An analysis of innocent interaction

Russ Harmer

An analysis of innocent interaction

Russ Harmer
CNRS & PPS, Université Paris 7

October 12, 2006

Abstract

We present two abstract machines for innocent interaction. The first, a rather complicated machine, operates directly on innocent strategies. The second, a far simpler machine, requires a "compilation" of the innocent strategies into "cellular" strategies before use. Given two innocent strategies, we get the same final result if we make them interact using the first machine or if we first cellularize them then use the other machine.

1 Introduction

Game semantics models programs as *strategies* for certain *games* played between two protagonists, Player and Opponent, respectively modelling a process and its environment/context . Roughly speaking, the game determines the "ground rules" (who can play what move and when) and usually corresponds to a *type*; for a given game, we then have many potential strategies (for Player) which correspond to different programs of the type in question.

In general, a strategy may depend arbitrarily on the "history of play" to date—what we typically call a *strategy of total information* (STI). On the other hand, many interesting classes of strategy restrict access to the history of play so that the strategy's behaviour depends only on some "partial history of play" to date. The *innocent* strategies [5, 6] used to model languages based on λ- and $\lambda\mu$-calculus fall into this category of *strategies of partial information* (SPIs).

Unlike interaction between STIs, interaction of SPIs poses some delicate questions. During interaction, does such an SPI "see" the entire history but somehow decide to only depend on part of it (a kind of self-censorship)? Or does the strategy only ever "see" the partial history it depends on (each strategy somehow censors the other)? And, in this latter case, once a strategy has made its move, how can it then provide the *next* partial history for the *other* strategy? In general, the partial history (received as input) doesn't suffice to reconstruct the next partial history (to be sent as input to the other strategy) induced by the move made by our strategy.

We present an analysis, in two stages, of this situation. In the first instance, we consider an "interaction architecture" which implements the second option (mutual censorship) by obliging our innocent strategies to interact via an intermediary: the *referee*.

The referee must keep track of the whole interaction. Each strategy receives its input (a partial history) from the referee, makes its move and sends this back to the referee. The referee, on receiving the strategy's response, updates the-interaction-to-date and then "reads off" from this the next partial history. It then sends this, as input, to the other strategy. And so on. (Of course, we could implement the referee trivially by just building up, move by move, the entire interaction—from which we can easily extract any desired partial history. Such a referee would censor both strategies, rather than each strategy censoring itself, but this still essentially corresponds to the first (self-censorship) option above.)

Our referee takes a different approach: it builds up, move by move, a *desequentialized* representation of the interaction. Technically, this consists of a tree-like data structure, constructed and shared by the two interacting strategies, plus an *access protocol* that prevents either strategy from accessing anything other than partial histories. So, each strategy censors the other; but only indirectly, via the agency of the referee. This interaction architecture/abstract machine resembles the PAM and its many variants [1–4] although with more of a desequentialized feel.

In the second stage of our analysis, we remark that, for a restricted class of strategies, the *cellular* strategies, we have no need for the referee. This leads to a far simpler interaction architecture/abstract machine (the CPAM) but, of course, one that can only be used some of the time. In order to better exploit this simple machine, we define a transformation that takes an innocent strategy and returns a cellular strategy that "does the same thing": we get the same result whether we interact innocent strategies directly with the PAM-like machine or we interact cellularized innocent strategies with the CPAM.

2 Innocent strategies and interaction

In this section, we briefly present the basic definitions of game semantics in order to formally define a class of SPIs, the innocent strategies. We then present our "desequentialized PAM".

2.1 Fundamental definitions

2.1.1 Pointing strings

Let Σ be a countable set. A ***pointing string*** over Σ is a string $s \in \Sigma^\star$ with pointers between the occurrences of s such that, if s_i (the ith symbol of s) points to s_j then $j < i$, *i.e.* pointers always point back to earlier occurrences, and we have *at most one* pointer from any given occurrence of s. We write $|s|$ for the length of s.

If $\Sigma' \subseteq \Sigma$ then we write $s \restriction \Sigma'$ for the *restriction* of s to Σ', *i.e.* the pointing string obtained by removing those occurrences of s from $\Sigma - \Sigma'$ and manipulating the pointers as follows: if a pointer points into the "forbidden zone" we keep following pointers until we either reemerge by pointing to some $s_z \in \Sigma'$, in which case s_i points to s_z in $s \restriction \Sigma'$, or we run out of pointers, in which case s_i has no pointer in $s \restriction \Sigma'$.

2.1.2 Arenas

The notion of *arena* provides us with an abstract setting to talk about pointing strings. Formally, we define an ***arena*** A to be a tuple $\langle M_A, \lambda_A, I_A, \vdash_A \rangle$ where

- M_A is a countable set of ***tokens***.
- $\lambda_A : M_A \to \{\mathsf{O}, \mathsf{P}\}$ ***labels*** each $m \in M_A$ as an Opponent or a Player token. We write $\overline{\lambda}_A$ for the "inverted" OP-labelling (exchange of O and P).
- I_A is a subset of $\lambda_A^{-1}(\mathsf{O})$ known as the ***initial moves*** of A.
- $\vdash_A$ is a binary ***enabling*** relation on M_A satisfying *alternation*: if $m \vdash_A n$ then $\lambda_A(m) \neq \lambda_A(n)$.

