Available online at www.sciencedirect.com
ScienceDirect
Annals of Pure and Applied Logic 151 (2008) 89–114
ANNALS OF PURE AND APPLIED LOGIC
www.elsevier.com/locate/apal

Angelic semantics of fine-grained concurrency☆

Dan R. Ghica[a], Andrzej S. Murawski[b,*]

[a] *School of Computer Science, University of Birmingham, Birmingham B15 2TT, UK*
[b] *Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, UK*

Available online 26 November 2007

Abstract

We introduce a game model for an Algol-like programming language with primitives for parallel composition and synchronization on semaphores. The semantics is based on a simplified version of Hyland–Ong-style games and it emphasizes the intuitive connection between the concurrent nature of games and that of computation. The model is fully abstract for *may*-equivalence.

Keywords: Shared-memory concurrency; Game semantics; May-equivalence

1. Introduction

Message-passing and shared memory are the two major paradigms of concurrent programming. The latter is closer to the underlying machine model, which makes it both more popular and more "low-level" (and more error-prone) than the former. This constitutes very good motivation for the study of such languages. Concurrent shared-variable programming languages themselves can come in several varieties:

- *Fine-grained* languages have designated atomic actions which are implemented directly by the hardware on which the program is executed. In contrast, *coarse-grained* programming languages can specify sequences of actions to appear as indivisible.
- Languages with *static process creation* execute statements in parallel and then synchronize on the completion of all the statements. Conversely, *dynamic process creation* languages can create wholly autonomous new threads of execution.
- The procedure invocation mechanism can be *call-by-name* or *call-by-value*.

Any combination of the features above is possible and yields interesting programming languages. In this paper we consider *fine-grained, static, call-by-name* languages. We found that this particular set of choices is most naturally suited to the particular semantic model we intend to present.

☆ An extended abstract of this article was presented at FoSSaCS'04.

* Corresponding author.
E-mail address: Andrzej.Murawski@comlab.ox.ac.uk (A.S. Murawski).

0168-0072/$ - see front matter
doi:10.1016/j.apal.2007.10.005

Our language comes very close to Brookes's Parallel Algol (PA) [6], which is a *coarse-grained, static, call-by-name* language. Whereas PA uses a coarse-grained **await** construct, we use fine-grained **semaphores**, with atomic operations **grab** and **release**. Additionally, unlike PA, our language allows side-effects in expressions. But otherwise our language is very similar to PA, and both are faithful to Reynolds's principles of combining call-by-name λ-calculus with local-variable imperative programming [18].

For sequential Algol, the combination of procedures and state gives rise to difficult semantic problems [17], which were first given an adequate solution relatively recently by Abramsky and McCusker using game semantics [2]. Their game model of Algol uses Hyland–Ong-style (HO) games which had previously been used to model sequential functional computation, notably the language PCF [13]. As it will be seen in this paper, the game model of concurrency is substantially simpler than that of sequentiality. One can think of sequentiality as a highly constrained and deterministic form of interleaving of concurrent actions, this being reflected by the nature of the rules governing HO-games. To model concurrency we renounce almost all the HO-rules, including the most basic one, the embodiment of sequentiality, *alternation*, and replace them with a single principle, akin to well-bracketing, that is an immediate reflection on the nature of static concurrency. The relative simplicity of our model is best illustrated by the direct definability proof. While the factorization method seems possible in principle, it would perhaps obscure the connection between the concurrent nature of computation and the concurrent nature of games.

Abramsky made the first attempt to model PA using resumption-style games [1], but the theoretical properties of that model have not been investigated. Concurrent games, using a *true concurrency* representation, have been used by Abramsky and Melliès [3] to model multiplicative–additive linear logic. Using a related notion of *asynchronous games*, Melliès [16] subsequently showed how the simply typed λ-calculus can be captured, by recasting the HO-constraint of *innocence* in terms of permutations on traces. His work thus connects *interleaved* game semantics with true concurrency models. We found that the interleaved representation is the most suitable for our language, because it deals more easily with the possibility of synchronization, which happens either inherently at process creation and termination, or explicitly through the usage of semaphores.

Laird's game model of synchronous message-passing concurrency [15] is the work most closely related to ours. It draws from the HO-model, and it also uses a non-alternating interleaved representation of concurrency. However, the technical differences are substantial. Laird's model introduces additional structure (*concurrency pointers*, to explicitly model threads) and additional conditions (*pointer-blindness*, to cut the model down) in order to set up a framework compatible with the PCF constraints (*visibility, innocence, well-bracketing*). By contrast, our approach is more direct and yields an explicit model, which seems more accessible.

