Electronic Notes in Theoretical Computer Science 45 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume45.html 26 pages

A Game Semantics of Idealized CSP

J. Laird

COGS, University of Sussex
Brighton BN1 9QH, UK
E-mail: jiml@cogs.susx.ac.uk

Abstract

A games semantics is described for a typed functional language which includes primitives for parallel composition and for synchronous communication on private channels. The semantics is based on a category obtained by extending "Hyland-Ong games" with a representation of multiple threads of control using "concurrency pointers" which express a new kind of causality relation between moves. The semantics is proved to be fully abstract for "channel-free" types with respect to a *may-and-must* equivalence for the finitary fragment, and with respect to *may*-equivalence for the whole language, using factorization results to reduce definability to the sequential case.

1 Introduction

Game semantics has been successfully used to give models of various sequential programming languages, with the distinctive feature that many of the underlying notions are intensional, combining ideas from concurrency theory and from traditional domain theory. A hierarchy of fully abstract models has emerged, based on the Hyland-Ong (HO) model of PCF [12] extended with non-functional (but still sequential) features such as mutable state and store [1,4] and control [13,14], to the point where there is now a thorough analysis of *sequential* functional computation which is to a large extent *based on concurrency*. Extending existing games models to give an interpretation of concurrent features is therefore a natural development, although it requires a substantial change in perspective, as existing models are based on representation of interaction as alternating *sequences* of tokens (moves).

One such construction — *concurrent games* [3] — has been used by Abramsky and Mellies to model multiplicative-additive linear logic (in which concurrency is implicit rather than otherwise). The concurrent games are based on a "true concurrency" representation of interaction as information flow, with no notion of moves as discrete pieces of information at all. This paper describes a different representation of concurrent behaviour — "multi-threaded games"

— which retains the token-based notion of interaction but adds a representation of *multiple threads of control*, allowing the rich structure of HO games models of side-effects to be exploited to model *explicit* concurrency features such as communication on private channels.

Multi-threaded games are used to give a semantics of a typed call-by-name λ-calculus with arithmetic plus a parallel composition operator and (ground-type) message passing along channels with locally declared names. The syntax of this language is essentially the same as that of Brookes' Idealized CSP [7] and so we have retained this name, although there are several differences in the operational semantics; message passing is synchronous (as opposed to asyncronous in [7]) and there is a key difference in the interpretation of the parallel composition operator; this could be summarized by saying that it is interpreted here as parallel composition of *processes*, and in [7] as parallel composition of *commands*.

Nonetheless, the semantic analysis provided by the games model can be regarded as complementary to the functor-category semantics described in [7]. Its distinguishing feature is that it is fully abstract with respect to may testing (for the full language), and with respect to may-and-must testing (for the recursion-free fragment). The full abstraction result is restricted to "channel-free" types, although it could be extended to all types by adding a "bad channel" constructor to the language, analogous to the "bad variable" constructor `mkvar` which has been added to Idealized Algol for the same purpose [1].

The basic framework of Hyland-Ong games gives us an interpretation of the λ-calculus with arithmetic. The extension to model concurrency has two key features; *multi-threading* and *synchronous message-passing*.

The interpretation of multiple threads of control requires the most radical extension to previous work in game semantics, which (with the exception of [3]) has built models of sequential languages by representing interaction of strategies as a *sequences* of moves. We add a new set of "concurrency pointers" to these sequences, so that they can be seen as representing the interaction of two strategies as a *tree* of moves. Retaining the sequential ordering on moves, however, allows us to capture *synchronizations* between moves in different branches of a tree. And by following back control pointers we can extract a "control thread" (a branch of the tree) for each move. These are ordinary sequences from the associated HO game, allowing a subcategory of "sequential strategies" isomorphic to the original HO games and strategies to be defined. There is a natural operation of parallel composition of strategies in the multi-threaded setting, based on interleaving of control threads. Moreover, we show (by factorization) that all branching of control threads in finitary strategies can be obtained from such parallel compositions.

Message passing along private channels is interpreted in "object-oriented" style, as in the functor-category model [7], and the games models of locally bound references [1,4] and exceptions [14]. In fact the interpretation of the chan type as the products of its "methods", send and recv, is precisely the same as the interpretation of the type var and its methods (assignment and dereferencing) in the games model of Idealized Algol [1]. Thus the only difference between the semantics of the two features is in the interpretations of the new-variable and new-channel declarations. Both are given in terms of composition with a strategy which violates some of the constraints obeyed by elements corresponding to purely functional terms, and captures the causal relationships between writing and reading or sending and receiving. The key property of the "new-channel" strategy is that it contains a synchronous communication between control threads, and as we show by a factorization result, it can be used to generate all such behaviour in finitary strategies in the model.

The remainder of the paper splits basically into two parts; Sections 2 – 4 define a sound games model of ICSP, and Section 5 refines it into a fully abstract semantics. Accordingly, the first part accomplishes the new constructions required to interpret threads and communication in games, whilst the second contains more technical material relating it to the original HO models. The detail of the organization is as follows:
— In section 2, a syntax, operational semantics and contextual equivalences are defined for ICSP.
— Section 3 describes the category of "multi-threaded games" which is the basis for the model.
— Section 4 gives the denotational semantics of ICSP in this category and shows that it is sound and adequate with respect to the operational semantics.
— Section 5 refines the semantics, and derives full abstraction results for it. First, a new condition — "pointer-blindness" — is used to cut down the model. Then, a subcategory of *sequential* strategies is defined; the sequential strategies which also satisfy (multi-threaded versions of) the constraints of *visibility, innocence and well-bracketing* are shown to form a category isomorphic to that used to model PCF in [12]. Finally, we give two factorization results to show that every finitary strategy in the model is definable as a term of ICSP, from which full abstraction follows.

2 Idealized CSP

Idealized CSP [7], or ICSP, is based on the call-by-name λ-calculus, with types generated from the ground types comm (commands), nat (natural numbers) and chan (channels for sending and receiving natural numbers). (We shall sometimes use B to represent the atomic or *basic* types comm and nat, and say that a type is *channel-free* if it is built from basic types with $\Rightarrow$.)
$T ::= \mathtt{comm} \mid \mathtt{nat} \mid \mathtt{chan} \mid T \Rightarrow T$
Terms are formed according to the following grammar:

$$M ::= x \mid \mathtt{skip} \mid \mathtt{0} \mid \mathtt{succ}\, M \mid \mathtt{pred}\, M \mid \mathtt{IF0}\, M \mid \lambda x.M \mid M\, M \mid \mathbf{Y}M$$

$$\mathtt{nil} \mid M;M \mid M \| M \mid \mathtt{newchan}\, M \mid \mathtt{send}\, M\, M \mid \mathtt{recv}\, M$$

The language combines the λ-calculus with concurrency primitives (based on CSP [10]) much as Idealized Algol [19] does with imperative features (an intellectual debt acknowledged in its name); declaring a new channel name and sending and receiving on it are analogous to declaring, allocating and deallocating a variable. Typing judgements for the concurrency features are as follows:

$$\frac{\Gamma\vdash M{:}B \quad \Gamma\vdash N{:}B}{\Gamma\vdash M\|N{:}B} \qquad \frac{}{\Gamma\vdash \mathtt{nil}{:}B} \qquad \frac{\Gamma\vdash M{:}\mathtt{comm} \quad \Gamma\vdash N{:}B}{\Gamma\vdash M;N{:}B}$$

$$\frac{\Gamma\vdash M{:}\mathtt{chan}\Rightarrow B}{\Gamma\vdash \mathtt{newchan}\,M{:}B} \qquad \frac{\Gamma\vdash M{:}\mathtt{chan} \quad \Gamma\vdash N{:}\mathtt{nat}}{\Gamma\vdash \mathtt{send}\,M\,N{:}\mathtt{comm}} \qquad \frac{\Gamma\vdash M{:}\mathtt{chan}}{\Gamma\vdash \mathtt{recv}\,M{:}\mathtt{nat}}$$

Given $M : \mathtt{nat}, N : B$, we shall also write $M; N$ for $((\mathtt{IF0}\ M)\ N)\,N$.

The operational semantics is given in terms of a "small-step" reduction relation on *configurations* of ICSP. A configuration $C = N_1, \ldots, N_k$ is a multiset of parallel *threads* — ICSP programs of the same (base) type — containing the free channel names $\mathrm{Ch}(C)$. (Equivalently, one could reduce a single program using structural congruences and scope extrusion.) The reduction rules use the notion of *evaluation context* to pick out a unique next redex for each thread which is not a value.

Definition 2.1 *Evaluation contexts are given by the following grammar:*

$$E[\cdot] ::= [\cdot] \mid E[\cdot]\,M \mid \mathtt{pred}\ E[\cdot] \mid \mathtt{succ}\ E[\cdot] \mid \mathtt{IF0}\ E[\cdot] \mid$$
$$E[\cdot]; M \mid \mathtt{send}\,M\,E[\cdot] \mid \mathtt{send}\,E[\cdot]\,n \mid \mathtt{recv}\,E[\cdot]$$

The small-step reduction rules are as follows:

$$\begin{gathered}
E[\lambda x.M\,N], C \longrightarrow E[M[N/x]], C \\
E[\mathbf{Y}M], C \longrightarrow E[M\,\mathbf{Y}M], C \\
E[\mathtt{pred}\ (\mathtt{succ}\ M)], C \longrightarrow E[M], C \\
E[\mathtt{IF0}\ \mathtt{0}], C \longrightarrow E[\lambda xy.x], C \\
E[\mathtt{IF0}\ (\mathtt{succ}\ \mathtt{n})], C \longrightarrow E[\lambda xy.y], C \\
E[\mathtt{skip}; M], C \longrightarrow E[M], C \\
E[M\|N], C \longrightarrow E[M], E[N], C \\
E[\mathtt{newchan}\,M], C \longrightarrow E[M\,c], C \ : \ c \notin Ch(E[\mathtt{newchan}\,M], C) \\
E[\mathtt{send}\,a\,\mathtt{n}], E'[\mathtt{recv}\,a], C \longrightarrow E[\mathtt{skip}], E[\mathtt{n}], C.
\end{gathered}$$

The operation `newchan` allows local or private channels to be declared — $\mathtt{newchan}\,M$ supplies a new channel name as an argument to $M : \mathtt{chan} \Rightarrow B$ and adds it to the environment. Evaluation of `send` is by reducing its second argument to a value (numeral) and then its first argument to a channel name, `recv` evaluates its argument to a channel name, and $\mathtt{send}\,a\,\mathtt{n}$ and $\mathtt{recv}\,a$ reduce in parallel to `skip` and `n` respectively. By contrast to [7], message passing is synchronous. The process $\texttt{nil} : B$ can be regarded as shorthand for either of the deadlocked processes $\texttt{newchan}\,\lambda c.\texttt{send}\,c\,\texttt{0} : \texttt{comm}$ or $\texttt{newchan}\,\lambda c.\texttt{recv}\,c : \texttt{nat}$. Note that evaluating a parallel composition duplicates the evaluation context. This may seem surprising, but makes more sense if we recall that the evaluation context is in effect a representation of the current continuation. So *parallel composition* of (ground-type) *processes* means "split the control thread in two and (concurrently) run one argument *with the current continuation* in each thread". Thus the parallel composition of two values (say `skip` and `skip`) reduces to a configuration consisting of two threads containing these values which cannot be further evaluated.