The ***empty arena*** $\mathbf{1}$ is defined to be $\langle \varnothing, \varnothing, \varnothing, \varnothing \rangle$. A ***flat arena*** has a single OQ-move and a set (possibly empty) of PA-moves, all of which are enabled by the O-move. For example, the boolean arena **bool** has an OQ, q, and two PAs, $\mathtt{tt}$ and $\mathtt{ff}$, with enabling relation $\mathsf{q} \vdash_{\mathbf{bool}} \mathtt{tt}$ and $\mathsf{q} \vdash_{\mathbf{bool}} \mathtt{ff}$. We similarly define $\bot$, **com** and **nat** as the flat arenas over $\varnothing$, $\{\mathtt{t}\}$ and $\{0, 1, 2, \ldots\}$ respectively.

2.1.3 Legal plays

A ***play*** in an arena A is a pointing string s, over alphabet M_A, which respects the structure of A in the following sense:

- if s_j points to s_i then $s_i \vdash_A s_j$;
- if s_i has no pointer then $s_i \in I_A$.

A ***legal play*** is a play which additionally satisfies ***alternation***: for $0 \leq i < |s|$, $\lambda_A(s_i) \neq \lambda_A(s_{i+1})$.

Each occurrence in a legal play s is an element m of M_A together with its pointer (unless $m \in I_A$); we call m plus its pointer a ***move*** of s. If s_j points to s_i we say that s_i ***justifies*** s_j or that s_j ***is justified by*** s_i. The first move of a legal play must be initial (since it cannot point to any previous move!) and hence is an O-move. A move that points to an initial move is called a ***secondary*** move. We write $\mathcal{L}_A$ for the set of all legal plays for the arena A.

The ***prefix*** ordering on strings extends obviously to legal plays so that $\mathcal{L}_A$ can be viewed as a partial order with least element ε, the empty string. For $s, t \in \mathcal{L}_A$, we write $s \sqsubseteq t$ (resp. $s \sqsubseteq^{\mathsf{O}} t$, resp. $s \sqsubseteq^{\mathsf{P}} t$) when s is a (resp. O-ending, resp. P-ending) prefix of t. We fix the convention that $\varepsilon \sqsubseteq^{\mathsf{P}} s$ for any $s \in \mathcal{L}_A$.

We write $s \wedge t$ for the ***longest common prefix*** of s and t, $\mathsf{ip}(s)$ for the ***immediate prefix*** of non-empty s and, provided the last move of s, written s_ω, has a pointer, we write $\mathsf{jp}(s)$ for the ***justifying prefix*** of s, *i.e.* that prefix of s ending with the move that *justifies* s_ω. We write $\mathsf{ie}(s)$ for the set of ***immediate extensions*** of s and, if $s \in \mathcal{L}_A$ and $m \in M_A$ such that s_ω enables m in A, we write $s \cdot m$ for the legal play obtained by adding m to the end of s, pointing to the last move.

2.1.4 Constructors on arenas

The ***product*** $A \times B$ of arenas A and B places the two arenas side-by-side with no possibility of interaction:

- $M_{A \times B} = M_A + M_B$
- $\lambda_{A \times B} = [\lambda_A, \lambda_B]$
- $I_{A \times B} = I_A + I_B$
- $\vdash_{A \times B} = \vdash_A + \vdash_B$

On the other hand, the ***par*** $A \parr B$, coalesces each pair of $I_A \times I_B$ as a single *new* initial move but otherwise preserves the existing structure of A and B:

- $M_{A \parr B} = (I_A \times I_B) + (M_A + M_B)$
- $\lambda_{A \parr B}(\mathsf{inl}((i_A, i_B)) = \mathsf{O}$
  $\lambda_{A \parr B}(\mathsf{inr}(\mathsf{inl}(a))) = \lambda_A(a)$
  $\lambda_{A \parr B}(\mathsf{inr}(\mathsf{inr}(b))) = \lambda_B(b)$
- $I_{A \parr B} = I_A \times I_B$
- $\mathsf{inl}((i_A, i_B)) \vdash_{A \parr B} \mathsf{inr}(\mathsf{inl}(a))$ iff $(i_A, i_B) \in I_{A \parr B}$ and $i_A \vdash_A a$
  $\mathsf{inl}((i_A, i_B)) \vdash_{A \parr B} \mathsf{inr}(\mathsf{inr}(b))$ iff $(i_A, i_B) \in I_{A \parr B}$ and $i_B \vdash_B b$
  $\mathsf{inr}(\mathsf{inl}(a)) \vdash_{A \parr B} \mathsf{inr}(\mathsf{inl}(a'))$ iff $a \vdash_A a'$
  $\mathsf{inr}(\mathsf{inr}(b)) \vdash_{A \parr B} \mathsf{inr}(\mathsf{inr}(b'))$ iff $b \vdash_B b'$

Finally, we have two complementary constructors, one that adds a new initial move and the other which removes the initial moves. We note by A^- the ***decapitation*** of arena A:

- $M_{A^-} = M_A - I_A$
- $\lambda_{A^-}(m) = \overline{\lambda}_A(m)$
- $I_{A^-} = \{m \in M_{A^-} \mid i \vdash_A m \wedge i \in I_A\}$
- $m \vdash_{A^-} n$ iff $m \vdash_A n$

and by A_* the operation of adding a new O root to A, making it the unique initial move of A_*:

- $M_{A_*} = \{*_A\} + M_A$
- $\lambda_{A_*}(\mathsf{inl}(*_A) = \mathsf{O}$
  $\lambda_{A_*}(\mathsf{inr}(a)) = \overline{\lambda}_A(a)$
- $I_{A_*} = \{*_A\}$
- $\mathsf{inl}(*_A) \vdash_{A_*} \mathsf{inr}(i_A)$ iff $i_A \in I_A$
  $\mathsf{inr}(a) \vdash_{A_*} \mathsf{inr}(a')$ iff $a \vdash_A a'$

The usual ***arrow*** $A \Rightarrow B$ of arenas A and B can be recovered by $A_* \parr B$.

