

Time and Games

Benjamin Lepercqey

PPS, Université Denis Diderot

Abstract. We add the notion of time to denotational models of the λ -calculus. The denotation is no longer constant through reduction, but rather decreases with respect to an appropriate order. Categorically, we use a monad over a cartesian category, an order over the morphisms of the Kleisli category, and a Galois connection to model β -reduction. We define a generic monad (time as a resource), and an instance of this construction in game semantics, where our timings are precise enough to simulate parallelism through interleaving.

1 Introduction

PCF is by nature an extensional language: a context can only see the extensional behavior of a sub-term. All it can do is “run” it on some arguments. If one on these runs does not terminate, then the whole program is stuck.

In an interactive setting (like sequential algorithms or the \mathcal{B} algebra of Van Oosten and Longley [10]), the context does not give the arguments at once. It rather waits for the sub-term to ask for information about them. It can then answer in the normal way, or stop the computation and do something else (catch). Still, if the sub-term loops without querying information about its environment, the whole program is stuck again.

We want the context to have an even more tighter control on its sub-terms: we want to separate strictly terms, that is, find contexts $C_{M,N}[\]$ such that $C_{M,N}[M] = 0$ and $C_{M,N}[N] = 1$ for terms M, N . This is stronger than the usual notion of separation, where a context such that $C[M] \uparrow$ and $C[N] \downarrow$ is said to separate M and N , although this information cannot be used by a program: the non-termination information is not (for a program’s view) a result. With this stronger notion of separation, the context knows the term are different (they lead to different *values*), thus it can react differently.

Let us take some examples in *PCF* (we write Ω for the infinite loop):

$$\begin{aligned} A &= \lambda x.1 \\ B &= \lambda x.\text{if } x \text{ then } 1 \text{ else } 1 \\ C &= \lambda x.\text{if } x \text{ then } 1 \text{ else } \Omega \end{aligned}$$

In PCF, there is no separating context for A and B (because $A \gtrsim B$ for the observational preorder) nor for any other (because the context would not be monotonic). In PCF with control operators, the context can test whether the term uses its argument or not with *catch*, so one can separate A and B . Nevertheless, it cannot separate B and C (a context can never learn anything about

Ω , because Ω never “answers”). If we want to strictly separate B and C , we need to have communications between a term and its context even when the terms does not refer explicitly to its environment. We add “dummy” communications that come periodically: we introduce time in the model.

Let us give a more concrete (although informal) example in the setting of Hyland and Ong games: $\llbracket 1 \rrbracket = \{\epsilon, q1\}$ and $\llbracket \Omega \rrbracket = \{\epsilon\}$ (Ω never answers). We add \perp (“the result is not yet ready”), and \top (“try some more”). \perp must be a player move (since it is the term which asks for more time) and \top on opponent move (the context allows the term to run for more time). The denotation of Ω will be $\{\epsilon, q\perp, q\perp\top\perp, q\perp\top\perp\top\perp \dots\}$: Ω does not stop the computation, it only takes an infinite time to answer. We can now distinguish B and C .

We encounter a problem here: if we want $\Omega = \text{fix } \lambda f.f$ to use time, we must change the definition of the fixpoint. We rather want $\text{fix } \lambda f.M$ to be slightly different from $M[f \setminus \text{fix } \lambda f.M]$: it should be slower (use more time). If one wants to be even more precise, one might also want application to take time, so β -equivalence might no longer be true. Thus, one could ask what would be a model in this perspective. It seems that, when M reduces to N , although M and N do not have the same denotation, their behavior is similar: we put an order on terms (*a priori* different from the observational order).

1.1 Plan of the Paper

We are left with several questions:

- How does this fit into traditional semantics?
- How should one measure time, that is: where should one put the \perp/\top moves?
- What is the defining power of this extension?

In section 2, we show that time can be described as a monad over a cartesian closed category together with an order, such that the curry/uncurry operators make a Galois connection. We give a generic construction to build a model with time out of a Cartesian closed category. In section 3, we apply this construction to the games model, define real-time strategies and give the Galois connection that make all the denotations of λ -terms real-time. In section 4, we investigate the third question in our model: although it seems that if we stick to purely functional programming style (innocent and well-bracketed strategies), the model is essentially the same, references allow the separation of many terms (β -equivalence is lost), and control operators allow precise slicing and therefore interleaving of several computations. Combining both constructions, one can even synchronize threads and communicate values between them, all of this in a manner transparent for the threads.

1.2 Related Work

Escardo [5] introduced PCF+timeout, a language in which a context could let a term run for some time and stop it. Time is defined as a number related to the reduction path of the term (the number of steps of reduction, or the number of recursion unfoldings, etc.) Our description is different in that time is

defined within the model, rather than from the operational semantics of a precise language. Longley conjectured that Kleene’s model K_1 was fully abstract for PCF+timeout+catch. In our example in games semantics, we would rather try to exploit the factorization theorems (PCF+state, PCF+control...). The part about interleaving of computations is a reformulation of the trampolined style of Ganz, Friedman and Wand [6]: it is more straightforward in that our Kleisli-construction provide a natural setting for the “bounce” operation, no program transformation is needed here. Finally, the communications between threads are similar to the modeling of references by Abramsky, Honda and McCusker [1].

