Innocent game semantics

Russ Harmer
CNRS & PPS, P7

August 2, 2006

1 Introduction

Game semantics, in its "modern" form, arose in the early 90s in the wake of sequential algorithms, process calculus and the geometry of interaction. The historical context of game semantics revolved around the long-standing problem of "full abstraction for PCF", PCF being an idealized functional language. After Plotkin's domain model (which he showed to be not fully abstract for PCF due to the presence of parallel functions) and Berry's stable model (also not fully abstract thanks to the Gustav function) came a radically different model: concrete data structures (CDSs) & sequential algorithms. This model is distinguished by being non-well-pointed: a category of CDSs and sequential algorithms exists, is a CCC, but its arrows are not *functions*. Game semantics shares this property; its arrows are *strategies* for playing in certain *games*.

At about the same time, Milner and Hoare were developing the first process calculi: CCS and CSP. While differing on many details, both calculi emphasize the importance of *interaction* as a mechanism for abstraction, represented syntactically as *parallel composition*, and of *hiding* as a way of making certain actions "internal" to a process (and hence invisible outside of that process), represented syntactically by a *restriction* operator. Game semantics borrows these ideas to define a notion of *composition* of strategies as *parallel composition plus hiding* which is associative and has identities, thus forming a category of games and strategies.

A little while later, Girard introduced linear logic via a decomposition of Berry's stable model. Shortly after, he introduced the geometry of interaction as a *dynamic semantics* of cut elimination (in fragments of linear logic, System F, *etc.*). Blass subsequently built a rudimentary game model of MLL which suffered from various problems, all more or less directly related to *polarities*. Kleene & Gandy's work on unimonotone functions used an extensional quotient on *oracles*, kinds of sequential algorithms, to define a class of sequential functions. The construction has a game-like flavour: the *bracketing condition* makes its first appearance here, long before the game models of PCF. Abramsky & Jagadeesan reworked the geometry of interaction in a categorical setting, leading to a game-like setting with composition based on parallel composition plus hiding. Finally, two game models of MLL appeared, one for MLL plus Mix due to Abramsky & Jagadeesan, the other due to Hyland & Ong for pure MLL.

The models of MLL marked the first appearance of game semantics in its modern form, but the three models that appeared shortly later—all fully abstract models of PCF—have been far more influential on the subsequent development of the field. These models are CCCs but not well-pointed, like the sequential algorithms model, but they easily give rise to well-pointed CCCs, where the fully abstract model really lives. Game semantics and sequential algorithms share the property that they model functional programs (as in PCF), not as mathematical functions, but as *processes*. Processes take time, they perform actions; a process typically has a context. Game semantics formalizes these ideas:

Imagine we have an as yet unspecified "game" with two protagonists, Opponent and Player. Player will represent our processes, Opponent their contexts. Each protagonist can play *moves*, some of which may depend on other moves. Once we have specified the moves and the dependencies between them, we have defined a *game*. Opponent and Player can now "play" in this game; we could require that Opponent starts and that they subsequently alternate. We model this as a *sequence* of moves called a *play*. A play thus represents one possible interaction between Opponent and Player. A *strategy* for Player consists of a set of plays: a "crib book" where Player "looks up" what move to play next as a function of "the move Opponent just played" or of "the whole of play to date", *etc.*

We develop these basic intuitions formally in the next section. Subsequent sections treat various subclasses of strategies of interest, starting with the *innocent* strategies that lie at the heart of the original PCF models.

2 The ambient SMCC

We employ standard notation and terminology for strings over an alphabet. We write s_i for the ith occurrence of s (provided it has one) and s_ω for its last occurrence. Also be aware that we make no notational distinction between an element of the alphabet, an occurrence in a string of said element and the string consisting only of that element.

2.1 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 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. [The latter can probably be relaxed.] We write $|s|$ for the length of (the underlying string 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 $s_i \in \Sigma'$ points to $s_j \in \Sigma'$ in s then we keep the same pointer in $s \restriction \Sigma'$; otherwise, we follow the pointer from s_j to some s_k and inductively apply the same reasoning. In other words, 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 $s \restriction \Sigma'$.

2.2 Arenas

An ***arena*** A is a tuple $\langle M_A, \lambda_A, I_A, \vdash_A \rangle$ where

- M_A is a countable set of ***tokens***.
- $\lambda_A : M_A \to \{\mathsf{O}, \mathsf{P}\} \times \{\mathsf{Q}, \mathsf{A}\}$ ***labels*** each $m \in M_A$ as belonging to Opponent or to Player and as a Question or an Answer. We write λ_A^{OP} (resp. λ_A^{QA}) for the composition with first (resp. second) projection and λ_A^{PO} for the "inverted" OP-labelling (first projection then exchange of O and P).
- I_A is a subset of $\lambda_A^{-1}(\mathsf{OQ})$ known as the ***initial moves*** of A.
- $\vdash_A$ is a binary ***enabling*** relation on M_A satisfying

  (e1) if $m \vdash_A n$ then $\lambda_A(m) \neq \lambda_A(n)$

  (e2) if $m \vdash_A n$ where $\lambda_A^{\mathsf{QA}}(n) = \mathsf{A}$ then $\lambda_A^{\mathsf{QA}}(m) = \mathsf{Q}$

The notion of arena provides an abstract setting in which to talk about pointing strings. Specifically, a ***play*** in an arena A is a pointing string s, over alphabet M_A, which satisfies

- OP-*alternation*: $\lambda_A^{\mathsf{OP}}(s_i) \neq \lambda_A^{\mathsf{OP}}(s_{i+1})$ for $0 \leq i < |s|$
- if s_j points to s_i then $s_i \vdash_A s_j$.

A ***legal play*** is a play where, for $1 \leq i \leq |s|$, if s_i has no pointer then $s_i \in I_A$.

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 simply that s_j ***points to*** s_i; more generally, if following pointers back from s_j arrives at s_i, we say that s_i ***hereditarily justifies*** s_j. The first move of a legal play must be initial (since it cannot point to any previous move!) and hence is an O-move and so OP-alternation just means: $\lambda_A^{\mathsf{OP}}(s_i) = \mathsf{O}$ if, and only if, i is odd.

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 s_ω has a pointer, $\mathsf{jp}(s)$ 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. Finally, if $s \in \mathcal{L}_A$ and $m \in M_A$ such that $s_\omega \vdash_A m$, we write $s \cdot m$ for the legal play obtained by adding m to the end of s, pointing to s_ω.

The ***empty arena* 1** is defined to be $\langle \varnothing, \varnothing, \varnothing, \varnothing \rangle$ so that $\mathcal{L}_{\mathbf{1}} = \{\varepsilon\}$. A ***flat arena*** has a single O-move and a set (possibly empty) of P-moves, all of which are enabled by the O-move. For example, the boolean arena **bool** has moves $\{\mathsf{q}, \mathsf{t}, \mathsf{ff}\}$ where $\lambda_{\mathbf{bool}}(\mathsf{q}) = \mathsf{OQ}$, $\lambda_{\mathbf{bool}}(\mathsf{t}) = \lambda_{\mathbf{bool}}(\mathsf{ff}) = \mathsf{PA}$, $I_{\mathbf{bool}} = \{\mathsf{q}\}$ with the enabling relation $\mathsf{q} \vdash_{\mathbf{bool}} \mathsf{t}$ and $\mathsf{q} \vdash_{\mathbf{bool}} \mathsf{ff}$. We can similarly define $\bot$, **com** and **nat** as the flat arenas over $\varnothing$, $\{\mathsf{t}\}$ and $\{0, 1, 2, \ldots\}$ respectively.

2.3 Constructors on arenas

The ***product*** $A \times B$ of arenas A and B is defined by:

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

This clearly respects the requirements for an arena. Note how *all* the structure of $A\times B$ is inherited from its constituent arenas; we place the arenas side-by-side with no possibility of "interaction" between them.

In contrast, the second major construction on arenas, the ***par*** $A \parr B$, adds in some new moves and structure. As for the product, the moves come from the disjoint union of those of A and B, but with the addition of a new set of initial moves; moves previously initial (in A or B) lose this status in $A \parr B$:

- $M_{A\parr B} = (I_A \times I_B) + (M_A + M_B)$
- $\lambda_{A\parr B}(\mathsf{inl}(\langle i_A, i_B\rangle)) = \mathsf{O}$;
  $\lambda_{A\parr B}(\mathsf{inr}(m)) = [\lambda_A, \lambda_B](m)$
- $I_{A\parr B} = I_A \times I_B$
- $\mathsf{inl}(\langle i_A, i_B\rangle) \vdash_{A\parr B} \mathsf{inr}(m)$ iff $i_A \vdash_A m$ or $i_B \vdash_B m$;
  $\mathsf{inr}(m) \vdash_{A\parr B} \mathsf{inr}(n)$ iff $m \vdash_A n$ or $m \vdash_B n$

The ***arrow*** $A \Rightarrow B$ is defined in terms of the par and the ***half lift*** operation which adds a new initial move, enabling all previously initial moves, and inverts the labelling in A:

- $M_{\downarrow A} = \{\star\} + M_A$
- $\lambda_{\downarrow A}(\mathsf{inl}(\star)) = \mathsf{O}$;
  $\lambda_{\downarrow A}(\mathsf{inr}(m)) = \lambda_A^{\mathsf{PO}}(m)$
- $I_{\downarrow A} = \{\star\}$
- $\mathsf{inl}(\star) \vdash_{\downarrow A} \mathsf{inr}(m)$ iff $m \in I_A$;
  $\mathsf{inr}(m) \vdash_{\downarrow A} \mathsf{inr}(n)$ iff $m \vdash_A n$

We now define $A \Rightarrow B$ as $(\downarrow A) \parr B$. Note that the (previously) initial moves of A become P-moves of $A \Rightarrow B$ enabled by the initial moves of B. Clearly this still satisfies the axioms of an arena. The ***negation*** $\neg A$ abbreviates $A \Rightarrow \bot$.

The only difference between the arena **bool** defined above and $(\bot \times \bot) \Rightarrow \bot$ lies in the labelling of the P-tokens as Questions (instead of as Answers). Note that we also have an arena $\bot \Rightarrow (\bot \times \bot)$, a kind of "upside-down version" of **bool**, and more generally $A \Rightarrow (A \times A)$, so we cannot think of arenas as (trees or) forests in general—even though all arenas in the simple type hierarchy over a collection of flat arenas *are* trees.

2.4 Strategies

So far, we've defined the notions of pointing string and arena in order to define the concept of "legal play" and seen that an arena A can be viewed as just a set of legal plays. When modelling a programming language with game semantics, arenas correspond to types. We thus consider a type as nothing more than a given set of *admissible traces*.

To give a game semantics to a program P of type T, we specify a subset $[\![P]\!] \subseteq [\![T]\!]$ of those admissible traces that the program can actually perform. More precisely, we associate to P a set of P-ending plays with intended interpretation that, if $sab \in [\![P]\!]$, then P continues by playing b (with appropriate pointer) if O continues by playing a (with appropriate pointer unless a is initial) after s. In other words, a strategy describes the deterministic behaviour of P parametrized by all deterministic, but unknown, possible behaviours of O. By convention, all programs contain the empty play ε as "starting point".

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

Set-theoretic inclusion $\subseteq$ determines a partial ordering on the set of all strategies for any given arena. Moreover, the union of a directed set of strategies yields its lub (w.r.t. the inclusion order above) so our partial order is a CPO. If we drop the second condition (of determinism), we can even take the union of an arbitrary set of strategies, *i.e.* we have a complete lattice of ***nondeterministic*** strategies. However, we will make little use of nondeterministic strategies in these notes.

Let us recap the development to date. The notion of arena corresponds to what we called a "game" in the introduction. However, historically, a game was defined to be an arena plus a (prefix-closed) subset of its legal plays, the idea being that only these *valid* plays could be played by a strategy for that game. In this document, we don't use this approach, instead using an "arena only" setting where we seek to constrain the behaviour of strategies directly, with explicitly given (and analysed) constraints, rather than relying on the valid plays of a game to restrict play.

2.5 Composition of strategies

Suppose we have strategies σ and τ for arenas A and $\neg A$ respectively. The response of τ (if any!) to the initial Question q must be some initial move of A and thus this P-move of τ can also be seen as an O-move for σ. The response of σ (if any!) to this can be seen as a new input for τ and so on. In other words, (with the exception of the initial move of $\neg A$) σ behaves as O for τ and τ behaves as O for σ.

Note, however, that we don't seem to have any way of stopping the interaction: every time a strategy plays a move, it immediately gets considered as the next input to the other; so either one strategy (eventually) fails to respond, thus bringing the whole interaction to a halt, *i.e.* deadlock; otherwise, the two strategies must engage in "infinite chattering", *i.e.* livelock.

We can avoid this problem by allowing partial interaction between strategies: rather than having σ and τ on arenas A and $\neg A$ respectively, we generalize to σ and τ on $A \Rightarrow B$ and $B \Rightarrow C$. Interaction only takes place in the "no man's land" of B and each strategy has a "private" region (A for σ, C for τ) where moves can be played without influencing the other strategy. In particular, if τ plays a move in C, the next O-move is not the responsibility of σ but of the external Opponent. On the other hand, if τ plays in B, this becomes a new input for σ in exactly the same way as before.

Such an idea is easily formalized with the notion of a ***legal interaction*** for arenas A, B and C: a pointing string u over alphabet $M_A + M_B + M_C$ such that $u \restriction A, B \in \mathcal{L}_{A \Rightarrow B}$, $u \restriction B, C \in \mathcal{L}_{B \Rightarrow C}$ and $u \restriction A, C$ satisfies OP-alternation. (It easily follows that $u \restriction A, C \in \mathcal{L}_{A \Rightarrow C}$.) We write $\mathcal{I}(A, B, C)$ for the set of all legal interactions for A, B and C (in that order).

By keeping track of who moves next in each component of a legal interaction, we obtain the following diagram that exposes the structure underlying interaction. We start in state OOO, *i.e.* O to play next in $A \Rightarrow B$, $B \Rightarrow C$ and $A \Rightarrow C$.

$$\begin{tikzcd}
A \Rightarrow B \quad B \Rightarrow C \quad A \Rightarrow C \\
\boxed{\mathsf{O} \quad \mathsf{P} \quad \mathsf{P}} \arrow[d, shift left] \arrow[dd, bend left=60, "B"] \\
\boxed{\mathsf{O} \quad \mathsf{O} \quad \mathsf{O}} \arrow[u, shift left, "C"] \arrow[d, shift left] \\
\boxed{\mathsf{P} \quad \mathsf{O} \quad \mathsf{P}} \arrow[u, shift left, "A"] \arrow[uu, bend left=60, "B"]
\end{tikzcd}$$

This shows that a legal interaction consists of a sequence of ***sandwiches*** of the form "O-move of $A \Rightarrow C$, followed by moves of B, followed by a P-move of $A \Rightarrow C$": if the number of B-moves is odd, one of the outer moves comes from A and the other from C whereas, if the number of B-moves is even, both outer moves come from the "same side". If $u \in \mathcal{I}(A, B, C)$ ends in state OOO, we define $\mathsf{jp}(u)$ to be the shortest prefix of u that witnesses $\mathsf{jp}(u \restriction A, C)$ and $\mathsf{trnct}(u)$ to be the shortest prefix of u witnessing $\mathsf{ip}(u \restriction A, C)$.

Of the three permissible states, only OOO corresponds to an even-length outer projection (O to move next in $A \Rightarrow C$)—but, since both inner projections also have even length in this state, we see that an even-length play in $A \Rightarrow C$ always comes from even-length plays of $A \Rightarrow B$ and $B \Rightarrow C$. The other two states, *i.e.* POP and OPP, correspond to having one even- and one odd-length inner projection, thereby yielding an odd-length outer projection: once Opponent makes a move in the outside, the interaction oscillates between these two states until such time as it "reemerges" from B with a Player move of $A \Rightarrow C$, taking us back to the OOO state.

The composite of $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ is defined by setting

$$\sigma \parallel \tau = \{u \in \mathcal{I}(A, B, C) \mid u \restriction A, B \in \sigma \wedge u \restriction B, C \in \tau\}$$

and then $\sigma ; \tau = \{u \restriction A, C \mid u \in \sigma \parallel \tau\}$. In words, $\sigma ; \tau$ consists of the external projections of all interactions that σ and τ mutually accept. Note that all actual interaction between σ and τ gets excised from their composite, leaving a sequence of external responses, by P in $A \Rightarrow C$, to moves of O in $A \Rightarrow C$—the "bread" of the sandwiches.

Proposition 2.5.1 *If σ and τ are nondeterministic strategies for $A \Rightarrow B$ and $B \Rightarrow C$ respectively then $\sigma ; \tau$ is a nondeterministic strategy for $A \Rightarrow C$.*

Proof Any $s \in \sigma ; \tau$ has a witness $u \in \mathcal{I}(A, B, C)$ satisfying $u \restriction A, B \in \sigma$ and $u \restriction B, C \in \tau$ so u is in the OOO state. Any even-length prefix t of s must then witnessed by a prefix v of u which will also be in the OOO state, hence $v \restriction A, B \in \sigma$ and $v \restriction B, C \in \tau$ and so $t \in \sigma ; \tau$. ■

This establishes that nondeterministic strategies compose; we now need to verify that determinism is preserved by composition. If we interact deterministic σ and τ for A and $\neg A$ respectively, a single interaction is traced out: O can never branch since no "no man's land" exists. In the more general case, where σ and τ are strategies for $A \Rightarrow B$ and $B \Rightarrow C$ respectively, O can branch (in $A \Rightarrow C$) and so more than one interaction becomes possible. However, each possible interaction witnesses a different play when projected to $A \Rightarrow C$.