2.1.5 Strategies

A ***strategy*** σ for an arena A, written $\sigma : A$, is a non-empty set of P-ending legal plays of A which satisfies

- *prefix-closure*: if $s \in \sigma$ and $s' \sqsubseteq^{\mathsf{P}} s$ then $s' \in \sigma$;
- *determinism*: if $s \in \sigma$ and $t \in \sigma$ then $s \wedge t \in \sigma$.

The second condition amounts to asking for $s \wedge t$ to end with a P-move; so only Opponent can branch nondeterministically.

We write $\mathsf{dom}(\sigma)$ for the ***domain*** of σ defined as $\bigcup_{s \in \sigma} \mathsf{ie}(s)$, the O-ending plays of A *accessible* to σ.

2.1.6 Legal interactions and program interactions

We define the *composition* of $\sigma : A_1 \Rightarrow A_2$ and $\tau : A_2 \Rightarrow A_3$ by generalizing the notion of legal play: a ***legal interaction*** over A_1, A_2 and A_3 is a pointing string u over $M_A + M_B + M_C$ such that $u \upharpoonright A_1, A_2 \in \mathcal{L}_{A_1 \Rightarrow A_2}$, $u \upharpoonright A_2, A_3 \in \mathcal{L}_{A_2 \Rightarrow A_3}$ and $u \upharpoonright A_1, A_3$ alternates. We write $\mathcal{I}(A_1, A_2, A_3)$ for the set of all legal interactions over A_1, A_2 and A_3.

This allows us to build a category with arenas as objects and strategies as arrows. However, in this paper, we rarely have need for the generality of legal interactions; we primarily consider closed interactions of base type. Formally, a ***program interaction*** for arena A and *flat* arena B is a legal play $s \in \mathcal{L}_{A \Rightarrow B}$ such that $s \upharpoonright A \in \mathcal{L}_A$ (and $s \upharpoonright B$ has a single initial move). We write $\mathcal{P}(A, B)$ for the set of all program interactions for A and B. For $\sigma : A$ and $\tau : A \Rightarrow B$, we define their set of interactions as

$$\sigma \mid \tau = \{u \in \mathcal{P}(A, B) \mid u \upharpoonright A \in \sigma \cup \mathsf{dom}(\sigma) \wedge u \in \tau \cup \mathsf{dom}(\tau)\}.$$

2.2 Innocence

2.2.1 The P-view

The P-***view*** of non-empty play $s \in \mathcal{L}_A$, noted $\ulcorner s \urcorner$, is defined in two stages. First we extract a subsequence of s with pointing structure defined only on (non-initial) O-moves:

- $\ulcorner s \urcorner = s_\omega$, if s_ω is an initial move;
- $\ulcorner s \urcorner = \ulcorner \mathsf{jp}(s) \urcorner \cdot s_\omega$, if s_ω is a non-initial O-move;
- $\ulcorner s \urcorner = \ulcorner \mathsf{ip}(s) \urcorner \, s_\omega$, if s_ω is a P-move.

In words, we trace back from the end of s, following pointers from O-moves, excising all moves under such pointers, and "stepping over" P-moves, until we reach an initial move. In general, a P-move $m \in s$ can "lose its pointer": if its justifier n lies strictly underneath an O-to-P pointer of $\ulcorner s_{<m} \urcorner$ then n does not occur in $\ulcorner s_{<m} \urcorner$. The second stage of the definition thus specifies that, if the justifier of a P-move in $\ulcorner s \urcorner$ gets excised in this way, it has *no justifier* in the P-view (and so $\ulcorner s \urcorner \notin \mathcal{L}_A$); otherwise it keeps the same justifier as in s.

2.2.2 The P-visibility condition

A legal play $s \in \mathcal{L}_A$ satisfies P-***visibility*** iff $\ulcorner s \urcorner \in \mathcal{L}_A$. In words, no P-move of $\ulcorner s \urcorner$ loses its pointer. Note that, for t some proper prefix of s, this doesn't prevent a P-move of $\ulcorner t \urcorner$ losing its pointer. We lift the definition of P-visibility to strategies in the obvious way: σ satisfies P-***visibility*** iff all $s \in \sigma$ do. So, for s in P-vis σ all $t \sqsubseteq^{\mathsf{P}} s$ do in fact satisfy P-visibility—since σ is closed under P-ending prefixes—so $\ulcorner t \urcorner \in \mathcal{L}_A$ for all the P-prefixes t of s.

2.2.3 Innocence as P-view-dependence

If $s, t \in \mathcal{L}_A$ where s ends with a P-move, satisfies P-vis and $\ulcorner \mathsf{ip}(s) \urcorner = \ulcorner t \urcorner$ then we denote by $\mathsf{match}(s, t)$ the unique extension of t satisfying $\ulcorner s \urcorner = \ulcorner \mathsf{match}(s, t) \urcorner$, *i.e.* add the last move of s to t using the "same" pointer as in s. We can do this since, by assumption, the last move of s points in $\ulcorner \mathsf{ip}(s) \urcorner = \ulcorner t \urcorner$, *i.e.* since $\mathsf{match}(s, t) = \mathsf{match}(\ulcorner s \urcorner, t)$.