2 Categorical description

2.1 Requirements

The behavior of our programs can be divided in two parts: the value of the program, and the time taken by the computation. Therefore we embed values into computations via a monad, as in [11]. We need a cartesian category \mathcal{G} with:

- a triple $(T, \eta, -^*)$ or equivalently a monad (T, η, μ) ,
- a tensorial strength $t_{A,B} : (TA) \times B \rightarrow T(A \times B)$ (we write $\bar{t}_{A,B} : A \times (TB) \rightarrow T(A \times B)$ for the corresponding morphism),
- a bifunctor $- \Rightarrow -$ covariant on the right, contravariant on the left.

In [11], the categories are cartesian closed. Here, we do not want closure (as it would collapse the β -equivalent terms), but only an ordering property. This is reminiscent of the categorical description of rewriting of λ -terms by Hilken in [8]: β -reduction makes the term smaller, and η -expansion leaves it unchanged. For the sake of simplicity, we do not give here the definitions in the 2-category. We need:

- an order \preceq on $\mathcal{G}(A, TB)$ compatible with $(\cdot)^*$ and $t_{A,B}$ in the following sense:

$$\forall a, a', b, \begin{cases} a \preceq a' \implies a^* \preceq (a')^* \\ a \preceq a' \implies b^* \circ a \preceq b^* \circ a' \\ a \preceq a' \implies t_{A,B} \circ \langle a, b \rangle \preceq t_{A,B} \circ \langle a', b \rangle \end{cases} \quad (1)$$

- two monotonic transformations $\varphi : \mathcal{G}(A \times B, TC) \rightarrow \mathcal{G}(B, T(A \Rightarrow C))$ and $\psi : \mathcal{G}(B, T(A \Rightarrow C)) \rightarrow \mathcal{G}(A \times B, TC)$ inducing a Galois connection, natural in the following sense: for all $f : A \times B \rightarrow TC$, $\sigma : B \rightarrow T(A \Rightarrow C)$ and $b : B' \rightarrow B, c : C \rightarrow C'$:

$$\begin{aligned} \varphi(Tc \circ f \circ (A \times b)) &= T(A \Rightarrow c) \circ \varphi f \circ b \\ \psi(T(A \Rightarrow c) \circ \sigma \circ b) &= Tc \circ \psi \sigma \circ (A \times b) \\ (\varphi f)^* &= \varphi(f^* \circ \bar{t}_{A,B}) \\ \psi(f^*) &= (\psi f)^* \circ \bar{t}_{A,B} \end{aligned} \quad (2)$$

Lemma 1. *If η is epi, the first two properties the latter ones.*

2.2 Interpretation of the simply-typed lambda-calculus

We define our category \mathcal{C} as the Kleisli category over \mathcal{G} : the objects are those of \mathcal{G} , $\mathcal{C}(A, B) = \mathcal{G}(A, TB)$, $g \bullet f = g^* \circ f = \mu \circ Tg \circ f$, and the identity on A is η_A .

Definition 2. We interpret the sequents $x_1 : A_1 \dots x_n : A_n \vdash M : A$ of the simply typed lambda-calculus as (for the sake of clarity, we give the arrows in \mathcal{G} , they actually appear as natural operations in \mathcal{C}):

$$[[A_1]] \times ([[A_2]] \times \dots \times ([[A_n]] \times 1) \dots) \xrightarrow{[[M]]^{x_n \dots x_1}} T[[A]]$$

where

- $[[x_i]]x_n \dots x_1 = \eta \circ \pi_1 \circ (\pi_2)^{i-1}$,
- $[[M]N]\rho = (\psi[[M]]\rho)^* \circ t_{A,B} \circ \langle [[N]]\rho, B \rangle$
- $[[\lambda x.M]]\rho = \varphi([[M]]\rho x)$.

Lemma 3. $[[M[x \setminus N]]]\rho = ([[M]]\rho x)^* \circ t_{A,B} \circ \langle [[N]]\rho, B \rangle$

The proof is very technical. All the required properties of φ, ψ are needed except the fact that is is a Galois connection, which be used in the following corollary:

Proposition 4. $[[\lambda x.M]N]\rho \succeq [[M[x \setminus N]]]\rho$

Proposition 5. Application and abstraction are monotonic for \succeq .

2.3 PCF

We recall the definition of PCF:

| | | |
|--------------------------|-------------|--|
| $t ::= x, y \dots$ | Variable | $(\lambda x.M)N \rightsquigarrow M[x \setminus N]$ |
| $\lambda x.t$ | Abstraction | if 0 then M else $N \rightsquigarrow M$ |
| $(t)u$ | Application | if $n + 1$ then M else $N \rightsquigarrow N$ |
| $0, 1 \dots$ | Constants | $\text{succ } n \rightsquigarrow (n + 1)$ |
| $\text{succ } t$ | Successor | $\text{pred } (n + 1) \rightsquigarrow n$ |
| $\text{pred } t$ | Predecessor | $\text{fix } M \rightsquigarrow (M)\text{fix } M$ |
| if t then u else v | Conditional | $M \rightsquigarrow M' \implies (M)N \rightsquigarrow (M')N$ |
| $\text{fix } t$ | Fixpoint | $M \rightsquigarrow M' \implies \text{succ } M \rightsquigarrow \text{succ } M'$ |
| | | $M \rightsquigarrow M' \implies \text{pred } M \rightsquigarrow \text{pred } M'$ |
| | | $M \rightsquigarrow M' \implies \text{if } M \text{ then } N \text{ else } P$ |
| | | $\rightsquigarrow \text{if } M' \text{ then } N \text{ else } P$ |