This is distinct from the interpretation of $M\|N$ in [7] which one might call "parallel composition of *commands*"; M and N are evaluated concurrently and the single value `skip` is returned if at least one evaluates to `skip`. So, for instance, the parallel composition of the commands `skip` and `skip` is equivalent to `skip`. Using parallel composition of processes with synchronous communication we can express parallel composition of commands $M, N : \texttt{comm}$ as:

$$\texttt{newchan}\,\lambda c.((M; \texttt{send}\,c\,\texttt{0})\|(N; \texttt{send}\,c\,\texttt{0})\|(\texttt{recv}\,c; \texttt{nil})).$$

Various other programming constructs such as Algol-style store and non-deterministic choice can be expressed in ICSP. A useful example; for any $n > 0$ define the program $\texttt{oracle}_n$ which erratically produces one of the numbers less than n.

$$\texttt{oracle}_n = \texttt{newchan}\,\lambda c.((\texttt{send}\,c\,\texttt{0}\|\texttt{send}\,c\,\texttt{1}\|\ldots\|\texttt{send}\,c\,(\texttt{pred n}); \texttt{nil}\|\texttt{recv}\,c)$$

For examples of solutions to more subtle programming problems involving concurrency see e.g. [7].

2.1 Convergence testing and operational equivalence

Observational equivalence is defined with respect to a simple test — having *at least one* convergent thread. This is equivalent to testing for exactly one convergent thread, or at least/exactly n threads for any finite $n > 0$. Since (as one would expect) $\texttt{nil}\|M$ will be denotationally equal to M, we cannot observe the existence of deadlocked threads so it will not be possible to test whether *all* threads converge.

Definition 2.2 *Define the predicate* $\Downarrow^{may}$ *(may convergence) on configurations of type* `comm` *as follows:*

$$\frac{}{C, \texttt{skip} \Downarrow^{may}} \qquad\qquad \frac{\exists C'.C \rightarrow C' \wedge C' \Downarrow^{may}}{C \Downarrow^{may}}$$

Terms $M, N : T$ *are* may-equivalent *(*$M \simeq^{may}_T N$ *if for all program contexts* $C[\cdot] : \texttt{comm}$*,* $C[M] \Downarrow^{may}$ *if and only if* $C[N] \Downarrow^{may}$*.*

May-equivalence is a natural equivalence to model using game semantics — as we shall see, it corresponds to trace-equivalence of strategies. However, for an inherently non-deterministic language such as ICSP it is rather weak — it leads to a failure to distinguish programs which always converge from those which may converge or diverge [9,8]. A stronger notion of equivalence can be derived from *may-and-must* testing.

Definition 2.3 *Define the predicate $\Downarrow^{must}$ (must convergence) on configurations:*

$$\frac{}{C, \texttt{skip} \Downarrow^{must}} \qquad \frac{\forall C'.(C \rightarrow C') \Longrightarrow C' \Downarrow^{must}}{C \Downarrow^{must}}$$

Terms $M, N : T$ are must-equivalent *if for all closing contexts $C[\cdot] : \texttt{comm}$, $C[M] \Downarrow^{must}$ if and only if $C[N] \Downarrow^{must}$. We shall write $M \simeq_T^{M\&M} N$ if M and N are may-equivalent* and *must-equivalent.*

Like games (and other) models of sequential languages, our semantics will reflect β equality as a denotational equivalence — e.g. we have $[\![\lambda x.x\ \texttt{skip}]\!] = [\![\texttt{skip}]\!]$. Unfortunately, in the presence of infinite reduction sequences (without fairness of evaluation), β-equality is not sound with respect to may-and-must testing. For example, $\lambda x.x\ \texttt{skip} \not\simeq^{M\&M} \texttt{skip}$ since $\mathbf{Y}\lambda y.y \| \lambda x.x\ \texttt{skip} \not\Downarrow^{must}$. This suggests that to give a denotational semantics of ICSP which is even adequate with respect to may and must testing (or a bisimulation equivalence), it will be necessary to incorporate the reduction behaviour of terms into the semantics. Here we shall just give the may-and-must result for the "finitary" fragment of ICSP for which evaluations always terminate; we shall call the language without the $\mathbf{Y}$-combinator *finitary ICSP* .

Proposition 2.4 *There is no infinite series of configurations of finitary ICSP $C_1, C_2, \ldots C_n, \ldots$ such that $C_1 \longrightarrow C_2 \longrightarrow \ldots \longrightarrow C_n \longrightarrow \ldots$.*

Proof. Is by a standard 'computability predicate" based argument. □

3 Multi-threaded Games

The games constructions which will be used to model ICSP are based on those given by Hyland and Ong [12] and Nickau [16] (and developed in [15,1,4,11]), in which states of the game — interactions between the system and the environment — are represented as *justified sequences* of moves. However, in order to model concurrency we shall enrich this structure with an additional causality relation between moves, expressed using "concurrency pointers". This allows a tree of multiple, interleaved "threads of control" to be represented as a single sequence. However, such a "multi-threaded sequence" carries additional information about the temporal ordering of events in different branches of the tree, and this extra information will be important in modelling syncronization.

The structure of a game (the moves, their labels, how they are related) is specified by its *arena.* This is defined essentially as in [12]; no new structure is required to support multi-threading. An arena A is a triple $\langle M_A, \vdash_A \subseteq$

$(M_A)_* \times M_A, \lambda_A : M_A \to \{Q, A\}\rangle$, where:
— M_A is a set of tokens called moves,
— $\vdash_A \subseteq (M_A)_* \times M_A$ is a relation called *enabling*,
— $\lambda^A : M_A \to \{Q, A\}$ is a function which labels moves as *answers* (A) or *questions* (Q), such that every answer has a unique enabling move which is a question.
We stipulate that a *unique* polarity for all of the moves in M_A can be inferred from the enabling relation using the rule:
m is an O-move if it is *initial* (i.e. $* \vdash m$), or enabled by a P-move,
m is a P-move if it is enabled by an O-move.

The simplest arenas are the *empty arena* $\mathbf{1}$, with no moves, and the *flat arenas* with a single, initial Opponent question q enabling a set of Player answers which are used to interpret basic types (here, as in [12] etc.); the interpretation of `comm` is the flat arena with a single answer ("execution of the command has successfully terminated") and the interpretation of `nat` is the flat arena with a set of answers indexed by the natural numbers. The product and function-space constructors from [12] also carry through to the multi-threaded setting.

Product For any set-indexed family of arenas $\{A_i \mid i \in I\}$, form the product $A = \Pi_{i \in I} A_i$ as follows:
- $M_{\Pi_{i \in I} A_i} = \coprod_{i \in I} M_{A_i}$,
- $\langle m, i\rangle \vdash_{\Pi_{i \in I} A_i} \langle n, j\rangle$ if $i = j$ and $m \vdash_{A_i} n$, and $* \vdash_{\Pi_{i \in I} A_i} \langle n, j\rangle$ if $* \vdash_{A_j} n$,
- $\lambda^{QA}_{\Pi_{i \in I} A_i}(\langle m, i\rangle) = \lambda_{A_i}(m)$.

We shall write A^k for $\Pi^k_{i=1} A$.

Function Space For arenas A_1, A_2, form $A_1 \Rightarrow A_2$ as follows:
- $M_{A_1 \Rightarrow A_2} = M_{A_1} + M_{A_2}$,
- $\langle m, i\rangle \vdash_{A_1 \Rightarrow A_2} \langle n, j\rangle$ if $i = j$ and $m \vdash n$
  or $m \in M_{A_2}$, $n \in M_{A_1}$ and $* \vdash_{A_2} m$ and $* \vdash_{A_1} n$,
  $* \vdash \langle m, i\rangle$ if $m \in M_{A_2}$ and $* \vdash_{A_2} m$,
- $\lambda^{QA}_{A_1 \Rightarrow A_2}(\langle m, i\rangle) = \lambda_{A_i}(m)$.

In general $l, m, n \ldots$ will be used to denote the moves of an arena, and $a, b, c, \ldots$ to denote *occurrences* of these moves in *sequences* r, s, t. The empty sequence is denoted ε, and for any sequences s, t, $s \sqsubseteq t$ means 's is a prefix of t', sa denotes the sequence s extended by the move a, and st the sequence s extended by the sequence t.

In a sequence of moves sa, a justification pointer for s is a pointer from a to some move in s which enables a. A *justified sequence* over an arena A is a sequence of elements of M_A in which each non-initial move a has a justification pointer. An *alternating* justified sequence t is one in which Opponent moves are always followed by Player moves, and vice-versa. We shall write J_A for the set of alternating justified sequences over A.

3.1 Multi-threaded sequences

We shall now define the *multi-threaded sequences* on which our model is based. They are formed by adding "concurrency pointers" to justified sequences.

Definition 3.1 *Let sa be a sequence of moves. A concurrency pointer for a is a pointer from a to some single (occurrence of a) move in s, which distinct from its justification pointer (if any). A move is* thread-initial *if it does not have a concurrency pointer. Let MT_A be the set of multi-threaded sequences — justified sequences with concurrency pointers — over the arena A. A justified sequence without concurrency pointers (that is, the standard notion of justified sequence used in [12]) is said to be "single-threaded".*

By tracing back concurrency pointers, we can extract a series of single-threaded subsequences, or "control threads" from each multi-threaded sequence.