A P-vis strategy σ is ***innocent*** iff

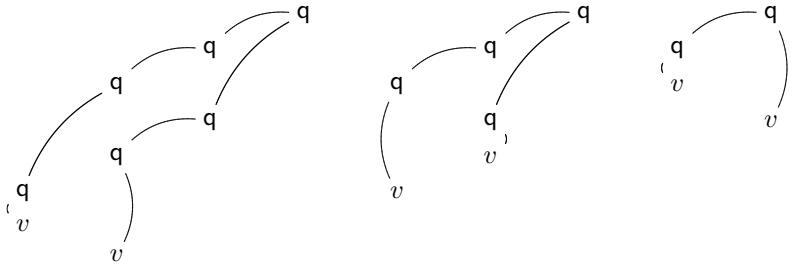
$$s \in \sigma \wedge t \in \mathsf{dom}(\sigma) \wedge \ulcorner \mathsf{ip}(s) \urcorner = \ulcorner t \urcorner \;\Rightarrow\; \mathsf{match}(s, t) \in \sigma.$$

An innocent strategy σ is thus completely determined by its ***view function*** $\ulcorner \sigma \urcorner$ defined to be $\{\ulcorner s \urcorner \mid s \in \sigma\}$: the plays of σ all arise as *interleavings* of entries of its view function. We can thus see innocent strategies as strategies of partial information, the "partial history" of a play to date being simply its P-view.

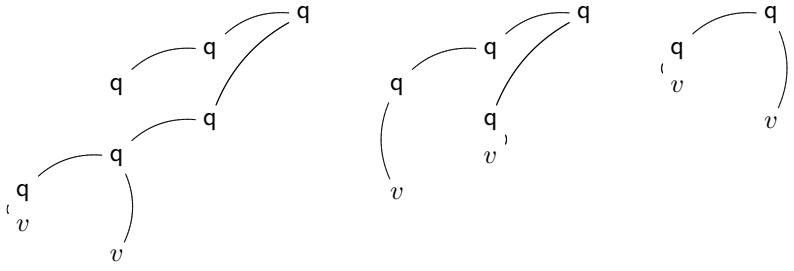
2.3 Innocent interaction

2.3.1 The role of the referee

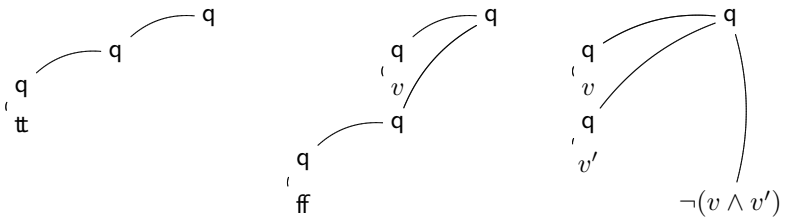
We begin with an example that illustrates the role of the referee. Consider Kierstead's term $\mathcal{K}_x = \lambda F(F)\lambda x(F)\lambda y(x)$:



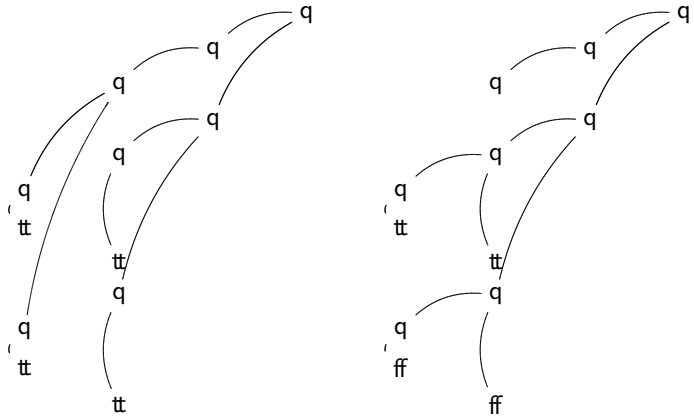
and its close cousin $\mathcal{K}_y = \lambda F(F)\lambda x(F)\lambda y(y)$:



with argument $\mathtt{NAND}_{\mathtt{tt};\mathtt{ff}} = \lambda f(\mathtt{if}\ (f)\mathtt{tt}\ (\mathtt{not}\ (f)\mathtt{ff})\ \mathtt{else}\ (\mathtt{if}\ (f)\mathtt{ff}\ \mathtt{tt}\ \mathtt{else}\ \mathtt{tt}))$:



The interactions start like this:



When $\mathtt{NAND}_{\mathtt{tt};\mathtt{ff}}$ plays the fifth move, its output P-view is just $\mathsf{q} \frown \mathsf{q}$ so the referee must insert the previous two moves in order to correctly send $\mathsf{q} \frown \mathsf{q} \frown \mathsf{q} \frown \mathsf{q} \frown \mathsf{q}$ as next input P-view for $\mathcal{K}_x/\mathcal{K}_y$. Similarly, with its last move, $\mathcal{K}_x$ has output P-view $\mathsf{q} \frown \mathsf{q} \frown \mathsf{q} \frown \mathsf{q} \frown \mathsf{q} \frown \mathsf{q} \frown \mathtt{tt} \frown \mathtt{tt}$, so the referee must insert the $\mathsf{q} \frown \mathtt{tt}$ arch so as to send $\mathsf{q} \frown \mathsf{q} \frown \mathtt{tt} \frown \mathsf{q} \frown \mathtt{tt}$ as next input P-view.

2.3.2 The desequentialized PAM

Given two view functions, the desequentialized PAM (DPAM) computes their interaction by making them communicate via a shared data structure, their *referee*. This data structure—a collection of nodes and two distinct tree structures on it—is built dynamically and is itself used to "guide" its own process of construction.