Lemma 2.5.2 (unique witness) *If σ and τ are strategies for $A \Rightarrow B$ and $B \Rightarrow C$ respectively then, for all $s \in \sigma ; \tau$, there exists a unique $u \in \mathcal{I}(A, B, C)$ such that $s = u \restriction A, C$, $u \restriction A, B \in \sigma$ and $u \restriction B, C \in \tau$.*

Proof If $s \in \sigma ; \tau$ has two distinct witnesses u_1 and u_2 then they must differ only in B. Set $v = u_1 \wedge u_2$; it must therefore be the case that v is in state OPP (resp. POP). But then τ (resp. σ) violates determinism at $v \restriction B, C$ (resp. $v \restriction A, B$) which cannot happen, by assumption. ■

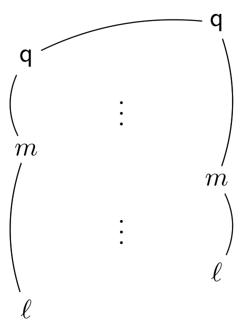
Proposition 2.5.3 *If σ and τ are strategies for $A \Rightarrow B$ and $B \Rightarrow C$ respectively then $\sigma ; \tau$ is a strategy for $A \Rightarrow C$.*

Proof Suppose we have $s, t \in \sigma ; \tau$ with unique witnesses u and v but where $r = s \wedge t$ has odd length. Let $w = u \wedge v$, an interaction that must be in state OPP or POP, so that $r = w \restriction A, C$. The longest prefix of w in state OOO uniquely witnesses the longest even-length common prefix of s and t and so either τ or σ would have to violate determinism at w. ■

2.6 Arenas and strategies as a category

We can organize the above development into a category $\mathbf{G}$: objects are arenas, arrows $f : A \to B$ are strategies $\sigma_f : A \Rightarrow B$. Given arrows $f : A \to B$ and $g : B \to C$, represented by strategies $\sigma_f : A \Rightarrow B$ and $\sigma_g : B \Rightarrow C$, we define $g \circ f : A \to C$ to be $\sigma_f ; \sigma_g$, the composite of σ_f and σ_g. We've just seen that this is well-defined. To complete the construction of our category, we must now verify that we have an identity arrow for each object and that composition is associative.

Identities For an arena A, a typical play of $\mathsf{id}_A : A \Rightarrow A$ looks like:

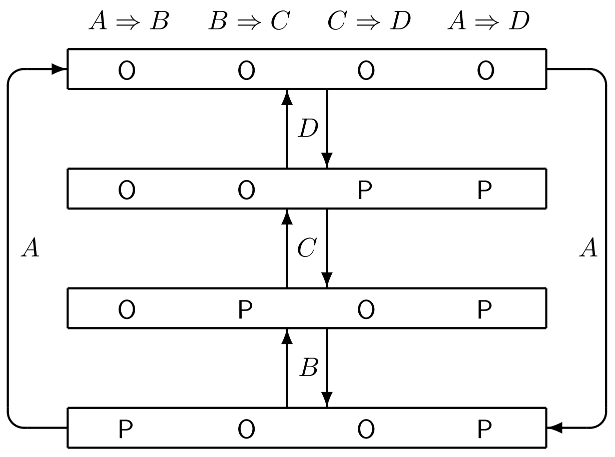


Every time O makes a move, P echoes that move with the same justification pointer on the "other side" of the arrow, the only exception being if O plays an initial move q. In this case, P responds with q on the left hand side, pointing to the q just played by O. An elementary proof establishes that, for any strategy $\sigma : A \Rightarrow B$, we have $\mathsf{id}_A \,;\sigma = \sigma = \sigma\,; \mathsf{id}_B$. So we have identities.

Associativity Suppose now that we have strategies $\sigma : A \Rightarrow B$, $\tau : B \Rightarrow C$ and $\upsilon : C \Rightarrow D$. We wish to show that $(\sigma\,;\tau)\,;\upsilon = \sigma\,;(\tau\,;\upsilon)$. To this end, consider any $s \in (\sigma\,;\tau)\,;\upsilon$. The play s comes from a unique witness $u \in \mathcal{I}(A,C,D)$ where $u \upharpoonright C,D \in \upsilon$ and $u \upharpoonright A,C \in \sigma\,;\tau$. In turn, $u \upharpoonright A,C$ comes from a unique $v \in \mathcal{I}(A,B,C)$ where $v \upharpoonright A,B \in \sigma$ and $v \upharpoonright B,C \in \tau$.

We need to combine u and v in order to construct a witness for s in $\sigma\,;(\tau\,;\upsilon)$. In order to do this, we extend the notion of legal interaction to four arenas: $w \in \mathcal{I}(A,B,C,D)$ iff $w \upharpoonright A,B$, $w \upharpoonright B,C$ and $w \upharpoonright C,D$ are legal in $A \Rightarrow B$, $B \Rightarrow C$ and $C \Rightarrow D$ respectively, and $w \upharpoonright A,D$ satisfies OP-alternation. (Indeed, the notion of legal interaction naturally extends to any *finite list* of arenas.)

Let $w \in \mathcal{I}(A,B,C,D)$. In between any two *consecutive* moves of $w \upharpoonright A,C$, we can have moves entirely from B or entirely from D but we cannot have moves from B *and* D. This can most easily be seen by deriving the extended state diagram for legal interactions:



A B-move can be played from states POOP and OPOP while a D-move only from OOOO and OOPP. Thus, in between any B-move and D-move of w, there must be an move from A or C. We call this property ***locality of transfer*** since it means that σ cannot directly "pass control" to υ; it must go via τ.

Lemma 2.6.1 (zipping) *Given $u \in \mathcal{I}(A, C, D)$ and $v \in \mathcal{I}(A, B, C)$ such that $u \upharpoonright A, C = v \upharpoonright A, C$, there exists a unique $w \in \mathcal{I}(A, B, C, D)$ such that $u = w \upharpoonright A, C, D$ and $v = w \upharpoonright A, B, C$.*

Proof Induction on $|u \upharpoonright A, C|$. If $u \upharpoonright A, C = \varepsilon$ then u plays only in D so $w = u$. Otherwise, let a be the final move of $u \upharpoonright A, C$. We apply the inductive hypothesis to $u' = u_{<a}$ and $v' = v_{<a}$, yielding unique $w' \in \mathcal{I}(A, B, C, D)$ such that $u' = w' \upharpoonright A, C, D$ and $v' = w' \upharpoonright A, B, C$.

If a is a P-move of $A \Rightarrow C$, u could continue with moves in D after a but a must be the final move of v. Dually, if a is an O-move of $A \Rightarrow C$, v could continue in B but u is in the POP state making D inaccessible. So the last move of $u \upharpoonright A, C$ is also the last move of at least one of u and v. We can thus extend our w' with a followed by the "tail" of u or v as appropriate, yielding $w \in \mathcal{I}(A, B, C, D)$ such that $u = w \upharpoonright A, C, D$ and $v = w \upharpoonright A, B, C$. ■

Clearly, the "mirror image" of this result—zipping $u \in \mathcal{I}(A, B, D)$ with $v \in \mathcal{I}(B, C, D)$—is proved in the same way. Associativity now follows:

Proposition 2.6.2 *If $\sigma : A \Rightarrow B$, $\tau : B \Rightarrow C$ and $\upsilon : C \Rightarrow D$ then $(\sigma\,;\tau)\,;\upsilon = \sigma\,;(\tau\,;\upsilon)$.*

Proof Let $s \in (\sigma\,;\tau)\,;\upsilon$. We apply zipping to the (unique) $u \in \mathcal{I}(A, C, D)$ and $v \in \mathcal{I}(A, B, C)$ witnessing $s \in (\sigma\,;\tau)\,;\upsilon$ and $u \upharpoonright A, C \in \sigma\,;\tau$. This yields a (unique) $w \in \mathcal{I}(A, B, C, D)$ such that $u = w \upharpoonright A, C, D$ and $v = w \upharpoonright A, B, C$. So $w \upharpoonright A, B \in \sigma$, $w \upharpoonright B, C \in \tau$ and $w \upharpoonright C, D \in \upsilon$.

By locality of transfer, we have that $w \upharpoonright B, D$ OP-alternates, hence $w \upharpoonright B, C, D \in \mathcal{I}(B, C, D)$. So $w \upharpoonright A, B \in \sigma$ and $w \upharpoonright B, D \in \tau\,;\upsilon$ witnessing $s \in \sigma\,;(\tau\,;\upsilon)$, the reverse inclusion being proved in the same way (using the "mirror image" zipping lemma). ■

This completes the proof that **G** does indeed form a category as claimed above. We sometimes refer to **G** as the *ambient* category since it provides the most general setting for "doing" game semantics. In the next section, we show how **G** can be seen as an SMCC. All subsequent models can be considered as subcategories of **G** where this symmetric monoïdal closed structure often restricts to a Cartesian closed structure.

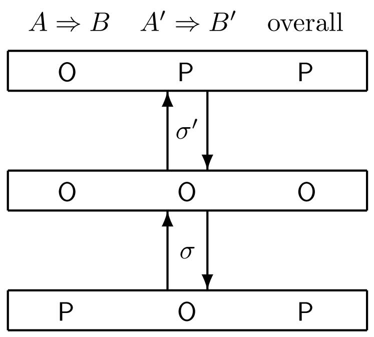
We already know that all homsets of **G** are CPOs. A straightforward argument establishes that composition of strategies is both monotone and continuous with respect to this CPO structure; so **G** "is" a CPO-enriched category.

2.7 Monoïdal closed structure

We extend the $\times$ constructor on arenas to a bifunctor on **G** as follows. Given arrows $\sigma : A \Rightarrow B$ and $\sigma' : A' \Rightarrow B'$, define $\sigma \times \sigma' : A \times A' \Rightarrow B \times B'$ to be

$$\{s \in \mathcal{L}_{(A \times A') \Rightarrow (B \times B')} \mid s \upharpoonright A, B \in \sigma \wedge s \upharpoonright A', B' \in \sigma'\}.$$

This definition makes sense since any $s \in \sigma \times \sigma'$ respects the following simplified version of the state diagram for legal interactions, *i.e.* the switching condition as a property of strategies rather than of games:



If we additionally have strategies $\tau : B \Rightarrow C$ and $\tau' : B' \Rightarrow C'$, a legal interaction $u \in \mathcal{I}(A \times A', B \times B', C \times C')$ where $u \upharpoonright A, B \in \sigma$, $u \upharpoonright A', B' \in \sigma'$, $u \upharpoonright B, C \in \tau$ and $u \upharpoonright B', C' \in \tau'$ can be split into $u \upharpoonright A, B, C \in \sigma \parallel \tau \subseteq \mathcal{I}(A, B, C)$ and $u \upharpoonright A', B', C' \in \sigma' \parallel \tau' \subseteq \mathcal{I}(A', B', C')$.

Lemma 2.7.1 (zipping) *Given $u \in \mathcal{I}(A, B, C)$, $u' \in \mathcal{I}(A', B', C')$ and $s \in \mathcal{L}_{(A\times A')\Rightarrow(C\times C')}$ where $s \upharpoonright A, C = u \upharpoonright A, C$ and $s \upharpoonright A', C' = u' \upharpoonright A', C'$, there exists a unique $w \in \mathcal{I}(A \times A', B \times B', C \times C')$ such that $s = w \upharpoonright A \times A', C \times C'$, $u = w \upharpoonright A, B, C$ and $u' = w \upharpoonright A', B', C'$.*

Proof A typical zipping argument. ■

This "vertical zipping" lemma yields bifunctoriality of $\times$ much as associativity of composition follows from "horizontal zipping" in the previous section.

Proposition 2.7.2 *Let $\sigma : A \Rightarrow B$, $\sigma' : A' \Rightarrow B'$, $\tau : B \Rightarrow C$ and $\tau' : B' \Rightarrow C'$ be arrows of* **G**. *Then we have $(\sigma \times \sigma') ; (\tau \times \tau') = (\sigma ; \tau) \times (\sigma' ; \tau')$.*

Proof If $s \in (\sigma ; \tau) \times (\sigma' ; \tau')$ then $s \upharpoonright A, C \in \sigma ; \tau$ (witnessed by some $u \in \mathcal{I}(A, B, C)$) and $s \upharpoonright A', C' \in \sigma';\tau'$ (witnessed by some $u' \in \mathcal{I}(A', B', C')$). Applying lemma 2.7.1 yields unique $v \in \mathcal{I}(A \times A', B \times B', C \times C')$ such that $s = v \upharpoonright A \times A', C \times C'$, $u = v \upharpoonright A, B, C$ and $u' = v \upharpoonright A', B', C'$. So $v \upharpoonright A \times A', B \times B' \in \sigma\times\sigma'$ and $v \upharpoonright B \times B', C \times C' \in \tau\times\tau'$ so $s = v \upharpoonright A \times A', C \times C' \in (\sigma\times\sigma');(\tau\times\tau')$.

If $s \in (\sigma \times \sigma') ; (\tau \times \tau')$ with witness $u \in \mathcal{I}(A \times A', B \times B', C \times C')$ then $u \upharpoonright A, B \in \sigma$ and $u \upharpoonright B, C \in \tau$ which implies that $u \upharpoonright A, B, C \in \mathcal{I}(A, B, C)$ and $u \upharpoonright A, C \in \sigma ; \tau$. Similarly $u \upharpoonright A', B', C' \in \mathcal{I}(A', B', C')$ and $u \upharpoonright A', C' \in \sigma' ; \tau'$. Hence $s = u \upharpoonright A \times A', C \times C' \in (\sigma ; \tau) \times (\sigma' ; \tau')$. ■

The unit object for $\times$ is the empty arena **1** equipped with the evident copycat strategies $\lambda_A : \mathbf{1} \times A \Rightarrow A$ and $\rho_A : A \times \mathbf{1} \Rightarrow A$. The fact that **1** is a terminal object of **G** means that we can build projections from $A \times B$ by $\pi_\ell = (\mathsf{id}_A \times !_B) ; \rho_A : A \times B \Rightarrow A$ and $\pi_r = (!_A \times \mathsf{id}_B) ; \lambda_A : A \times B \Rightarrow B$.

The re-tagging natural isomorphisms $(M_A + M_B) + M_C \cong M_A + (M_B + M_C)$ and $M_A + M_B \cong M_B + M_A$ in **Set** induce the associativity and commutativity maps $\alpha_{ABC} : (A \times B) \times C \Rightarrow A \times (B \times C)$ and $\gamma_{AB} : A \times B \Rightarrow B \times A$ in **G**.

Finally, since the only difference between $(A \times B) \Rightarrow C$ and $A \Rightarrow (B \Rightarrow C)$ again lies in the tagging of the disjoint unions, $\mathbf{G}(A \times B, C) \cong \mathbf{G}(A, B \Rightarrow C)$, the familiar *currying* isomorphism which we write as $\Lambda(-)$. Uncurrying (for all A and B) $\mathsf{id}_{A \Rightarrow B}$ yields $\epsilon_{AB} : (A \Rightarrow B) \times A \Rightarrow B$, the (family of) *evaluation* map(s) of $\mathbf{G}$ such that, for any $\sigma : A \times B \Rightarrow C$, we have

$$\sigma = (\Lambda(\sigma) \times \mathsf{id}_B) \,;\, \epsilon_{BC}.$$

This establishes that $\mathbf{G}$ is an SMCC (symmetric monoïdal closed category). In an SMCC, each object B can be associated to a functor $(\times B)$ by defining $(\times B)A = A \times B$ and $(\times B)f = f \times \mathsf{id}_B$. The property of being an SMCC can then equivalently be stated as: for all objects B, the functor $(\times B)$ has a right adjoint. This implies a natural isomorphism

$$\frac{(\times B)A \to C}{\overline{A \to (B \Rightarrow)C}}$$

where $(B \Rightarrow)$ is the (specified) right adjoint of $(\times B)$. So, when we define the notion of an SMCC, we don't use $\times$ as a bifunctor; we only use the derived family of functors, one for each object. As a result, the definition of SMCC guarantees a family of right adjoints but doesn't directly specify a "bifunctor" related to the family of right adjoints as $\times$ is related to the family $(\times B)$.