Definition 3.2 *The* control thread *of the last move in a sequence s is defined by induction on length as follows:*
$\mathsf{CT}(a) = a$*, if a is thread-initial,*
$\mathsf{CT}(satb) = \mathsf{CT}(sa)b$ *if b has a concurrency pointer to a.*

A *multi-threaded legal sequence* is a finite sequence with concurrency and justification pointers, such that each thread is an alternating justified sequence:

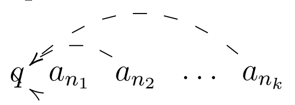
- every O-move which is not thread-initial has a concurrency pointer to a P-move, and vice-versa,
- every non-initial move has a justification pointer to a move in its control thread.

Definition 3.3 *For an arena A, define the set of multi-threaded legal sequences as follows: $LM_A = \{s \in MT_A \mid \forall t \sqsubseteq s.\mathsf{CT}(t) \in J_A\}$.*

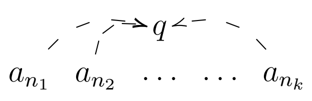
Definition 3.4 *A legal sequence (multi-threaded or single-threaded) is well-opened if it contains at most one initial Opponent move. Let WM_A be the set of well-opened (multi-threaded) legal sequences on A:*

$$WM_A = \{s \in LM_A \mid ta \sqsubseteq s \wedge \vdash a \Longrightarrow t = \varepsilon\}$$

For example, the well-opened *single-threaded* sequences on the flat arena $[\![\texttt{nat}]\!]$ are at most three moves long, and have the form ε, q or qa_i for some $i \in \omega$. By contrast, for any finite sequence of natural numbers $n_1 n_2 \ldots n_k$ there is a unique sequence



in which each (Player) answer a_i has a concurrency (and justification pointer) back to q. This sequence is a representation of a tree of control threads:



— as we shall see, it is a trace in the strategy $[\![\texttt{n}_1 \| \texttt{n}_2 \| \ldots \| \texttt{n}_k]\!]$.

3.2 Representing concurrent interaction

The representation of trees as sequences with pointers can easily be seen to be non-unique; in the example above, any permutation of the answers $a_{n_1}, \ldots, a_{n_n}$ will give a trace in $[\![\mathtt{n}_1 \| \mathtt{n}_2 \| \ldots \| \mathtt{n}_k]\!]$. We can recover uniqueness by deeming that all such interleavings of control-threads are equivalent, but one might then ask: what is the point of representing multiple control threads in a sequential form in the first place? The answer is that the sequential ordering of moves is required in order to synchronize events between different threads. However, since Idealized CSP itself contains only limited facilities for such synchronizations, multi-threaded sequences still do not give unique representations of interactions between the corresponding strategies. Thus Player-strategies will have only limited power to observe and control the actual ordering of moves in different threads. Player can wait until a given O-move has occurred before giving a response (in ICSP it is possible to suspend evaluation of a thread until an event occurs in another thread). However, Player cannot:

- force one P-move to occur before another (Player or Opponent) move, or
- *observe* the order in which two contiguous O-moves occur.

To reflect these constraints, we define a preorder $\precsim$, such that $s \precsim t$ if s can be obtained from t by migrating P moves forward, and migrating O-moves back. We shall require that strategies be closed under $\precsim$ — in other words, if t is a trace of a strategy and $s \precsim t$ then s must also be a trace of the strategy.

Definition 3.5 *Let $\precsim$ be the least preorder on multi-threaded legal sequences such that for all $sab \cdot t, sba \cdot t \in LM_A$ such that $\lambda^{OP}(a) = O$ or $\lambda^{OP}(b) = P$, $sab \cdot t \precsim sba \cdot t$.*

The original representation of sequential, deterministic Player-strategies [12] is as sets of even-length sequences in which the final move represents the response of Player to the preceding sequence. As observed in [8] this form of representation is not sufficient to model a simple non-deterministic functional language up to may *and must* equivalence; for example, it identifies a strategy which always converges with one which may converge or diverge. In the multi-threaded setting, the absence of the alternation condition complicates matters further; it is no longer the case that the space of positions is partitioned between those in which it is Opponent's turn to move, and those in which it is Player's turn to move. So Player could play a string of moves without waiting for Opponent's response, or might require several O-moves to prompt a single move in reply. A multi-threaded strategy σ on an arena A will be represented as the set of its traces at which it (possibly) halts, either waiting for ever, or for some more O-moves.

Definition 3.6 *Given a set $\sigma \subseteq LM_A$, let $T_\sigma = \{s \in LM_A \mid \exists t \in \sigma . s \sqsubseteq t\}$.*

When σ is a strategy, T_σ represents the set of reachable *traces* of σ.

Definition 3.7 *A (multi-threaded) strategy $\sigma : A$ is a subset of LM_A subject* *to the following conditions:*

- $\varepsilon \in \sigma$,
- *σ is closed with respect to $\precsim$ — $s \precsim t$ and $t \in \sigma$ implies $s \in \sigma$,*
- *the extension of a reachable position with an O-move is reachable — if $s \in T_\sigma$ and a is an O-move such that $sa \in LM_A$ then $sa \in T_\sigma$,*
- *in any reachable position, σ must either play a P-move or wait for another O-move — if $s \in T_\sigma$ then either $s \in \sigma$ or there is some P-move a such that $sa \in T_\sigma$.*

A single-threaded strategy on A is a subset of L_A satisfying these conditions (closure under $\precsim$ is clearly trivial).

To give a fully abstract semantics for may-testing for the full language we shall consider strategies only up to equivalence of reachable traces.

Definition 3.8 *$\sigma : A$ is* trace-equivalent *to $\tau : A$ ($\sigma \approx_A \tau$) if $T_\sigma = T_\tau$.*

We shall now define a category of arenas and strategies. First, extend the notion of *restriction* to multi-threaded sequences so that it "mends" both justification and concurrency pointers.

Definition 3.9 *Given $s \in L_{(A_1 \Rightarrow A_2) \Rightarrow A_3}$, $s \upharpoonright (A_i, A_j)$ ($i < j$) is a multi-threaded sequence on $A_i \Rightarrow A_j$ defined as follows:*
$\varepsilon \upharpoonright (A_i, A_j) = \varepsilon$
$sa \upharpoonright (A_i, A_j) = s \upharpoonright (A_i, A_j)$ *if* $a \notin M_{A_i}, M_{A_j}$
$sa \upharpoonright (A_i, A_j) = (s \upharpoonright (A_i, A_j))a$ *if* $a \in M_{A_i}, M_{A_j}$.
The justifier of a in $sa \upharpoonright (A_i, A_j)$ is the most recently played move from A_i or A_j which hereditarily justifies *a. (If there is no such move, then a is initial.) The concurrency pointer from a points to the most recently played move (if any) from A_i or A_j which is in $\mathsf{CT}(sa)$.*

Lemma 3.10 *If $s \in LM_{(A_1 \Rightarrow A_2) \Rightarrow A_3}$ then $\mathsf{CT}(s \upharpoonright (A_i, A_j)) = \mathsf{CT}(s) \upharpoonright (A_i, A_j)$.*

As in other models, canonical morphisms are *copycat* strategies which simply copy Opponent moves between different parts of a game. (However, there is a difference between the treatment of concurrency and justification pointers; the copy of an O-move a has a concurrency pointer to a itself, and a justification pointer to the move of which the justifier of a was a copy.) The identity strategy is the prime example.

Definition 3.11 *Say that $s \in LM_{A \Rightarrow A}$ is a* copycat *sequence if for all $t \sqsubseteq^{even} s$, $t \upharpoonright A^- = t \upharpoonright A^+$, and every P-move in s has a concurrency pointer to the preceding O-move. Then $\mathtt{id}_A = \{s \in LM_{A \Rightarrow A} \mid \exists t. s \precsim t \wedge t \,\text{is a copycat}\}$.*

The 'parallel composition with hiding' [5] of strategies is defined as follows.

Definition 3.12 *Given $\sigma : A_1 \to A_2$ and $\tau : A_2 \to A_3$, let $\sigma;\tau = \{t \in LM_{A_1 \Rightarrow A_3} \mid \exists s \in LM_{(A_1 \Rightarrow A_2) \Rightarrow A_3}. t = (s \upharpoonright A_1, A_3) \wedge (s \upharpoonright A_1, A_2) \in \sigma \wedge (s \upharpoonright A_2, A_3) \in \tau\}$*

Lemma 3.13 *If $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$, then $\sigma;\tau : A \Rightarrow C$.*

Proof. The key points are that if $s \in LM_{(A_1 \Rightarrow A_2) \Rightarrow A_3}$, then $s \restriction A_i, A_j$ is a legal sequence, and closure under $\precsim$. The former follows from the single-threaded case [12], using Lemma 3.10. The latter is proved by showing that, given $s \in LM_{(A_1 \Rightarrow A_2) \Rightarrow A_3}$ and $t \in LM_{A_1 \Rightarrow A_3}$, such that $s \restriction A_1, A_3 \precsim t$, we can find $r \in LM_{(A_1 \Rightarrow A_2) \Rightarrow A_3}$ such that $r \restriction A_1, A_3 = t$, $s \restriction A_1, A_2 \precsim r \restriction A_1, A_2$ and $s \restriction A_2, A_3 \precsim r \restriction A_2, A_3$. We derive r from s by migrating P-moves in A_3 forward, and O-moves in A_1 back, and swapping contiguous O-moves and contiguous P-moves.

The proof that composition is associative follows the standard argument from [12], which is little altered by the presence of concurrency pointers and absence of the alternation condition.

Proposition 3.14 *For all $\sigma : A \Rightarrow B, \tau : B \Rightarrow C, \rho : C \Rightarrow D$ $\sigma;(\tau;\rho) = (\sigma;\tau);\rho$.*

Definition 3.15 *To construct a cartesian closed category, however, we need the notion of* well-opened *strategy. A well-opened strategy $\sigma : A$ is a subset of WM_A (Definition 3.4) satisfying the conditions laid down in Definition 3.7 with respect to well-opened sequences rather than legal sequences.*

Composition of well-opened strategies $\sigma : A \Rightarrow B$ with $\tau : B \Rightarrow C$ is by "promotion" of σ, followed by parallel composition with hiding, where promotion is defined as follows.

Definition 3.16 *For a (multi-threaded legal) sequence s containing an initial move a, define $s \restriction a$ to be the the (multi-threaded legal) subsequence of s hereditarily justified by a:*
$sa \restriction a = a$,
$sb \restriction a = (s \restriction a)b$ if a hereditarily justifies b (b has a concurrency pointer to a move in $s \restriction b$ by visibility),
$sb \restriction a = s \restriction a$ otherwise.
For well-opened $\sigma : A$, define $\sigma^\dagger : A$ as follows:
$s \in \sigma^\dagger$ if and only if for all initial moves a in s, $s \restriction a \in \sigma$.