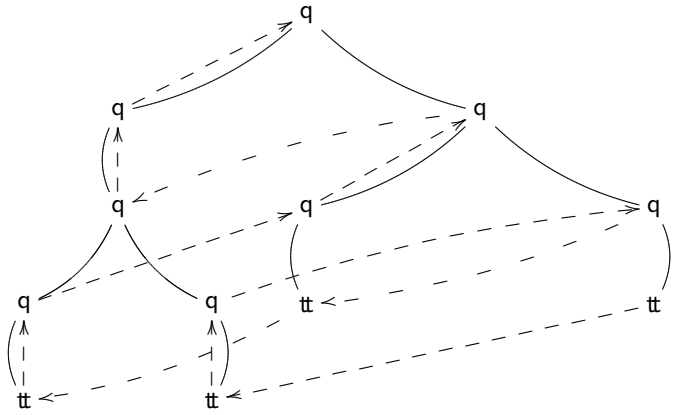
In outline, the DPAM interrogates the two view functions alternately whilst maintaining the referee data structure (on the basis of the responses of the view functions). Each time a view function makes a move, the DPAM adds a new node (to the referee) with two edges: the c-ptr and the j-ptr. It then offers this new node to the other view function.

When a view function is offered a node, it can read off its current P-view by traversing the referee: from its own nodes, it follows the c-ptrs and from the other's nodes, it follows the j-ptrs. We call this the *access protocol* to the referee: a view function has read/write access to c-ptrs from its nodes, only write access to its j-ptrs but read access to those of the other, and no access to the other's c-ptrs.

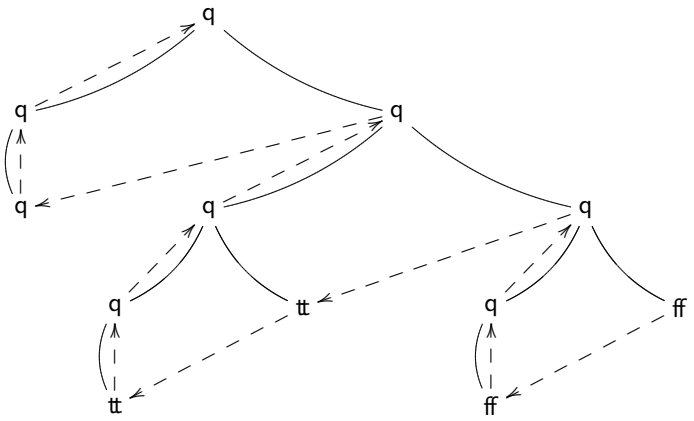
So, given view functions $\ulcorner\sigma\urcorner : A$ and $\ulcorner\tau\urcorner : A \Rightarrow B$, we build their ***referee*** recursively by:

- when a view function is offered a node, it reads off its current P-view;
- assuming it responds to this P-view, the DPAM creates a new node n that encodes the *token* just played; it then adds n's *c-ptr*, which points to the input/offered node, and its *j-ptr*, which points to that node of the referee (in the "read off" P-view) encoding the justifier of n;
- the DPAM offers n to the other view function

If we apply the DPAM procedure to $\mathcal{K}_x$ and $\mathtt{NAND}_{\mathtt{tt};\mathtt{ff}}$, it constructs the following referee (we only show the subtree associated to the interaction of §2.3.1) where the dotted arrows are the c-ptrs:



If we apply it to $\mathcal{K}_y$ and $\mathtt{NAND}_{\mathtt{tt};\mathtt{ff}}$, we get:



At first sight, these "2-trees" don't seem very desequentialized. After all, given any node in either, we can trace back the entire interaction by always following the c-ptrs. However, in the DPAM, the access protocol prevents this: each view function has access only to its own c-ptrs—and so can only traverse its P-views.

3 Cellular strategies

The DPAM correctly implements interaction of innocent strategies, using only the view functions (not the full set of legal plays). However, the overhead of maintaining the referee inevitably slows the machine down. Could we implement (something sufficient to simulate) innocent interaction without using a referee?

In this section, we present the class of cellular strategies. We can interact such strategies using a radically simplified DPAM, the CPAM (cellular pointer abstract machine), which needs no referee. We then describe a "cellularization" process that converts an innocent strategy into a cellular one that "does the same thing". We can thus interact two innocent strategies by first cellularizing both, then running the results in the CPAM.

3.1 Cellular strategies

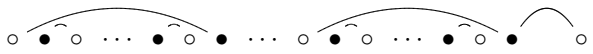
3.1.1 The OP-view

The OP-***view*** of $s \in \mathcal{L}_A$, written $\lceil s \rfloor$, is defined by:

- $\lceil s \rfloor = s_\omega$, if s_ω is an initial move;
- $\lceil s \rfloor = \lceil \mathsf{jp}(s) \rfloor \cdot s_\omega$, otherwise.

In words, we simply follow back the "chain" of pointers to the initial move.

If s satisfies P-visibility, the OP-view $\lceil s \rfloor$ forms a subsequence of the P-view $\lceil s \rceil$. Indeed, we can consider the P-view as an *annotation* of the OP-view which inserts, underneath each O $\curvearrowleft$ P arch of $\lceil s \rfloor$, a sequence of P $\curvearrowleft$ O arches:



3.1.2 The O-views and the view

The O-***view*** of $s \in \mathcal{L}_A$, written $\lfloor s \rfloor$, is defined by:

- $\lfloor \varepsilon \rfloor = \varepsilon$;
- $\lfloor s \rfloor = \lfloor \mathsf{jp}(s) \rfloor \cdot s_\omega$, if s_ω is a P-move;
- $\lfloor s \rfloor = \lfloor \mathsf{ip}(s) \rfloor \ s_\omega$, if s_ω is an O-move.

