strategies, it is natural to start from the first category of games and strategies ever considered in the literature: the category of Conway games defined in [11] which is indeed compact closed. In fact, the starting point of our work is precisely the observation that this category is very deeply related to the game model of general references introduced by Abramsky, Honda and McCusker.

One missing ingredient in Conway games (however) is a notion of *well-bracketing* which enables to constraint the strategies of arena games. Hence, we take advantage of our recent construction of a compact closed category of Conway games and *well-bracketed* strategies, see [19]. In fact, our construction goes much further than this, since we are able to see the very notion of *well-bracketing* as an effect of *linearity conditions* on games and strategies. Although this logical aspect of the construction is fundamental, its role is very limited here. So, we will avoid discussing the details of the construction. Let us simply mention that in order to define a notion of bracketing on Conway games, while keeping the self-dual nature of Conway games, one needs to replace the traditional notions of *question* and *answer* by the more conceptual (and expressive) notions of *query* and *response* attached to the moves. See [19] and [26] for details.

Memory access modality

It appears that the category of call-by-value games defined by Abramsky, Honda and McCusker is isomorphic to a well-understood subcategory of the Kleisli category induced by the replication modality on our category of bracketed games. In fact, we could easily use their full abstraction result directly in order to obtain a full abstraction result for our model. But this is not really our point, here. The only novelty of our work is that the strategy $\mathtt{cell}_A$ interpreting a memory cell of type A is obtained in a nice and conceptual way in our formulation.

The existence of a trace operator and a tensorial negation on the category $\mathbf{Conway}^-$ enables to construct a very primitive notion of memory cell – called *linear memory* cell because it can be written and read only once. The replication modality of tensor logic enables then to replicate this linear memory cell in order to transform it into a *constant memory* cell, where one may write only once, but may then read as many times as desired.

The very last step is thus to construct a general memory cell. Here, we take advantage of the presence of linearity in our model, which enables to capture the difference between the copy management performed by a usual memory cell and by a constant memory cell. This difference is nicely axiomatized by the existence of two particular natural transformations

$$\xi_{A,B} \ : \ !(A \otimes !B) \ \longrightarrow \ !A \otimes !B \qquad\qquad \alpha_A \ : \ A \otimes !A \ \longrightarrow \ !A$$

defining what we call a memory access modality. These two maps will enable us to construct in basic little steps the familiar notion of memory cell:

- the *linear memory cell* which can be written and read only once – its construction requires the negation of tensor logic, and the compact closed structure of Conway games,
- the *constant memory cell* which can be written only once, but read an infinite number of times – this construction requires the replication modality of tensor logic,
- the familiar *memory cell* which can be written and read an infinite number of times – this construction requires the assumption that that the replication modality is equipped with a memory access map.

Moreover, the resulting notion of memory cell coincides with the strategy $\texttt{cell}_A$ defined by Abramsky, Honda and McCusker in the model of general references.

Bad variables in game semantics of states

One embarrassing point about this game-theoretic approach on references is that one needs to extend the programming language with a *bad variable constructor* in order to achieve the expected full abstraction result, characterizing programs as visible (or thread-independent) well-bracketed strategies. This particular problem is nicely explained and corrected in [18] and [21] for ground type references, by refining the notion of arena game. The main idea is to constrain the use of `read` and `write` moves by introducing new relations on plays and strategies to express an aspect of uniformity characterizing the interaction of every memory cell strategy. The main idea is that in absence of bad variables, each variable operation, whether a `Read` or a `Write`, produces the same side-effects while it is being completed.

Confronted to this difficulty, and the wish to extend the original notion of reference with the ability to compare names, the temptation is high to escape from the original framework. This may be done in two directions:

- shift to a monadic approach, where the strategies interpreting the imperative programs are innocent, rather than visible or thread-independent. This is the solution recently explored in [27] in a nominal setting,
- shift to a game model with an explicit notion of state in the game. This is the solution recently advocated in [16] and [15].

We hope that our algebraic approach to general references will enable to regulate the field, and to relate these apparently diverging approaches to game semantics.

Related works

There are not so many axiomatic approaches to programming languages based on game semantics. Let us mention three of them however. Abramsky gave a purely axiomatic proof of definability for PCF [1] followed by an axiomatic proof of full completeness for models of ML polymorphic types, in collaboration with Lenisa [3]. A few years later, Laird worked out an axiomatic proof of definability for a language with higher-order store [14]. His construction starts from the definition of an asymmetric tensor product on games, discussed with the first author a few years

$$[Var]\ \frac{}{\Gamma, x : A \vdash x : A} \qquad [Bool]\ \frac{}{\Gamma \vdash b : \mathrm{Bool}} \qquad [Nat]\ \frac{}{\Gamma \vdash n : \mathrm{Nat}} \qquad [Unit]\ \frac{}{\Gamma \vdash \mathtt{skip} : \mathrm{Unit}}$$

$$[Abs]\ \frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x.M : \alpha \to \beta}\ (x \notin \Gamma) \qquad [App]\ \frac{\Gamma \vdash M : \alpha \to \beta \quad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \beta}$$

$$[Succ]\ \frac{\Gamma \vdash M : \mathrm{Nat}}{\Gamma \vdash \mathtt{succ}\ M : \mathrm{Nat}} \qquad [Prec]\ \frac{\Gamma \vdash M : \mathrm{Nat}}{\Gamma \vdash \mathtt{prec}\ M : \mathrm{Nat}} \qquad [If]\ \frac{\Gamma \vdash M : \mathrm{Bool} \quad \Gamma \vdash M_i : \alpha (i = 1, 2)}{\Gamma \vdash \mathtt{ifzero}\ M\ \mathtt{then}\ M_1\ \mathtt{else}\ M_2 : \alpha}$$

$$[Pair]\ \frac{\Gamma \vdash M_i : \alpha_i (i = 1, 2)}{\Gamma \vdash \langle M_1, M_2 \rangle : \alpha_1 \times \alpha_2} \qquad [Proj]\ \frac{\Gamma \vdash M : \alpha_1 \times \alpha_2}{\Gamma \vdash \pi_i(M) : \alpha_i (i = 1, 2)}$$

$$[New]\ \frac{}{\Gamma \vdash \mathtt{new}_A : \mathrm{ref}[A]} \qquad [MkVar]\ \frac{\Gamma \vdash M : A \to \mathrm{Unit} \quad \Gamma \vdash M : \mathrm{Unit} \to A}{\Gamma \vdash \mathtt{mkvar}\ MN : \mathrm{ref}[A]}$$

$$[Assign]\ \frac{\Gamma \vdash M : \mathrm{ref}[A] \quad \Gamma \vdash N : A}{\Gamma \vdash M := N : \mathrm{Unit}} \qquad [Deref]\ \frac{\Gamma \vdash M : \mathrm{ref}[A]}{\Gamma \vdash\ \mathtt{deref}(M) : A}$$

Fig. 1. Typing rules of **IdeaML**

before. This asymmetry is indeed an important ingredient underlying the interactive behavior of higher-order store. In this paper, we prefer to derive the asymmetry from the existence of a pair of left-to-right and right-to-left coercion strategies

$$\neg\neg A \otimes \neg\neg B \qquad \Longrightarrow \qquad \neg\neg(A \otimes B)$$

reflecting the fact that the continuation monad $\neg\neg$ is strong but not commutative. Accordingly, the existence of a retraction

$$!A \multimap !B \quad \trianglelefteq \quad !(!A \multimap B)$$

mentioned in [14] is very similar to our notion of memory access. It will be interesting to understand how the logical and algebraic framework developed in this paper enables to clarify these axiomatic approaches to higher order store.