This bifunctor does always exist however: for $f : A \to B$ and $g : C \to D$, $f \Rightarrow g$ maps an arrow $h : B \to C$ to $f \,;\, h \,;\, g$. In the particular case of our category $\mathbf{G}$, we extend the $\Rightarrow$ constructor (on arenas) to strategies in the following way [V. Danos, private communication]:

Given $\sigma : A \Rightarrow B$ and $\upsilon : C \Rightarrow D$, define

$$\sigma \mid\mid\mid \upsilon = \{u \in \mathcal{I}(A, B, C, D) \mid u \restriction A, B \in \sigma \wedge u \restriction C, D \in \upsilon\}$$

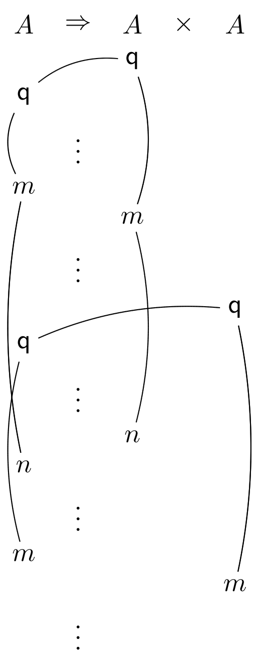
and for $u \in \mathcal{I}(A, B, C, D)$, define a legal play u^* of $(B \Rightarrow C) \Rightarrow (A \Rightarrow D)$ by (trivial relabelling of moves and) replacing all pointers from initial moves of A by their pointers in $u \restriction A, D$. Now define $\sigma \Rightarrow \upsilon : (B \Rightarrow C) \Rightarrow (A \Rightarrow D)$ by $\{u^* \mid u \in \sigma \mid\mid\mid \upsilon\}$. This construction defines a functor from $\mathbf{G}^{op} \times \mathbf{G}$ to $\mathbf{G}$, *i.e.* a bifunctor of "mixed variance", contravariant on the left, covariant on the right:

Suppose we additionally have $\sigma' : A' \Rightarrow B'$, $\tau : B \Rightarrow C$ and $\tau' : B' \Rightarrow C'$. If $s \in (\sigma' \Rightarrow \tau) \,;\, (\sigma \Rightarrow \tau')$, it has some witness $u \in \mathcal{I}(C \Rightarrow A', B \Rightarrow B', A \Rightarrow C')$. Since $u \restriction C \Rightarrow A', B \Rightarrow B' \in \sigma' \Rightarrow \tau$, it must have the form u_1^* for some $u_1 \in \mathcal{I}(B, C, A', B')$. In a similar way, $u \restriction B \Rightarrow B', A \Rightarrow C'$ has the form u_2^* for some $u_2 \in \mathcal{I}(A, B, B', C')$ such that $u_1 \restriction B, B' = u_2 \restriction B, B'$. A mundane zipping, inserting u_1 into u_2, yields a legal interaction in $\mathcal{I}(A, B, C, A', B', C')$ which, by hiding B and B', in turn yields the $v \in \mathcal{I}(A, C, A', C')$ such that $s = v^*$. So $s \in (\sigma \,;\sigma') \Rightarrow (\tau \,;\tau')$. The same reasoning "going backwards" proves the reverse inclusion.

2.8 Single-threaded strategies

We've already seen that the empty arena $\mathbf{1}$ is a terminal object of the category $\mathbf{G}$. So, for any $\sigma : A \Rightarrow B$, $!_A = \sigma;!_B$, *i.e.* all strategies commute with the $!_A$s.

We also have, for any arena A, a "comultiplication" strategy $\Delta_A : A \Rightarrow A \times A$ which interleaves the two copies of A on the RHS into the one copy on the LHS:



We want to characterize those strategies that commute with comultiplication, *i.e.* those $\sigma : A \Rightarrow B$ satisfying $\sigma \,;\, \Delta_B = \Delta_A \,;\, \sigma \times \sigma$, since such strategies form a Cartesian closed subcategory of $\mathbf{G}$ (for which the family of strategies Δ_A becomes a *natural* transformation).

A ***thread*** t of A is a legal play with at most one initial move so that t is either ε or some $s \in \mathcal{L}_A$ whose only initial move is its first move. A legal play $s \in \mathcal{L}_A$ is ***well-opened*** iff no initial move is ever repeated (so we can have several *different* initial moves). Beware that this definition conflicts with much of the literature of game semantics which conflates the concepts of thread and well-opened play. We write $\mathcal{L}_A^{\mathsf{wo}}$ for the set of all well-opened plays of A.

Let $s \in \mathcal{L}_A$ and consider that initial move m of s that hereditarily justifies the last move of s. The subsequence of all moves hereditarily justified by m—the "connected component" of s relative to m and the justification pointers—is called the ***current thread*** of s and written $\lceil s \rceil$. The current thread *need not* be a legal play. We say that $s \in \mathcal{L}_A$ is ***well-threaded*** iff, for all $s' \sqsubseteq^{\mathsf{P}} s$, s'_ω is justified by some move of $\lceil \mathsf{ip}(s') \rceil$; in words, P never changes thread. The current thread of a well-threaded play is always legal.

If we have a well-threaded strategy σ (*i.e.* all $s \in \sigma$ are well-threaded) let $\lceil \sigma \rceil$ be $\{\lceil s \rceil \mid s \in \sigma\}$. This defines a set of even-length legal plays closed under prefixes. In the other direction, if T is a coherent (*i.e.* for any $t_1, t_2 \in T$, $t_1 \wedge t_2$ has even length) set of even-length threads of A, define

$$\begin{aligned}
\mathsf{ST}_0(T) &= \{\varepsilon\} \\
\mathsf{ST}_{i+1}(T) &= \{sab \in \mathcal{L}_A \mid s \in \mathsf{ST}_i(T) \wedge \lceil sab \rceil \in T\} \\
\mathsf{ST}(T) &= \bigcup_{i \in \mathbf{N}} \mathsf{ST}_i(T).
\end{aligned}$$

It can readily be shown that $\sigma \subseteq \mathsf{ST}\lceil\sigma\rceil$ and $\lceil\mathsf{ST}(T)\rceil \subseteq T$. If $\sigma = \mathsf{ST}\lceil\sigma\rceil$, we say that σ is a ***single-threaded*** strategy and if $T = \lceil\mathsf{ST}(T)\rceil$, we say that T is a ***thread function***. We can additionally establish that $\mathsf{ST}\lceil\sigma\rceil$ is always the *least* single-threaded strategy containing well-threaded σ (and dually that $\lceil\mathsf{ST}(T)\rceil$ is the *greatest* thread function contained in T) which implies a one-2-one correspondence between single-threaded strategies and thread functions: $\sigma = \mathsf{ST}\lceil\sigma\rceil$ and $\lceil\mathsf{ST}(T)\rceil = T$.

An elementary (but quite long) proof establishes that single-threaded strategies correspond exactly to those σ commuting with $!_A$ and Δ_A.

$$\begin{tikzcd}
A \arrow[r, "\Delta_A"] \arrow[d, "\sigma"'] & A \times A \arrow[d, "\sigma\times\sigma"] \\
B \arrow[r, "\Delta_B"'] & B \times B
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[d, "\sigma"'] \arrow[dr, "!_A"] & \\
B \arrow[r, "!_B"'] & \mathbf{1}
\end{tikzcd}$$

For general reasons, the lluf subcategory of $\mathbf{G}$ consisting of arenas and single-threaded strategies is Cartesian closed. For σ and τ composable single-threaded strategies, the *threads* of their composite (which determine the actual composite $\sigma;\tau$) come from interaction between *threads* of τ and *plays* of σ: $\lceil\sigma\,;\tau\rceil = \sigma;\lceil\tau\rceil$.

3 Innocent strategies

The behaviour of strategies as defined above may depend arbitrarily on the entire history of play; we can think of them as "strategies of total information". Single-threaded strategies impose the restriction that the strategy's behaviour depend only on the current thread—what we might call a "strategy of partial information". In this section, we introduce a more subtle class of restricted strategies whose behaviour depends only on a subsequence of the current thread (determined by the justification pointers) called the P-*view*. These *innocent* strategies form a subcategory $\mathbf{I}$ of $\mathbf{G}$ where, as for single-threaded strategies, the monoïdal structure becomes a bona fide product and it follows that $\mathbf{I}$ is a CCC.

3.1 P-views and P-visibility

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 (whence we stop).

In general, a P-move $m \in s$ can "lose its pointer": if its justifier n lies (strictly) underneath an O-to-P pointer in $\ulcorner s_{<m} \urcorner$ then n does not occur in $\ulcorner s_{<m} \urcorner$. We nonetheless complete the definition of P-view by specifying 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.

We say that 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 this implies that a P-move of $\ulcorner t \urcorner$ may lose its pointer for t some proper prefix of s. We lift the definition of P-visibility pointwise to strategies: σ satisfies P-***visibility*** iff all $s \in \sigma$ do. Note that, for s in P-vis σ as opposed to arbitrary P-vis s, 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.

A move in a legal interaction $u \in \mathcal{I}(A, B, C)$ is a ***generalized* O*-move*** (resp. ***generalized* P*-move***) in ***component*** $\mathcal{L}$ (resp. $\mathcal{R}$) iff it is an O- (resp. P-)move of $A \Rightarrow B$ (resp. $B \Rightarrow C$). So an O-move of $A \Rightarrow C$ or any move of B is a generalized O-move (in the appropriate component) while a P-move of $A \Rightarrow C$ or any move of B is a generalized P-move (in the appropriate component). A move in B is thus a generalized O-move in one component and a generalized P-move in the other. Note that a generalized P-move is *always* immediately preceded by a generalized O-move in the *same component.* Finally, we say that moves from A and C are ***external*** moves of u and that moves from B are ***internal*** moves of u.

We extend the notion of P-***view*** to legal interaction $u \in \mathcal{I}(A, B, C)$ with the following inductive definition.

$$\begin{aligned}
\ulcorner un \urcorner &= n, && \text{if } n \text{ is an initial move of } C;\\
\ulcorner umvn \urcorner &= \ulcorner um \urcorner \cdot n, && \text{if } n \text{ is an external O-move of } u \text{ justified by } m;\\
\ulcorner um \urcorner &= \ulcorner u \urcorner m, && \text{if } m \text{ is a generalized P-move.}
\end{aligned}$$

Recall that a legal interaction can be viewed as a sequence of sandwiches: O-move of $A \Rightarrow C$, moves of B, P-move of $A \Rightarrow C$. The P-view of a legal interaction thus consists of a subsequence of sandwiches determined by the pointers from external O-moves of u. Each sandwich of u is either removed entirely or left untouched in the P-view.

Just as the P-view of a legal play can lose pointers from P-moves and so need not itself be a legal play, $\ulcorner u \urcorner$ can lose pointers from its *generalized* P-moves and so need not be a legal interaction. However, if u results from the interaction of two P-vis strategies, *i.e.* all P-ending prefixes of *both* internal projections $u \upharpoonright A, B$ and $u \upharpoonright B, C$ satisfy P-visibility, then the following lemma guarantees that $\ulcorner u \urcorner$ is legal:

Lemma 3.1.1 *If $u \in \mathcal{I}(A, B, C)$ such that $u \upharpoonright A, B \in \sigma$ and $u \upharpoonright B, C \in \tau$ for* P*-vis σ and τ, and m is a generalized* P*-move of u in component*[*] *X, then $\ulcorner u_{\leqslant m} \urcorner \in \mathcal{I}(A, B, C)$.*

Proof By induction on the length of $u_{\leqslant m}$; two cases, depending on the move n immediately preceding m in u.

[*]We use X as a metavariable ranging over $\{\mathcal{L}, \mathcal{R}\}$ when we neither know nor care which of the two components is being referred to, whence we use Y to denote *the other* component.

If n is a generalized P-move of u (necessarily in component Y), we apply the inductive hypothesis to get $\ulcorner u_{<m} \urcorner \in \mathcal{I}(A,B,C)$. If n is an O-move of $A \Rightarrow C$, either it's initial (in which case the claim is trivial—this encompasses the base case) or it's justified by ℓ, in which case we apply the inductive hypothesis to $u_{\leqslant \ell}$ to get $\ulcorner u_{<m} \urcorner = \ulcorner u_{\leqslant \ell} \urcorner \cdot n \in \mathcal{I}(A,B,C)$. So $\ulcorner u_{<m} \urcorner \in \mathcal{I}(A,B,C)$.

By P-visibility, m points in $\ulcorner u_{<m} \restriction X \urcorner$. The last move of this, n, points to a generalized P-move m' in X which occurs in $\ulcorner u_{<m} \urcorner$. In turn, m' is immediately preceded by a generalized O-move in X, n', which also occurs in $\ulcorner u_{<m} \urcorner$. If we continue in this way, we find that $\ulcorner u_{<m} \restriction X \urcorner$ is a subsequence of $\ulcorner u_{<m} \urcorner$. Hence we can attach m to the end of $\ulcorner u_{<m} \urcorner$ with correct justification pointer, yielding $\ulcorner u_{\leqslant m} \urcorner \in \mathcal{I}(A,B,C)$ as required. ■

Note that, for $u \in \mathcal{I}(A,B,C)$ satisfying the hypotheses of this lemma, $\ulcorner u \urcorner \restriction A, C = \ulcorner u \restriction A, C \urcorner$. So, as an immediate corollary of this lemma, we have that P-visibility is preserved by composition: the composite of P-vis $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ is again a P-vis strategy since, if $s \in \sigma \,;\, \tau$, its unique witness $u \in \mathcal{I}(A,B,C)$ satisfies the hypotheses of the above lemma; hence $\ulcorner u \urcorner \in \mathcal{I}(A,B,C)$ and so $\ulcorner s \urcorner = \ulcorner u \urcorner \restriction A, C \in \mathcal{L}_{A \Rightarrow C}$.

The fact that P-vis strategies compose illustrates (in one particular case) how the arena-only approach to game semantics [advocated here] loses none of the generality of the arena-plus-plays approach:

A typical game (in the latter approach) consists of an arena together with its set of P-vis plays; this suffices to describe models of PCF, FPC or Idealized Algol, with or without control operators. In such a game, we *can only play* P-vis plays; in other words, P-visibility is enforced by the objects. In the arena-only setting, we restrict attention to the subcategory of P-vis strategies, but keeping the same underlying objects, *i.e.* just arenas. So, while non-P-vis plays could still be played in principle, they are *never played* by P-vis strategies; here P-visibility is enforced by the arrows.

3.2 Innocence defined by a Galois connection

If we upgrade the notion of view to being a *property* of plays—s *is* a P-view iff $s = \ulcorner s \urcorner$—then we can speak unambiguously of "a set of P-views". A (resp. nondeterministic) strategy containing only P-views is sometimes called a ***view function*** (resp. ***view relation***). If V is a set of P-ending P-views, define

$$\begin{aligned} T_0(V) &= \{\varepsilon\} \\ T_{n+1}(V) &= \{s \in \mathcal{L}_A \mid \mathsf{ip}(\mathsf{ip}(s)) \in T_n(V) \wedge \exists t \in V.\ \ulcorner s \urcorner = t\} \\ \mathsf{tr}(V) &= \bigcup_{n \in \mathbf{N}} T_n(V) \end{aligned}$$

This builds a (potentially nondeterministic) strategy whose response to an O-ending play s depends only on its P-view $\ulcorner s \urcorner$. In other words, every play of $\mathsf{tr}(V)$ is an interleaving of P-views in V that only interleaves "OP pairs" of moves (so as to preserve OP-alternation). We call this an OP-***interleaving***. Conversely, a P-vis strategy σ induces a view relation $\ulcorner \sigma \urcorner = \{\ulcorner s \urcorner \mid s \in \sigma\}$. In general, $\ulcorner \sigma \urcorner$ may be nondeterministic, even if σ is deterministic, but we always have $\sigma \subseteq \mathsf{tr}(\ulcorner \sigma \urcorner)$.

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.* extend t by s_ω with the "same" pointer as in s. We can do this since s_ω points in $\ulcorner \mathsf{ip}(s) \urcorner = \ulcorner t \urcorner$ (by assumption).

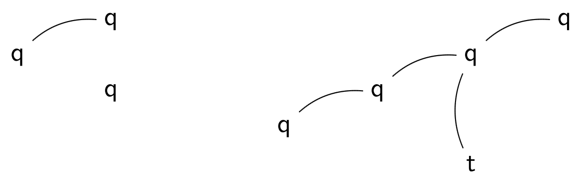
A *deterministic* P-vis strategy σ for which $\sigma = \mathsf{tr}(\ulcorner \sigma \urcorner)$ is called an ***innocent*** strategy. Such strategies can be directly characterized by

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

In words, the response of an innocent strategy depends only on the current P-view, cf. the above definition of tr. One can go on to show that $\ulcorner \mathsf{tr}(V) \urcorner \subseteq V$ and hence that tr and fn form a Galois connection. So $\mathsf{tr} \circ \mathsf{fn}$ is a closure operator (returning the smallest innocent strategy containing its input) and $\mathsf{fn} \circ \mathsf{tr}$ is an interior operator (returning the largest view function contained in its input). In this way, we establish a one-to-one correspondence between view functions and innocent strategies.

A view function represents the *minimum* information, the set of P-views, that determines an innocent strategy. We could redefine the category of innocent strategies using view functions as arrows—an innocent strategy for $A \Rightarrow B$ is a view function for the same arena—but the restatement of the definition of composition in terms of view functions makes the accompanying proofs much more intricate and difficult to understand.

In some sense, a view function perfectly captures the idea of an innocent strategy in isolation, *i.e.* a term all by itself, but to compose innocent strategies with the usual "parallel composition plus hiding" definition (as opposed to a "view by view" definition), we need more information in our arrows than is contained in a view function: in general, we need to close under OP-interleaving, *i.e.* apply $\mathsf{tr}(-)$, *e.g.* when Opponent plays a move such as the last moves of



with no pointer or which points beyond the immediately preceding move, the resulting play cannot be a P-view and so, from the parallel-composition-plus-hiding perspective, a view function could never respond.