To have model for full PCF, we need (in the Kleisli category):

- an object N with arrows $\underline{n} : 1 \rightarrow N$,
- $\text{succ}, \text{pred} : 1 \rightarrow N \Rightarrow N$ such that $\text{succ} \circ \underline{n} \succeq \underline{n+1}$ and $\text{pred} \circ \underline{n+1} \succeq \underline{n}$,
- an arrow $\text{if} : N \times (N \times N) \rightarrow N$ such that $\text{if} \circ \langle \underline{0}, \langle p, q \rangle \rangle \succeq p$ and $\text{if} \circ \langle \underline{n+1}, \langle p, q \rangle \rangle \succeq q$,
- a transformation Y that maps arrows of $\mathcal{C}(\Gamma, A \Rightarrow A)$ to arrows of $\mathcal{C}(\Gamma, A)$, such that $Yf \succeq \psi f \circ \langle Yf, \Gamma \rangle$.

We then define the denotation of a term in the obvious way. The choice of the arrows reflect time taken by primitives, tests and recursion unfoldings. These conditions are needed to have the following property:

Proposition 6. *If $M \rightsquigarrow^* N$ and the free variables of M, N are in ρ , then $\llbracket M \rrbracket \rho \succeq \llbracket N \rrbracket \rho$: a reduced term runs faster than a long form.*

2.4 The Canonical Time Monad

In this section, we define a monad on any CCC. The definitions for the order and the Galois connection are left (as they seem to depend on the nature of the CCC). Nevertheless, we can prove general, interesting results about these models. We assume in the following that we have a CCC \mathcal{G} : its cartesian product is written \times , its curryfication C and its uncurryfication U .

If we think of time as a resource given by the context to the term, it is natural to turn terms of type A into terms of type $T \Rightarrow A$, that is functions that take time (of type T) and give a value of type A . It is easy to check that the following gives a monad and a tensorial strength ($\delta_T : T \Rightarrow T \times T$ is the diagonal morphism):

$$\begin{aligned}\mu &= (\delta_T \Rightarrow A) \circ C(UU id_{T \Rightarrow T \Rightarrow A} \circ \alpha) \\ \eta &= C\pi_2 \\ t_{A,B} &= C(U(T \Rightarrow A) \times B)\end{aligned}$$

All that is left to define in order to have a model of PCF is what time is (the object T), what “faster” means (the order \preceq on $T \Rightarrow _$) and what uses time: the Galois connection (time taken by application), the fixpoint transformation (time taken by recursion unfoldings), and the arrows for the constants and primitives (time taken by primitives).

We add some conditions on T and \preceq (remember that \mathcal{C} is the Kleisli category built over \mathcal{G}):

- there exists an arrow $\text{tick} : 1 \rightarrow T$ in \mathcal{G} (time can be “created” in \mathcal{G}),
- we define, for $f \in \mathcal{C}[A, B]$, $\pi f = \psi f \circ \langle \text{tick}, A \rangle$, and require that $f \preceq g$ only if $\pi f = \pi g$.

The second condition means that \preceq is only about the time part of f, g . It implies that if M reduces to N , then $\pi \llbracket M \rrbracket = \pi \llbracket N \rrbracket$, so the image by π of the model (with time) in \mathcal{C} is a model (without time) in \mathcal{G} . More precisely:

Theorem 7. *Contextual separation is essentially the same as in \mathcal{G} : for all closed PCF terms $M, N : A$, if $\pi \llbracket M \rrbracket = \pi \llbracket N \rrbracket$, then for all contexts C , $\pi \llbracket C[M] \rrbracket = \pi \llbracket C[N] \rrbracket$.*

Proof. It is a consequence of the Kleisli construction. We make an induction on C , the only interesting case being the application. Since a context applied to a term sees only the “extensional” part of the term A , it cannot see the time

requests T_M , it can only forward it:

$$\begin{array}{ccc}
\Gamma & \xrightarrow{\llbracket (P)M \rrbracket} & T_{M,P} \Rightarrow C \\
\downarrow \text{to}\langle \llbracket M \rrbracket, \Gamma \rangle & & \uparrow \mu \\
T_M \Rightarrow (A \times \Gamma) & \xrightarrow{T_M \Rightarrow \llbracket P \rrbracket} & T_M \Rightarrow T_P \Rightarrow C
\end{array}$$

Theorem 7 shows that we need to introduce a specific construction for time inside the language in order to use time. We add the typing rules:

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : T \Rightarrow T}{\Gamma \vdash \text{timing } M N : A} \quad \frac{\Gamma \vdash M : T \quad \Gamma \vdash N : A}{\Gamma \vdash M ; N : A}$$

timing M ($\lambda\tau.N$) means that we execute M , replacing the times ticks by calls to N . N takes the “normal” tick (from the context) and gives a tick back. We use a separate type for ticks, with no constructor, so that the only value that N can return is τ^1 . Of course, it might do something before returning τ : raise an exception, update a reference, etc. We define $\llbracket \text{timing } M N \rrbracket_\rho$ accordingly, as the composition (in the λ -calculus sense) in the original CCC \mathcal{G} : we have $\llbracket M \rrbracket : \Gamma \rightarrow T \Rightarrow A$ and $\llbracket N \rrbracket : \Gamma \rightarrow T \Rightarrow T$, so $U\llbracket M \rrbracket \circ \langle U\llbracket N \rrbracket, \pi_2 \rangle : T \times \Gamma \rightarrow A$. We set

$$\llbracket \text{timing } M N \rrbracket = C(U\llbracket M \rrbracket \circ \langle U\llbracket N \rrbracket, \pi_2 \rangle)$$

We also require the model to provide a morphism $\text{seq} : T \times A \rightarrow A$ to implement the semicolon.

How the terms of PCF+**timing** will behave is tightly linked to the precise usage of T . We need to complete our model to give precise semantics to PCF+**timing**. Although some elements are missing, we can give the generic intuition behind the monad and the **timing** operator. We translate the terms of PCF+**timing** into terms of PCF with a special type T :

- a term $M : A$ is turned into $\tilde{M} = \lambda\tau.M' : T \Rightarrow A$, where occurrences of τ in M' represent time ticks,
- \tilde{x} does not use time,
- **timing** $M N$ becomes $\lambda\tau.(\tilde{M})(\tilde{N})\tau$,
- the other constructions add or remove time, by adding or removing occurrences of τ .

Of course, given the actual Galois connection and arrows for primitives, the last sentence can be made more precise. In the next section, we introduce a Galois connection that puts (roughly) an occurrence of τ on each β -redex: $(\lambda x.x)(\lambda y.y)0$ becomes $\lambda\tau.(\lambda x.\tau;x).(\lambda y.\tau;y)0$. We reduce it by β -reduction, and we get $\lambda\tau.\tau;\tau;0$: this is the value 0, computed in two steps.

A natural choice for T would be the unit type of ML: we count ticks, which carry no intrinsic value. We would still want T to be a special type: in a language with a unit type, there should be no confusion between commands (of type

¹ or another τ' given in the same fashion by some enclosing **timing** _ _

unit) and time ticks (of type T). Nevertheless, one could imagine T to be more complex, For example, with $T = N$, the possibilities for the timing function would be much richer: one could “create” time (giving back constants), compute things through the ticks, etc. We focused here on the simplest choice for T , which allows many interesting variations.

In this light, the projection π only amounts to applying the translated term to $() : T$. For our example $(\lambda\tau.\tau; \tau; 0)() = (); () ; 0 = 0$.

In [5], Escardo defines precisely the operational semantics, then defines time as the number of steps of reduction (or the number of recursion unfoldings, etc.), then he defines the model according to this particular timing. The problem is, if we change the *presentation* of the operational semantics (say, split a rule in two), then we change the whole meaning of the term. In the present paper, we chose to define time inside the model. We believe that the timings we get this way are more natural, since the reduction rules of PCF do not reflect at all the real implementation of functional languages, and the aim of the models of sequentiality is precisely to describe the behavior of terms using only “low-level” operations. It would of course still be better to give operational semantics corresponding to the model, for instance through the above translation.

In the sequel, we give an instance of this construction in the games model.

3 Category of timed games

3.1 Arenas, plays and strategies

We use the variant of Hyland and Ong games described in [7]: it allows a modular description with constraints on strategies (as in [2] and contrary to [9]), without the categorical description of the exponentials (as in [2]), which would require both a co-Kleisli construction for the ! comonad and the Kleisli construction for the time monad. Thus we have a uniform presentation for different classes of strategies (innocent, well-bracketed, etc.) in a relatively simple setting.

Definition 8. *An arena is a triple $\langle M_A, \lambda_A, \vdash_A \rangle$ where*

- M_A is a set of moves,
- λ_A is the labeling function, from M_A to $\{O, P\} \times \{Q, A\}$ (opponent, player, question, answer),
- \vdash_A is the enabling relation, such that
 - if $n \vdash_A m$ and $n \neq m$, then $\lambda_A^{O,P}(n) \neq \lambda_A^{O,P}(m)$,
 - if $m \vdash_A m$, then $\lambda_A(m) = OQ$ and there is no $n \neq m$ such that $n \vdash_A m$ (m is an initial move),
 - if $n \vdash_A m$ and m is an answer, then n is a question.

Definition 9. *A justified sequence in an arena A is a finite sequence on M_A , together with a pointer from each move m to a preceding move n , such that $n \vdash_A m$. The hereditary justifier of a move m is the unique initial move m_0 such that there exists a sub sequence $m_0 m_1 \dots m_k$ where m_i justifies m_{i+1} and $m = m_k$.*

A legal play is a justified sequence where player and opponent alternate. We write L_A the set of legal plays.

Definition 10. If sm has odd length, the current thread $\lceil sm \rceil$ is the subsequence of moves hereditarily justified by the hereditary justifier of m (the connected component of m). If sm has even length, sm is well-threaded at m if the justifier of m occurs in $\lceil s \rceil$. A play is well-threaded if it is well-threaded at each player move. The current thread of a legal play is also a legal play.