In contrast to a P-view which always has a unique initial move, an O-view may contain many initial moves. A play $s \in \mathcal{L}_A$ satisfies O-***visibility*** iff $\lfloor s \rfloor \in \mathcal{L}_A$, *i.e.* we lose no O-pointers in $\lfloor s \rfloor$.

The ***short*** O-***view*** of $s \in \mathcal{L}_A$, written $\llcorner s \lrcorner$, is defined by:

- $\llcorner s \lrcorner = s_\omega$, if s_ω is a secondary move;
- $\llcorner s \lrcorner = \llcorner \mathsf{jp}(s) \lrcorner \cdot s_\omega$, if s_ω is a non-secondary P-move;
- $\llcorner s \lrcorner = \llcorner \mathsf{ip}(s) \lrcorner \ s_\omega$, if s_ω is an O-move

Note that, for $s \in \mathcal{L}_A$, its short O-view $\llcorner s \lrcorner$ is *not* a legal play of A but *is* a legal play of A^-, the decapitation of A.

Let u be a program interaction between innocent $\sigma : A$ and $\tau : A \Rightarrow B$. If τ played the last move of u then $\llcorner u \lrcorner = \ulcorner u \upharpoonright A \urcorner$, *i.e.* the next input P-view for σ is just the short O-view of u. If σ played the last move of u then $\mathsf{q} \cdot \lfloor u \upharpoonright A \rfloor = \ulcorner u \urcorner$ (where q, the initial move of u, justifies all the initial-in-A moves of u), *i.e.* the next input P-view for τ is essentially just the O-view of $u \upharpoonright A$.

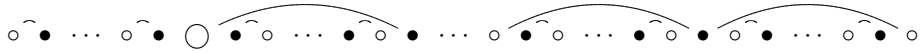
If s satisfies P- *and* O-visibility, the OP-view is contained in the O-view (as well as in the P-view). So an O-view can be seen as an "annotated OP-view" in much the same way as a P-view:

Earlier on, we defined the function $\mathsf{match}(s,t)$ to formalize what we intuitively mean by "extend t with the last move of s". We now define a new function $\mathsf{match}^\star(s,t)$ for P-ending, P-vis $s \in \mathcal{L}_A$ and $t \in \mathcal{L}_A$ such that $\ulcorner \mathsf{jp}(s) \urcorner = \ulcorner t \urcorner$: $\mathsf{match}^\star(s,t)$ denotes the extension of t with the suffix of the P-view of s that lies underneath the pointer from s_ω to $\mathsf{jp}(s)_\omega$. In other words, instead of adding just the last move of s to t, this adds the last "chunk" of the P-view of s to t, so that $\ulcorner \mathsf{match}(s,t) \urcorner = \ulcorner s \urcorner$. Similarly, for O-ending, O-vis $s \in \mathcal{L}_A$ and $t \in \mathcal{L}_A$ such that $\lfloor \mathsf{jp}(s) \rfloor = \lfloor t \rfloor$, we define $\mathsf{match}^\star(s,t)$ to be that extension of t with the last "chunk" of the O-view of s.

Finally, we define the ***view*** $\tilde{s}$ of P- and O-vis $s \in \mathcal{L}_A$ to be the obvious "superposition" of the P-view and the O-view:

- $\tilde{s} = \lfloor s \rfloor$, if s_ω is an initial move;
- $\tilde{s} = \mathsf{match}^\star(s, \widetilde{\mathsf{jp}(s)})$, otherwise.

The view of s can thus be seen as a doubly annotated OP-view:



where $\bigcirc$ is the initial move of the OP-view.

3.1.3 View-dependence

Let σ be a strategy for A satisfying O- and P-visibility. We say that σ is ***view-dependent*** iff

$$s \in \sigma \wedge t \in \mathsf{dom}(\sigma) \wedge \widetilde{\mathsf{ip}(s)} = \tilde{t} \;\Rightarrow\; \mathsf{match}(s,t) \in \sigma.$$

A view-dependent strategy thus depends on the view in exactly the same way that an innocent strategy depends on the P-view. In particular, innocence implies view-dependence: a P-view is a view where all O-moves are justified by the immediately preceding move.

In passing, let us note that the class of view-dependent strategies is closed under composition, giving rise to an SMCC (not a CCC) that contains the innocent CCC and is contained in the usual "ambient SMCC" of arenas and all strategies.

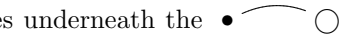
A view-dependent strategy σ is ***cellular*** iff it satisfies OP-visibility. Note that (the behaviour of) a cellular strategy still *depends on* the entire view; only its choice of justification pointer is restricted by the condition. However, this already rules out *copycat* strategies (the usual identity arrows in categories of games) for all but the most trivial of arenas. In consequence, cellular strategies do not form a category—even though the condition *is* preserved by composition.

A cellular strategy is uniquely determined by its ***cellular view function*** which maps O-ending views (inputs) to P-ending views (outputs).

3.2 Cellular interaction

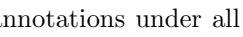
3.2.1 The CPAM

During DPAM interaction, each view function reads off, from the referee, its current input P-view. Once the view function responds, the DPAM updates the referee to reflect the new "state of interaction". We need to do this because the view function cannot in general determine, from its input P-view and response, the next P-view for the other view function. Typically, an input P-view plus the view function's response looks something like:

In general, the next input P-view contains moves underneath the  arch. But these moves don't appear in the current P-view. This kind of thing happens all the time in DPAM interaction; the c-ptrs of the referee allow us to recover these "missing" moves.