2 A language with general references

Types and terms

In this section, we recall the call-by-value language with higher-order store considered by Abramsky, Honda and McCusker. We call this language **IdeaML** for clarity's sake. Its types are defined by the grammar below

$$A, B ::= \mathrm{Unit} \mid \mathrm{Nat} \mid A \to B \mid A \times B \mid \mathrm{ref}[A]$$

The terms and typing rules of **IdeaML** are recalled in Figure 1, where Γ is a typing context which associates a type to every variable. Note that the constructor `mkvar` is here to compensate for the presence of *bad variables*. This problem, first identified by John Reynolds [25], comes from terms of type $\mathrm{ref}[A]$ which do not have a memory

$$\frac{}{(V,L,\sigma)\Downarrow(V,L,\sigma)} \qquad \frac{M_1\Downarrow V_1 \quad M_2\Downarrow V_2}{\langle M_1,M_2\rangle\Downarrow\langle V_1,V_2\rangle} \qquad \frac{M\Downarrow\langle V_1,V_2\rangle}{\pi_i(M)\Downarrow V_i (i=1,2)}$$

$$\frac{N\Downarrow V \quad M\Downarrow\lambda x.V'}{MN\Downarrow V'[V/x]} \qquad \frac{M_1\Downarrow V_1 \quad M_2\Downarrow V_2}{\texttt{mkvar}\ M_1\ M_2\Downarrow\texttt{mkvar}\ V_1\ V_2}$$

$$\frac{M\Downarrow 0 \quad M_1\Downarrow V}{\texttt{ifzero}\ M\ \texttt{then}\ M_1\ \texttt{else}\ M_2\Downarrow V} \qquad \frac{M\Downarrow n \quad M_2\Downarrow V \quad (n\neq 0)}{\texttt{ifzero}\ M\ \texttt{then}\ M_1\ \texttt{else}\ M_2\Downarrow V}$$

$$\frac{M\Downarrow V}{\texttt{succ}\ M\Downarrow V+1} \qquad \frac{M\Downarrow V}{\texttt{prec}\ M\Downarrow V-1} \qquad \frac{}{(\texttt{new}_A,L,\sigma)\Downarrow(l,L\cup(l:A),\sigma)}\ l\notin L$$

$$\frac{(M,L,\sigma)\Downarrow(l,L',\sigma') \quad (N,L',\sigma')\Downarrow(V,L'',\sigma'')}{(M:=N,L,\sigma)\Downarrow(\texttt{skip},L'',\sigma''(x\mapsto V))} \qquad \frac{(M,L,\sigma)\Downarrow(l,L',\sigma') \quad \sigma'(l)=V}{(\texttt{deref}(M),L,\sigma)\Downarrow(V,L',\sigma')}$$

$$\frac{M\Downarrow\texttt{mkvar}\ V_1\ V_2 \quad N\Downarrow V \quad V_1(V)\Downarrow\texttt{skip}}{M:=N\Downarrow\texttt{skip}} \qquad \frac{M\Downarrow\texttt{mkvar}\ V_1\ V_2 \quad V_2(\texttt{skip})\Downarrow V}{\texttt{deref}(M)\Downarrow V}$$

Fig. 2. Operational semantics of **IdeaML**

interpretation. The constructor `mkvar` allows to create bad variables from a term of type $A \to$ Unit (the writing method) and from a term of type Unit $\to A$ (the reading method).

Operational semantics

We also recall the operational semantics of the language. A *configuration* is defined as a triple (M, L, σ) where M is a term, L is a set of memory locations, and σ is a partial function, called memory state, which goes from locations to values of the appropriate type. As in the original paper by Abramsky, Honda and McCusker, we choose here a *big step* semantics. We thus need a notion of value, which are terms of the form:

$$V ::= n \mid b \mid x \mid \texttt{skip} \mid \lambda x.V \mid \langle V_1, V_2\rangle$$

When a configuration (M, L, σ) reduces to a value (V, L', σ'), we note

$$(M,L,\sigma)\Downarrow(V,L',\sigma')$$

For a more convenient notation, when the evolution of the memory state can be left implicit, we adopt the convention that

$$\frac{M\Downarrow V \quad M'\Downarrow V'}{M''\Downarrow V''}$$

is an abbreviation for

$$\frac{(M,L,\sigma)\Downarrow(V,L',\sigma') \quad (M',L',\sigma')\Downarrow(V',L'',\sigma'')}{(M'',L,\sigma)\Downarrow(V'',L'',\sigma'')}$$

Figure 2 describes inductively the operational semantics of our language.

3 Bracketed Conway games

The category **Conway** of bracketed Conway games was introduced in [19]. As we will see, it is compact-closed, and its full subcategory $\mathbf{Conway}^-$ of Opponent-starting games is symmetric monoidal closed and traced – this providing the appropriate model of tensor logic in order to perform our algebraic reconstruction of the arena game model of **IdeaML**.

3.1 Conway games.

A *Conway game* A is an oriented rooted graph (V_A, E_A, λ_A) consisting of (1) a set V_A of vertices called the *positions* of the game; (2) a set $E_A \subset V_A \times V_A$ of edges called the *moves* of the game; (3) a function $\lambda_A : E_A \to \{-1, +1\}$ indicating whether a move belongs to Opponent (-1) or Proponent $(+1)$. We note $\star_A$ the root of the underlying graph. A Conway game is said to be *negative* (resp. *positive*) when all its move starting from the root belongs to Opponent (resp. Proponent).
A *play* $s = m_1 \cdot m_2 \cdot \ldots \cdot m_{k-1} \cdot m_k$ of a Conway game A is a path $s : \star_A \twoheadrightarrow x_k$ starting from the root $\star_A$

$$s : \star_A \xrightarrow{m_1} x_1 \xrightarrow{m_2} \ldots \xrightarrow{m_{k-1}} x_{k-1} \xrightarrow{m_k} x_k$$

Two paths are parallel when they have the same initial and final positions. A play is *alternating* when

$$\forall i \in \{1, \ldots, k-1\}, \qquad \lambda_A(m_{i+1}) = -\lambda_A(m_i).$$

We note Play_A the set of plays of a game A.

3.2 Bracketed Conway games.

A bracketed Conway game is defined as a Conway game equipped with

- a finite set $Q_A(x)$ of *queries* for each position $x \in V_A$ of the game,
- a function $\lambda_A(x) : Q_A(x) \to \{-1, +1\}$ which assigns to every query in $Q_A(x)$ a polarity which indicates whether the query is made by Opponent (-1) or Proponent $(+1)$,
- for each move $x \xrightarrow{m} y$, a *residual* relation

$$[m] \subset Q_A(x) \times Q_A(y)$$

satisfying:

$$r[m]r_1 \quad \text{and} \quad r[m]r_2 \implies r_1 = r_2$$
$$r_1[m]r \quad \text{and} \quad r_2[m]r \implies r_1 = r_2$$

The definition of residuals is then extended to paths $s : x \twoheadrightarrow y$ in the usual way – by composition of relations. We then define

$$r[s] \stackrel{\text{def}}{=} \{r' \mid r[s]r'\} \quad \text{and} \quad [s]r \stackrel{\text{def}}{=} \{r' \mid r'[s]r\}.$$

We say that a path $s : x \twoheadrightarrow y$:

- *complies with a query* $r \in Q_A(x)$ when r has no residual after s — that is, $r[s] = \emptyset$,
- *initiates a query* $r \in Q_A(y)$ when r has no ancestor before s — that is, $[s]r = \emptyset$.

We require that a move m only initiates queries of its own polarity, and only complies with queries of the opposite polarity. In order to formalize that a residual of a query is intuitively the query itself, we also require that two parallel paths s and t induce the same residual relation: $[s] = [t]$. Finally, we require that there are no queries at the root: $Q_A(\star) = \emptyset$.

3.3 Resource function.

Extending Conway games with queries enables the definition of a resource function

$$\kappa_A = (\kappa_A^+, \kappa_A^-)$$

which counts, for every path $s : x \twoheadrightarrow y$, the number $\kappa_A^+(s)$ (respectively $\kappa_A^-(s)$) of Proponent (respectively Opponent) queries in $r \in Q_A(y)$ initiated by the path s — that is, such that $[s]r = \emptyset$.

3.4 Dual.

Every bracketed Conway game A induces a *dual* game A^* obtained simply by reversing the polarity of moves and queries.

3.5 Tensor product.

The tensor product $A \otimes B$ of two bracketed Conway games A and B is essentially the asynchronous product of the two underlying graphs. More formally, it is defined as:

- $V_{A\otimes B} = V_A \times V_B$,
- its moves are of two kinds :

$$x \otimes y \to \begin{cases} z \otimes y \text{ if } x \to z \text{ in the game } A \\ x \otimes z \text{ if } y \to z \text{ in the game } B, \end{cases}$$

- $Q_{A\otimes B}(x \otimes y) = Q_A(x) \uplus Q_B(y)$,

the polarities of moves and queries in the game $A \otimes B$ is inherited from games A and B and the *residual relation* is defined pointwise.