The player view is defined by:

- $\lceil sm \rceil = m$ if m is an initial move,
- $\lceil smtn \rceil = \lceil s \rceil mn$ if n is an O -move and m justifies n ,
- $\lceil sm \rceil = \lceil s \rceil m$ if m is a P -move.

A play $sm \in L_A^{even}$ is visible at m if the justifier of m occurs in $\lceil s \rceil$. It is visible if it is visible at each player move.

The product and arrow arenas are defined in the usual way. We define the lifted arena A_\perp :

- $M_{A_\perp} = \{q, a\} + M_A$,
- $\lambda_{A_\perp}(q) = OQ$, $\lambda_{A_\perp}(a) = PA$, $\lambda_{A_\perp}(m) = \lambda_A(m)$ otherwise,
- $m \vdash_{A_\perp} n$ iff $m = n = q$ or $m = q, n = a$ or $m = a, n \vdash_A n$ or $m \neq n, m \vdash_A n$.

Definition 11. A strategy σ over an arena A is a non-empty, even-prefix closed set of even-length legal plays over A such that if $sab, sac \in \sigma$, then $sab = sac$ (determinism).

Definition 12. A strategy is visible if whenever $sab \in \sigma$, then b is justified in $\lceil sa \rceil$. A strategy is well-bracketed if the player answers are justified by the last unanswered opponent question in the view. A strategy is single-threaded if all plays of σ are well-threaded and player moves depend only on the current thread: if $sab, t \in \sigma$ and $\lceil sa \rceil = \lceil ta \rceil$, then there exists a unique way to extend ta with b such that $\lceil sab \rceil = \lceil tab \rceil$. A visible strategy is innocent if player moves depend only on the view: if $sab, t \in \sigma$ and $\lceil sa \rceil = \lceil ta \rceil$, then there exists a unique way to extend ta with b such that $\lceil sab \rceil = \lceil tab \rceil$.

Theorem 13. Arenas and single-threaded (resp. innocent) strategies form a CCC. Well-bracketed strategies induce sub-CCCs.

3.2 The Time Monad

Since \mathcal{G} is cartesian closed, we can apply the results of the previous section, to get a monad with a tensorial strength, but we still need to define an ordering, a Galois connection, the fixpoint transformation and the arrows for the constants. One easily checks that, whatever choice we make for T , η is epi: the strategy never plays in T , so the arena *always* stays empty. Eventually, we choose T to be the arena 1_\perp (two moves: one initial opponent question q , which justifies the player answer a).

Definition 14. Let s, t be plays of $\Gamma \Rightarrow T \Rightarrow A$: we define $s \preceq t$ iff

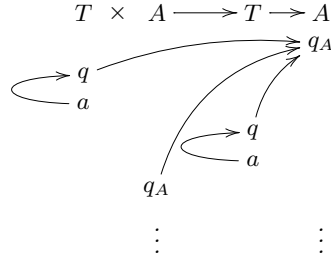
- $s \upharpoonright \Gamma, A = t \upharpoonright \Gamma, A$, and

- s is a sub-sequence of t i.e. there exists an injective monotonic map $\phi : \mathbb{N} \rightarrow \mathbb{N}$ such that $s_i = t_{\phi_i}$ and if s_i justifies s_j then t_{ϕ_i} justifies t_{ϕ_j} .

This means that t is s where some moves were inserted in T . This extends to an order on the strategies of $\Gamma \rightarrow T \Rightarrow A$: $\sigma \preceq \tau$ if σ is Egli-Milner smaller than τ : $\forall s \in \sigma, \exists t \in \tau, s \preceq t$ and $\forall t \in \tau, \exists s \in \sigma, s \preceq t$.

Lemma 15. \preceq is compatible with composition and with the monadic constructions (equations 1).

Lemma 16. $\text{seq} : T \times A \rightarrow A$ is the sequence operator:



Up to here, the definitions were quite canonical. There is some freedom in the choice of $\varphi, \psi, Y, \text{if}, \text{succ}$ and pred , depending on what one wants to observe. This is like in [5], in which several models/languages are given, according to what “events” are counted: each reduction step, or only recursion unfoldings. In the light of our translation into PCF with an extra argument, it is actually even closer to [6], where the yields are explicitly put in the program. Here, we choose where to put the occurrences of τ . One can think of time resolution being more or less fine-grained.

In the present paper, we give the somehow most precise (although certainly not the most realistic) timing of strategies: we control exactly the number of moves played between two moves by the context.

3.3 Real-time strategies

Definition 17. A strategy $\sigma : A \Rightarrow B$ is a real-time strategy if player never answers to an opponent move in A with a move in A .

The intuition (which is an easy consequence of the above restriction) is the strategy never plays twice in a row in A : it must make a move in B .

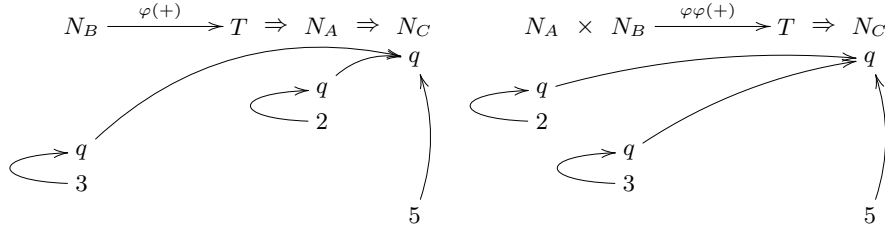
Proposition 18. The composition, product of real-time strategies are also real time, and so are the identities and the projections. Thus, the arenas with the real-time strategies make a sub cartesian category \mathcal{G}_{rt} of \mathcal{G} .