3.3 Innocent strategies

In the previous section, we have shown a way of defining innocent strategies as being those (necessarily) P-vis strategies entirely determined by their view function, *i.e.* those strategies (that can be) generated by "completing" a P-vis strategy (by applying the closure operator of the previous section). We also noted an equivalent characterization: a P-vis strategy σ is innocent iff

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

In this definition, the play $s \in \sigma$ could be any legal play but in fact it suffices to consider just P-views (since $\mathsf{match}(s,t) = \mathsf{match}(\ulcorner s \urcorner, t)$). This observation leads to a third, perhaps simpler, characterization of innocence as P-visibility plus

$$s \in \mathcal{L}_A \wedge \mathsf{ip}(s) \in \mathsf{dom}(\sigma) \wedge \ulcorner s \urcorner \in \ulcorner \sigma \urcorner \;\Rightarrow\; s \in \sigma.$$

To see the equivalence of these two definitions, let σ be a P-vis strategy for A. If σ satisfies the first definition, suppose we have some $s \in \mathcal{L}_A$ such that $\mathsf{ip}(s) \in \mathsf{dom}(\sigma)$ and $\ulcorner s \urcorner \in \ulcorner \sigma \urcorner$. So, for some $t \in \sigma$, $\ulcorner t \urcorner = \ulcorner s \urcorner$ and hence $s = \mathsf{match}(t, \mathsf{ip}(s)) \in \sigma$ as required. For the other direction, if $s \in \sigma$, $t \in \mathsf{dom}(\sigma)$ where $\ulcorner \mathsf{ip}(s) \urcorner = \ulcorner t \urcorner$ then $\mathsf{match}(s,t) \in \mathcal{L}_A$, $\mathsf{ip}(\mathsf{match}(s,t)) = t \in \mathsf{dom}(\sigma)$ and $\ulcorner \mathsf{match}(s,t) \urcorner = \ulcorner s \urcorner \in \ulcorner \sigma \urcorner$. Hence $\mathsf{match}(s,t) \in \sigma$ as required.

The next lemma plays a vital role in the proof that innocent strategies compose.

Lemma 3.3.1 *If $u \in \mathcal{I}(A,B,C)$ such that $u \upharpoonright A,B \in \sigma$ and $u \upharpoonright B,C \in \tau$ for P-vis σ and τ, and m is a generalized O-move of u in component X, then $\ulcorner u_{\leqslant m} \upharpoonright X \urcorner = \ulcorner \ulcorner u_{\leqslant m} \urcorner \upharpoonright X \urcorner$.*

Proof By induction on the length of $u_{\leqslant m}$. If m is an initial move of $u \upharpoonright X$ then $\ulcorner u_{\leqslant m} \upharpoonright X \urcorner = m = \ulcorner \ulcorner u_{\leqslant m} \urcorner \upharpoonright X \urcorner$. Otherwise, m is either an O-move of $A \Rightarrow C$ or a generalized P-move in component Y. In either case, the justifier n of m occurs in $\ulcorner u_{\leqslant m} \urcorner$.

We finish by calculating

$$\begin{aligned}
\ulcorner \ulcorner u_{\leqslant m} \urcorner \upharpoonright X \urcorner &= \ulcorner \ulcorner u_{\leqslant n} \urcorner \upharpoonright X \urcorner \cdot m \\
&= \ulcorner \ulcorner u_{<n} \urcorner \upharpoonright X \urcorner n \cdot m \\
&= \ulcorner u_{<n} \upharpoonright X \urcorner n \cdot m \\
&= \ulcorner u_{\leqslant n} \upharpoonright X \urcorner \cdot m \\
&= \ulcorner u_{\leqslant m} \upharpoonright X \urcorner
\end{aligned}$$

using the inductive hypothesis and the definition of P-view. ∎

Essentially, this lemma says that $\ulcorner u \urcorner$ not only witnesses $\ulcorner u \upharpoonright A, C \urcorner$ but that it contains all the information needed for two interacting *innocent* strategies to *build a witness* for $\ulcorner u \upharpoonright A, C \urcorner$.

This allows us to extend the function match to legal interactions. Suppose we have $u, v \in \sigma \| \tau \subseteq \mathcal{I}(A,B,C)$ where σ and τ are innocent strategies, u witnesses $s \in \sigma \,;\, \tau$ and v is the minimum witness of $t \in \mathsf{dom}(\sigma \,;\, \tau)$—a legal interaction ending with an external O-move. Suppose further that $\ulcorner v \urcorner = \ulcorner \mathsf{trnct}(u) \urcorner$ where $\mathsf{trnct}(u)$ is that legal interaction obtained by removing the "tail" of generalized P-moves from u, *i.e.* the minimum witness of $\mathsf{ip}(s)$.

Let X be the component where the last move of v occurs. By the above lemma, we know that $\ulcorner v \upharpoonright X \urcorner = \ulcorner \ulcorner v \urcorner \upharpoonright X \urcorner = \ulcorner \ulcorner \mathsf{trnct}(u) \urcorner \upharpoonright X \urcorner = \ulcorner u \upharpoonright X \urcorner$. Since σ and τ are innocent, the response (in component X) to $\ulcorner v \upharpoonright X \urcorner$ is the same as to $\ulcorner u \upharpoonright X \urcorner$. This reasoning can be iterated, alternating between the two components, to extend v with the same "tail" of generalized P-moves that was removed from u to get $\mathsf{trnct}(u)$. We thus end up with a legal interaction $\mathsf{match}(u,v)$ witnessing that $t' \in \mathsf{ie}(t)$ such that $\ulcorner t' \urcorner = \ulcorner s \urcorner$.

Proposition 3.3.2 *If $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ are innocent strategies then so is their composite $\sigma\,;\tau : A \Rightarrow C$.*

Proof We use the third (and final) characterization of innocence. Suppose $s \in \mathcal{L}_{A\Rightarrow C}$ where $\mathsf{ip}(s) \in \mathsf{dom}(\sigma\,;\tau)$ and $\ulcorner s \urcorner \in \ulcorner \sigma\,;\tau \urcorner$. Then we have some $t \in \sigma\,;\tau$, witnessed by $v \in \sigma \parallel \tau$, such that $\ulcorner t \urcorner = \ulcorner s \urcorner$. Moreover, we have a minimum witness u for $\mathsf{ip}(s)$.

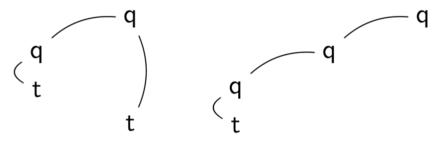
So $\ulcorner u \urcorner \restriction A, C = \ulcorner \mathsf{ip}(s) \urcorner = \ulcorner \mathsf{ip}(t) \urcorner = \ulcorner \mathsf{trnct}(v) \urcorner \restriction A, C$ and, by the unique witness lemma 2.5.2, we conclude that $\ulcorner u \urcorner = \ulcorner \mathsf{trnct}(v) \urcorner$. Hence $\mathsf{match}(v, u)$ witnesses $\mathsf{match}(t, \mathsf{ip}(s)) = s \in \sigma\,;\tau$. ■

We have now established that innocent strategies compose and so form a subcategory **I** of **G**. It can easily be checked that **G**'s SMCC structure carries over to **I**. More importantly, this SMCC structure in fact restricts to a CCC structure on **I**:

Given innocent strategies $\sigma : A \Rightarrow B$ and $\tau : A \Rightarrow C$, the strategy $\langle \sigma, \tau \rangle$, defined as $\Delta_A\,;(\sigma \times \tau)$, clearly satisfies $\langle \sigma, \tau \rangle\,;\pi_\ell = \sigma$ and $\langle \sigma, \tau \rangle\,;\pi_r = \tau$ and is unique (among innocent, or even single-threaded strategies) in doing so. The closed structure of **G** carries over immediately so that **I** is a CCC.

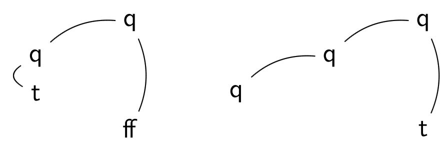
3.4 Typically non-innocent behaviours

A deterministic P-vis strategy σ can violate innocence in two quite different ways. In the arena (`com` $\Rightarrow$ `com`) $\Rightarrow$ `com`, the strategy `ccc` defined by (taking the even-length prefix closure of) the plays



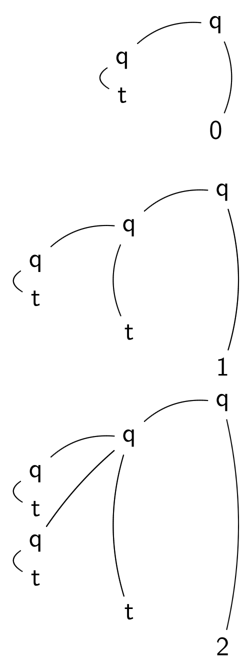
fails to be innocent since the second play could be extended with the Answer `t` (in the middle), yielding the same view as the first play, yet the strategy would fail to respond. This implies a sensitivity to information unavailable in the view, in this case whether the argument (of type `com`) to the input (of type `com` $\to$ `com`) has been tested for convergence or not.

Note how this differs from a strictness test: strictness (or otherwise) depends on the input, not on the input's argument(s). We can detect strictness with $\lambda f \mu\alpha$ (`conv` $(f)[\alpha]$`t ff`) of type (`com` $\to$ `com`) $\to$ `bool`, a standard (and perfectly innocent) idiom of μPCF:



Our second example of a (deterministic P-vis) strategy, with behaviour sensitive to information unavailable in the P-view, lives in the arena (`com` $\Rightarrow$ `com`) $\Rightarrow$ `nat`.

The strategy ccn doesn't merely depend on whether or not the argument to its input converges, but *how many times* it is asked to converge:



etc. This kind of behaviour, unthinkable in PCF, corresponds to the following "copy counting" idiom of Idealized Algol:

$$\mathtt{new}\ v := 0\ \mathtt{in}\ (f\ (\mathtt{inc}\ v;\ \mathtt{t}));\ !v$$

How does this violation of innocence differ from the first example above? In that example, we find that the view relation $\ulcorner\mathsf{ccc}\urcorner$ is in fact a view *function* whereas $\ulcorner\mathsf{ccn}\urcorner$ really is a relation. In particular, this implies that ccc, unlike ccn, can be "completed" to an innocent strategy.

This suggests that ccc remains fairly close to innocence. Indeed, whenever ccc has to play a move, its choice of move depends only on the P-view—just as an innocent strategy—but the decision of *whether or not to play the move* at all depends on more than the view: an innocent strategy *must* respond to a P-view in the domain of its view function, but ccc in the same situation *may* choose to "give up"—an "error" or "deadlock" rather than divergence.

However, for strategies like ccn, not only the question of whether (or not) to reply but the *choice of move* itself may depend on information lying outside of the view, this "loss of information" manifesting itself as nondeterminism in the view function. This suggests that the passage from PCF to Idealized Algol could go via an intermediate language of "PCF with refusals" corresponding to those non-innocent strategies whose view relations are nonetheless deterministic. It doesn't seem too far-fetched to imagine a connection with the "λ-calculus with multiplicities" (where refusal would signal the exhaustion of a resource).

3.5 I as a CPO-enriched category

Given an arena A, we define an order on the set of all innocent strategies for A by setting, for all σ and τ for A,

$$\sigma \leq \tau \ \text{ iff } \ \ulcorner\sigma\urcorner \subseteq \ulcorner\tau\urcorner.$$

This clearly defines a partial order with least element the "undefined" strategy $\{\varepsilon\}$. In fact, this ordering coincides with the usual subset-inclusion ordering: $\sigma \subseteq \tau$ if, and only if, $\ulcorner\sigma\urcorner \subseteq \ulcorner\tau\urcorner$. Moreover, if σ_1 and σ_2 are composable innocent strategies and $\sigma_2 \leq \sigma_2'$, then any $s \in \sigma_1 \,;\, \sigma_2$ will also be in $\sigma_1 \,;\, \sigma_2'$, from which it immediately follows that composition of strategies is monotone with respect to our ordering.

Let Δ be a directed set of innocent strategies for A. The lub of Δ defined by

$$\bigsqcup \Delta = \bigcup_{\sigma \in \Delta} \sigma$$

can easily be seen to be an innocent strategy. So the set of all innocent strategies for an arbitrary arena A is a CPO. To see that composition is continuous with respect to these lubs, suppose we have an innocent strategy $\sigma : A \Rightarrow B$ plus a directed set Δ of innocent strategies for $B \Rightarrow C$. We wish to establish that $\sigma \,;\, \bigsqcup \Delta = \bigsqcup(\sigma \,;\, \Delta)$. Since, for all $\tau \in \Delta$, we have $\sigma \,;\, \tau \leq \sigma \,;\, \bigsqcup \Delta$ we must also have $\bigsqcup(\sigma \,;\, \Delta) \leq \sigma \,;\, \bigsqcup \Delta$. For the other direction, if $s \in \sigma \,;\, \bigsqcup \Delta$ then $s \in \sigma \,;\, \tau$ for some $\tau \in \Delta$ hence $s \in \bigsqcup(\sigma \,;\, \Delta)$ and so $\sigma \,;\, \bigsqcup \Delta \leq \bigsqcup(\sigma \,;\, \Delta)$ as required.

We can concisely summarize the discussion so far by saying that **I**, our CCC, "is CPO-enriched". We can go further, showing that homsets are ω-algebraïc CPOs (have bases of compact elements).

Recall that an element x of a CPO X is ***compact*** iff, for all directed subsets Δ of X, if $x \leq \bigsqcup \Delta$ then $x \leq z$ for some $z \in \Delta$. We write $\mathcal{K}(X)$ for the compact elements of X. Our CPO X is ***algebraïc*** iff, for all $x \in X$, the set $D_x = \{k \in \mathcal{K}(X) \mid k \leq x\}$ is directed and $x = \bigsqcup D_x$. An algebraic CPO with countable $\mathcal{K}(X)$ is called ω-algebraic.

We would like to characterize those innocent strategies that are compact in a given homset. To this end, let Δ be an arbitrary directed set of innocent strategies for arena A. We begin by remarking that any innocent σ with *finite* view function $\ulcorner\sigma\urcorner$ is compact: for each $s \in \ulcorner\sigma\urcorner$, we must have $s \in \ulcorner\sigma_s\urcorner$ for some $\sigma_s \in \Delta$ and, since directed sets have upper bounds for any finite subset, we infer a $\tau \in \Delta$ such that $\sigma \leq \tau$ as required.

The converse also holds: if σ is compact then it must have finite view function. To see this, consider the set D_σ of all innocent τ with finite view function such that $\tau \leq \sigma$. This is a directed set and σ is obviously an upper bound, so $\bigsqcup D_\sigma \leq \sigma$. But, for each $s \in \sigma$, we have an innocent strategy σ_s induced by closing $\{s' \in \mathcal{L}_A \mid s' \sqsubseteq^{\mathsf{even}} s\}$ with $\mathsf{tr} \circ \mathsf{fn}$. This strategy has finite view function and hence lies in D_σ, so $s \in \bigsqcup D_\sigma$ implying that $\sigma \leq \bigsqcup D_\sigma$.

In words, *any* innocent σ can be expressed as the lub of a directed set of innocent strategies with finite view functions. This means that no strategy with infinite view function can possibly be compact and, putting this together with the above, we see that the compact elements are precisely those innocent strategies with finite view function. This proves that the CPO of innocent strategies for A is ω-algebraïc.

3.6 The O-visibility condition

The ***long* O-*view*** of $s \in \mathcal{L}_A$, noted $\lfloor s \rfloor$, is defined dually to the P-view: we follow pointers from P-moves so this time O-moves can lose pointers:

- $\lfloor \varepsilon \rfloor = \varepsilon$
- $\lfloor s \rfloor = \lfloor \mathsf{jp}(s) \rfloor \cdot s_\omega$, 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, a long O-view may contain many initial moves. A play $s \in \mathcal{L}_A$ satisfies O-***visibility*** iff $\lfloor s \rfloor \in \mathcal{L}_A$—so we lose no O-pointers in $\lfloor s \rfloor$—and satisfies simply ***visibility*** iff it satisfies P- and O-visibility.

Lemma 3.6.1 *Let $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ be* P*-vis strategies and suppose that $u \in \sigma \| \tau$ such that, for all external* O*-moves o of u, we have that $u_{\leqslant o} \restriction A, C$ satisfies* O*-visibility. Then, for a non-initial external move m in component X, $\lfloor u_{\leqslant m} \restriction Y \rfloor$ extends $\lfloor \mathsf{jp}(u_{\leqslant m}) \restriction Y \rfloor$.*

Proof By induction on the length of $u_{\leqslant m}$ (for m an external move). The base case for m a P-move consists of a single sandwich which has the required property by P-visibility in X. The base case for m an O-move consists of a sandwich plus an external O-move which trivially has the required property.

If m is an external P-move in component X, its justifier occurs in $\ulcorner u_{<m} \restriction X \urcorner$. If we trace back $\lfloor u_{<m} \restriction Y \rfloor$, it starts out just like $\ulcorner u_{<m} \restriction X \urcorner$ until this latter steps back onto an *external* O-move o in X. At this point, we can apply the inductive hypothesis to $u_{\leqslant o}$ so that $\lfloor u_{\leqslant o} \restriction Y \rfloor$ extends $\lfloor \mathsf{jp}(u_{\leqslant o}) \restriction Y \rfloor$. (In other words, following back the O-view of $u_{\leqslant o} \restriction Y$ can never jump past the justifier of o.) If we continue in this way, we eventually arrive at the last move of $\mathsf{jp}(u_{\leqslant m}) \restriction Y$ and so $\lfloor u_{\leqslant m} \restriction Y \rfloor$ extends $\lfloor \mathsf{jp}(u_{\leqslant m}) \restriction Y \rfloor$ as required.