We can now define categories $\mathcal{GS}$ and $\mathcal{GM}$ with *arenas* as objects, and multi-threaded and single-threaded strategies on $A \Rightarrow B$ as morphisms from A to B, respectively. Composition is defined $\sigma \cdot \tau = \sigma^\dagger;\tau$. (The well-opened identity strategy on A, $\mathtt{Id}_A$, is the restriction of $\mathtt{id}_A$ to well-opened sequences.)

Proposition 3.17 *$\mathcal{G_M}^\dagger$ is a category.*

Proof. We use the following facts: for any A, $\mathtt{Id}_A^\dagger = \mathtt{id}_A$, and for any well-opened σ, $\sigma^\dagger;\mathtt{Id}_B = \sigma$.
For well-opened strategies $\sigma : A \Rightarrow B, \tau : B \Rightarrow C$, $\sigma^\dagger;\tau^\dagger = (\sigma^\dagger;\tau)^\dagger$.

Moreover, the CCC structure on single-threaded games [15,4] transfers smoothly to $\mathcal{GM}$; the operation $\times$ is a cartesian product with copycat strategies π_i :

$A_1 \times A_2 \to A_i$ for $i = 1, 2$:

$$\pi_i = \{s \in W_{A_1^- \times A_2^- \Rightarrow A_i^+} \mid \forall t \sqsubseteq^{even} s.t\restriction A_i^- = t\restriction A_i^+\}.$$

Definition 3.18 *For well-opened strategies $\sigma : A \to B, \tau : A \to C$, define:* $\langle \sigma, \tau \rangle : A \to B \times C = \{s \in LM_{A \Rightarrow B \times C} \mid (s\restriction A, B \in \sigma \wedge s\restriction C = \varepsilon) \vee (s\restriction A, C \in \tau \wedge s\restriction B = \varepsilon)\}$*.*

The exponential is the function-space arena $\Rightarrow$ — there are obvious copycat strategies:
$\mathsf{App} : A \times (A \Rightarrow B) \to B$ and $\Lambda : (A \times B) \Rightarrow C \to A \Rightarrow (B \Rightarrow C)$.

Proposition 3.19 *$\mathcal{GM}, \mathbf{1}, \times, \Rightarrow$ is a cartesian closed category.*

To model recursion, we quotient $\mathcal{GM}$ by trace-equivalence (which must be preserved by composition, since each equivalence class contains the strategy which halts on all traces in that class). The result is a cpo-enriched category which is the basis for the may-testing model of ICSP.

Proposition 3.20 *Let $\mathcal{GM}_{/\approx}$ be the category in which morphisms are $\approx$-equivalence classes of strategies from $\mathcal{GM}$ (with the obvious definition of composition: $[\sigma] \cdot [\tau] = [\sigma \cdot \tau]$). Then $\mathcal{GM}_{/\approx}$ is a cartesian closed category which is* cpo-enriched *with the ordering $[\sigma] \leq [\tau]$ if $T_\sigma \subseteq T_\tau$.*

4 Semantics of ICSP

As in [1] the interpretation of *sequential composition* is by defining $[\![\Gamma \vdash M; N]\!] = \langle [\![\Gamma \vdash M]\!], \Gamma \vdash N]\!] \rangle; \mathsf{seq}$, where $\mathsf{seq} : \mathtt{comm} \times \mathtt{comm} \to \mathtt{comm}$ is a strategy which interrogates each of its arguments once, in turn, before making a response. So a typical *control thread* of play in seq is as follows.

$$\begin{array}{ccccc}
[\![\mathtt{comm}]\!] & \times & [\![\mathtt{comm}]\!] & \Rightarrow & [\![\mathtt{comm}]\!] \\
 & & & & q \\
q & & & & \\
a & & & & \\
 & & q & & \\
 & & a & & \\
 & & & & a
\end{array}$$

However, in the multi-threaded setting, we need to be able to define seq as a strategy which behaves in the same way in each control thread. More generally we wish to give an embedding of the original model of PCF in the category of single-threaded strategies into $\mathcal{GM}$.

Definition 4.1 *Given single-threaded $\sigma : A$, define the multi-threaded strategy* $\mathsf{multithread}(\sigma) : A = \{s \in LM_A | \ \forall r \sqsubseteq s.\mathsf{CT}(r) \in T_\sigma \wedge \forall t.((t \precsim s \vee s \precsim t) \Longrightarrow \mathsf{CT}(t) \in \sigma)\}$*.*

Lemma 4.2 *The operation* $\mathsf{multithread}$ *is a functor from $\mathcal{GS}$ to $\mathcal{GM}$ which preserves cartesian closed structure.*

Proof. The proof that $\mathsf{multithread}(\sigma); \mathsf{multithread}(\tau) = \mathsf{multithread}(\sigma;\tau)$ is based on Lemma 3.10. $\square$

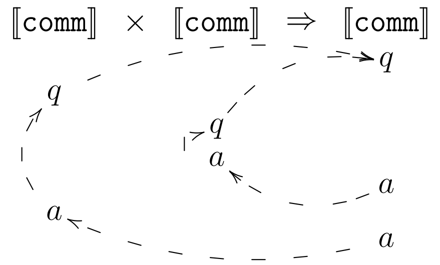
Thus we can define the interpretation of sequential composition and the conditional as the image under $\mathsf{multithread}$ of their single-threaded counterparts.

Parallel composition of strategies is by *interleaving* their responses to the initial move (preserving the concurrency pointers). Say that an *arena* is well-opened if it has an unique initial move.

Definition 4.3 *Let $s = ab_1 \ldots b_m$ and $t = ac_1 \ldots c_n$ be sequences in LM_A, where A is a well-opened arena. A* tail-interleaving *of s and t is a sequence $ar \in LM_A$ such that r is an interleaving of $b_1 \ldots b_m$ with $c_1 \ldots c_n$ which preserves justification and concurrency pointers — e.g. if b_j points to a in s then b_j points to a in ar, or if b_j points to b_i in s then b_j points to b_i in r. We shall write $s|t$ for the set of tail-interleavings of s and t.*

Proposition 4.4 *For any well-opened arena A and (pointer-blind) strategies $\sigma, \tau : A$ the parallel composition of σ and τ — $\sigma|\tau = \bigcup\{s|t \mid s \in \sigma \wedge t \in \tau\}$ — is a well-defined (pointer-blind) strategy.*

So, for instance, we have a general *parallel composition* morphism for arenas with unique initial moves, $\mathsf{para}_A : A \times A \to A = \pi^l_A|\pi^r_A$. A typical play (with concurrency pointers) of $\mathsf{para}_{[\![\mathtt{comm}]\!]}$ is as follows:


$[\![\mathtt{comm}]\!] \times [\![\mathtt{comm}]\!] \Rightarrow [\![\mathtt{comm}]\!]$
q
q
q
a
a
a
a

We define $[\![\Gamma \vdash M \| N : B]\!] = [\![\Gamma \vdash M]\!] | [\![\Gamma \vdash N]\!]$. We have the property that $(\sigma|\tau) \times \rho; \mathsf{seq} = (\sigma \times \rho; \mathsf{seq}) | (\tau \times \rho; \mathsf{seq})$ — in other words for all $M, N : \mathtt{comm}$, $[\![(M\|N); L]\!] = [\![(M; L)\|(N; L)]\!]$.

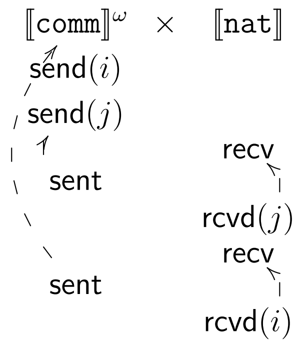
So it remains to give the semantics of locally bound channels. This is based on viewing elements of type `chan` as 'objects' defined by their 'methods' — in this case `send` and `recv`. This was suggested as an interpretation for reference types by Reynolds [18] and used to give a a functor-category semantics for idealized CSP by Brookes [7]. The interpretation described here is particularly close to the game semantics of store in Idealized Algol [1].

- We define $[\![\mathtt{chan}]\!] = [\![\mathtt{comm}]\!]^\omega \times [\![\mathtt{nat}]\!]$ (which is the same as the interpretation of the type `var` given in [1]).
- For sending messages there is a sequential and innocent strategy $\mathsf{write} : [\![\mathtt{nat}]\!] \times [\![\mathtt{comm}]\!]^\omega \to [\![\mathtt{comm}]\!]$ described in [1] which responds to the initial move by asking the question in $[\![\mathtt{nat}]\!]$, given the answer n it asks the initial question in the nth part of the product $[\![\mathtt{comm}]\!]^\omega$. When this is answered, it answers the initial question.

We define $[\![\Gamma \vdash \texttt{send}\, M\, N]\!] = \langle [\![\Gamma \vdash M]\!]; \pi_l, [\![\Gamma \vdash N]\!]\rangle; \mathsf{write}$, which is precisely the same as the interpretation of assignment in [1].

- We define $[\![\Gamma \vdash \texttt{recv}\, M]\!] = [\![\Gamma \vdash M]\!]; \pi_r$ (the interpretation of deallocation in [1]).

Thus the only part of the semantics of channels which is non-functional — and moreover the only part which differs from the game semantics of store in Idealized Algol — is the interpretation of new channel generation. This is defined by composition with a (pointer-blind) strategy $\mathsf{ccell} : [\![\texttt{nat}]\!] \times [\![\texttt{comm}]\!]^\omega$ which is similar to the cell strategy used to interpret `new` in the model of Idealized Algol [1] in the way that it causes interaction between the two read/write or send/receive components of $[\![\texttt{chan}]\!]$ but the significant difference is that communication between sending and receiving is concurrent and synchronous rather than sequential. A typical play of ccell (with concurrency pointers) is given below (the questions in the ith "send component" $[\![\texttt{comm}]\!]^\omega$ have been labelled $\mathsf{send}(i)$, and their answers as $\mathsf{sent}(i)$, the question in the "receive" component $[\![\texttt{nat}]\!]$ has been labelled recv, and its ith answer $\mathsf{rcvd}(i)$).