With our restriction on the plays, composition is in fact much simpler: there are always at most two moves played in the center arena between moves in the outside arenas. This is a kind of soundness property: we cannot hide a long computation by composition.

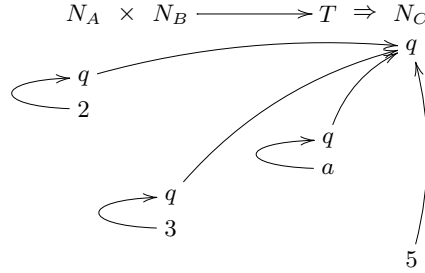
Proposition 19. *The monadic morphisms η and μ and the tensorial strength t are real-time, and if σ is real-time, then so is $T \Rightarrow \sigma$. Thus, we can build the Kleisli category \mathcal{C} over \mathcal{G}_{rt} .*

3.4 The Galois Connection

We give two maps such that φ and ψ are monotonic and that $\psi\varphi f \succeq f$ and $\varphi\psi f \preceq f$. The idea is simple: the usual curry/uncurry operations in games only amounts to a relabeling of moves. Here the problem is that in general the plays we get by uncurrying do not respect the new condition (they play several times in a row on the left side). For example, the plus operation:



But thanks to the monad, there is always a possible player move on the right side: ask in T .



On the contrary, we can “compress” the curried version of a play, by removing the two moves in T that come right before a move in N_A .

Let us be more formal. First, remember that U is the usual uncurryfication and C is the curryfication. In the rest of the section, σ is a strategy of $B \rightarrow T \Rightarrow A \Rightarrow C$ and τ is a strategy of $A \times B \rightarrow T \Rightarrow C$. The sequences we get by applying U , are not real-time. If m is a move in A , we call $I(m)$ the initial move in C that hereditarily justifies it. We define θs as $U(s)$ where all opponent moves m_A in A are replaced by $q_T a_T m_A$ where q_T is justified by $I(m_A)$.

We define $\psi\tau$ as the closure of $\{\theta t \mid t \in \tau\}$ by pair prefix.

Proposition 20. *If τ is a strategy of $B \Rightarrow (T \Rightarrow A \Rightarrow C)$, then $\psi\tau$ is a real-time strategy of $(A \times B) \Rightarrow T \Rightarrow C$. Moreover, if τ is innocent (resp. well-bracketed), so is $\psi\tau$.*

We define φ the opposite way: we remove the moves in T occurring just before a move in A . The visibility condition ensures that we do not “mix” several

threads: if $u = sq_T a_T s' m_A$ is a play, we know that m_a is justified in $\lceil sq_T a_T s \rceil = \lceil s \rceil q_T a_T s$, so it must be in the same “thread” as q_T .

Now, let s be a play of $A \times B \Rightarrow T \Rightarrow C$. We transform $C(s)$ by removing all sub-sequences $q_T a_T$ occurring just before an opponent move in A : we call ρs the resulting sequence.

We need to be careful here: we do not want to transform plays that end with q_T because we do not know yet if we have to remove it or not. We need to “look ahead” a little: we define $\varphi\sigma$ as the closure by pair prefix of $\{\rho s \mid s \in \pi\sigma\}$, where $\pi\sigma$ is the set of $s \in \sigma$ such that

- s does not end with q_T or
- s ends with q_T and $sa_T q_T \in \sigma$.

Proposition 21. *if σ is a strategy of $A \times B \Rightarrow (T \Rightarrow C)$, then $\varphi\sigma$ is well defined, and it is a strategy of $B \Rightarrow (T \Rightarrow A \Rightarrow C)$. Moreover, if σ is innocent (resp. well-bracketed), so is $\varphi\sigma$.*

Lemma 22. *ψ and φ respect the naturality conditions (equations 2).*

Proof. Since the only transformation on A is the identity, the moves in A are exactly the same on both sides of the equality, so the added (resp. deleted) moves are the same too.

Proposition 23. *$\sigma \preceq \psi\varphi\sigma$, and $\varphi\psi\sigma = \sigma$, so φ, ψ define a Galois connection.*

Proof. Let σ be a strategy of $A \times B \Rightarrow (T \Rightarrow C)$. Let s be a play of σ . ρs is $U(s)$ where some move were added. These moves and only these moves are deleted by θ , so $\theta\rho s = s$. On the contrary, if $s \in f$, we prove that $\rho\theta s \succeq s$. θ deletes some moves of s , all of which are put back by ρ . But, in the general case, moves in A do not come after moves in T , so θ deletes less than ρ adds, and $\rho\theta s \succeq s$.

Note that there is an obvious dual definition: put the moves in T after (and not before) moves in A . It is a bit easier to show that curryfication yields a deterministic strategy (since we do not have to look ahead), but our Galois connection corresponds to a more natural transformation of the λ -terms.