The unique bracketed Conway game 1 whose underlying Conway game is the game with a unique position $\star$ and no move is the neutral element of the tensor product. As usual in game semantics, every play s of the game $A \otimes B$ can be seen as the interleaving of a play $s_{|A}$ of the game A and a play $s_{|B}$ of the game B.

3.6 Strategies.

A *strategy* σ of a bracketed Conway game A is defined as a non empty set of *alternating plays* of even length such that (1) every non empty play starts with an Opponent move; (2) σ is closed by even length prefix; (3) σ is *deterministic*, i.e. for all plays s, and for all moves m, n, n',

$$s \cdot m \cdot n \in \sigma \ \wedge \ s \cdot m \cdot n' \in \sigma \ \Rightarrow \ n = n'.$$

We write $\sigma : A \to B$ to indicate that σ is strategy over the game $A^* \otimes B$.

3.7 Well-bracketed plays.

An alternating play s is well-bracketed (from the Player's point of view) in the game A when for every factorization $s = s_1 \cdot m \cdot t \cdot n \cdot s_2$ with m an Opponent move and n a Player move, one has

$$\kappa_A^+(m \cdot t \cdot n) = 0 \quad \text{implies} \quad \kappa_A^-(m \cdot t \cdot n) = 0$$

An alternating play s is well-bracketed (from the Opponent's point of view) in the game A when it is well-bracketed from the Player's point of view in the dual game A^*. An alternating play s is well-bracketed in the game A when it is well-bracketed from the Player and the Opponent's point of view.

3.8 Well-bracketed strategies.

A strategy σ is *well-bracketed* when all its plays $s \in \sigma$ are well-bracketed (from Player's point of view).

3.9 Composition.

We define the composite of two strategies $\sigma : A \to B$ and $\tau : B \to C$ in the usual fashion based on "parallel composition plus hiding". The proof that the composite of two well-bracketed strategies is well-bracketed is far from immediate: the interested reader will find in [19].

3.10 Identity morphism.

We define the identity morphism id_A on a game A by the expected copycat strategy on the game $A^* \otimes A$, in the same way as André Joyal defined it in [11]. The copycat strategy replies to any Opponent move in one of the component A^* or A by the dual move in the other component.

3.11 The category of bracketed Conway games.

The category **Conway** has bracketed Conway games as objects, and well-bracketed strategies σ of $A^* \otimes B$ as morphisms $\sigma : A \to B$. The resulting category **Conway** is compact-closed in the sense of [13]. The unit $\eta_A : 1 \to A \otimes A^*$ and counit $\varepsilon_A : A^* \otimes A \to 1$ are defined as the expected copycat strategies.

From now on, we will consider the full subcategory $\mathbf{Conway}^-$ of negative bracketed Conway games. By negative game A, we mean a game where Opponent starts. This category $\mathbf{Conway}^-$ is symmetric monoidal, and inherits a trace operator from the ambient category of bracketed Conway games. Not only that: the category $\mathbf{Conway}^-$ is symmetric monoidal closed. As such, it defines a model of tensor logic for every object $\bot$ selected as tensorial pole in the category $\mathbf{Conway}^-$. The tensorial negation below is induced by defining $\bot$ as the game with a unique move, which is played by Opponent, and initiates no query.

3.12 Tensorial negation.

The negation $\neg A$ of a negative game is the pointed game obtained by lifting the dual game A^* with a unique Opponent move that does not initiate any query.

3.13 Replication modality.

Every negative Conway game A induces a duplicable game $!A$ which should be seen as some kind of infinite tensor product of the game A. We define $!A$ as follows

- its positions are finite words $w = x_1 \cdots x_k$ whose letters are positions x_i of the game A which are different from the root; the intuition is that a letter x_i describes the current position in the i^{th} copy of A;
- its root is the empty word $\star_{!A} = \varepsilon$;
- a move $w \to w'$ is either a move played in one copy:

$$w_1\ y\ w_2 \to\ w_1\ y'\ w_2$$

  where $y \to y'$ is a move of the game A and $y \neq \star_A$; or an opening of a new copy:

$$w \to w \cdot y$$

  where $\star_A \to y$ is a move of the game A strating from the root $\star_A$.
- its queries at the position $w = x_1 \cdots x_n$ are the pairs (i, r) composed by an index $1 \leq i \leq n$ and a query r at the position x_i of the game A. In particular, there is no query at the empty position ε.

The polarities of moves and queries are inherited from those of the game A in the obvious way; and the residual relation is defined as for the tensor product. Using the limit construction of free exponentials described in [23,26], we are able to show that the game $!A$ is the free commutative comonoid generated by the game A. In this way, we have just defined a model of tensor logic (without coproducts) in a traced monoidal category equipped with a notion of well-bracketing.

3.14 Accommodating the additives.

At this point, there remains to define a notion of finite coproduct in the category $\mathbf{Conway}^-$. Unfortunately, the category $\mathbf{Conway}^-$ does not have coproducts. Instead, it has all products, noted $\&_i A_i$, defined as the direct sum of the underlying graphs. So, we apply the *family construction*, described in [5], in order to freely add the coproducts to our category. One extremely pleasant aspect of tensor logic is that the category $Fam(\mathbf{Conway}^-)$ defines a model of tensor logic with sums and a replication modality inherited from the category $\mathbf{Conway}^-$. Namely, the exponential modality on the category $Fam(\mathbf{Conway}^-)$ is defined pointwise, and thus transports an object $(A_i)_{i\in I}$ of the category $Fam(\mathbf{Conway}^-)$ to the object $!(A_i)_{i\in I} = (!A_i)_{i\in I}$.

3.15 Notation.

Before interpreting the language **IdeaML** in our game semantics, we find useful to polarize formulas, so that *negative* formulas are interpreted in the category $\mathbf{Conway}^-$ of Opponent-starting Conway games, whereas the *positive* formulas are interpreted in the category

$$\mathbf{Conway}^+ \quad \stackrel{\text{def}}{=} \quad (\mathbf{Conway}^-)^{\text{op}}$$

of Player-starting Conway games. In order to distinguish between a positive and a negative formula, one has to clone every connective of tensor logic. Typically, the tensorial negation $\neg$ is cloned as two functor

$$\stackrel{P}{\uparrow} : \mathbf{Conway}^- \longrightarrow \mathbf{Conway}^+ \qquad \stackrel{O}{\downarrow} : \mathbf{Conway}^+ \longrightarrow \mathbf{Conway}^-$$

This leads us to draw the mnemonic table:

$\mathbf{Conway}^-$	$\stackrel{O}{\downarrow} A^+$	$A^- \otimes B^-$	$A^- \,\&\, B^-$	$!A^-$
$\mathbf{Conway}^+$	$\stackrel{P}{\uparrow} A^-$	$A^+ \parr B^+$	$A^+ \oplus B^+$	$?A^+$

Note in particular that the replication modality ! does not alter the polarity of the negative formula, in contrast to what happens in usual polarized logic. This point is not only a key aspect of tensor logic: it is also necessary in order to achieve our reconstruction the memory cell in the next section.

4 Full abstraction by comparison

In this section, we establish a full abstraction theorem for our game semantics of the language **IdeaML**. We achieve this by reformulating the game model designed by Abramsky, Honda and McCusker as a subcategory of the Kleisli category $\mathbf{Conway}^-_!$ induced by the replication comonad ! on the category $\mathbf{Conway}^-$ of negative bracketed Conway games.

Recall that Abramsky, Honda and McCusker construct a category **AHM** with arena games as objects, and well-bracketed strategies on the game $A \Rightarrow B$ as mor-

phisms from A to B, see [2] for details. An arena is a triple $A = (M_A, \lambda_A, \vdash_A)$ where M_A is a set of *moves*, $\lambda_A = (\lambda_A^{OP}, \lambda_A^{QA}) : M_A \to \{O, P\} \times \{Q, A\}$ is a *labelling* function and $\vdash_A$ is a justification relation between the moves. A strategy of such an arena is then defined as a set of justified, alternating and well-bracketed plays, in the tradition of [10].