Informally, the behaviour of ccell can be described in the following terms. It must respond to any play in which there is both a unanswered $\mathsf{send}(i)$ question and an unanswered recv question by matching up such pairs of questions by giving the answer $\mathsf{rcvd}(i)$ to the recv question, and the answer sent to $\mathsf{send}(i)$.

Because answers can be exchanged between any pair of open `send` and `recv` moves, ccell is implicitly non-deterministic. In order to satisfy the alternation condition, the `send` and `recv` moves must always be in different threads — as one would expect, as synchronous message passing requires the sender and recipient to be in different threads. Formally, ccell can be defined as follows.

- Let the *balanced* sequences of ccell, $B_{\mathsf{ccell}} \subseteq \mathsf{ccell}$, be the least set of sequences in $LM_{\mathtt{chan}]\!]}$ containing ε and closed under $\precsim$ and the following rule: if $s \in B_{\mathsf{ccell}}$, then $s \cdot \mathsf{send}(i) \cdot \mathsf{recv} \cdot \mathsf{sent} \cdot \mathsf{rcvd}(i) \in B_{\mathsf{ccell}}$ (where sent has concurrency and justification pointers to $\mathsf{send}(i)$, and $\mathsf{rcvd}(i)$ to recv).
- Let the "waiting to send" sequences of ccell, S_{ccell}, be the least superset of B_{ccell} such that if $s \in S_{\mathsf{ccell}}$, then $s \cdot \mathsf{send}(i) \in S_{\mathsf{ccell}}$.
- Let the set of "waiting to receive" sequences, R_{ccell}, be the least superset of B_{ccell} such that if $s \in R_{\mathsf{ccell}}$, then $s \cdot \mathsf{recv} \in R_{\mathsf{ccell}}$.

Now let $\mathsf{ccell} = \{s \in LM_{[\![\mathsf{chan}]\!]} \mid \exists t.s \precsim t \wedge (t \in S_{\mathsf{ccell}} \vee t \in R_{\mathsf{ccell}})\}$ and define:

$$[\![\Gamma \vdash \mathtt{newchan}\, M : B]\!] = ([\![\Gamma \vdash M : \mathsf{chan} \Rightarrow B]\!] \times \mathsf{ccell}); \mathsf{App}.$$

4.1 Soundness of the semantics

We shall say that a strategy on $[\![\mathtt{comm}]\!]$ is *may convergent* if it can answer the initial question at least once, and *must convergent* if it always gives some response (i.e. does not wait) in response to Opponent's initial move.

Definition 4.5 *For a strategy $\sigma : [\![\mathtt{comm}]\!]$, define $\sigma \downarrow^{may}$ if $qa \in T_\sigma$ and $\sigma \downarrow^{must}$ if $q \notin \sigma$.*

Soundness of the interpretation — i.e. correspondence between the notions of may and must convergence in the operational and denotational semantics — can now be established. First, commutativity and associativity of parallel composition, and commutativity of new-channel declaration means that the denotation of a configuration can be defined without ambiguity.

Definition 4.6 *For a configuration $C = M_1 : \mathtt{comm}, \ldots, M_n : \mathtt{comm}$ such that $Ch(C) = c_1, \ldots, c_k$, define $[\![C]\!] = [\![\mathtt{newchan}\, \lambda c_1. \ldots \mathtt{newchan}\, \lambda c_k.M_1 \| \ldots \| M_n]\!]$.*

Say that a configuration is *converged* if it has the form $C, \mathtt{skip}$, and *deadlocked* if it is not converged and cannot be further reduced. So if C is converged, then $[\![C]\!] \downarrow^{may}$ and $[\![C]\!] \downarrow^{must}$ and if C is deadlocked then $[\![C]\!] \not\downarrow^{may}$ and $[\![C]\!] \not\downarrow^{must}$. Soundness of the semantics with respect to may-testing thus boils down to the following fact.

Proposition 4.7 *If $C \longrightarrow C'$ then $T_{[\![C']\!]} \subseteq T_{[\![C]\!]}$.*

Proof. The proof is based on the standard properties of a cpo-enriched cartesian closed category together with the following lemma.

Lemma 4.8 $[\![E[M \| N] = [\![E[M] \| E[N]]\!]$
$[\![(\mathtt{newchan}\ \lambda c.M) \| N]\!] = [\![\mathtt{newchan}\, \lambda c.(M \| N)]\!]$ $(c \notin FV(N)$
$[\![(\mathtt{newchan} \lambda c.M); N]\!] = [\![\mathtt{newchan}\, \lambda c.(M; N)]\!]$, $(c \notin FV(N))$
$[\![\mathtt{newchan}\, \lambda c.E[\mathtt{send}\ c\ v] \| E'[\mathtt{recv}\ c] \| M]\!] \subseteq [\![\mathtt{newchan}\, \lambda c.E[\mathtt{skip}] \| E'[v] \| M]\!]$.

Soundness of the finitary fragement with respect to must-testing is expressed by the following proposition.

Proposition 4.9 *For any non-converged and non-deadlocked configuration C of* finitary *ICSP, $[\![C]\!] \downarrow^{must}$ if $[\![C']\!] \downarrow^{must}$ for all C' such that $C \longrightarrow C'$.*

The proof is based on the equalities/inclusions given in Lemma 4.8 together with the following lemma, (established by analysis of the strategy ccell).

Lemma 4.10 *For terms $M_1, M_2, \ldots M_n$, define $\|_{i \le n} M_i = M_1 \| M_2 \| \ldots \| M_n$. Suppose we have terms $M^i_j = E^i_j[\mathtt{send}\, a_i\, v^i_j] : j \le m_i$ and $N^i_k = D^i_k[\mathtt{recv}\, a_i] : k \le l_i$, for $i \le n$, where each $E^i_j[\cdot]$ and $D^i_k[\cdot]$ is an evaluation context. Then $\|_{i \le n}((\|_{j \le m_i} M^i_j) \| (\|_{k \le l_i} N^i_k)) =$*

$$\bigcup_{I\leq n,J\leq m_I,K\leq l_J} \|_{i\leq n}((\|_{j\leq m_i}M(I,J))^i_j)\|(\|_{k\leq l_i}N(I,J,K)^i_k))$$

where $M(I,J)^i_j = E^i_j[\texttt{skip}]$, *if* $i = I$ *and* $j = J$
$M(I,J)^i_j = M^i_j$ *otherwise,*
and $N(I,J,K)^i_k = E^i_k[v^I_J]$ *if* $i = I$ *and* $k = K$,
$N(I,J,K)^i_k = N^i_k$, *otherwise.*

We can now prove Proposition 4.9 by induction on reduction-sequence length. Given C such that if $C \longrightarrow C'$ then $[\![C]\!] \downarrow^{must}$, suppose that there is some reduction $C \longrightarrow C'$ which is not an instance of the communication rule. Then by Lemma 4.8, $[\![C]\!] = [\![C']\!]$ and $[\![C]\!] \downarrow^{must}$ by hypothesis. On the other hand, if there is no reduction which is not an instance of communication, then all threads have the form $E[\texttt{send}\ c\ n]$ or $E[\texttt{send}\ c\ n]$ and hence by Lemma 4.10, $q \notin [\![C]\!] = \bigcup_{C':C\longrightarrow C'}[\![C']\!]$ as required.

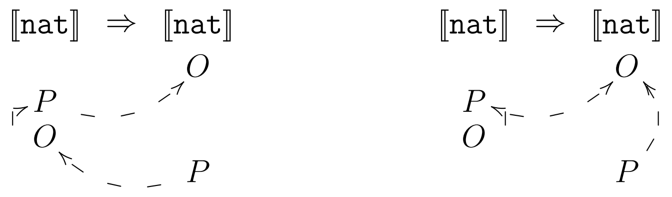
Proposition 4.11 *If* M : $\texttt{comm}$ *is a program of ICSP then* $M \Downarrow^{may}$ *if and only if* $[\![M]\!] \downarrow^{may}$ *and if* M *is a program of finitary ICSP then* $M \Downarrow^{must}$ *if and only if* $[\![M]\!] \downarrow^{must}$.

Proof. $M \Downarrow^{may}$ implies $[\![M]\!] \downarrow^{may}$ and $M \Downarrow^{must}$ implies $[\![M]\!] \downarrow^{must}$ by induction on reduction. For terms of finitary ICSP, the converse follows from the fact that evaluation always terminates (Proposition 2.4). To show that $[\![M]\!] \Downarrow^{may}$, then $M \Downarrow^{may}$ for terms containing the $\mathbf{Y}$ combinator, we use the fact that $[\![C[\mathbf{Y}M]\!] \Downarrow^{may}$ if and only if there is some finite k such that $[\![C[M^k]]\!] \Downarrow^{may}$ to reduce to the result for finitary ICSP. (Where M^k is the kth approximant to $\mathbf{Y}M$, i.e. $M^0 = \Omega$, $M^{k+1} = M\ (M^k$.) □

5 Definability and Full Abstraction

Our model of ICSP is not fully abstract for may or must equivalence. This is because the presence of control pointers allows distinctions to be made between strategies which do not correspond to distinctions in the language. In this section, we will show how the model can be cut down by eliminating these distinctions, and hence achieve a fully abstract result by proving that every compact strategy is definable.

The constraint which we require says that Player cannot (directly) observe the concurrency pointers from O-moves. This corresponds to the fact that in ICSP it is not directly observable in which *location* (i.e. which thread) computation is occurring. For example, $\lambda x.x : \texttt{nat} \Rightarrow \texttt{nat}$ is observationally equivalent to $\texttt{newchan}\,\lambda c.\lambda x.((\texttt{send}\,c\,x;\texttt{nil})\|\texttt{recv}\,c)$, even though they are interpreted as distinct strategies (and hence there is a strategy on $[\![(\texttt{nat} \Rightarrow \texttt{nat}) \Rightarrow \texttt{comm}]\!]$ which can distinguish them).



$[\![\lambda x.x]\!]$ $\quad$ $[\![\texttt{newchan}\,\lambda c.\lambda x.((\texttt{send}\,c\,x;\texttt{nil})\|\texttt{recv}\,c)]\!]$.

Definition 5.1 *Let $\sim_O$ to be the least equivalence relation on multi-threaded* legal *sequences closed under the condition:*
$sa_1t_1a_2t_2br \sim_O sa_1t_1a_2t_2br$, if b is an O-move with a concurrency pointer to a_1 in the first sequence, and to a_2 in the second sequence, and the sequences are otherwise identical.
A strategy σ is (concurrency) pointer-blind *if it is closed with respect to $\sim_O$ — i.e. $s \in \sigma$ and $s \sim_O t$ implies $t \in \sigma$.*

It is straightforward to show that the pointer-blind strategies form a subcategory of $\mathcal{GM}$ which contains the denotations of all of the terms of ICSP. Pointer-blindness induces the following "intrinsic equivalences" between strategies which are indistinguishable by a pointer-blind observer.