3.5 PCF constants

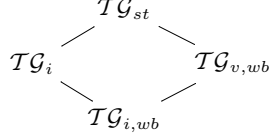
Definition 24. *We define succ , pred , and if as the smallest real-times strategies which meet the requirements.*

succ and pred are as expected. if is different from the usual strategy in that it has to ask for time after it got the tested value and before it asks the **then** or **else** branch.

We define the fixpoint in the usual way: given a strategy $\sigma : \Gamma \rightarrow A \Rightarrow A$, when define a chain of strategies $f_i : \Gamma \Rightarrow A$ (for the inclusion ordering): $f_0 = \emptyset$ and $f_{i+1} = \psi\sigma \circ t \circ \langle f_i, \Gamma \rangle$. We finally take $Y\sigma = \bigcup f_i$.

We have all the elements needed to prove the following theorem:

Theorem 25. *The games and the single-threaded (resp. innocent, visible and well-bracketed, innocent and well-bracketed) real-time strategies are a model with time of PCF. We have the usual models hierarchy:*



4 Defining power of the real-time strategies

It seems that the model of innocent and well bracketed real-time strategies does not allow more separation than the original innocent and well bracketed strategies model: although two β -equivalent terms can have a different denotation (timing $_ \lambda\tau.\Omega$ returns the result if it took no time to compute it, and loops otherwise), this information is about observational ordering, which a purely functional context cannot use. We were not able to formally prove it, though.

4.1 Non innocent strategies

If we allow non-innocent strategies, we can count the ticks. Something like:

$$\text{new } x = 0 \text{ in timing } M (\lambda\tau.x := x + 1; \tau); \text{deref } x$$

With this “profiling” operation, we can separate terms that compute the same value in a different number of steps, like 1 and $(\lambda x.x + 1)0$.

4.2 Non Well-Bracketed Strategies and Interleaving

If we relax the well-bracketedness condition, we can stop the computation when the term uses one of its arguments. In particular, if we can stop it when it uses time, we can separate terms that take no time to compute from the others.

More interesting, in a language with sums, we can stop a computation and restart it where it was stopped: we define the strategy $\text{catch}_{A,C}$ of type $((A \Rightarrow B) \Rightarrow C) \Rightarrow (A + C)$ where C is a flat type (like N or 1): $\text{catch}_{A,C} M$ gives the result (of type C) if M does not use its argument, or the argument of type A given to it. Figure 4.2 shows its maximal threads (the dots stand for the copycat strategy). We compose this with the proper innocent and well-bracketed strategy to get a strategy catch_A of type $((A \Rightarrow B) \Rightarrow C) \Rightarrow (T \Rightarrow A)$: it behaves as above if it catches a value of type A and loops infinitely otherwise (asking again and again in T). Now, we define for $M : N$, $M' : T \Rightarrow N$:

$$\begin{aligned}
 \text{embed } M &= \lambda\tau.\text{timing } M (\lambda\tau'.\tau) \\
 \text{step } M' &= \text{catch}_{T \Rightarrow N, N} \lambda\alpha.\text{timing } M' (\text{catch}_T \lambda\beta.\alpha\beta)
 \end{aligned}$$

Intuitively, if we apply step to $M : T \Rightarrow N$, it answers either that the computation is not finished, giving the remainder, or that it is finished, giving the result.

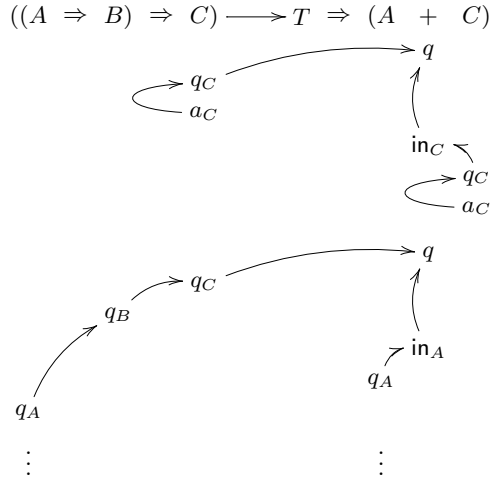


Fig. 1. The strategy catch

We have all the tools we need to implement schedulers: we can run term for a time slice, then do something else, then restart the term where it was stopped and so on. This is quite similar to the trampolined style of Ganz, Friedman and Wand [6]. Actually, the types are just the same, but thanks to the Kleisli construction, the thunks do not appear explicitly in the term: we do not need a transformation, all the embedding is done inside the model. Following their construction we have several schedulers:

- see-saw: alternates between two (or more) continuations, to allow simulation of parallelism through interleaving,
- more complicated cases with dynamic thread creation etc.

In the sequel, we write `parallelMN` the term implementing the interleaved execution of M and N with the see-saw scheduler.