The three authors declare that a strategy σ is *thread-independent* when it plays according each "thread" or "connected component" of the game independently. In other words, a thread-independent strategy σ plays according to all the moves of the game which are hereditarily justified by the same initial move. A more conceptual way to think of these strategies is to observe that they are precisely the comonoid morphisms between arenas, seen as commutative comonoids in the category **AHM**, see [8] for details.

Our point is that every such thread-independent strategy σ from A to B in the category **AHM** may be seen as a strategy of the form $!A \multimap B$ in the category $\mathbf{Conway}^-$ of negative Conway games. We formalize this intuition below. Let us define $\mathbf{AHM_{thread}}$ the category whose objects are arenas and whose morphisms from A to B are well-bracketed and thread-independent strategies on the game $A \Rightarrow B$. It is by definition a subcategory of **AHM**.

We define a faithful (but not full) functor

$$\mathcal{U} \quad : \quad \mathbf{AHM} \quad \longrightarrow \quad \mathbf{Conway}^-$$

which transports an arena $A = (M_A, \lambda_A, \vdash_A)$ to the negative bracketed Conway game $\mathcal{U}(A)$ defined as follows:

- the positions of $\mathcal{U}(A)$ are the plays of the arena game A,
- there is a move m between two plays s and s' precisely when $s' = s \cdot m$,
- the polarity of moves is inherited from the polarity of those of A,
- a move $s \xrightarrow{m} s'$ initiates a query when m is a question in the game A;
- a move n complies with a query initiated by the move m when n is an answer and m justifies n.

Every strategy σ of the game $A \to B$ in the category **AHM** is transported to the corresponding strategy σ of the game $\mathcal{U}(A) \to \mathcal{U}(B)$ in the category $\mathbf{Conway}^-$ of negative bracketed Conway games. The functor $\mathcal{U}$ is obviously faithful.

Let us briefly explain why the functor $\mathcal{U}$ is not full. This is a subtle point which deserves to be clarified. The anomaly is minor however, and comes in fact from the very definition of well-bracketed strategies in arena games – as strategies σ containing only well-bracketed plays, where *well-bracketed* means that the play is well-bracketed from the point of view of Player *and* Opponent. Given two arena games A and B, it is generally possible to find an alternating play s in the arena game $A \Rightarrow B$ which is well-bracketed from the point of view of Player but not from the point of view of Opponent. Then, any well-bracketed strategy τ which contains the play s in the Conway game $\mathcal{U}(A) \multimap \mathcal{U}(B)$ cannot be the image of a well-bracketed strategy $\sigma : A \to B$ in the category **AHM** of arena games.

This anomaly is easy to correct however, since every well-bracketed strategy τ on a Conway game contains a largest well-bracketed strategy $\widehat{\tau}$ of the form $\mathcal{U}(\sigma)$, defined as the set of plays $s \in \tau$ which are not only well-bracketed from the point of view of Player, but also from the point of view of Opponent. The strategies of the form $\mathcal{U}(\sigma)$ are then characterized as the strategies τ such that the equality $\tau = \widehat{\tau}$ holds.

The functor $\mathcal{U} : \mathbf{AHM} \to \mathbf{Conway}^-$ lifts to a faithful functor

$$\widetilde{\mathcal{U}} \ : \ \mathbf{AHM_{thread}} \ \longrightarrow \ \mathbf{Conway}^-_!$$

such that the following diagram commutes

$$\begin{array}{ccc} \mathbf{AHM_{thread}} & \xrightarrow{\widetilde{\mathcal{U}}} & \mathbf{Conway}^-_! \\ \downarrow & & \downarrow \\ \mathbf{AHM} & \xrightarrow{\mathcal{U}} & \mathbf{Conway}^- \end{array}$$

Observe that the category $\mathbf{AHM_{thread}}$ defines a subcategory of $\mathbf{Conway}^-_!$ in which all the bracketed Conway games of the form $\mathcal{U}(A)$ are bracketed in the usual sense... that a move in the game can initiate *at most* one query.

At this stage, Abramsky, Honda and McCusker shift to the game semantic interpretation of a call-by-value language, by applying the family construction on the category **AHM**, exactly as we have done in order on the category of negative Conway games $\mathbf{Conway}^-$ in order to obtain a model of tensor logic with sums and replicative modality. From this, we deduce that the category introduced by Abramsky, Honda and McCusker in order to give a fully abstract model of **IdeaML** lives inside the category $Fam(\mathbf{Conway}^-_!)$. Moreover, the morphisms of this underling category are characterized by the equality $\tau = \hat{\tau}$. Besides, the contextual equivalence of the category $Fam(\mathbf{Conway}^-_!)$ identifies two strategies σ and τ containing the same well-bracketed plays:

$$\sigma \sim \tau \quad \Longleftrightarrow \quad \hat{\sigma} = \hat{\tau}.$$

Hence, the category $Fam(\mathbf{Conway}^-_!)$ provides a fully abstract model of **IdeaML** which coincides with the model provided by Abramsky, Honda and McCusker after this mild quotient by contextual equivalence.

5 An algebraic account of memory cells

The main novelty of our approach to general references is that it enables us to reconstruct the interactive strategy interpreting the memory cell in a purely conceptual way. In fact, we work gradually, and define three different paradigms of memory cell, using tensor logic and the compact closed structure of the underlying category of Conway games. A pleasant aspect of the approach is to relate the copy management performed by the familiar memory cell to the existence of a natural transformation

on the replication modality

$$\xi_{A,B} \quad : \quad !(A \otimes !B) \quad \longrightarrow \quad !A \otimes !B$$

satisfying two coherence laws.

5.1 An analysis of the strategy $\mathtt{cell}_A$.

Before constructing the strategies associated to each kind of cell, it seems useful to recall how the language **IdeaML** is interpreted in the category $Fam(\mathbf{Conway}^-)$, following [2]. Abramsky, Honda and McCusker interpret the call-by-value language **IdeaML** using a cartesian closed category equipped with a strong monad. This monad, defined on the family category, is described in our model by

$$T(A_i)_{i\in I} \stackrel{\text{def}}{=} (\overset{O}{\downarrow} \oplus_i \overset{P}{\uparrow} A_i)$$

seen as a singleton family in the category $Fam(\mathbf{Conway}^-)$. The unit of this monad is obtained from the unit of the continuation monad and from injections. The strength of the monad T is easily deduced from the strength of the continuation monad. We note

$$TA \otimes TB \underset{\mathtt{right}_{A,B}}{\overset{\mathtt{left}_{A,B}}{\rightleftarrows}} T(A \otimes B)$$

the left-to-right and right-to-left coercions of the strong monad T. The interpretation of a typing judgment

$$x_1 : X_1, \cdots x_n : X_n \vdash M : A$$

is given by a morphism

$$!\{X_1\} \otimes \cdots \otimes !\{X_n\} \;\longrightarrow\; T\{A\}$$

where $\{A\} = (A_i)_{i\in I}$ is the translation of the type A in the category $Fam(\mathbf{Conway}^-)$. Such a morphism in the category $Fam(\mathbf{Conway}^-)$ may be seen as a strategy which lets Opponent choose a component Y_i in each family X_i, and then plays according to a strategy

$$!Y_1 \otimes \cdots \otimes !Y_n \quad \longrightarrow \quad \overset{O}{\downarrow} \oplus_i \overset{P}{\uparrow} A_i$$

in the category $\mathbf{Conway}^-$. The translation of the arena corresponding to the type ref[A] is the bracketed game

$$\{\mathrm{ref}[A]\} \stackrel{\text{def}}{=} !(\&_i \overset{O}{\downarrow} (\overset{P}{\uparrow} 1 \parr ?A_i^*)) \otimes ! \, T(A_i) \stackrel{\text{def}}{=} !\mathtt{Write}_{!A} \otimes !\mathtt{Read}_A$$

seen as a singleton family in the category $Fam(\mathbf{Conway}^-)$. The formulas

$$\mathtt{Write}_A \;=\; \&_i \overset{O}{\downarrow} (\overset{P}{\uparrow} 1 \parr A_i^*) \qquad\qquad \mathtt{Read}_A \;=\; T(A_i)_{i\in I}$$

$$\{\Gamma \vdash \texttt{new}\ x : A, y : B\ \texttt{in}\ M\} = \{\Gamma \vdash \texttt{new}\ y : B, x : A\ \texttt{in}\ M\} \tag{1}$$

$$\{\Gamma, x : \mathrm{ref}[A] \vdash \texttt{new}\ y : B\ \texttt{in}\ x := V; M\} = \{\Gamma, x : \mathrm{ref}[A] \vdash x := V; \texttt{new}\ y : B\ \texttt{in}\ M\} \tag{2}$$