If m is an external O-move in component X, external O-visibility implies that its justifier lies in $\lfloor u_{<m} \restriction A, C \rfloor$. As we trace back $\lfloor u_{\leqslant m} \restriction Y \rfloor$, we apply the inductive hypothesis to $u_{\leqslant p}$ for all the external P-moves p of $\lfloor u_{<m} \restriction A, C \rfloor$ (in component X) that lie between m and its justifier. This establishes, at each such point, that $\lfloor u_{\leqslant p} \restriction Y \rfloor$ extends $\lfloor \mathsf{jp}(u_{\leqslant p}) \restriction Y \rfloor$. Hence $\lfloor u_{\leqslant m} \restriction Y \rfloor$ extends $\lfloor \mathsf{jp}(u_{\leqslant m}) \restriction Y \rfloor$ as required. ■

This lemma basically says that the pointers from *external* moves in component X constrain the pointers in component Y, even though external moves in X cannot be seen from Y!

In a P-vis strategy, not all plays necessarily satisfy O-visibility. However, in order to build a category of P-vis strategies, we have no need for this extra generality: when P-vis σ and τ interact, σ appears O-vis from τ's point of view and τ appears O-vis from σ's point of view, so we can safely ignore all plays violating O-visibility:

Lemma 3.6.2 *Let $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ be* P*-vis strategies and suppose that $u \in \sigma \| \tau$ such that, for all external* O*-moves o of u, we have that $u_{\leqslant o} \restriction A, C$ satisfies* O*-visibility. Then, for any generalized* O*-move m of u in component X, we have that $u_{\leqslant m} \restriction X$ satisfies* O*-visibility.*

Proof If m is an external move, it's either initial (whence we have nothing to prove) or it points in $\lfloor u_{<m} \upharpoonright A, C \rfloor$ in component X. For each P-move of $\lfloor u_{<m} \upharpoonright A, C \rfloor$ in component Y, we apply the above lemma, thus establishing that m points in $\lfloor u_{<m} \upharpoonright X \rfloor$ as required.

Otherwise, m is a generalized O-move in X which is also a generalized P-move in Y and so, by P-visibility, points in $\ulcorner u_{<m} \upharpoonright Y \urcorner$. We apply the above lemma to each external O-move of $\ulcorner u_{<m} \upharpoonright Y \urcorner$, thus establishing that m points in $\lfloor u_{<m} \upharpoonright X \rfloor$ as required. ■

This implies, for innocent (or just P-vis) strategies $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$, that any $s \in \sigma \,;\, \tau$ that happens to satisfy O-visibility necessarily comes from an interaction between O-vis plays of σ and τ. So, if we write $\mathcal{O}(\sigma)$ for the set of all O-vis plays of P-vis σ, we can build a category [G. McCusker, private communication]:

Proposition 3.6.3 *We have a category with arenas as objects and strategies satisfying (*P*-visibility and) $\sigma = \mathcal{O}(\sigma)$ as arrows.*

Proof We define the composite $\tau \circ \sigma$ of such $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ by $\mathcal{O}(\sigma \,;\, \tau)$ and the identity arrow 1_A by $\mathcal{O}(\mathsf{id}_A)$. Clearly, $1_B \circ \sigma = \sigma = \sigma \circ 1_A$. For associativity, if we additionally have $\upsilon : C \Rightarrow D$, we calculate:

$$\begin{aligned}
\upsilon \circ (\tau \circ \sigma) &= \mathcal{O}(\mathcal{O}(\sigma \,;\, \tau) \,;\, \upsilon) \\
&= \mathcal{O}(\mathcal{O}(\sigma \,;\, \tau) \,;\, \mathcal{O}(\upsilon)) \\
&= \mathcal{O}((\sigma \,;\, \tau) \,;\, \upsilon) \\
&= \mathcal{O}(\sigma \,;\, (\tau \,;\, \upsilon)) \\
&= \mathcal{O}(\mathcal{O}(\sigma) \,;\, \mathcal{O}(\tau \,;\, \upsilon)) \\
&= \mathcal{O}(\sigma \,;\, (\upsilon \circ \tau)) \\
&= (\upsilon \circ \tau) \circ \sigma
\end{aligned}$$

using associativity of ; and lemma 3.6.2. ■

This defines a category of P- and O-vis strategies which, qua category, differs inessentially from the category of P-vis strategies. However, when we look at the concrete representations of the arrows, we see a clear difference: a strategy satisfying $\sigma = \mathcal{O}(\sigma)$ cannot interact with a non-P-vis strategy whereas a strategy satisfying only P-vis remains capable of interacting with a non-P-vis strategy. This relates to a rather deep property of game semantics:

Game semantics has defined a hierarchy of models of (idealized) programming languages. Most of these models contain the (fully abstract) game semantics for PCF and, by relaxing various constraints placed on that model, we find (fully abstract) models for a range of extensions of PCF. However, the interpretation of a PCF term *remains unchanged* in the passage from one model to another. Indeed, the semantics of a PCF term contains many superfluous plays—superfluous for the purposes of presenting the fully abstract model of PCF—but when we relax a given constraint on the PCF model, some of these previously superfluous plays may become relevant: the fewer the constraints we place on P, the more P can observe of the behaviour of O. So rather than "superfluous" we should perhaps say "invisible"; and a play "invisible" for PCF may well be "visible" in an extension of PCF.

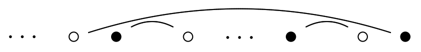
3.7 The annotated O-view

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 that 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.

The ***annotated (long)*** **O-*view*** of P- and O-vis $s \in \mathcal{L}_A$, written $\lfloor \widetilde{s} \rfloor$, is defined inductively:

- $\lfloor \widetilde{\varepsilon} \rfloor = \varepsilon$
- $\lfloor \widetilde{s} \rfloor = \lfloor \widetilde{\mathsf{ip}(s)} \rfloor s_\omega$, if s_ω is an O-move
- $\lfloor \widetilde{s} \rfloor = \mathsf{match}^\star(s, \lfloor \widetilde{\mathsf{jp}(s)} \rfloor)$, if s_ω is an P-move

In words, the annotated (long) O-view traces back the (long) O-view but, instead of *excising* all moves underneath each P-to-O pointer, it retains that suffix of the current P-*view* "enclosed" by the pointer:



The assumption of O-visibility for s implies that $\lfloor \widetilde{s} \rfloor \in \mathcal{L}_A$. So $\lfloor \widetilde{s} \rfloor$ in fact consists of an interleaving of P-views of A. If $s \in \sigma$ for an innocent σ then $\lfloor \widetilde{s} \rfloor$ exposes what input P-view σ requires in order to produce $\lfloor s \rfloor$ as output O-view.

We previously defined $\mathsf{match}(u,v)$ to formalize what we mean by "extend v with the tail of generalized P-moves of u". In order to define $\mathsf{match}^\star(u,v)$, consider $u \in \sigma \parallel \tau$ (for P-vis $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$) and $v \in \mathcal{I}(A,B,C)$ such that $\ulcorner \mathsf{jp}(u) \urcorner = \ulcorner v \urcorner$. We define $\mathsf{match}^\star(u,v)$ to be the extension of v with the last "chunk" of the P-view of u, *i.e.* extend v with the suffix of $\ulcorner u \urcorner$ lying underneath the pointer from u_ω.

The P-view of a legal interaction picks out a subsequence of sandwiches according to the following scheme: track back through the current sandwich until we reach an external O-move o; then follow o's pointer and recursively apply, until we reach an initial O-move. The (long) O-view [defined below] of a legal interaction u picks out its subsequence of sandwiches differently: it applies the above procedure (for the P-view) but only until we reach that external O-move o that justifies the last move of u—assumed to be an external P-move; then move to the external P-move immediately preceding o and recursively apply.

Suppose that our $u \in \sigma \parallel \tau$ additionally satisfies external O-visibility: for all external O-moves o of u, we have $u_{\leqslant o} \upharpoonright A, C \in \mathcal{L}_{A \Rightarrow C}$. We define the ***(long)*** **O-*view*** of u, written $\lfloor u \rfloor$, inductively:

- $\lfloor \varepsilon \rfloor = \varepsilon$
- $\lfloor u \rfloor = \lfloor \mathsf{ip}(u) \rfloor u_\omega$, if u_ω is an external O-move
- $\lfloor u \rfloor = \mathsf{match}^\star(u, \lfloor \mathsf{jp}(u) \rfloor)$, if u_ω is an external P-move

Note that $\lfloor u \rfloor$ is an interleaving of P-views of legal interactions just as $\lfloor \widetilde{s} \rfloor$ is an interleaving of P-views of legal plays. External O-visibility guarantees that $\lfloor u \rfloor \in \mathcal{I}(A, B, C)$. Moreover, $\ulcorner u \urcorner$ is a subsequence of $\lfloor u \rfloor$ and $\lfloor u \rfloor$ clearly witnesses $\lfloor u \upharpoonright \widetilde{A, C} \rfloor$.

4 Game semantics of PCF

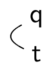
The game semantics of PCF is always presented in a "categorical style" so that a typed term-in-context $x_1 : T_1, \ldots, x_n : T_n \vdash M : T$ is modelled as an arrow $[\![M]\!] : [\![T_1]\!] \times \cdots \times [\![T_n]\!] \to [\![T]\!]$ in our CCC. (We cannot model the above term as a strategy for $[\![T]\!]$ parametrized by an environment mapping the free variables x_i to strategies in $[\![T_i]\!]$ since $\mathbf{I}$ is not well-pointed: we have no way of building the interpretation of a λ-abstraction.) The CCC structure of $\mathbf{I}$ guarantees us a model of the simply typed λ-calculus over our choice of base types: a variable is modelled by the appropriate projection from its context, λ-abstraction by currying and application by the eval map. In what follows, we use base types `com`, `bool` and `nat` modelled by the flat arenas **com**, **bool** and **nat**.

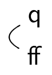
4.1 Arithmetic strategies

The base type constants of PCF, such as

$$\frac{}{\Gamma \vdash \mathtt{tt} : \mathtt{bool}} \qquad \frac{}{\Gamma \vdash \mathtt{ff} : \mathtt{bool}}$$

for the booleans, are modelled by strategies that never play in Γ. Instead, they respond immediately to the initial Question with the appropriate Answer:



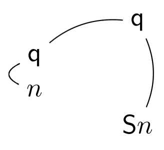


PCF typically contains a few constructs for manipulating `nat`s and `bool`s. A fairly minimal choice consists of "successor", "predecessor" and "test for zero":

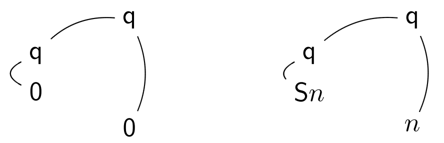
$$\frac{\Gamma \vdash M : \mathtt{nat}}{\Gamma \vdash (\mathtt{succ}\ M) : \mathtt{nat}} \qquad \frac{\Gamma \vdash M : \mathtt{nat}}{\Gamma \vdash (\mathtt{pred}\ M) : \mathtt{nat}} \qquad \frac{\Gamma \vdash M : \mathtt{nat}}{\Gamma \vdash (\mathtt{zero?}\ M) : \mathtt{bool}}$$

In all three of these rules, the term M is modelled as a strategy for $[\![\Gamma]\!] \Rightarrow \mathbf{nat}$. We wish to define $[\![(\mathtt{succ}\ M)]\!]$, $[\![(\mathtt{pred}\ M)]\!]$ and $[\![(\mathtt{zero?}\ M)]\!]$ as compositions of $[\![M]\!]$ with appropriate strategies on $\mathbf{nat} \Rightarrow \mathbf{nat}$ (for `succ` and `pred`) or $\mathbf{nat} \Rightarrow \mathbf{bool}$ (for `zero?`).

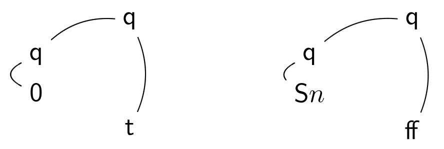
The strategy $\mathsf{succ} : \mathbf{nat} \Rightarrow \mathbf{nat}$ is the innocent strategy generated by the view function containing all plays of the form



where n is not a move but a variable ranging over moves and $\mathsf{S}n$ stands for the move following n; *e.g.* if n stands for the move $\mathsf{2}$ then $\mathsf{S}n$ stands for the move $\mathsf{3}$. Similarly, the strategy $\mathsf{pred} : \mathbf{nat} \Rightarrow \mathbf{nat}$ is induced by the view function containing



and $\mathsf{zero?} : \mathbf{nat} \Rightarrow \mathbf{bool}$ is induced by



so we can define:

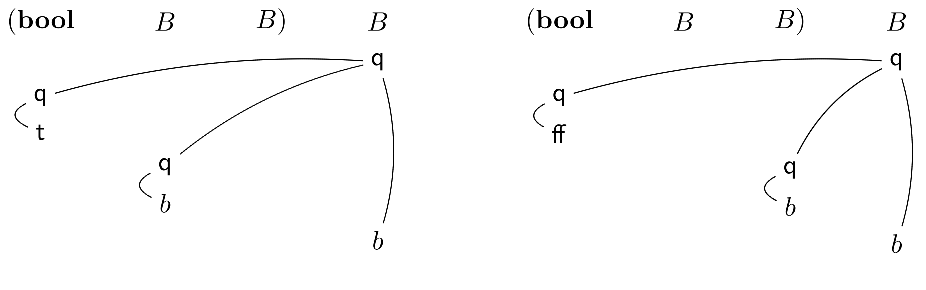
- $[\![(\texttt{succ}\ M)]\!] = [\![M]\!]\ ;\mathsf{succ}$
- $[\![(\texttt{pred}\ M)]\!] = [\![M]\!]\ ;\mathsf{pred}$
- $[\![(\texttt{zero?}\ M)]\!] = [\![M]\!]\ ;\mathsf{zero?}$

4.2 Control-flow strategies

We next consider the `if` construct of PCF which, without loss of generality, we can restrict to choosing between two terms of base type:

$$\frac{\Gamma \vdash M : \texttt{bool} \qquad \Gamma \vdash N : B \qquad \Gamma \vdash L : B}{(\texttt{if}\ M\ N\ \texttt{else}\ L) : B}$$

In order to model this, we need an innocent strategy $\mathsf{if} : \mathbf{bool} \times B \times B \Rightarrow B$ that interrogates its boolean in order to decide whether to behave as first or second projection:



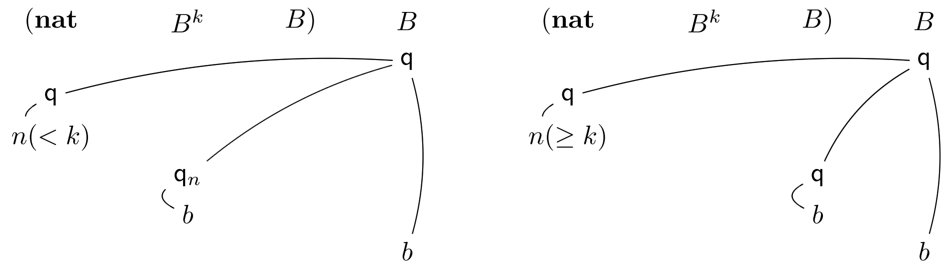
Given $[\![M]\!] : [\![\Gamma]\!] \Rightarrow \mathbf{bool}$, $[\![N]\!] : [\![\Gamma]\!] \Rightarrow [\![B]\!]$ and $[\![L]\!] : [\![\Gamma]\!] \Rightarrow [\![B]\!]$, we thus define

- $[\![(\texttt{if}\ M\ N\ \texttt{else}\ L)]\!] = \langle [\![M]\!], [\![N]\!], [\![L]\!] \rangle\ ;\mathsf{if}$

In a similar way, we define the semantics of the `case` construct of PCF

$$\frac{\Gamma \vdash M : \mathtt{nat} \quad \Gamma \vdash \vec{N} : B \quad \Gamma \vdash L : B}{(\mathtt{case}\ M\ \vec{N}\ \mathtt{else}\ L) : B}$$

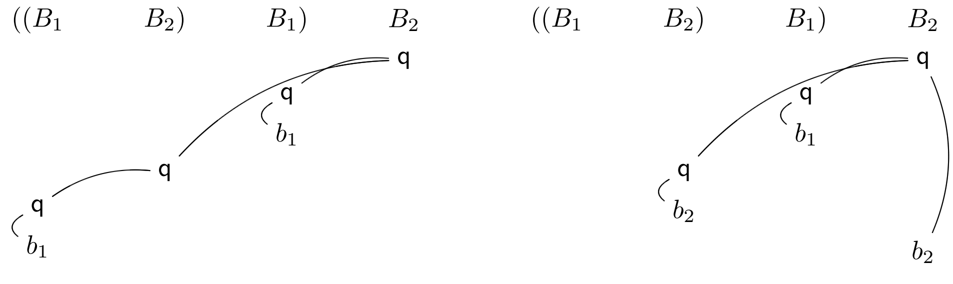
by composition with $\mathsf{case} : \mathbf{nat} \times [\![\vec{B}]\!] \times [\![B]\!] \Rightarrow [\![B]\!]$, the generalization of `if` to handle branching on the first k natural numbers (the terms $\vec{N}$) plus a default case (the term L) if the natural number returns k or more:



The `if` and `case` constructs both evaluate one distinguished term (of type `bool` or `nat` as appropriate) in order to choose, from a given collection of terms, which to evaluate next. Note, however, that the value used to make that choice does not get passed to the chosen continuation. In contrast, the `let` construct of PCF only has one possible continuation after evaluation of its distinguished term; but the value obtained is passed to the continuation:

$$\frac{\Gamma \vdash M : B_1 \quad \Gamma, x : B_1 \vdash N : B_2}{\Gamma \vdash (\mathtt{let}\ x = M\ \mathtt{in}\ N) : B_2}$$

In order to define the semantics of `let`, we first define, for base types B_1 and B_2, the "delayed copycat" $\mu_{B_1 B_2} : [\![B_1 \Rightarrow B_2]\!] \Rightarrow [\![B_1 \Rightarrow B_2]\!]$ as:



If we have a strategy $\sigma : [\![\Gamma]\!] \Rightarrow [\![B_1 \Rightarrow B_2]\!]$, it may interrogate its argument B_1 multiple times. If we compose σ with $\mu_{B_1 B_2}$, this has the effect of "linearizing" σ in B_1: the delayed copycat first evaluates the argument in B_1 "once and for all"; each time σ subsequently requests that argument, $\mu_{B_1 B_2}$ simply repeats the value already computed.