4.3 Delay, Synchronization, Communication

Thanks to the structure of the model, we can make delays completely transparent: the term tries to read some variable, which asks for time for a while before returning the value. A term without explicit reference to time *does not know*, this is completely internalized in the model: it just passes the ticks between the variable and the context. In the same fashion, it is easy to synchronize two threads with the help of non-innocent strategies: let `semaphore` be the strategy corresponding to the following pseudo-ML code (in \mathcal{G} , it has type $T \Rightarrow (\text{com} \times \text{com})$, where $\text{com} = 1_{\perp}$, so, in the Kleisli category, it has type $\text{com} \times \text{com}$):

```
let x = ref 0 in lambda tau.
  (incr x; while x<>0 do tau done, decr x; while x<>0 do tau done)
```

Now, we can use the two different sides for two threads. When one of them is asked, it loops asking for time (which, again, unless explicitly specified through the use of `timing _`, *the context does not see*) until the other side is asked too, that is until the other thread has reached the synchronization point. Let us take an example, again in pseudo-ML:

```
let sync1, sync2 = semaphore tau in
parallel (...; sync1; ...) (...; sync2; ...)
```

`sync1` and `sync2` are plain variables in the terms, which do not even need to know that they are actually synchronizations. When the computation reaches one of them, it waits (asking for more time) until the other is reached too, then they return `()` at their next time tick, and the computation can continue.

Actually, in order to have a single-threaded strategy, we need to lift the arena $\text{com} \times \text{com}$. A typical play of semaphore would then look like figure 2. We cannot compose it directly with strategies arising from term, since it is a lifted form: it would fit much better in a call-by-value description, but the categorical description of another model with two monads was out of the scope of this extended abstract. We would need special construct in our call by name language.

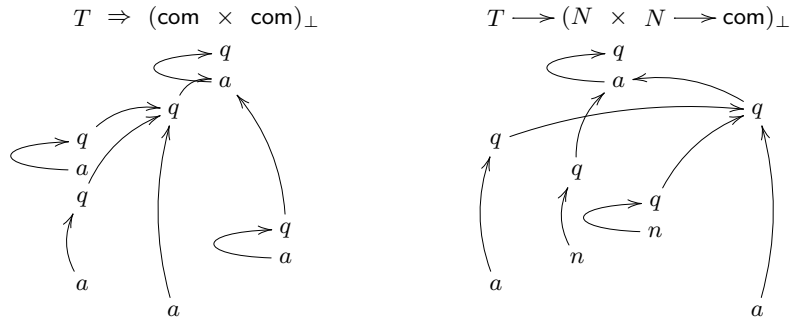


Fig. 2. Typical plays of semaphore and channel

Very easily, we can pass values on this channel: replace $\text{com} \times \text{com}$ by $A \times (A \rightarrow \text{com})$: a couple of values, one that reads a value of type A on the channel, and one that writes a value of type A on the channel and returns com : the strategy channel waits for both sides to be ready, then act as the identity between both A s. Figure 2 describes a typical play. Again, the latency is hidden unless the term explicitly uses the non-functional feature `timing`.

All of this looks a lot like the cell strategy of Abramsky, Honda and McCusker in [1]. The main difference is that we do not stop if the value is first read (we wait until it is initialized) and that we do not answer immediately to a `write` command (we wait until there is a corresponding `read`). Actually, using a variant of their cell strategy, one can implement a multiple-communication channel (with the help of dedicated `read` and `write` constructs).

5 Conclusion and further work

We introduced a notion of time in the λ -calculus like languages, and defined the notion of model for such a language. Although we presented one particular model in the present paper, it appears that there are many other possibilities:

- another monad (but $T \Rightarrow _$ seems to be the most natural choice),
- we chose the example of games semantics, but there must be equivalent presentations in any interactive setting, maybe even in static settings (such as hypercoherences with the multiset exponentials, see [4, 3])
- what we measure: we give here a Galois connection that make the denotation real-time, but one could choose other notions of time, modifying the semantics accordingly (put time on recursion, on primitives etc.)

We showed that non well-bracketed strategies allowed the simulation of parallelism through interleaving, and that combined with references, one could synchronize threads and pass values. The natural extensions would be to allow non determinism in strategies, or to encode the full π -calculus. There are many open problems about the intrinsic power of such a description: do the factorization theorems for non innocent or non well-bracketed strategies apply to this model? What is the exact separating power of the non innocent strategies?

References

1. Samson Abramsky, Kohei Honda, and Guy McCusker. A fully abstract game semantics for general references. In *Proceedings, Thirteenth Annual IEEE Symposium on Logic in Computer Science*, 1998.
2. Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Information and Computation*, 163:409–470, 2000.
3. Nuno Barreiro and Thomas Ehrhard. Anatomy of an extensional collapse. 1997.
4. T. Ehrhard. Hypercoherence: A strongly stable model of linear logic. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 83–108. Cambridge University Press, 1995. Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993.
5. Martin H. Escardó. A metric model of PCF. Laboratory for Foundations of Computer Science, University of Edinburgh. Unpublished research note, presented at the Workshop on Realizability Semantics and Applications, 1999, April 1998.
6. Steven E. Ganz, Daniel P. Friedman, and Mitchell Wand. Trampolined style. In *Proc. 1999 ACM SIGPLAN International Conference on Functional Programming*, pages 18–27, Paris, September 1999.
7. Russ Harmer. *Games and full abstraction for nondeterministic languages*. PhD thesis, University of London, 1999.
8. Barnaby P. Hilken. Towards a proof theory of rewriting: the simply-typed 2- λ calculus. Technical Report UCAM-CL-TR-336, University of Cambridge, Computer Laboratory, May 1994.
9. J. Martin E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Information and Computation*, 163(2):285–408, December 2000.
10. J. Longley. The sequentially realizable functionals. Technical Report ECS-LFCS-98-402, University of Edinburgh, 1998.
11. Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.