Definition 5.2 *Define the equivalences $\equiv^{may}, \equiv^{M\&M}$ on strategies $\sigma, \tau : A$: $\sigma \equiv^{may} \tau$ if for all $\rho : A \rightarrow [\![\texttt{comm}]\!]$, $\sigma;\rho \downarrow^{may}$ if and only if $\tau;\rho \downarrow^{may}$. $\sigma \equiv^{M\&M} \tau$ if if for all $\rho : A \rightarrow [\![\texttt{comm}]\!]$, $\sigma;\rho \downarrow^{must}$ if and only if $\tau;\rho \downarrow^{must}$.*

We shall show that every compact pointer-blind strategies on a (channel-free) type-object is definable, up to equivalence, as a term of ICSP, from which it folows by a standard argument that two terms are may (or may-and-must) equivalent if and only if their denotations are may (may-and-must) equivalent. First, however, note that we can characterize $\equiv^{may}$ and $\equiv^{M\&M}$ directly in terms ofa relationship on traces which is a complement of $\sim_O$.

Definition 5.3 *Let $\sim_P$ be the least equivalence relation on multi-threaded sequences such that $sa_1t_1a_2t_2br \sim_P sa_1t_1a_2t_2br$ if b is a Player move with a concurrency pointer to a_1 in the first sequence and to a_2 in the second, and the sequences are otherwise identical.*
Define the equivalence $\sim$ on sets of sequences: $\sigma \sim \tau$ if $\forall s \in \sigma.\exists t \in \tau.s \sim t$ and $\forall t \in \tau.\exists s \in \sigma.s \sim \tau$.

Proposition 5.4 *$\sigma \equiv^{may} \tau$ if $T_\sigma \sim T_\tau$ and $\sigma \equiv^{M\&M} \tau$ if $\sigma \sim \tau$.*

Proof. Suppose that $T_\sigma \not\sim T_\tau$, then w.l.o.g. there exists $s \in T_\sigma$ such that for all $t \in T_\tau$, $t \not\sim_P s$. Assume that s is maximal (with this property) with respect to $\precsim$. Let $\rho : A \rightarrow [\![\texttt{comm}]\!]$ be the strategy generated by the prefixes of $qsa \in A \Rightarrow [\![\texttt{comm}]\!]$ (where q is the initial question in $[\![\texttt{comm}]\!]$ and a is its answer) — i.e. ρ is generated by $\{t \in LM_{[\![T \Rightarrow \texttt{comm}]\!]} \mid \exists r \sqsubseteq qsa.t \precsim r\}$. Then $qa \in T_{\sigma;\rho}$ and $qa \notin T_{\tau;\rho}$, since if $qs'a \precsim qsa$, then $s \precsim s'$ and hence $s' \notin \tau$ by maximality of s with respect to $\precsim$. Hence $\sigma \not\equiv^{may} \tau$ as required.

Suppose $\sigma \not\sim \tau$, then either $T_\sigma \not\sim T_\tau$ and hence $\sigma \not\equiv^{M\&M} \tau$ by the argument above, or else $T_\sigma \sim T_\tau$. Then (w.l.o.g.) we can find a trace on which σ diverges, but τ always makes another move — i.e. there exists $s \in \sigma$ such that $s \notin \tau$. Let $R \subseteq WM_{[\![T]\!]}$ be the set of minimal length sequences r such that $r \in \tau$ and $s \sqsubseteq r$. Then let ρ be the strategy which converges when it encounters any trace from R which is distinct from s, and diverges otherwise — i.e. ρ is generated by the set of sequences $\{t \in LM_{A \Rightarrow [\![\mathtt{comm}]\!]} \mid t \precsim qs \vee \exists r \in R.\exists u \sqsubseteq r.u \neq s \wedge rt \precsim qua\}$. Then $q \in \sigma$, but $q \notin \tau$, so $\sigma \not\equiv^{M\&M} \tau$ as required.

To show that $\equiv^{may}$ and $\equiv^{M\&M}$ correspond to $\simeq^{may}$ and $\simeq^{M\&M}$ it suffices to show that all finitary pointer-blind strategies are definable (up to $\sim_P$) as ICSP terms. What is a finitary strategy ? All strategies on type-objects contain infinitely many sequences, so finiteness is defined in terms of the smallest set required to generate these sequences.

Definition 5.5 *A* generator *for a strategy σ is a set of sequences $S \subseteq \sigma$ such that for all $t \in \sigma$ there exists $sr \precsim t$ such that $s \in S$ and r contains only O-moves. We say that σ is* finitary *if it has a finite generator S, and that it is bounded by k if the length of all sequences in S is less than k.*

Finitary strategies suffice to distinguish between denotations of ICSP terms. With respect to may-equivalence, this follows from the observation that the distinguishing strategy ρ defined in the first part of the proof of Proposition 5.4 has a finitary generator. With respect to must-equivalence, finitary strategies suffice to distinguish *finite-branching* strategies.

Definition 5.6 *A strategy σ is finite-branching if for each $s \in T_\sigma$, the set of minimal length t such that $t \in \sigma$ and $s \sqsubseteq t$ is finite.*

It is straightforward to show that all terms of finitary ICSP have finite-branching denotations. Clearly, if σ and τ are finite-branching then the distinguishing strategy ρ defined in the first part of the proof of Proposition 5.4 has a finitary generator.

Lemma 5.7 *For any strategies $\sigma, \tau : A$, $\sigma \equiv^{may} \tau$ if and only if for all finitary $\rho : A \to [\![\mathtt{comm}]\!]$, $\sigma;\rho \downarrow^{may}$ if and only if $\tau;\rho \downarrow^{may}$. And if σ and τ are finite-branching $\sigma \equiv^{M\&M} \tau$ if if for all finitary $\rho : A \to [\![\mathtt{comm}]\!]$, $\sigma;\rho \downarrow^{must}$ if and only if $\tau;\rho \downarrow^{must}$.*

The first part of the proof of definability of finitary pointer-blind strategies is a characterisation of the elements which are definable *without* parallel composition and message passing — in effect, the image of the original HO model of PCF under the embedding multithread. This requires the notion of *sequentiality* for multi-threaded strategies — a sequential strategy is one which never spawns new threads of control.

Definition 5.8 *A multi-threaded sequence s is* Player sequential *if every O-move in s is the target of at most one concurrency pointer: i.e. if $tarc, tar'c' \sqsubseteq s$ where c, c' are P-moves with concurrency pointers to a, then $rc = r'c'$.*
A strategy σ is sequential *if every $s \in \sigma$ is Player sequential.*

The *visibility* and *innocence* conditions on strategies from [12] can be generalised to the multi-threaded sequential strategies by applying the *view*-function [12] to each control thread (which is consistent with the intuition that the view represents a certain kind of "relevant history of the sequence" from Player's perspective).

Definition 5.9 *The Player view $\ulcorner s \urcorner$ of a justified sequence s is a sequence with justification pointers, defined inductively on the length of s, as follows:*
$\ulcorner \varepsilon \urcorner = \varepsilon$,
$\ulcorner sa \urcorner = \ulcorner s \urcorner a$, *if a is a Player move,*
$\ulcorner sa \urcorner = a$, *if a is an initial Opponent move,*
$\ulcorner sa \cdot tb \urcorner = \ulcorner s \urcorner ab$, *if b is an Opponent move justified by a.*

Definition 5.10 *A strategy σ obeys the* visibility *condition if for every $s \in T_\sigma$, $\ulcorner \mathsf{CT}(s) \urcorner$ is a well-formed justified sequence.*

A (sequential) strategy is *innocent* if its response in each control-thread depends only on the P-view of that thread. For a multi-threaded strategy σ, let $\ulcorner \sigma \urcorner = \{\ulcorner \mathsf{CT}(s) \urcorner \mid s \in \sigma\}$. (If σ is single-threaded, $\ulcorner \sigma \urcorner = \{\ulcorner s \urcorner \mid s \in \sigma\}$.)

Definition 5.11 *A* sequential *strategy σ is* innocent *(and deterministic) if it satisfies the visibility condition and $\ulcorner \sigma \urcorner$ is evenly branching: i.e. if $s, t \in \ulcorner \sigma \urcorner$ and $s \sqcap t$ is odd-length then $s = t$.*

Note that although it is *not* the case in general that $\mathsf{multithread}(\sigma)$ is a pointer-blind strategy, that this does hold if σ is innocent; if an O-move is in the $P-$view, then the preceding move in the view is its justifier, which must be no later than the target of its concurrency pointer.

Definition 5.12 *From a multi-threaded innocent strategy σ, define the single-threaded innocent strategy* $\mathsf{threads}(\sigma) = \{\mathsf{CT}(s) \mid s \in \sigma\}$.

The following lemma follows from Lemma 3.10.

Lemma 5.13 *For single-threaded $\sigma : A \to B, \tau : B \to C$,* $\mathsf{threads}(\sigma); \mathsf{threads}(\tau) = \mathsf{threads}(\sigma; \tau)$.

Lemma 5.14 *For any sequential and innocent σ,* $\mathsf{multithread}(\mathsf{threads}(\sigma)) = \sigma$*, and for all single-threaded and innocent τ,* $\mathsf{threads}(\mathsf{multithread}(\tau)) = \tau$.

Hence $\mathsf{threads}$ is an isomorphism between the multi-threaded, innocent and sequential strategies, and the innocent and single-threaded strategies.

Proposition 5.15 *The sequential and innocent strategies form a subcategory of $\mathcal{GM}$ which is isomorphic to the category of single-threaded games and innocent strategies.*

The definability result for ICSP - $\{\|, \mathtt{newchan}\}$ requires one further constraint — the *bracketing condition* on strategies, which requires that each Player-answer must be justified by the most recent open question. (This is the only point at which the question/answer distinction is used — it is not required to define the fully abstract model, only in the proof of definability via the

Hyland-Ong model.)