Given $[\![M]\!] : [\![\Gamma]\!] \Rightarrow [\![B_1]\!]$ and $[\![N]\!] : [\![\Gamma, B_1]\!] \Rightarrow [\![B_2]\!]$, we define

- $[\![(\mathtt{let}\ x = M\ \mathtt{in}\ N)]\!] = \langle [\![M]\!], (\Lambda_{B_1}[\![N]\!]) \mathbin{;} \mu_{B_1 B_2} \rangle \mathbin{;} \mathsf{eval}_{\mathsf{B_1 B_2}}$

In words, we linearize N in B_1 and then apply it, in the usual way, to M.

4.3 Recursively defined PCF terms

To define recursive processes, PCF provides, at all types T, a special form $\texttt{fixpt}_T$ taking an input of type $T \to T$ and returning its least fixed point:

$$\frac{\Gamma \vdash M : T \to T}{\Gamma \vdash (\texttt{fixpt}\ M) : T}$$

Given $[\![M]\!] : [\![\Gamma]\!] \to [\![T \to T]\!]$, we construct an ω-chain of strategies in $[\![\Gamma]\!] \to [\![T]\!]$ with the following inductive definition

- $f_0 = \bot_{\Gamma \to T}$
- $f_{n+1} = \langle \mathsf{id}_\Gamma, f_n \rangle\ ;\ \Lambda^{-1}([\![M]\!])$

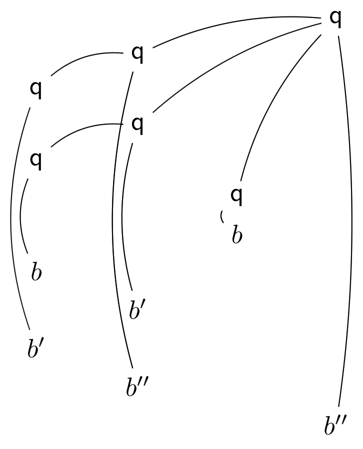
and then set $[\![\Gamma \vdash (\texttt{fixpt}\ M) : T]\!] = \bigsqcup_{i \in \mathbf{N}} f_i$. Essentially, this definition just computes the limit of the ω-chain $\bot \leq f(\bot) \leq f(f(\bot)) \leq \cdots$

4.4 Example PCF processes

Nesting Suppose we wish to compute $f(f(x))$ for some function f and initial input x. The obvious PCF term

$$\lambda f x\ (f\ (f\ x)) : (\texttt{bool} \to \texttt{bool}) \to \texttt{bool} \to \texttt{bool}$$

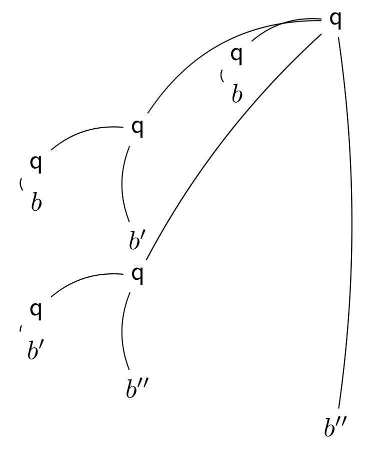
doesn't evaluate x until the control stack, representing the nested applications of f, has been built. This process can be seen in the structure of the strategy denoting this term, in particular the way that the pointers associating Answers with Questions get longer and longer:



Sequencing An alternative to pushing control stack instead evaluates x straight away, applies f to that and then reapplies f to that. While avoiding the need for a stack, this requires intermediate names for each "step" in the sequence:

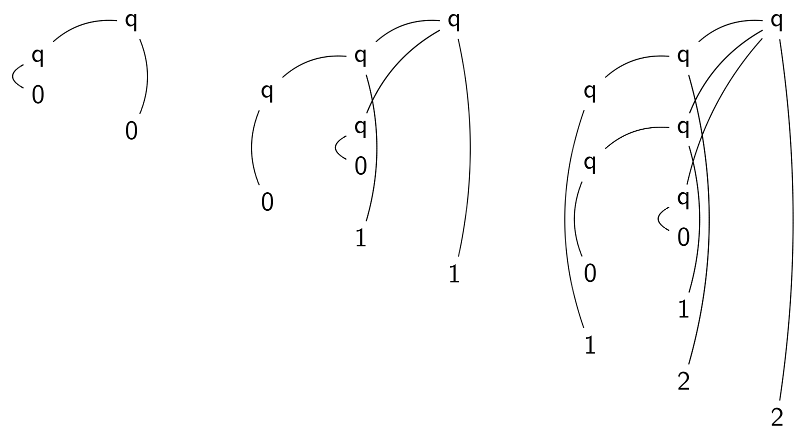
$$\lambda f x\ (\texttt{let}\ z = x\ \texttt{in}\ (\texttt{let}\ z = (f)z\ \texttt{in}\ (f)z)).$$

This term describes an inside-out version of the previous strategy:



Base type recursion In PCF, we never need recursion to define processes of base type. However, we can extend of PCF with an *erratic choice* operator M `or` N that chooses nondeterministically between M and N; we exploit this to give a first example of the use of `fixpt` that builds a base type process:

Typical interactions between $\texttt{fixpt}_{\texttt{nat}}$ and λx(0 `or` (`succ` x)) look like



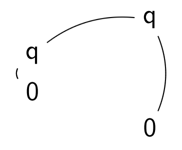
so that (`fixpt` λx (0 `or` (`succ` x))) denotes that process of type `nat` that *may* return *any* natural number. However, this process doesn't terminate if `or`, *unknown to us*, always selects its right-hand input. This matches the pessimistic assessment of may&must testing that this program *may* diverge.

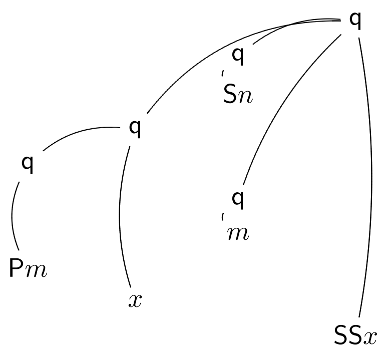
On the other hand, if `or` uses a fair coin to decide, at run-time, which of its two arguments to interrogate, the probability of divergence in the above example is nil: as far as probabilistic testing is concerned, our program always converges. In general, "qualitative" (`or` with may&must testing, *i.e.* an unfair coin with probabilistic testing) and "quantitative" (a fair coin with probabilistic testing) nondeterminism describe completely different situations: the first can be seen as "interacting with a *deterministic but unknown* Opponent" whereas, in the second case, Opponent *really is* nondeterministic: Opponent's choices are resolved at run-time and *nobody* knows in advance what's going to happen.

The double function A play in the strategy for

$$\texttt{dbl} = \lambda f x\ (\texttt{case}\ x\ 0\ \texttt{else}\ (\texttt{succ}\ (\texttt{succ}\ (f\ (\texttt{pred}\ x)))))$$

follows one of these two templates:





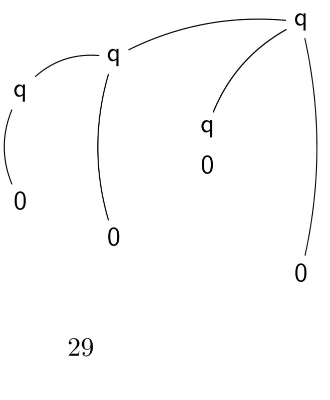
In words, the process begins by evaluating the head variable x: if this gives 0, it returns result 0; otherwise it applies f to (`pred` x), finally returning that value plus two.

The application of f to (`pred` x) results in *reevaluation* of x, even though we just evaluated it, even though the value just computed for x lies in the view! (Compare the "shape" of the above strategy with that of cft defined earlier.)

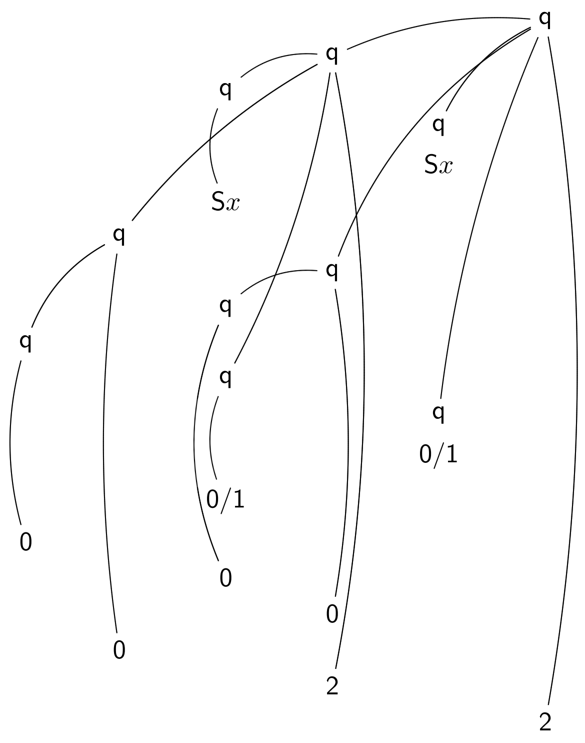
This makes the lack of any *uniformity* assumption highly relevant: the second time x gets evaluated, it may in principle return a different result to the first time. It turns out that this cannot happen in any PCF (indeed μPCF) context but such behaviour expresses the very essence of PCF extended with references (if only at base types) or nondeterminism of any kind.

Nonuniformity raises another question: the second value of x (the occurrence m) could be 0—but what value do we choose for P0? Typically it's defined to be 0 but one could equally well argue for "undefined" or "error".

In any case, let's calculate the semantics of (`fixpt dbl`) by supplying $[\![\texttt{dbl}]\!]$ to `fixpt`. We build the set of plays interpreting (`fixpt dbl`) inductively, starting with input 0. In this first interaction, we make no recursive calls (hence T_2 never gets used) and merely copycat between T_0 and (the first copy of) T_1:

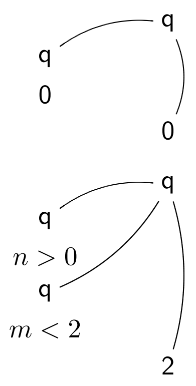


The interaction becomes more interesting if Opponent instead supplies a non-zero input at the first time of asking and either `0` or `1` the second time:

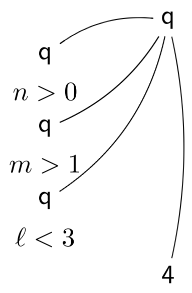


The initial non-zero input provokes a recursive call which opens a second copy of `dbl` whose (eventual) input is `pred` of the second input. In the above example, this means that the inner copy of `dbl` gets input `0` which "bottoms out" the recursion, yielding `2`. Note that, even if the first input were `2`, `3` or `2004`, if the second input is less than 2, the recursion bottoms out[*]—and the final result doesn't depend on the *inputs received* but only on the *number of recursive calls* made. In general, if the nth input is strictly less than n, this causes the recursion to bottom out and apply `succ` $2(n-1)$ times to `0`.

In general, the game semantics of (`fixpt dbl`) looks like:



[*]This depends on the choice that (`pred` 0) evaluates to `0`.



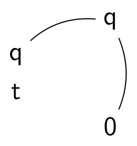
etc. Recall that, in the absence of nonuniformity, *every* input will be the same. This means that we supply a "constant stream" such as $1, 1$ or $2, 2, 2$ or $3, 3, 3, 3$ with the effect that input stream $n, n, ..., n$ (of length $n+1$) sets up n recursive calls, the $n{+}1$th input causes the recursion to stop and the result $2((n{+}1){-}1) = 2n$ is eventually returned.

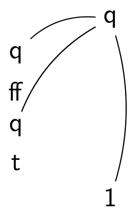
In the presence of nonuniformity, strange things can happen. For example, the application (`fixpt dbl`)(`0 or 2`) can evaluate to `0`, to `4` but also to `2` (if the first input were `2` but the second `0`). Similar possibilities exist in an Algol-like language with references at base type.

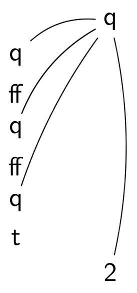
cft revisited Recall the "count for true" term defined earlier:

$$f : \texttt{bool} \to \texttt{nat}, x : \texttt{bool} \vdash (\texttt{if}\ x\ \texttt{0}\ \texttt{else}\ (\texttt{succ}\ (f\ x))) : \texttt{nat}$$

λ-abstracting x and f and applying `fixpt` yields a process that repeatedly asks x for input until such time as x receives `tt`, whence the process terminates returning the number of `ff`s received before the `tt`:



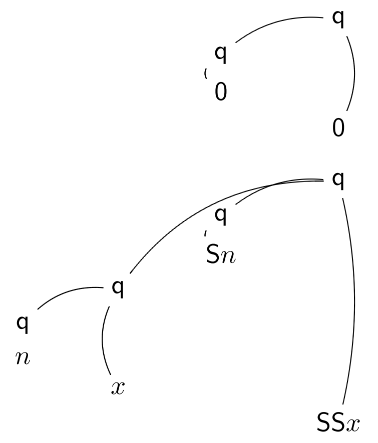




etc. This process exhibits subtle behaviour in the presence of nonuniformity (in the form of nondeterminism or state). However, in PCF or μPCF, x always receives the same input. Under such circumstances, we may as well evaluate x once and for all (*i.e.* use the μ-transformed version of cft) which makes it clear that, as far as PCF/μPCF are concerned, cft behaves the same as

$$\lambda x\ (\texttt{if}\ x\ 0\ \texttt{else}\ \Omega).$$

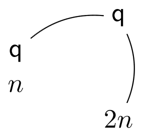
double revisited Unlike "count for true" (which remains rather uninteresting in the absence of nonuniformity) the "double" function would, if anything, make more sense once μ-transformed:



Notice in particular that the application of f to (`pred` x) in the `else` clause no longer reevaluates x (cf. the example of μ-transforming cft) and so, by taking the fixed point of

$$\texttt{dbl2} = \lambda f x\ (\texttt{let}\ z = x\ \texttt{in}\ (\texttt{case}\ z\ 0\ \texttt{else}\ (\texttt{succ}\ (\texttt{succ}\ (f\ (\texttt{pred}\ z))))))$$

(*i.e.* the term corresponding to the μ-transformed $[\![\texttt{dbl}]\!]$), we get a much more intuitive semantics of the double function where the input is evaluated exactly once:



This strategy *can* therefore be defined in PCF (if extended with `let`) by a perfectly finitary term, despite our earlier concerns that it would have required an infinite `case` form. It follows from "uniformity" that (`fixpt dbl2`) cannot be distinguished from (`fixpt dbl`) by *any* context of PCF or μPCF—but, in the presence of any kind of nonuniformity, these terms can easily be distinguished: *e.g.* (`fixpt dbl2`)(`0 or 2`) *cannot* evaluate to `2`.