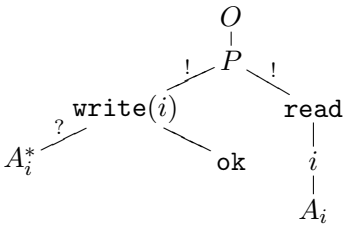
$$\{\Gamma \vdash \texttt{new}\ x : A, y : B\ \texttt{in}\ x := V_1; y := V_2; M\} = \{\Gamma \vdash \texttt{new}\ x : A, y : B\ \texttt{in}\ y := V_2; x := V_1; M\} \tag{3}$$

$$\{\Gamma \vdash \texttt{new}\ x : A\ \texttt{in}\ x := V; (\lambda y.M)(\texttt{deref}(x))\} = \{\Gamma \vdash \texttt{new}\ x : A\ \texttt{in}\ x := V; (\lambda y.M)V\} \tag{4}$$

$$\{\Gamma \vdash \texttt{new}\ x : A\ \texttt{in}\ x := V_1; x := V_2; M\} = \{\Gamma \vdash \texttt{new}\ x : A\ \texttt{in}\ x := V_2; M\} \tag{5}$$

Fig. 3. Semantics of the memory cell

denote the type of the write and read methods for the family $A = (A_i)_{i \in I}$. When $I = \{i\}$ is a singleton, the game $T\{\mathrm{ref}[A]\}$ can be depicted as the arena



The interactive strategy $\texttt{cell}_A$ implementing the memory cell may be described informally as follows:

- the strategy $\texttt{cell}_A$ reacts to the first move O by playing the move P,
- if the next move played by Opponent is `read`, the strategy does not respond,
- the strategy $\texttt{cell}_A$ reacts to the move $\texttt{write}(i)$ by playing the move `ok`,
- when Opponent plays the move `read` and when the last writing move which has been played is $\texttt{write}(i)$, the strategy $\texttt{cell}_A$ answers i,
- when Opponent plays an initial move of A_i in the read component, the strategy $\texttt{cell}_A$ answers by playing an initial move in a new copy of the game $?A_i^*$, which corresponds to the last move $\texttt{write}(i)$ played,
- then, the strategy $\texttt{cell}_A$ copycats the Opponent moves in A_i (resp. A_i^*) by playing the corresponding moves in A_i^* (resp. A_i).

We will now reconstruct the strategy $\texttt{cell}_A$ in three elementary steps.

5.2 Linear memory cell and dual object.

First, we construct the strategy $\texttt{lin\_cell}_A$ of type

$$1 \xrightarrow{\ \texttt{lin\_cell}_A\ } T\big(\texttt{Write}_A \otimes \texttt{Read}_A\big), \tag{6}$$

which interprets the linear memory cell. The construction of this strategy relies on the tensorial negation and on the existence of dual objects. Those two things enable to build a right inverse to the strength $t_{A,B,C}$

$$\mathfrak{t}_{A,B,C} \ : \overset{O}{\downarrow} (A \parr \overset{P}{\uparrow} (B \otimes C)) \ \longrightarrow \overset{O}{\downarrow} (A \parr \overset{P}{\uparrow} B) \otimes C.$$

The construction starts from an elementary property of duals in a monoidal closed category, which the interested reader will find mentioned by Peter May [17].

Proposition 5.1 *Let $(\mathcal{C}, \otimes, 1, \multimap)$ be a symmetric monoidal closed category. If the object C possesses a dual object C^* in the category $\mathcal{C}$, then the canonical morphism*

$$f_{A,B,C} \quad : \quad (A \multimap B) \otimes C \quad \longrightarrow \quad A \multimap (B \otimes C)$$

is an isomorphism, for every object A and B.

We use a variant of this property, where the closure is replaced by a tensorial negation. As starting point, we observe that the strength

$$\overset{O}{\downarrow}(A \parr \overset{P}{\uparrow} B) \otimes C \quad \xrightarrow{t_{A,B,C}} \quad \overset{O}{\downarrow}(A \parr \overset{P}{\uparrow}(B \otimes C))$$

which exists in the category $\mathbf{Conway}^-$ extends in fact to the whole category **Conway**. This point is subtle: note in particular that the strategy $t_{A,B,C}$ is partial when the game C is positive, since it does not answer when Opponent plays his first move in the left hand side game C. For that reason, the algebraic and logical status of the strength in the whole category **Conway** remains somewhat enigmatic at that level of description.

The strategy $\mathfrak{t}_{A,B,C}$ in the category **Conway** may be defined as follows:

$$\begin{array}{rl} \overset{O}{\downarrow}(A \parr \overset{P}{\uparrow}(B \otimes C)) & \xrightarrow{-\otimes \eta_C} \quad \overset{O}{\downarrow}(A \parr \overset{P}{\uparrow}(B \otimes C)) \otimes C^* \otimes C \\ & \xrightarrow{t \otimes C} \quad \overset{O}{\downarrow}(A \parr \overset{P}{\uparrow}(B \otimes C \otimes C^*)) \otimes C \\ & \xrightarrow{(-\otimes \varepsilon_C)\otimes C} \overset{O}{\downarrow}(A \parr \overset{P}{\uparrow} B) \otimes C. \end{array}$$

When the game A is positive and the games B and C are negative, the resulting strategy defines a morphism in the category $\mathbf{Conway}^-$, which is right inverse to the strength. This right inverse morphism will be at the heart of our definition of the strategy $\mathtt{lin\_cell}_A$. The construction starts by composing the injection of the singleton family (A_i) into the family $(A_i)_{i \in I}$ with the unit of the monad T

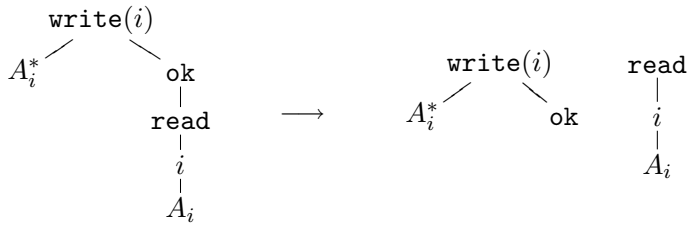
$$A_i \quad \longrightarrow \quad (A_i)_{i \in I} \quad \longrightarrow \quad T(A_i)_{i \in I} = \mathtt{Read}_A. \tag{7}$$

We then take its image by the tensorial negation and apply the law defining the tensorial negation, so as to obtain the morphism

$$\chi_{(A_i)} \quad : \quad 1 \quad \longrightarrow \quad \overset{O}{\downarrow}\big(A_i^* \parr \overset{P}{\uparrow} \mathtt{Read}_A\big).$$

Then, we use the right inverse of the strength for $A = A_i^*$, $B = 1$ and $C = \mathtt{Read}_A$

and define a strategy of domain and codomain



in the arena games notation. Every play s played by this strategy starts with the moves

$$\texttt{write}(i) \cdot (\texttt{write}(i))^* \cdot i^* \cdot i$$

followed by a copycat strategy. So, the strategy is the copycat strategy which does not answer in the case Opponent decides to read the memory cell before writing in it.

At this stage, there only remains to take the product on every index i and to compose with the unit of the continuation monad, in order to obtain the strategy $\texttt{lin\_cell}_A$ described by Equation (6). The linear dereference method is defined by the copycat strategy on $T\texttt{Read}_A$, and the linear assign method is provided by

$$\texttt{assign}_{\texttt{lin}A} \stackrel{\text{def}}{=} \texttt{eval}^{\ P}_{A,\uparrow 1} \quad : \quad A \otimes \texttt{Write}_A \longrightarrow T1$$

where

$$\texttt{eval}_{A,B} \quad : \quad A \otimes \overset{O}{\downarrow}(A^* \parr B) \quad \to \quad \overset{O}{\downarrow} B$$

is obtained by applying the law defining the tensorial negation to the identity morphism

$$\overset{O}{\downarrow}(A^* \parr B) \quad \xrightarrow{\text{id}} \quad \overset{O}{\downarrow}(A^* \parr B).$$

At this point, we show that this linear memory cell is well behaved in the sense that it satisfies the first four equations of Figure 3. Those four equations are well known because they constitute four of the five equations validated by a memory cell in the work of Abramsky, Honda and McCusker [2]. They ensure together the soundness of a linear memory cell.