However, imagine now an alternative interaction architecure (to the DPAM) where the strategies exchange *views*, not P-views, but where both strategies must be cellular. If a strategy receives an input view (from the other strategy), it knows its response (by view-dependence) and the fact that its response points in the OP-view means that the next view (for the other strategy) can be simply "read off" without any need for a referee:

If we take the input view and extend it with the strategy's response

we almost have another view. If we erase the annotations under all the  arches of the OP-view that lie under the pointer from the last move, we get a view, the next input for the other strategy:

So, given cellular view functions $\sigma : A$ and $\tau : A \Rightarrow B$ (for B a base type), we define their ***CPAM*** interaction by:

- when a view function receives a view, it adds its next move (if any) to this view and erases no-longer-necessary annotations;
- unless the last move was in B, it sends the resulting view to the other view function (which does the same thing).

3.2.2 Cellularization

Given innocent $\sigma : A$ and $\tau : A \Rightarrow B$ (where B is a flat arena), we transform their set of interactions $\sigma \mid \tau$ into a new set $\sigma ©\tau$ of interactions satisfying OP-visibility. We define this inductively on the length of u:

- $\varepsilon^{©} = \varepsilon$;
- $ua^{©} =$
  - $u^{©}a$, if a points in the OP-view;
  - $u^{©}\ulcorner u \restriction \sigma \urcorner$, if a (is not OP-vis and) is played by τ;
  - $u^{©}u'a$, if a (is not OP-vis and) is played by σ; and where u' is the last segment of $\mathsf{jp}(u)^{©}$ with its first two moves removed.

The set $= \{u \restriction A \mid u \in \sigma ©\tau\}$ determines a deterministic cellular view function (by taking the set of views of plays in the set) and so can be "closed", yielding a cellular strategy $\sigma^{©}$. We can similarly define a cellular $\tau^{©}$ for $A \Rightarrow B$. So, if we interact $\sigma^{©}$ and $\tau^{©}$ in the CPAM, they play exactly the transformed interactions above and so we get the same final result (if any) as if we'd interacted σ and τ in the DPAM.

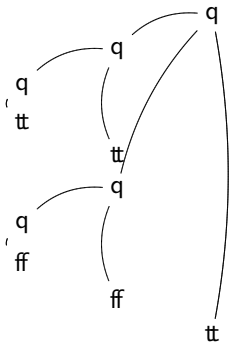
We've defined here a kind of "relative cellularization": we fix σ *and* τ and obtain $\sigma^{©}$ and $\tau^{©}$ that interact "just like" σ and τ. But, if we now cellularize the same τ with some other σ', we (usually) obtain a *different* cellularization of τ, *i.e.* some $\tau^{©'}$ which interacts correctly with $\sigma'^{©}$.

Intuitively, we could define an "absolute" cellularization of τ by cellularizing it with *all* possible σs and taking the big union of all the resulting $\tau^{©}$s. However, unlike in the relative case, the proof that the resulting strategy determines a deterministic view function is highly nontrivial. We hope to present this in a future version of this paper.

3.3 Examples of cellularization

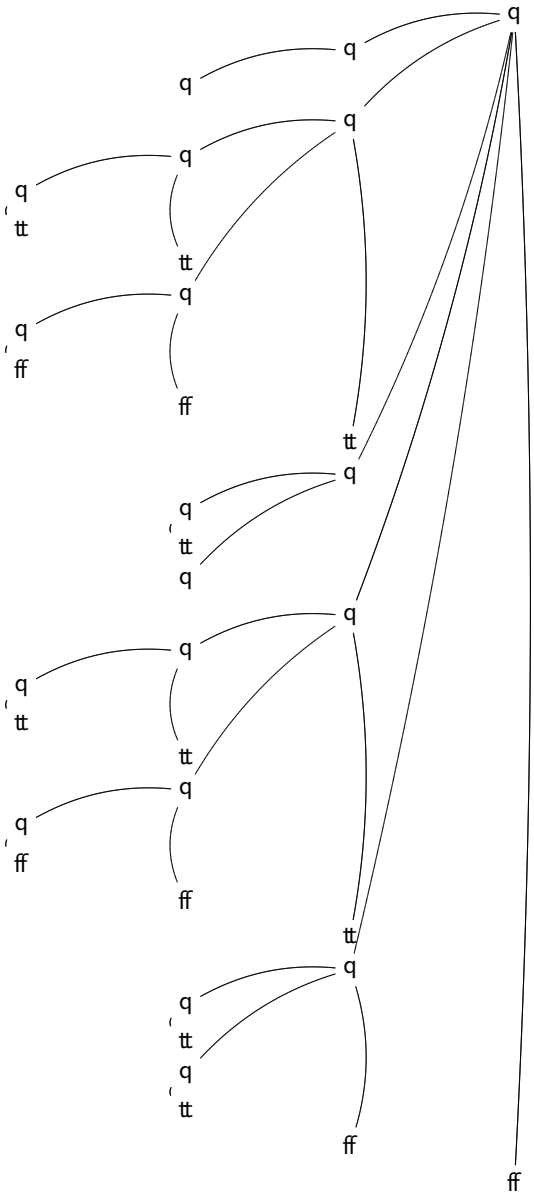
3.3.1 K_y vs. nand

We give the transformation of $\mathtt{NAND} \mid \mathcal{K}_y$ below. The cellular interaction starts out exactly like the corresponding innocent interaction. However, once NAND completes the subcomputation



$\mathcal{K}_y$ would violate cellularity (in the innocent interaction), so we insert the whole of the next P-view for `NAND`. The interaction remains cellular until that same subcomputation completes a second time, whence $\mathcal{K}_y$ inserts the (now longer) next P-view for `NAND`.