Definition 5.16 *Define the* pending questions *(a set containing at most one move) of a justified sequence as follows:*
$\mathsf{pending}(\varepsilon) = \{\}$,
$\mathsf{pending}(sb) = \{b\}$, *if* $\lambda(b) = Q$,
$\mathsf{pending}(sbtc) = \mathsf{pending}(s)$ *if* $\lambda(c) = A$ *and* c *is justified by* b.
A (single-threaded) sequence s *is well-bracketed if for all* $ratb \sqsubseteq s$, *if* $\lambda(b) = A$ *and* b *is justified by* a, *then* $\mathsf{pending}(sat) = \{a\}$. *A strategy* σ *is* well-bracketed *if for every* $s \in T_\sigma$, $\ulcorner s \urcorner$ *is well-bracketed.*

The isomorphism between innocent, sequential strategies, and innocent, single-threaded strategies extends to the well-bracketed strategies. This gives a proof of the following definability result for innocent, sequential strategies, which is a minor adaptation of the definability theorem for PCF [12], and precisely analogous to definability in Idealized Algol without bad variables [1].

Proposition 5.17 *If* Γ *is a* $\mathtt{chan}$*-free context and* T *is a* $\mathtt{chan}$*-free type, and* $\sigma : [\![\Gamma, \mathtt{chan}^k]\!] \rightarrow [\![T]\!]$ *is a sequential and innocent strategy such that* $\ulcorner \sigma \urcorner$ *is finite, then there is a term* $\Gamma, \mathtt{chan}^k \vdash M_\sigma : T$ *of ICSP -* $\{\|, \mathtt{newchan}\}$ *such that* $\sigma = [\![M_\sigma]\!]$.

The astute reader will have observed that in fact all denotations of terms of ICSP obey both the visibility and well-bracketing conditions. However, it is not necessary to impose these conditions to get a fully abstract model; as we shall show, every pointer-blind strategy is equivalent (with respect to $\sim$) to one which obeys both conditions. In other words, we can simulate sequential side-effects such as first-class continuations and reference cells using concurrent ones (these results correspond to syntactic encodings of these features using threads and messages).

Proposition 5.18 *For every pointer-blind strategy* $\sigma : A$, *there exists a well-bracketed (pointer-blind) strategy* $\widehat{\sigma}$ *such that* $\sigma \sim \widehat{\sigma}$.

To prove this proposition, we use a result proved in [13] — all violations of the bracketing condition can be expressed in terms of the single-threaded strategy Peirce by *factorization* — together with the fact that there is a well-bracketed multi-threaded strategy $\widehat{\mathsf{Peirce}}$ such that $\mathsf{Peirce} \sim \widehat{\mathsf{Peirce}}$.

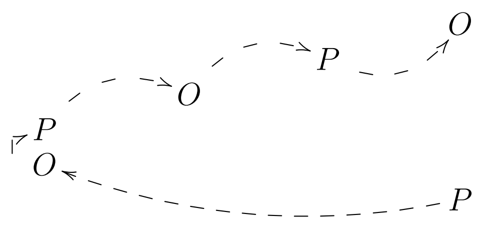
Lemma 5.19 *There is a strategy* $\mathsf{Peirce} : [\![((\mathtt{nat} \Rightarrow \mathtt{nat}) \Rightarrow \mathtt{nat}) \Rightarrow \mathtt{nat}]\!]$ *such that for all finitary strategies* $\sigma : A$ *there is a* well-bracketed *strategy* $\widetilde{\sigma} : ((([\![\mathtt{nat} \Rightarrow \mathtt{nat}) \Rightarrow \mathtt{nat}) \Rightarrow \mathtt{nat}]\!] \rightarrow A$ *such that* $\mathsf{Peirce}; \widetilde{\sigma} = \sigma$.

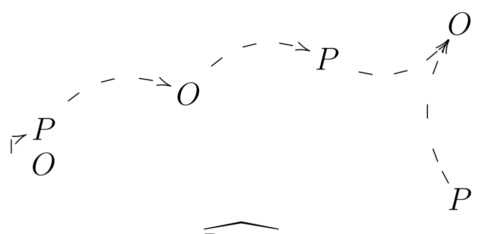
Proof. Peirce is formally defined in [13]; a key "typical play" is shown below. The extension of the factorization proof from single-threaded strategies to multi-threaded strategies is straightforward. □

Lemma 5.20 *There is a well-bracketed (non-sequential) strategy* $\widehat{\mathsf{Peirce}} = [\![\mathtt{newchan}\, \lambda c.\lambda f.(f\ \lambda x.(\mathtt{send}\, c\, x); \mathtt{nil}) \| \mathsf{recv} x]\!]$ *such that* $\widehat{\mathsf{Peirce}} \sim \mathsf{Peirce}$.

This is proved by comparing the plays of Peirce and $\widehat{\mathsf{Peirce}}$, for example:

$$((\llbracket \mathtt{nat} \rrbracket \Rightarrow \llbracket \mathtt{nat} \rrbracket) \Rightarrow \llbracket \mathtt{nat} \rrbracket) \Rightarrow \llbracket \mathtt{nat} \rrbracket \qquad ((\llbracket \mathtt{nat} \rrbracket \Rightarrow \llbracket \mathtt{nat} \rrbracket) \Rightarrow \llbracket \mathtt{nat} \rrbracket) \Rightarrow \llbracket \mathtt{nat} \rrbracket$$





Peirce $\widehat{\mathsf{Peirce}}$.

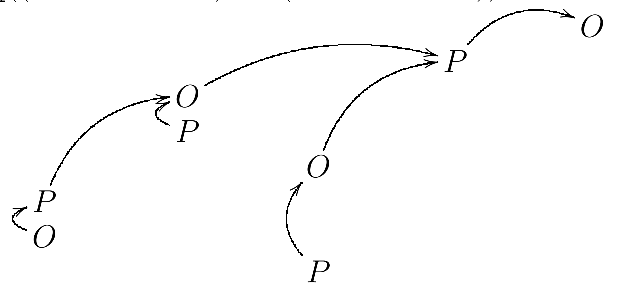
To prove Proposition 5.18 let $\widehat{\sigma} = \widehat{\mathsf{Peirce}}; \widetilde{\sigma} \sim \mathsf{Peirce}; \sigma = \sigma$.

Similar results apply with respect to the *visibility* condition.

Proposition 5.21 *There is a sequential strategy* $\mathsf{ucell} : \llbracket((\mathtt{comm} \Rightarrow \mathtt{comm}) \Rightarrow (\mathtt{comm} \Rightarrow \mathtt{comm})) \Rightarrow \mathtt{comm}\rrbracket$ *such that for all finitary strategies* $\sigma : A$ *there is a strategy* $\widetilde{\sigma} : \llbracket((\mathtt{comm} \Rightarrow \mathtt{comm}) \Rightarrow (\mathtt{comm} \Rightarrow \mathtt{comm})) \Rightarrow \mathtt{comm}\rrbracket \Rightarrow A$ *which satisfies the visibility condition, such that* $\mathsf{ucell}; \widetilde{\sigma} = \sigma$*.*

In essence, ucell is a curried version of the cell strategy which is used to factorize the visibility condition in [4] (except that it can store only one reference). A typical control thread of play runs as follows (solid arrows represent *justification* pointers).

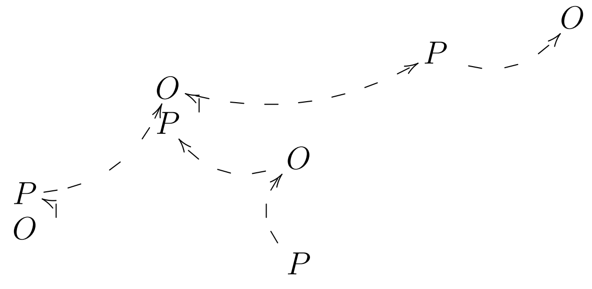
$$\llbracket((\mathtt{comm} \Rightarrow \mathtt{comm}) \Rightarrow (\mathtt{comm} \Rightarrow \mathtt{comm})) \Rightarrow \mathtt{comm}\rrbracket$$



Lemma 5.22 *There is a multi-threaded strategy* $\widehat{\mathsf{ucell}}$ *such that* $\widehat{\mathsf{ucell}} \sim \mathsf{ucell}$ *and* $\widehat{\mathsf{ucell}}$ *satisfies visibility.*

$\widetilde{\mathsf{ucell}} = \llbracket \lambda f.\mathtt{newchan}\, \lambda c.(f\ (\lambda x.\mathtt{recv}\, c; x; \mathsf{recv} c; \mathtt{nil} \| \mathtt{skip}))\ ((\mathtt{send}\, c\, 0); \mathtt{send}\, c\, 0)\rrbracket$
A typical play of $\widetilde{\mathsf{ucell}}$ (with concurrency pointers) is as follows.

$$\llbracket((\mathtt{comm} \Rightarrow \mathtt{comm}) \Rightarrow (\mathtt{comm} \Rightarrow \mathtt{comm})) \Rightarrow \mathtt{comm}\rrbracket$$



To prove Proposition 5.21, let $\widehat{\sigma} = \widehat{\mathsf{ucell}}; \widetilde{\sigma} \sim \mathsf{ucell}; \widetilde{\sigma} = \sigma$.

5.1 Factorization

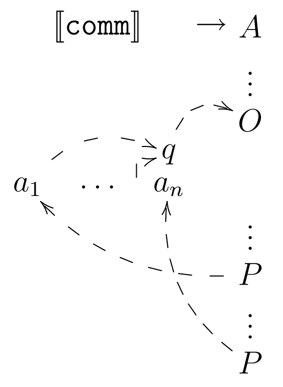
To prove the full abstraction results it remains to show that every finitary strategy satisfying pointer-blindness, well-bracketing and visibility can be reduced to the composition of an innocent and sequential strategy with a finite number of ccells and parallel compositions. First, a factorization of each finitary multi-threaded strategy into a sequential strategy plus parallel composition.

Definition 5.23 *Given $sa \sqsubseteq t \in LM_A$, define* $\mathsf{branches}(sa, t)$ *to be the number of moves in t which point to a. For finitary σ, let*
$\mathsf{branches}(\sigma) = \max(\{\mathsf{branches}(sa, t) \mid sa \sqsubseteq t \in T_\sigma \wedge \lambda^{OP}(a) = O\})$.

Thus σ is sequential if and only if $\mathsf{branches}(\sigma) \leq 1$.