The formalism provided by tensor logic enables us to deduce the validity of those four equations in a purely categorical way, using commutative diagrams of an algebraic flavor. In particular, we never have to refer to the strategies underlying the canonical morphisms.

(i) Equality (1) comes from the fact that the diagram

$$A \otimes B \xrightarrow{\eta_A \otimes \eta_B} TA \otimes TB \underset{\texttt{right}_{A,B}}{\overset{\texttt{left}_{A,B}}{\rightrightarrows}} T(A \otimes B)$$

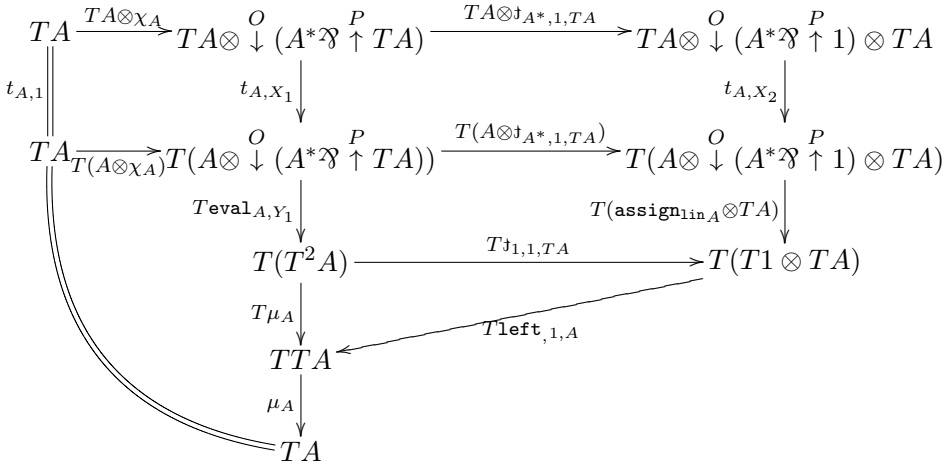
commutes, where the two morphisms $\texttt{left}_{A,B}$ and $\texttt{right}_{A,B}$ are the left-to-right and right-to-left coercions induced by the strong monad T,

(ii) Equality (2) follows directly from the coherence laws (associativity and multiplication) satisfied by the monad T,

(iii) Equality (3) follows from the bifunctoriality of the tensor product,

(iv) Equality (4) is more difficult to derive, because it involves diagrammatic properties of the morphism $\natural$. In order to simplify, we only describe here the key diagrams in the case of a singleton family, $(A_i)_{i\in I} = (A)$. Namely, we use the fact that the morphism $\natural$ is constructed from the strength of the tensorial negation and from the unit and counit of the compact closed structure.

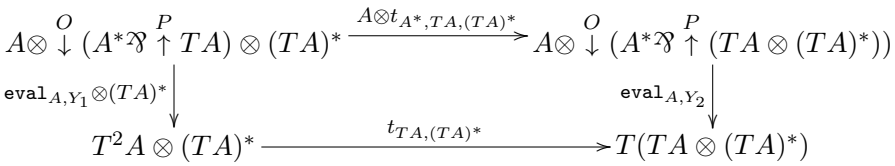
More precisely, this equation reduces to the commutative diagram



where

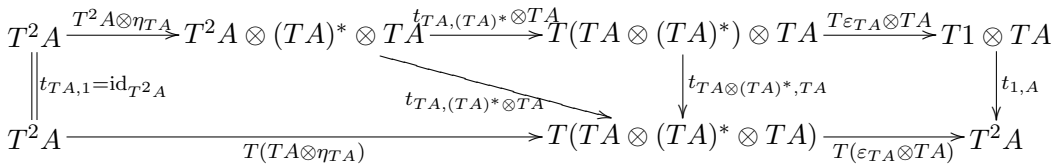
$$X_1 = \overset{O}{\downarrow}(A^*\parr \overset{P}{\uparrow} TA) \qquad X_2 = \overset{O}{\downarrow}(A^*\parr \overset{P}{\uparrow} 1)\otimes TA \qquad Y_1 = \overset{P}{\uparrow} TA.$$

The two squares on the top commute by naturality of the strength of the monad T. The bottom-left triangle commutes as a variation on the zigzag identity involving the tensorial negation. By expanding the definition of $\natural$ and using the naturality of the unit and counit coming from the compact closed structure, the commutation of the square involving `eval` can be reduced to the commutation of the square,



where $Y_2 = \overset{P}{\uparrow}(TA\otimes (TA)^*)$. In the same way, to see that the down-right triangle commutes, we have to come back to the definition of $\natural$. The commutation is then

reduced to the commutation of the diagram



The left square commutes by naturality of the strength, the triangle in the middle commutes by associativity of the strength, and the square on the right commutes again by naturality of the strength.

5.3 Constant cell and replication modality.

We describe below the strategy $\mathtt{const\_cell}_A$ which interprets the constant memory cell in which one can write only once, but read infinitely many times. We construct it by taking the image through ! of the morphism (7) obtained by composing an injection and the unit of the monad T

$$!A_i \longrightarrow !A = (!A_i)_{i\in I} \longrightarrow !\mathtt{Read}_A$$

instantiated at the family $A = (A_i)_{i\in I}$. We then apply the same transformation as for the linear memory cell, and obtain a morphism

$$1 \xrightarrow{\quad\mathtt{const\_cell}_A\quad} T(\mathtt{Write}_{!A} \otimes !\mathtt{Read}_A). \tag{8}$$

This strategy has only one writing stage, and then throws a new copy on the same thread each time Opponent asks for reading the value. The dereference method is given by the dereliction

$$\mathtt{deref}_A \stackrel{\text{def}}{=} \varepsilon_{TA} \quad : \quad !\mathtt{Read}_A \longrightarrow TA$$

while the constant assign method is given by

$$\mathtt{assign}_{\mathtt{const}\,A} \stackrel{\text{def}}{=} \mathtt{eval}^{\,P}_{!A,\uparrow 1} \quad : \quad !A \otimes \mathtt{Write}_{!A} \longrightarrow T1$$

Using the naturality of the dereliction of the replication modality, one easily establishes that the constant memory cell validates the first four equations of Figure 3 from the fact that the linear memory cell satisfies them.

5.4 Memory cell and temporality.

At this point, we are ready to describe the memory cell which can be written and read an infinite number of times. The most natural idea is to think of such a memory cell as an infinite number of constant memory cells. This gives rise to a strategy of type

$$1 \longrightarrow !(\mathtt{Write}_{!A} \otimes !\mathtt{Read}_A)$$

where the scheduling of each copy is explicit. The whole point of the memory cell is precisely to hide this management at the level of types. In order to achieve this, it is necessary to take advantage of the high level of control available in the strategies between Conway games, in order to define a *memory access* strategy

$$\xi_{\mathtt{Write}_{!A},\mathtt{Read}_A} \quad : \quad !(\mathtt{Write}_{!A} \otimes !\mathtt{Read}_A) \quad \to \quad !\mathtt{Write}_{!A} \otimes !\mathtt{Read}_A$$

whose task is to enforce that every time a read access is performed on the memory cell, it reads the last written value. We start by defining this strategy in the language of Conway games, before describing the series of algebraic properties it satisfies. It is worth observing already that since the strategy is not innocent (not even visible), there is no hope to define it using structural morphisms. Given a play s of the game $!A \otimes !B$

$$s \quad : \quad \star_{!A\otimes !B} \twoheadrightarrow (w_A, w_B)$$

where $w_A \in V_{!A}$ and $w_B \in V_{!B}$, we define the position of the game $!(A \otimes !B)$

$$(w_A, w_B) \downarrow s \quad \stackrel{\text{def}}{=} \quad (a_1, w_1) \cdots (a_n, w_n)$$