$\mathcal{K}_y$ vs. `NAND`:

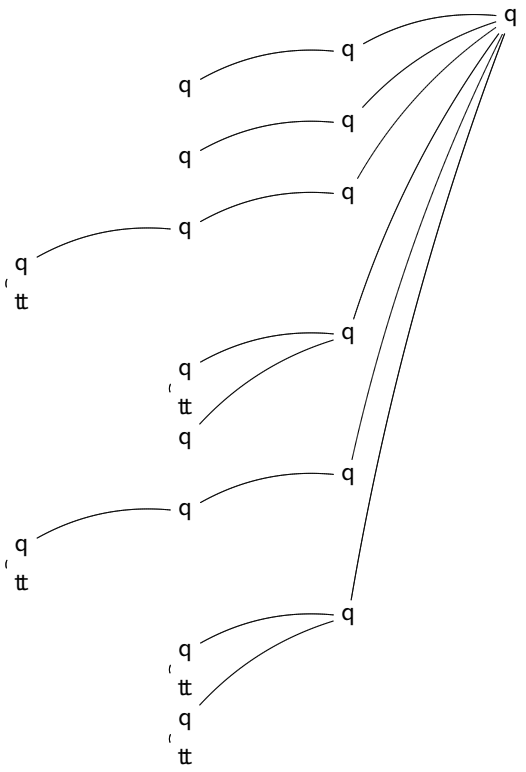


This examples nicely illustrates the way that cellular interaction works incrementally: $\mathcal{K}_y$ gradually "unpicks" `NAND`, view by view, and the interaction goes a bit further every time $\mathcal{K}_y$ has "learnt a new view" of `NAND`—witness the sequence of P-views in the first copy of `NAND`: $\mathsf{q} \frown \mathsf{q}$; $\mathsf{q} \frown \mathsf{q} \frown \mathsf{tt} \frown \mathsf{q}$; $\mathsf{q} \frown \mathsf{q} \frown \mathsf{tt} \frown \mathsf{q} \frown \mathsf{ff} \frown \mathsf{tt}$.

3.3.2 K_x vs. nand

The noncellular behaviour of $\mathcal{K}_x$ becomes apparent much sooner than that of $\mathcal{K}_y$. After just the fifth move, $\mathcal{K}_x$ must insert $\mathsf{q} \frown \mathsf{q} \frown \mathsf{q}$ to provide the correct input P-view for `NAND`. On receiving `NAND`'s response, $\mathcal{K}_x$ must once again insert the next P-view, *etc.* Indeed, the cellularized $\mathcal{K}_x$ spends much of its time "switching" between the two copies of `NAND`.

$\mathcal{K}_x$ vs. `NAND` [beginning segment]:



4 Conclusions and future work

In summary, we've presented two abstract machines for interacting respectively innocent and cellular strategies. We've outlined the procedure of cellularization that permits us to interact innocent strategies in the CPAM. The resulting interactions in some sense untangle the "pointer spaghetti" of typical innocent interactions, exposing perhaps some potential deeper symmetry between Opponent and Player in the guise of view-dependent strategies. In any case, the category of view-dependent strategies seems worthy of further attention, the most pressing need being for a syntax, perhaps using first-class contexts, at least for describing head normal forms.

Another natural next step would be to implement the two machines presented here and compare their performance, in particular to see whether the efficiency of the CPAM (no referee) comes at too high a cost (the cellularized strategies contain *very* long plays in general).

This work owes a considerable intellectual debt to the work of Padovani [8] and Loader [7]. Both papers use "transfering terms" (now known as "cellular terms") and a transformation of λ-terms into cellular terms in order to establish decidability of the contextual equivalence on the minimal model (Padovani) and on unary PCF (Loader). However, our work also differs from theirs in that our notion of "cellular" restricts *and* extends innocence: we have a stronger visibility condition but a weaker view-dependence condition.

Acknowledgements I would like to thank Pierre Clairambault, Pierre-Louis Curien, Thierry Joly and Vincent Padovani for numerous discussions related to the subject of this paper.

References

[1] T. Coquand. A semantics of evidence for classical arithmetic. *J. Symb. Logic*, 60(1):325–337, 1995.

[2] P.-L. Curien and H. Herbelin. Computing with abstract Böhm trees. In M. Sato and Y. Toyama, editors, *Fuji International Symposium on Functional and Logic Programming (FLOPS '98)*. World Scientific, Singapore, 1998.

[3] P.-L. Curien and H. Herbelin. Abstract machines for dialogue games. *Panoramas et synthèses*, 2006. To appear.

[4] V. Danos, H. Herbelin, and L. Regnier. Game semantics & abstract machines. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science (LICS '96)*, pages 394–405. IEEE Computer Society Press, 1996.

[5] R. Harmer. Innocent game semantics. Lecture notes, 2004–2006.

[6] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 163:285–408, 2000.

[7] R. Loader. Unary PCF is decidable. *Theoretical Computer Science*, 206:317–329, 1998.

[8] V. Padovani. Decidability of all minimal models. In *TYPES '95: Selected papers from the International Workshop on Types for Proofs and Programs*, London, UK, 1996. Springer-Verlag.