Mutual recursion The program below (in vanilla R5RS Scheme) computes b^n, exploiting the ability of `letrec` to define `od?` and `ev?` in a *mutually recursive* fashion.

```
(define (fastexp b n)
  (letrec
      ((od? (lambda (x)
               (if (= x 0) #f (ev? (- x 1)))))
       (ev? (lambda (x)
               (if (= x 0) #t (od? (- x 1)))))
       (sqr (lambda (x) (* x x)))
       (exp (lambda (n)
               (if (= n 0) 1 (if (ev? n)
                                  (sqr (exp (/ n 2)))
                                  (* b
                                     (exp (- n 1))))))))
    (exp n)))
```

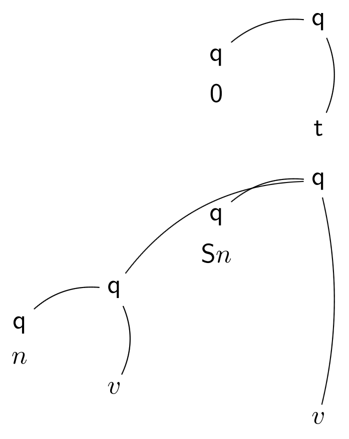
How can such a program be written in PCF? Or, more precisely, how can `od?` and `ev?` be expressed in terms of `fixpt` alone? We begin, by analogy with the use of `fixpt` in defining the function `double`, by introducing two free variables, f and g, standing for `od?` and `ev?` respectively:

$$\begin{aligned} F\ f\ g &= \lambda z\ (\texttt{let}\ x = z\ \texttt{in}\ (\texttt{if}\ (\texttt{zero?}\ x)\ \texttt{ff}\ \texttt{else}\ (g\ (\texttt{pred}\ x)))) \\ G\ f\ g &= \lambda z\ (\texttt{let}\ x = z\ \texttt{in}\ (\texttt{if}\ (\texttt{zero?}\ x)\ \texttt{t}\ \texttt{else}\ (f\ (\texttt{pred}\ x)))) \end{aligned}$$

The use of `let` avoids reevaluation of x in the `else` continuation, cf. `double`. We next eliminate the two free variables, f and g, by taking appropriate `fixpt`s. We can start by eliminating f (as in F') or g (as in G'):

$$\begin{aligned} F'\ g &= (\texttt{fixpt}\ \lambda f\ (F\ f\ g)) \\ G'\ f &= (\texttt{fixpt}\ \lambda g\ (G\ f\ g)) \end{aligned}$$

We model these as strategies on $(\texttt{nat} \to \texttt{bool}) \to (\texttt{nat} \to \texttt{bool})$. For example, a play in $[\![G']\!]$ follows one of these two templates:

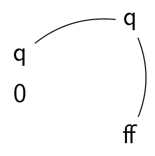


Note that, since g doesn't occur in G, the `fixpt` actually does nothing except eliminate g.

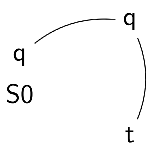
In the second step, we eliminate the remaining free variable:

$$\begin{aligned} \mathcal{F} &= (\texttt{fixpt}\ \lambda f\ (F\ f\ (G'\ f))) \\ \mathcal{G} &= (\texttt{fixpt}\ \lambda g\ (G\ (F'\ g)\ g)) \end{aligned}$$

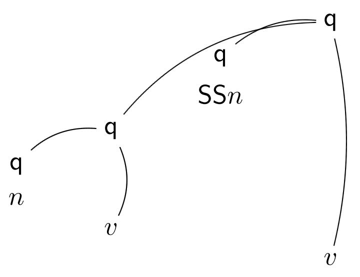
The subterm $(F\ f\ (G'\ f))$ (resp. $(G\ (F'\ g)\ g)$) introduces the second base case in $\mathcal{F}$ (resp. $\mathcal{G}$):



(resp. t)



(resp. ff)



The terms $\mathcal{F}$ and $\mathcal{G}$ can then easily be seen to define the functions `od?` and `ev?` as desired. The mutually *tail recursive* nature of `od?` and `ev?` facilitates the process of "eliminating a free variable with `fixpt`" since a recursive call to one of the functions can easily be converted to a call to the other, *e.g.* a call to (`ev`? 3) gets replaced by (`od`? 2). More generally, any collection of mutually tail recursive functions describes a finite state automaton; each function corresponds to a state, each tail call to a state transition.

A slightly more subtle example of mutual recursion, which doesn't fit the above pattern of mutual tail recursion, can be found by rephrasing the Lucid-style dataflow definition

```
fib = 0 fby aux
aux = 1 fby fib+aux
```

(`fby` meaning "followed by") of the Fibonacci series as two mutually recursive functions

```
(letrec
    ((fib (lambda (n)
            (if (= n 0) 0 (aux (- n 1)))))
     (aux (lambda (n)
            (if (= n 0) 1 (+ (fib (- n 1))
                             (aux (- n 1)))))))
  ...)
```

where `fib` remains tail recursive but `aux`, in referring to itself and to `fib`, sets up a more complex interdependency.

As for odd/even, we have two terms, each with two free variables, and eliminate mutual recursion in two steps. However, unlike odd/even, the two fixed points $\mathcal{F}$ and $\mathcal{G}$ describe qualitatively different processes: eliminating `fib` from `aux` yields the "standard" tree recursive definition of Fibonacci (albeit starting from 1 and 1) whereas eliminating `aux` from `fib` engenders a rather subtle process (since `aux` refers to itself) that computes Fibonacci starting from 0 and 1.

Accumulating paramaters Consider a simple programming problem such as adding two natural numbers (using only "test for zero" plus `succ` and `pred`). The natural *recursive* solution

```
(define (add1 x y)
  (let ((z x))
    (if (zero? z) y (succ (add1 (pred z) y)))))
```

rather redundantly passes its second argument `y` all the way through the winding up of the recursion, only to repeatedly increment it during the unwinding phase. Unfortunately, under a call-by-name evaluation discipline, this second argument is accessed for the *first time* by the innermost recursive copy of `add1`; this provokes a "cascade" of Questions, terminated by a request for the second argument at the "top level", which serves only to pass the "top level" input all the way back to the innermost copy [that triggered the original cascade]. Only then can the control stack unwind, calculating result `5` for `(add1 2 3)` in our particular example.

At first sight, the following slight variation

```
(define (add2 x y)
  (let ((z x))
    (if (zero? z) y (add2 (pred z) (succ y)))))
```

seems to avoid the need for a control stack—the *current continuation* always being trivial. However, under a call-by-name evaluation strategy, even this (syntactically) tail recursive term only evaluates its second argument when it hits the base case (*i.e.* when `x` becomes zero). In effect, once the base case is reached, we know *how many times* we need to apply `succ` but we don't know what *value* this stack of `succ`s should be applied to. As such, the process engendered by `add2` cannot reasonably be considered iterative, despite its tail recursive appearance. On the other hand, the tail recursive nature of `add2` does allow a certain optimization: once the innermost copy receives the value `5` for its second argument, this can be returned directly to the top level.

A properly *iterative* solution requires neither a control stack nor an implicit stack of `succ`s. This avoids the Question cascades of `add1` and `add2` by *accumulating* the second argument: at the moment that the innermost recursive copy of `add3` asks for its argument `y`, it has already calculated the desired result!

```
(define (add3 x y)
  (let ((z1 x) (z2 y))
    (if (zero? z1) z2 (add3 (pred z1) (succ z2)))))
```

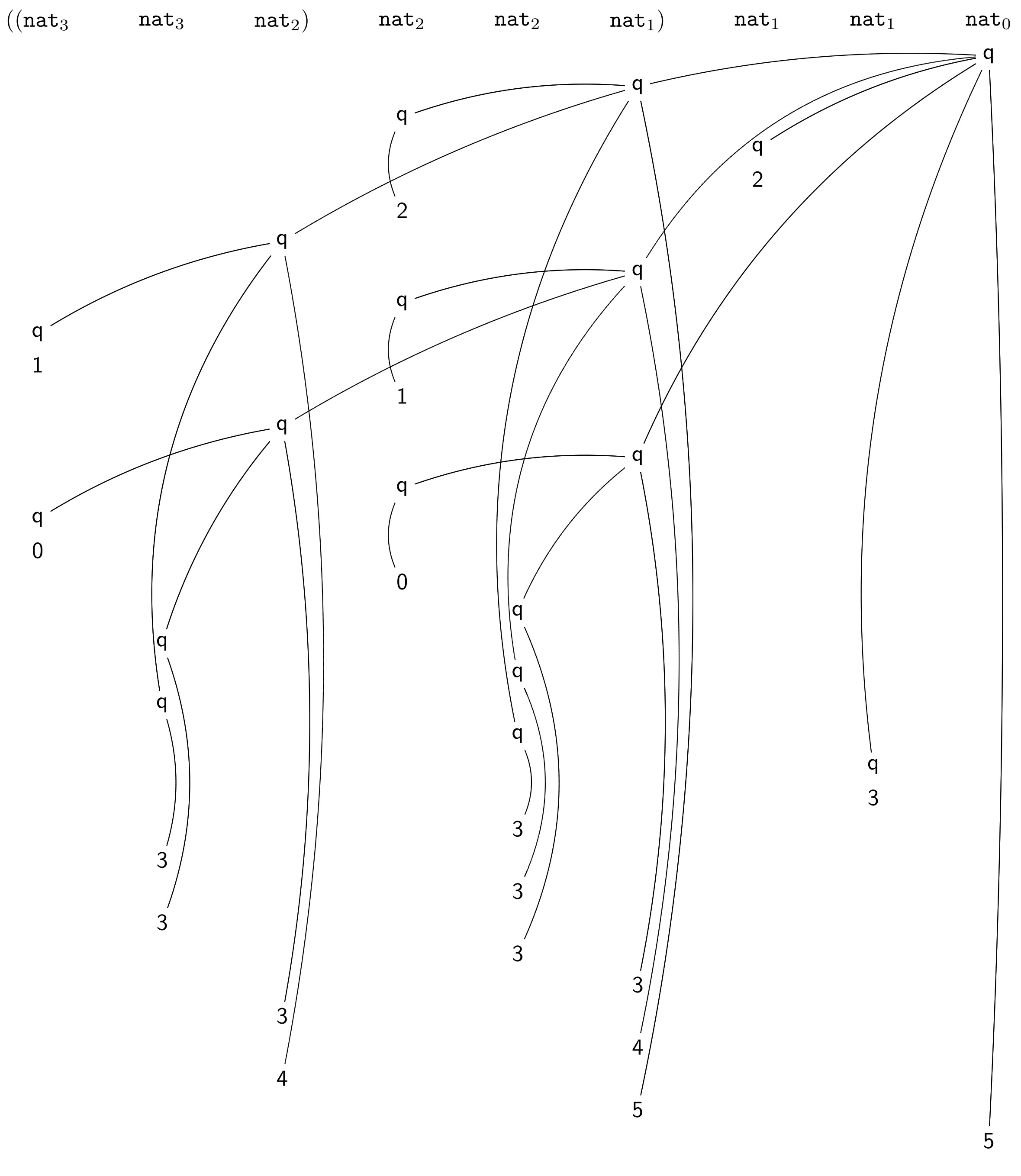


Figure 1: a trace of `fixpt` vs. $\lambda f y \mu x (\texttt{case } x\ y \texttt{ else } (\texttt{succ } (f)(\texttt{pred } x) y))$

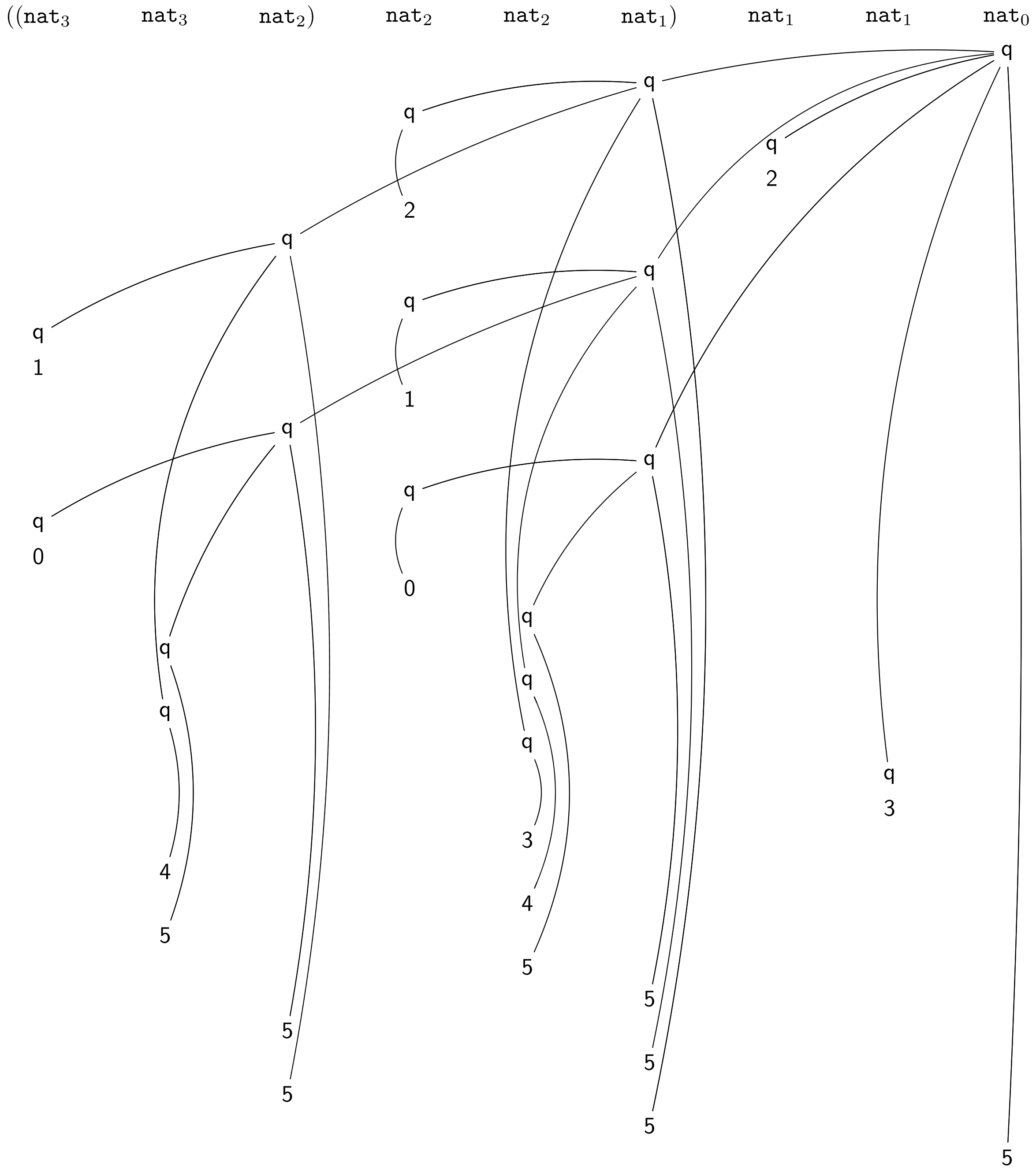


Figure 2: a trace of `fixpt` vs. $\lambda f y \mu x(\texttt{case}\ x\ y\ \texttt{else}\ (f)(\texttt{pred}\ x)(\texttt{succ}\ y))$

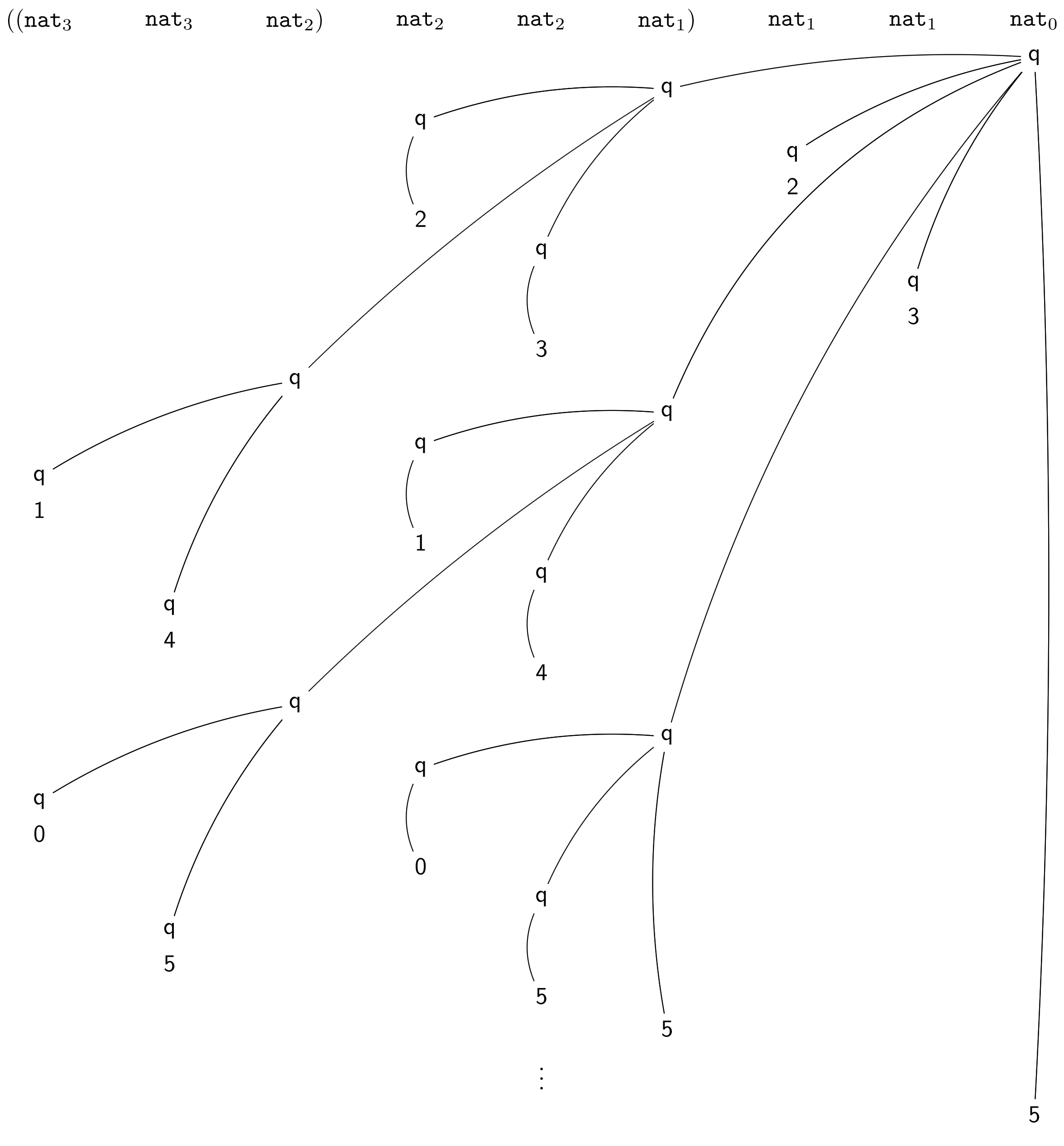
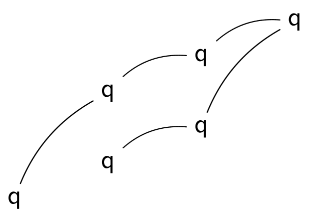


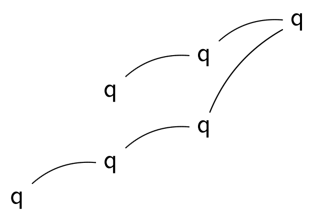
Figure 3: a trace of `fixpt` vs. $\lambda f \mu x y($`case` x y `else` $(f)($`pred` $x)($`succ` $y))$

4.5 The importance of justification pointers

In the (relatively) simple examples seen so far, the use of pointers may seem minimal, perhaps even unnecessary. In general, however, they play an essential disambiguating role. For example, compare the infamous λ-term of Kierstead $\mathcal{K}_x = \lambda F(F)\lambda x(F)\lambda y(x)$ with its close cousin $\mathcal{K}_y = \lambda F(F)\lambda x(F)\lambda y(y)$, viewed as strategies on $((\bot \to \bot) \to \bot) \to \bot$:



vs.