where $a_i \in V_A$ and $w_i \in V_{!B}$ for all $1 \leq i \leq n$, by requiring that

$$w_A \quad = \quad a_1 \cdots a_n \qquad\qquad w_B \quad = \quad w_1 \cdots w_n$$

and that the copies of the component B appearing in the position w_i have been opened by the play s after the copy of the component A corresponding to the position a_i, and before the copy of the component A corresponding to the position a_{i+1}. So, the transformation of (w_A, w_B) into $(w_A, w_B) \downarrow s$ consists in gathering together as w_i all the copies of the component B opened by the play s between the i-th and $(i+1)$-th copies of the component A, witnessed by the positions a_i and a_{i+1}. This scheduling enables us to define the memory access function

$$f \quad : \quad \mathrm{Play}_{!A\otimes !B} \to \mathrm{Play}_{!(A\otimes !B)}$$

as the unique function

$$s : \star_{!A\otimes !B} \twoheadrightarrow (w, w') \quad \mapsto \quad f(s) : \star_{!(A\otimes !B)} \twoheadrightarrow (w, w') \downarrow s$$

such that the plays s and $f(s)$ have the same underlying moves. This memory access function induces a memory access strategy

$$\xi_{A,B} \quad : \quad !(A \otimes !B) \to \; !A \otimes !B$$

defined as

$$\xi_{A,B} \quad \stackrel{\text{def}}{=} \quad \{s \mid \forall t \prec s \;,\; t_{|!(A\otimes !B)} = f(t_{|!A\otimes !B})\}$$

where s and t are even length paths of the game $!(A \otimes !B) \multimap (!A \otimes !B)$ and t is a prefix of s. We will also need to consider the strategy

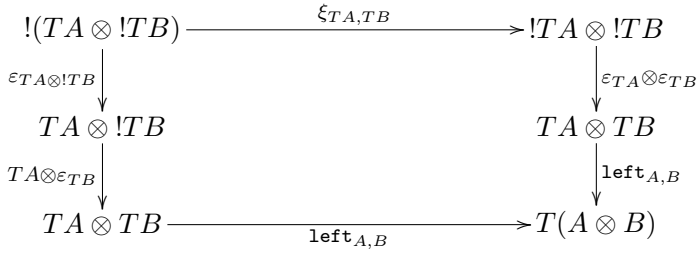
$$\alpha_A \; : \; A \otimes !A \; \longrightarrow \; !A$$

consisting of a copycat between the first opened copy of the game $!A$ on the right-hand side and the game A, followed by a copycat between the other copies of the game $!A$ and the game $!A$ on the left-hand side. We will now axiomatize the properties satisfied by the memory access strategy, properties that will be enough to show the validity of this strategy (for clarity, we omit explicit mention to the associativity in the monoidal category).

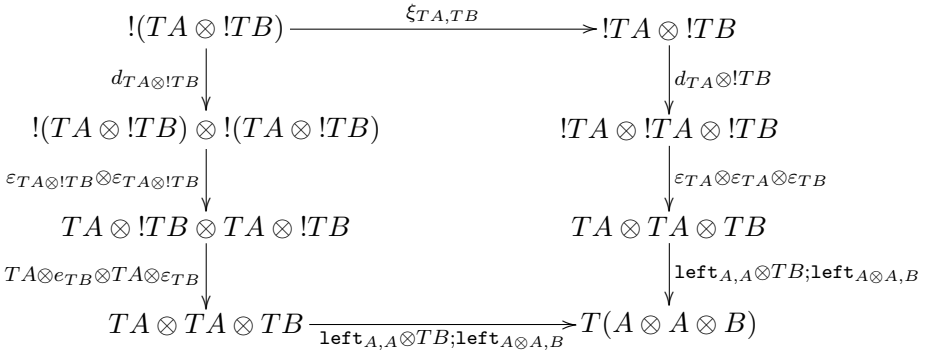
A *memory access modality* is defined as a replication modality ! equipped with two natural transformations ξ and α of component

$$\xi_{A,B} \ : \ !(A \otimes !B) \longrightarrow !A \otimes !B \qquad\qquad \alpha_A \ : \ A \otimes !A \longrightarrow !A$$

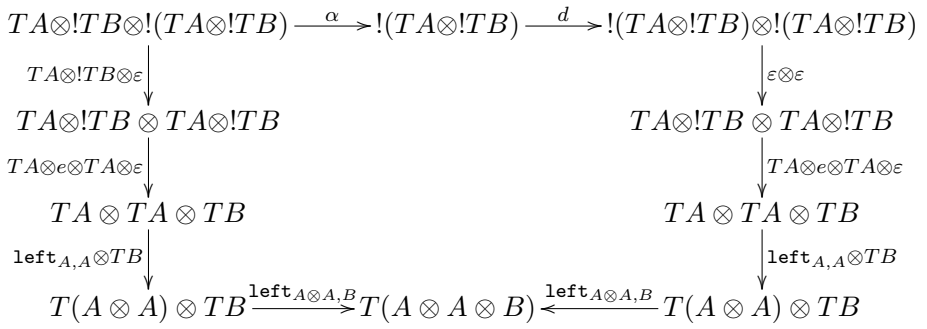
such that the diagrams



(9)



(10)



(11)

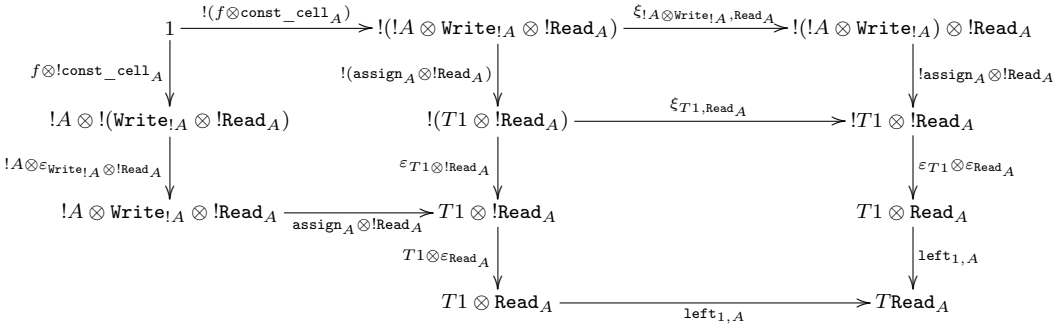
commute, where $\varepsilon_A : !A \to A$ denotes the unit and $d_A : !A \to !A \otimes !A$ denotes the multiplication of the replication comonad ! while e_A denotes the counit of the commutative comonoid $!A$. We let the reader check that the memory access strategy ξ satisfies the Diagrams (9) and (10) and that the strategy α satisfies the Diagram (11). This enables us to define the memory cell as the strategy

$$\mathtt{cell}_A \quad \stackrel{\text{def}}{=} \quad \xi_{\mathtt{Write}_{!A},\mathtt{Read}_A} \quad \circ \quad !\mathtt{const\_cell}_A$$

Coarsely speaking, Diagram (9) states that the memory cell behaves as the constant cell on the first copy. Diagram (10) states that the copy in B are linked to the last opened copy of A. It is worth stressing here the crucial use of the strength of the monad T in order to express the order in which the different copies are opened and played.

At this point, there remains to show that the notion of memory access is sufficient to guarantee that the strategy $\mathtt{cell}_A$ interpreting the memory cell, validates the five equations exhibited in the work of Abramsky, Honda and McCusker [2], see Figure 3. The first three equations are deduced from the same ingredients as for the linear or constant cell. Namely, we use properties of the tensor product and of the strong monad T.

Equation (4) is more difficult to derive, and follows essentially from Diagram (9). The proof is purely diagrammatic. The basic idea is to establish that for a unique assignment, the memory cell behaves as the constant memory cell, and then, to take advantage of the fact that we already know that the constant memory cell validates Equation (4). Technically speaking, the proof is based on the diagram



which is shown to commute for every morphism $f : 1 \to !A$. The left top square commutes by naturality of the counit ε, the right top square commutes by naturality of the memory access ξ, and the bottom square commutes as an instance of Diagram (9). This diagram states that a single use of the assign or dereference method induces the same interactive behaviour for the memory cell as for the constant cell.