Proposition 5.24 *Let $\sigma : A$ be a finitary strategy such that* $\mathsf{branches}(\sigma) = n$. *Then there exists a finitary sequential strategy* $\mathsf{sq}(\sigma) : [\![\mathtt{comm}]\!] \rightarrow A$ *such that:* $[\![\|_{i \leq n}\mathtt{skip}]\!]; \mathsf{sq}(\sigma) = \sigma$.

Proof. The idea behind the factorization is simple. In response to each O-move a in A, $\mathsf{sq}(\sigma)$ makes a move with a concurrency pointer to a in $\mathtt{comm}$, prompting Opponent (playing as $[\![\|_{i \leq n}\mathtt{skip}]\!]$) to give n answers. Then $\mathsf{sq}(\sigma)$ plays as σ, except that where σ plays a move with a pointer a, $\mathsf{sq}(\sigma)$ plays the same move with a pointer to a fresh instance of one of these answers. For example, a factorized play run as follows:



Let q be the initial question in $[\![\mathtt{comm}]\!]$ and $a_1, a_2, \ldots a_n$ its answers. Suppose that $\forall sa \sqsubseteq t.\mathsf{branches}(sa, t) \leq n$.
Define a Player-sequential sequence $\mathsf{sq}(t)$, by the following induction:
$\mathsf{sq}(\varepsilon) = \varepsilon$,
$\mathsf{sq}(sb) = \mathsf{sq}(s)bqa_1a_2 \ldots a_n$, if b is an O-move,
$\mathsf{sq}(sbtc) = \mathsf{sq}(sbt)\widehat{c}$ (where $\widehat{c}$ points to a_i), if c is a P-move pointing to b, and $\mathsf{branches}(sb, sbtc) = i$.
Thus $\mathsf{sq}(\_)$ is a map from LM_A to $LM_{[\![\mathtt{comm}]\!] \rightarrow A}$ with the following properties: $\mathsf{sq}(s)$ is player-sequential, $\mathsf{sq}(s) \upharpoonright A = s$ and $\mathsf{sq}(s) \upharpoonright [\![\mathtt{comm}]\!] \in [\![\|_{i \leq n}\mathtt{skip}]\!]$. Let $\mathsf{sq}(\sigma) = \{t \in LM_{[\![\mathtt{comm}]\!] \Rightarrow A} \mid \exists s \in \sigma.t \precsim \mathsf{sq}(s)\}$.

The factorization of sequential strategies on into innocent strategies via the $\mathtt{chan}$ type and ccell strategy is more complex. It resembles the factorization of innocence via the cell strategy [1] in that it uses ccell to encode information about the entire history of the play, but it also uses synchronization in a fundamental way *and* incorporates the factorization of non-determinism via an "oracle" given in [8]. The factorized strategy runs two threads in parallel, a "master strategy" which operates solely by sending and receiving messages in a context of channels; it observes Opponent moves and dictates Player moves in the main arena via communication with a (generalized) "slave strategy" which moves back and forth between the channels and the main arena, reporting on the progress of play in the latter to the master strategy and executing its commands.
For a finitary strategy σ let $\mathsf{responses}(\sigma)$ be the greatest number of different P-moves which σ can give in response to one position — i.e. $\mathsf{responses}(\sigma) = \max\{|\{sa \ : \ sa \in T_\sigma \wedge \lambda^{OP}(a) = P\}| \ : \ s \in T_\sigma\}$.

Proposition 5.25 *For each arena A and $k \in \omega$ there exists a finitary, sequential and innocent (deterministic) strategy* $\mathsf{slave}_k : \mathtt{nat} \times \mathtt{nat} \times \mathtt{chan}^{k+2} \rightarrow A$ *such that if $\sigma : A$ is a finitary strategy bounded by k and* $\mathsf{responses}(\sigma) = n$, *then there is an innocent strategy* $\mathsf{mas}(\sigma) : (\mathtt{nat} \times \mathtt{nat}) \times \mathtt{chan}^{k+2} \times \mathtt{nat} \rightarrow A$ *and such that* $\mathtt{oracle}_n \times [\![0\|1]\!] \times \mathsf{ccell}^{k+2}; (\mathsf{mas}(\sigma)|\mathsf{slave}_k) = \sigma$.

Proof. Factorized plays proceed as follows. The slave responds to each Opponent move a in A by trying to send an encoding of it (and its view) on channel 1. The master maintains a single open receive question on channel 1, allowing it to learn the entire history of play in A (up to $\precsim$ and $\sim_O$). Once the master and slave have concluded a successful communication on channel 1, the master sends the slave a number $2 < i \leq k+2$ on channel 2. This number is the name of a private channel, in which the slave immediately plays a receive move, waiting for communication from the master. When the master has observed that play in A has taken place to which σ would respond with one of the P-moves $b_0, b_2, \ldots b_m$, he splits the thread of control using the parallel composition $[\![0\|1]\!]$, in a special case of the sequentialization factorization of Proposition 5.24. In one of the threads thus created, he maintains the surveillance of channel 1 by repeating the receive question. In the other, he uses the oracle to generate $j \leq m$ non-deterministically. If the move b_j has a pointer to a_i the master then sends an encoding of b_j on channel i which is received by the waiting slave, who plays the move b_j with a pointer to a_i as required.

The factorization theorems together with Proposition 5.17 yield the following definability result.

Corollary 5.26 *Every finitary (pointer-blind) strategy σ over a channel-free type-object $[\![T]\!]$ is definable (up to $\sim$) as an ICSP term $M_\sigma : T$.*

The definability result means that the "intrinsic equivalence" on strategies corresponds to observational equivalence. Thus we have a fully abstract model of ICSP via the standard argument [6,12].

Theorem 5.27 *Let T be a channel-free type. For all closed terms $M, N : T$ of ICSP, $M \simeq^{may} N$ if and only if $[\![M]\!] \equiv^{may} [\![N]\!]$ if and only if $T_{[\![M]\!]} \sim T_{[\![N]\!]}$. For all closed terms $M, N : T$ of finitary ICSP, $M \simeq^{M\&M} N$ if and only if $[\![M]\!] \equiv^{M\&M} [\![N]\!]$ if and only if $[\![M]\!] \sim [\![N]\!]$.*

6 Conclusions

The semantics of ICSP described here can be considered a first step in an attempt to describe concurrency more generally using game semantics. A first priority must be to extend the full abstraction result, either to may-and-must testing for recursively defined process or to a bisimulation equivalence It seems clear that to do so requires the incorporation of the reductions of the operational semantics into the games model. This is certainly possible; one can introduce "silent moves" which belong to neither Player nor Opponent and correspond directly to reductions. It seems likely, however, that this extra "syntactic" element may make the denotational semantics more difficult to reason about.

Another desirable development would be an extension of the full abstraction result to types including channels. Recent work by McCusker on the game semantics of Idealized Algol without bad variables suggests that this is possible using an ordering on legal plays, although in the concurrent case it seems rather more complicated.

Variations on the language itself can be considered. For instance, asynchronous message passing as in Brookes' model [7]. Or we could consider a call-by-value language with thread-identifiers and passing of functions as messages (a kind of "core CML" [17]). It is straightforward to give a call-by-value semantics in the style described here (either directly, or by using the $\mathsf{Fam}(\mathcal{C})$ construction [2]), and to extend the interpretation of message passing to higher-order.

As we have described a framework for modelling concurrency using HO-games, there is plenty of scope for more general developments. Varying the definition of the preorder $\precsim$ should yield the opportunity to model concurrency features which give different powers to syncronize events across threads. The relationship with the games semantics of sequential effects such as control and state offers the chance to study the interaction between these features and concurrent ones, including some expressiveness isssues which have been hinted at here.

References

[1] S. Abramsky and G. McCusker. Linearity, Sharing and State: a fully abstract game semantics for Idealized Algol with active expressions. In P.W. O'Hearn and R. Tennent, editors, *Algol-like languages*. Birkhauser, 1997.

[2] S. Abramsky and G. McCusker. Call-by-value games. In M. Neilsen and W. Thomas, editors, *Computer Science Logic:* 11^{th} *Annual workshop proceedings*, LNCS, pages 1–17. Springer-Verlag, 1998.

[3] S. Abramsky and P.-A. Mellies. Concurrent games and full completeness. In *Proceedings of the 14th annual Symposium on Logic In Computer Science, LICS '99*, 1999.

[4] S. Abramsky, K. Honda, G. McCusker. A fully abstract games semantics for general references. In *Proceedings of the 13th Annual Symposium on Logic In Computer Science, LICS '98*, 1998.

[5] S. Abramsky, R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59:543–574, 1994.

[6] S. Abramsky, R. Jagadeesan and P. Malacaria. Full abstraction for PCF. Accepted for publication in Information and Computation, 1996.

[7] S. Brookes. Idealized CSP: Combining procedures with communicating processes. In *Proceedings of MFPS '97*, Electronic notes in Theoretical Computer Science. Elsevier-North Holland, 1997.

[8] R. Harmer and G. McCusker. A fully abstract games semantics for finite non-determinism. In *Proceedings of the Fourteenth Annual Symposium on Logic in Computer Science, LICS '99*. IEEE Press, 1998.

[9] M. Hennessy, and E. Ashcroft. A mathematical semantics for a non-deterministic typed λ-calculus. *Theoretical Computer Science*, 11:227–245, 1980.

[10] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[11] K. Honda and N. Yoshida. Game theoretic analysis of call-by-value computation. In *Proceedings of 24th International Colloquium on Automata, Languages and Programming*, volume 1256 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[12] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III, 1995. To appear in Theoretical Computer Science.

[13] J. Laird. Full abstraction for functional languages with control. In *Proceedings of the Twelfth International Symposium on Logic In Computer Science, LICS '97*, 1997.

[14] J. Laird. A fully abstract game semantics of local exceptions. In *Proceedings of the Sixteenth International Symposium on Logic In Computer Science, LICS '01*, 2001. To appear.

[15] G. McCusker. *Games and full abstraction for a functional metalanguage with recursive types*. PhD thesis, Imperial College London, 1996.

[16] H. Nickau. Hereditarily sequential functionals. In *Proceedings of the Symposium on Logical Foundations of Computer Science:Logic at St. Petersburg*, LNCS. Springer-Verlag, 1994.

[17] J. Reppy. *Higher Order Concurrency*. PhD thesis, Cornell University, 1992.

[18] J. Reynolds. Syntactic control of interference. In *Conf. Record 5th ACM Symposium on Principles of Programming Languages*, pages 39–46, 1978.

[19] J. Reynolds. The essence of Algol. In *Algorithmic Languages*, pages 345–372. North Holland, 1981.