The only difference between these two strategies lies in the target of the final pointer: does it point to the first or second copy of the input? And yet these terms can exhibit quite different behaviours. For example, it turns out that $\mathcal{K}_x$ and $\mathcal{K}_y$ are contextually equivalent in unary PCF but are distinguishable in just about any conceivable extension of this language. A contextual separation becomes possible in unary μPCF with

$$[]\ \lambda f \mu\alpha(\mathtt{conv}\ (f)[\alpha]\mathtt{t}\ \Omega)$$

or in unary PCF with an error [boolean PCF where all `else` clauses are just `ff` so we think of `ff` as an error value that can only be propogated] with

$$[]\ \lambda f(\mathtt{if}\ (f)\mathtt{t}\ (\mathtt{if}\ (f)\mathtt{ff}\ \Omega\ \mathtt{else}\ \mathtt{ff})\ \mathtt{else}\ \mathtt{ff})$$

and in boolean PCF, they can be *strictly* separated, *i.e.* the following context sends $\mathcal{K}_x$ to `t` and $\mathcal{K}_y$ to `ff`:

$$[]\ \lambda f(\mathtt{if}\ (f)\mathtt{t}\ (\mathtt{if}\ (f)\mathtt{ff}\ \mathtt{ff}\ \mathtt{else}\ \mathtt{t})\ \mathtt{else}\ (\mathtt{if}\ (f)\mathtt{ff}\ \mathtt{t}\ \mathtt{else}\ \mathtt{t})).$$

The fact that $\mathcal{K}_x$ and $\mathcal{K}_y$ cannot be distinguished in unary PCF should not be taken to mean that pointers play no role in that language. To see this, consider the difference between

$$X_1X_2 = \lambda F(F)\lambda x_1x_2(F)\lambda y_1y_2(\mathtt{conv}\ x_1\ x_2)$$

and

$$X_1Y_2 = \lambda F(F)\lambda x_1x_2(F)\lambda y_1y_2(\mathtt{conv}\ x_1\ y_2)$$

when interacting with a nested context such as

$$[]\ \lambda f(f)((f)\mathtt{t}\Omega)((f)\Omega\mathtt{t}).$$

The game semantics of X_1X_2 and X_1Y_2 differ only in the pointers determining the choice between x_2 and y_2 (in the bodies of the `seq` forms) and yet placing X_1X_2 in this context results in divergence while X_1Y_2 happily converges. This happens because X_1Y_2 only ever looks at the left input of the inner-left copy of f and at the right input of the inner-right copy of f (both of which are defined) whereas X_1X_2 performs a more systematic “search” of its context, in particular looking at the *right* input of the inner-*left* copy of f.

5 Definability and full abstraction for PCF

5.1 Definability

Most "full abstraction" results use the same basic idea: establish that all *finite* elements of the model are definable in the language in question; then show that any two *semantically distinct* terms M and N can be separated by some *finite* context C; conclude that the term defining C distinguishes M from N. In this section, we present such finite definability results, firstly for all innocent strategies, then for "well bracketed" innocent strategies (no non-local control flow) and finally for "rigid" innocent strategies (no local control flow).

5.1.1 Classical Böhm trees

Below we present Herbelin's [2] convenient language, based on $\lambda\mu$-calculus and known as CBT, for defining compact innocent strategies.

$$\begin{array}{rcl} F & ::= & \lambda\vec{x}\mu\alpha(E) \\ E & ::= & \Omega \mid [\alpha]\mathtt{t} \mid [\alpha]\mathtt{ff} \mid \mathtt{if}\ (x)\vec{F}\ E\ \mathtt{else}\ E \end{array}$$

For the sake of a simple syntax, we consider the finitary version of the language with base type `bool` plus constants `t` and `ff` and the `if-then-else` form. This can easily be extended to base type `nat` plus constants and a `case` form. Terms E, called *executables*, we type as sequents $\Gamma\ ;\ \Delta \vdash E$ and terms F we type as $\Gamma\ ;\ \Delta \vdash F : T$ where Γ is a list of variable-simple type pairs, Δ is a list of name-base type pairs and T is a simple type. We assume a divergent executable Ω, available regardless of the syntactic context.

CBT

$$\frac{}{\Gamma\ ;\ \Delta\ \vdash\ \Omega} \qquad \frac{}{\Gamma\ ;\ \Delta, \alpha : \mathtt{bool}\ \vdash\ [\alpha]\mathtt{t}} \qquad \frac{}{\Gamma\ ;\ \Delta, \alpha : \mathtt{bool}\ \vdash\ [\alpha]\mathtt{ff}}$$

$$\frac{\Gamma\ ;\ \Delta\ \vdash\ F_1 : T_1\ \cdots\ \Gamma\ ;\ \Delta\ \vdash\ F_n : T_n \qquad \Gamma\ ;\ \Delta\ \vdash\ E_0 \qquad \Gamma\ ;\ \Delta\ \vdash\ E_1}{\Gamma, x : T_1 \to \cdots \to T_n \to \mathtt{bool}\ ;\ \Delta\ \vdash\ (\mathtt{if}\ (x)\vec{F}\ E_0\ \mathtt{else}\ E_1)}$$

$$\frac{\Gamma, x_1 : T_1 \ldots x_n : T_n\ ;\ \Delta, \alpha : \mathtt{bool}\ \vdash\ E}{\Gamma\ ;\ \Delta\ \vdash\ \lambda\vec{x}\mu\alpha(E) : T_1 \to \cdots \to T_n \to \mathtt{bool}}$$

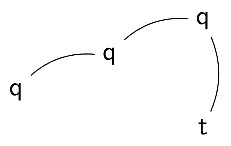
Theorem 5.1.1 *Every compact innocent strategy σ on an arena A interpreting a simple type (over* `bool`*) is definable in CBT.*

The theorem is easily proved by induction on the size of the view function of σ; see [1, 2] for more details. The two base cases—no response or immediate answer to the initial question of A—correspond to basic executables. Otherwise, σ splits into two substrategies, $\mathtt{args}(\sigma)$ and $\mathtt{cnts}(\sigma)$ standing for "arguments to σ" and "continuations from σ", both of which have size strictly smaller than σ (and so are CBT-definable by inductive hypothesis) and which can be combined by the `if` form to recover the original σ.

5.1.2 Intuitionistic Böhm trees

The passage from μPCF to PCF invokes Hyland & Ong's *bracketing condition* to prevent precocious answers from being played. Specifically, if $s \in \mathsf{dom}(\sigma)$ then the ***pending question*** of s is the first Opponent question encountered while tracing back the P-view of s. A strategy σ is ***well-bracketed*** iff, for all $t \in \sigma$ ending with a Player answer, the justifier of that answer is the pending question of t. Equivalently, the answer points in that suffix of the P-view terminated by the pending question.

The following play violates the bracketing condition since its final occurrence points past the pending question (the third question played), instead answering the initial question directly.



A simplified version of the argument for P-visibility establishes that composing well-bracketed σ and τ always yields well-bracketed $\sigma\,;\tau$. Well-bracketed strategies thus form a subcategory of $\mathbf{I}$ (the identities always being well-bracketed) which is again a CCC. Unlike the general case of an innocent strategy, in a well-bracketed σ, the substrategy $\mathtt{args}(\sigma)$ *must* pass control to the other substrategy $\mathtt{cnts}(\sigma)$; we cannot pre-empt $\mathtt{cnts}(\sigma)$ by throwing a value direct to the "top level" (as does the non-bracketed example above).

The bracketing condition thus renders the μ-binder and the naming construct redundant, the following simplified language IBT being sufficient to establish definability for well-bracketed compact innocent strategies.

$$\begin{array}{lcl} F & ::= & \lambda\vec{x}(E) \\ E & ::= & \Omega \mid \mathtt{t} \mid \mathtt{ff} \mid \mathtt{if}\ (x)\vec{F}\ E\ \mathtt{else}\ E \end{array}$$

We type executables E by sequents $\Gamma \vdash E : \mathtt{bool}$ and terms F as before. Alternatively, one could type executables as in CBT but with the requirement that Δ, the list of name-base type pairs, has (at most) one entry.

IBT

$$\frac{}{\Gamma \ \vdash\ \Omega : \mathtt{bool}} \qquad \frac{}{\Gamma \ \vdash\ \mathtt{t} : \mathtt{bool}} \qquad \frac{}{\Gamma \ \vdash\ \mathtt{ff} : \mathtt{bool}}$$

$$\frac{\Gamma \vdash F_1 : T_1 \ \cdots\ \Gamma \vdash F_n : T_n \qquad \Gamma \vdash E_0 : \mathtt{bool} \qquad \Gamma \vdash E_1 : \mathtt{bool}}{\Gamma, x : T_1 \to \cdots \to T_n \to \mathtt{bool} \vdash (\mathtt{if}\ (x)\vec{F}\ E_0\ \mathtt{else}\ E_1) : \mathtt{bool}}$$

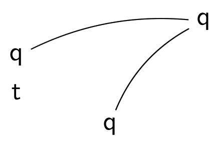
$$\frac{\Gamma, x_1 : T_1 \ldots x_n : T_n \ \vdash\ E : \mathtt{bool}}{\Gamma \ \vdash\ \lambda\vec{x}(E) : T_1 \to \cdots \to T_n \to \mathtt{bool}}$$

Theorem 5.1.2 *Every compact, well-bracketed innocent strategy σ on an arena A interpreting a simple type (over* $\mathtt{bool}$*) is definable in IBT.*

5.1.3 Rigid Böhm trees

We've already seen how the bracketing condition rules out any use of non-local control operators. In this section, we introduce a "dual" constraint which limits the use of *local* control operators (such as `if-then-else` or `case`) to defining *unary functions*: continuations of `case` statements cannot themselves be `case`s.

Analogously to the above definition of the pending question of s, define the *extant answer* of s to be the first OA-occurrence encountered while tracing back the P-view. This induces a suffix of $\ulcorner s \urcorner$ which we call the *rigid view.* A strategy σ is ***rigid*** iff, for all $t \in \sigma$ ending with a question, the justifier of that question occurs in the rigid view. Thus, a violation of rigidity corresponds to the guts of an `if-then-else` or `case` form:



The extant answer of the view qqt is clearly its last occurrence t yet the strategy responds with a question pointing beyond that, to the initial question. In an arena where *only questions can enable*, being rigid implies that once an answer is given, no more questions can be asked. In other words, each view of a rigid strategy contains either no answers at all, exactly one Player answer (its final occurrence) or one Opponent and one Player answer (the last two occurrences).

As for the bracketing condition, a simple reworking (see [1]) of the argument for P-visibility establishes that we have a subcategory of rigid strategies. The grammar below picks out that fragment of CBT, called RBT, that corresponds to this subcategory—those terms of CBT with no "cascaded" `case`s:

$$\begin{array}{rcl} F & ::= & \lambda\vec{x}\mu\alpha(E) \\ E & ::= & C \mid \texttt{if } (x)\vec{F}\ C \texttt{ else } C \\ C & ::= & \Omega \mid [\alpha]\texttt{t} \mid [\alpha]\texttt{ff} \end{array}$$

We type continuations C and executables E in the same way as executables in CBT.

RBT

$$\frac{}{\Gamma\ ;\ \Delta\ \vdash\ \Omega} \qquad \frac{}{\Gamma\ ;\ \Delta, \alpha : \texttt{bool}\ \vdash\ [\alpha]\texttt{t}} \qquad \frac{}{\Gamma\ ;\ \Delta, \alpha : \texttt{bool}\ \vdash\ [\alpha]\texttt{ff}}$$

$$\frac{\Gamma\ ;\ \Delta\ \vdash\ F_1 : T_1\ \cdots\ \Gamma\ ;\ \Delta\ \vdash\ F_n : T_n \qquad \Gamma\ ;\ \Delta\ \vdash\ C_0 \qquad \Gamma\ ;\ \Delta\ \vdash\ C_1}{\Gamma, x : T_1 \to \cdots \to T_n \to \texttt{bool}\ ;\ \Delta\ \vdash\ (\texttt{if } (x)\vec{F}\ C_0 \texttt{ else } C_1)}$$

$$\frac{\Gamma, x_1 : T_1 \ldots x_n : T_n\ ;\ \Delta, \alpha : \texttt{bool}\ \vdash\ E}{\Gamma\ ;\ \Delta\ \vdash\ \lambda\vec{x}\mu\alpha(E) : T_1 \to \cdots \to T_n \to \texttt{bool}}$$

Theorem 5.1.3 *Every compact, rigid innocent strategy σ on an arena A interpreting a simple type (over* `bool`*) is definable in RBT.*

Unlike the bracketing condition which, as we've seen, restricts the behaviour of $\texttt{args}(\sigma)$, rigidity constrains $\texttt{cnts}(\sigma)$: when an answer is received, rigidity prevents σ from posing a new question—so the continuation C either diverges or immediately plays an answer.

5.1.4 The bracketing–rigidity diamond

We've now seen, in isolation, how bracketing and rigidity restrict the definitional power of innocent strategies. We can organize these results into the diamond (where B = bracketed, R = rigid, I = all innocent) of subcategories induced by imposing all combinations of B and R:

$$\begin{tikzcd}
 & I & \\
R \arrow[ur] & & B \arrow[ul] \\
 & B \wedge R \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

So the category B corresponds to the IBT language, R to the RBT language and I evidently to CBT. The base of the diamond corresponds to another Böhm tree language with a particularly simple interpretation: computation consists of asking a sequence of questions and then answering them in reverse order, *i.e.* RBT without the possibility of skipping questions. More generally, this category harbours a fully abstract model of the simply typed λ-calculus (over some collection of base types) extended with constants for values of base type and primitive *unary* functions between base types.

This classification of innocent strategies as a diamond allows us to judge the "nature" of strategy by examining in which of the four categories it lives. For example, if a strategy lives in B but not in R, all its plays are well-bracketed but it must contain non-rigid plays. So, if a strategy lives only in I, we deduce that it contains (at least) one play violating bracketing and another violating rigidity. However, this does *not* necessarily mean that, during interaction with another strategy, both bracketing and rigidity will be violated. Indeed, since one constraint applies only to answers and the other only to questions, we cannot *simultaneously* violate both. Nonetheless, it can happen that, for some $s \in \sigma$, we have prefixes $s_r \sqsubseteq s$ and $s_b \sqsubseteq s$ where s_r (resp. s_b) violates rigidity (resp. bracketing). In such circumstances, we have few qualms about placing σ in I. On the other hand, we could easily construct a strategy τ which begins by interrogating a Boolean argument and subsequently violates B iff that argument evaluated to $\texttt{t}$ and violates R iff it evaluated to $\texttt{ff}$. Clearly τ never violates B *and* R in the course of a single interaction with its context, yet τ only lives in category I since it obviously violates the entry requirements for B and R.

In summary, the diamond of categories can help us to better understand some, but not all, aspects of the behaviour of innocent strategies: it can guarantee the "good behaviour" of a strategy and can warn of possible "bad behaviour". This "map" of innocence could be usefully complemented by an analysis, as hinted above, where we focus on the behaviour of a strategy in a *given* context, rather than in all contexts.

References

[1] V. Danos and R. Harmer. The anatomy of innocence. In *Proceedings, Tenth Annual Conference of the European Association for Computer Science Logic*. Springer Verlag, 2001.

[2] H. Herbelin. Games and weak-head reduction for classical PCF. In P. de Groote and J. R. Hindley, editors, *Third International Conference on Typed Lambda Calculi and Applications, TLCA '97, Nancy, France, April 2-4, 1997, Proceedings*, volume 1210 of *Lecture Notes in Computer Science*, pages 214–230. Springer, 1997.