Finally, Equation (5) follows essentially from our assumption that Diagram (10) commutes, which ensures that a memory cell always reads the last written value. Just as in the case of Equation (4), the proof is purely diagrammatic. The basic idea of the proof is to establish that for every pair of morphisms $f, g : 1 \to !A$, whenever one assigns the value f and then the value g to the memory cell, and then one reads

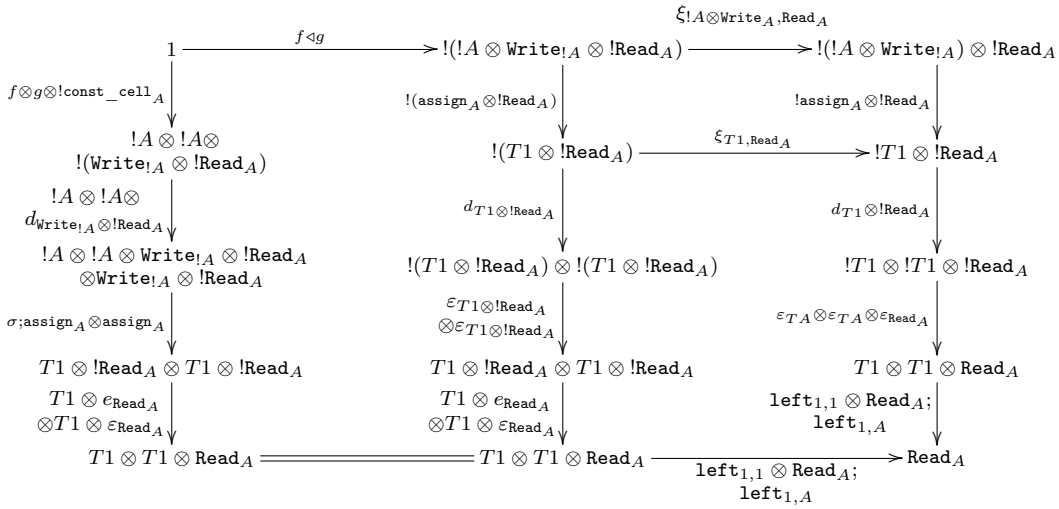
the cell, one gets the value g. In order to establish this fundamental property, it appears convenient to introduce the morphism

$$f \triangleleft g \quad : \quad 1 \quad \longrightarrow \quad !(!A \otimes \mathtt{Write}_{!A} \otimes !\mathtt{Read}_A)$$

defined as

$$f \triangleleft g \stackrel{\text{def}}{=} (f \otimes \mathtt{const\_cell}_A) \otimes !(g \otimes \mathtt{const\_cell}_A); \alpha_{!A \otimes \mathtt{Write}_{!A} \otimes !\mathtt{Read}_A}$$

in the category $Fam(\mathbf{Conway}^-)$. The proof is based on the following diagram



which is shown to commute for every pair of morphisms f and g: the right top square commutes by naturality of ξ, the right bottom square is an instance of Diagram (10) and we leave the reader check the left square postcomposed with the morphism

$$T1 \otimes T1 \otimes \mathtt{Read}_A \xrightarrow{\mathtt{left}_{1,1} \otimes \mathtt{Read}_A} T1 \otimes \mathtt{Read}_A \xrightarrow{\mathtt{left}_{1,A}} \mathtt{Read}_A$$

commutes, this fact itself following from our assumption that Diagram (11) commutes.

6 Conclusion and future works

We have defined a compact closed category of games, equipped with a notion of well-bracketing. This category offers an alternative way to think of the fully abstract model of the call-by-value language with higher-order store defined by Abramsky, Honda and McCusker. One interesting point about our approach is that the treatment of the memory cell is done diagrammatically, thus avoiding the direct and somewhat uncomfortable definitions and proofs in the original paper. We use only a fragment of our linear type system, corresponding to the traditional notion of "bracket" in arena games. We believe that this offers an opportunity to refine the

type system of the original programming language, in order to integrate the hugely extended notion of control provided by linear logic. It would be also interesting to analyze the models defined by McCusker [18] and Murawski [21] in order to avoid the bad variable constructor `mkvar` in the definition of the language. More generally, we believe that much remains to be done in order to understand the algebraic structure of memory in game semantics. Ideally, this algebraic analysis should connect the game models based on visible or thread-independent strategies studied in this paper, with the monadic approach to general references recently advocated by Tzevelekos [27] which is based on innocent strategies and a state monad defined by recursion on all types.

Acknowledgement

The two authors are grateful to Russ Harmer, Masahito Hasegawa, Martin Hyland and Luke Ong for stimulating discussions and encouragement.

References

[1] S. Abramsky. *Axioms for Definability and Full Completeness*. MIT Press, 1999.

[2] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general reference. In *13th IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1998.

[3] S. Abramsky and M. Lenisa. Axiomatizing Fully Complete Models for ML Polymorphic Types. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, volume 1893. Springer Lecture Notes in Computer Science, 2000.

[4] S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for idealized algol with active expressions, 1997.

[5] Samson Abramsky and Guy McCusker. Call-by-value games. In Mogens Nielsen and Wolfgang Thomas, editors, *6th Annual Conference of the European Association for Computer Science Logic*, volume 1414 of *Lecture Notes in Computer Science*. Springer, 1998.

[6] Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *3th Annual Conference of the European Association for Computer Science Logic*, volume 933 of *LNCS*, Poland, June 1995. Springer-Verlag.

[7] G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20:265–321, 1982.

[8] Russ Harmer. *Games and Full Abstraction for Nondeterministic Languages*. PhD thesis, University of London, 2000.

[9] Masahito Hasegawa. On traced monoidal closed categories. *Mathematical Structures in Computer Science*, 2008. Published online on 27 October 2008.

[10] Martin Hyland and Luke Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 163(2):285–408, December 2000.

[11] André Joyal. Remarques sur la théorie des jeux à deux personnes. *Gazette des Sciences Mathématiques du Québec*, 1(4):46–52, 1977. English version by Robin Houston.

[12] André Joyal, Ross Street, and Domnic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(447-468):184, 1996.

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

[14] J. Laird. A categorical semantics of higher order store. In *Proceedings, 9th Conference on Category Theory and Computer Science*, 2002.

[15] J Laird. A game semantics of names and pointers. *Annals of Pure and Applied Logic*, 2009.

[16] Paul Blain Levy. Global state considered helpful. In *Proceedings of Mathematical Foundations of Programming Semantics (MFPS 2008), Electronic Notes in Computer Science*, volume 218, pages 241–259, 2008.

[17] Peter May. Duality in bicategories and topological applications. A brief memorial talk for Saunders MacLane, april 2006. http://www.math.uchicago.edu/∼may/.

[18] Guy McCusker. On the semantics of Idealized Algol without the bad variable constructor. *Proceedings of Mathematical Foundations of Programming Semantics (MFPS 2003), Electronic Notes in Computer Science*, 83, 2003.

[19] Paul-André Melliès and Nicolas Tabareau. Resource modalities in game semantics. In *22th IEEE Symposium on Logic in Computer Science*, pages 389–398, 2007.

[20] Robin Milner. Action Calculi V: reflexive molecular forms. Unpublished, third draf, June 1994.

[21] A.S. Murawski. Bad Variables Under Control. In *21th Annual Conference of the European Association for Computer Science Logic*, 2007.

[22] C.-H. L. Ong and S. Sanjabi. Fully abstract semantics of additive aspects by translation. In *Proceedings of Sixth International Conference on Aspect-Oriented Software Development (AOSD 2007)*, 2007.

[23] Paul-André Melliès and Nicolas Tabareau and Christine Tasson. An explicit formula for the free exponential modality of linear logic. In *36th International Colloquium on Automata, Languages and Programming (ICALP)*, 2009.

[24] Uday Reddy. Global state considered unnecessary: An introduction to object-based semantics. *Journal of Lisp and Symbolic Computation*, 9:7–76, 1996.

[25] John Reynolds. Syntactic control of interference. *5th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 39–46, 1978.

[26] Nicolas Tabareau. *Modalités de ressources et contrôle en logique tensorielle*. PhD thesis, Université Paris Diderot, 2008.

[27] N. Tzevelekos. Full abstraction for nominal general references. In *22th IEEE Symposium on Logic in Computer Science*, pages 399–410, 2007.