Other work on denotational models for shared-variable concurrency we consider related to ours are Brookes's full abstraction result for a transition-trace model of a ground-type programming language [5] and his relational parametric model of PA [6]. Also interesting is Roeckl and Sangiorgi's process semantics of PA using the π-calculus [19]. A representation of our game model into the π-calculus seems possible, which would give a fully abstract π-calculus model of PA.

1.1. Syntax

The types are generated by the grammar

$$\beta ::= \mathbf{exp} \mid \mathbf{com} \mid \mathbf{var} \mid \mathbf{sem} \qquad \theta ::= \beta \mid \theta \to \theta$$

where β stands for base types.

The type judgements, given below, are of the form $\Gamma \vdash M : \theta$, where Γ maps identifiers to types. $\star$ stands for any of the usual unary or binary operators (e.g. **succ**, **pred**). We use letters C, E, S, V, F to stand for commands, expressions, semaphores, (assignable) variables and functions, respectively. M will be used for terms of any type, n for integer constants and c for base-type constants.

Expressions:

$$\frac{\{\Gamma \vdash E_1 : \mathbf{exp}\} \qquad \Gamma \vdash E_2 : \mathbf{exp}}{\Gamma \vdash \{E_1\} \star E_2 : \mathbf{exp}} \qquad \frac{\Gamma \vdash V : \mathbf{var}}{\Gamma \vdash !V : \mathbf{exp}}$$

Commands:

$$\frac{\Gamma \vdash C_1 : \mathbf{com} \qquad \Gamma \vdash C_2 : \mathbf{com}}{\Gamma \vdash C_1 \,||\, C_2 : \mathbf{com}}$$

$$\frac{\Gamma \vdash C_1 : \mathbf{com} \qquad \Gamma \vdash M_2 : \theta}{\Gamma \vdash C_1; M_2 : \theta} \quad \theta \in \{\mathbf{com}, \mathbf{exp}\}$$

$$\frac{\Gamma \vdash V : \mathbf{var} \qquad \Gamma \vdash E : \mathbf{exp}}{\Gamma \vdash V := E : \mathbf{com}}$$

$$\frac{\Gamma, x : \mathbf{var} \vdash M : \theta}{\Gamma \vdash \mathbf{newvar}\, x := n \,\mathbf{in}\, M : \theta} \quad \theta \in \{\mathbf{com}, \mathbf{exp}\}$$

Synchronization:

$$\frac{\Gamma \vdash S : \mathbf{sem}}{\Gamma \vdash \mathbf{grab}(S) : \mathbf{com}} \qquad \frac{\Gamma \vdash S : \mathbf{sem}}{\Gamma \vdash \mathbf{release}(S) : \mathbf{com}}$$

$$\frac{\Gamma, x : \mathbf{sem} \vdash M : \theta}{\Gamma \vdash \mathbf{newsem}\, x := n \,\mathbf{in}\, M : \theta} \quad \theta \in \{\mathbf{com}, \mathbf{exp}\}$$

PCF:

$$\frac{\Gamma \vdash E : \mathbf{exp} \qquad \Gamma \vdash M_i : \theta}{\Gamma \vdash \mathbf{ifzero}\, E \,\mathbf{then}\, M_1 \,\mathbf{else}\, M_2 : \theta} \quad i = 1, 2 \qquad \theta \in \{\mathbf{com}, \mathbf{exp}\}$$

$$\frac{\Gamma \vdash F : \theta \to \theta' \qquad \Gamma \vdash M : \theta}{\Gamma \vdash FM : \theta'} \qquad \frac{\Gamma, x : \theta \vdash M : \theta'}{\Gamma \vdash \lambda x : \theta . M : \theta \to \theta'}$$

MK:

$$\frac{\Gamma \vdash M : \mathbf{exp} \to \mathbf{com} \qquad \Gamma \vdash E : \mathbf{exp}}{\Gamma \vdash \mathbf{mkvar}\, ME : \mathbf{var}} \qquad \frac{\Gamma \vdash C_1 : \mathbf{com} \qquad \Gamma \vdash C_2 : \mathbf{com}}{\Gamma \vdash \mathbf{mksem}\, C_1 C_2 : \mathbf{sem}}$$

We also have constants

$$n : \mathbf{exp} \qquad \mathbf{skip} : \mathbf{com} \qquad \mathbf{fix}_\theta : (\theta \to \theta) \to \theta$$

as well as the axiom $x : \theta \vdash x : \theta$.

Note that $C_1; M_2$ as well as **ifzero** E **then** M_1 **else** M_2 can be defined as syntactic sugar for all types. For example, given $M_2 : \mathbf{var}$, we can define $C_1; M_2$ by

$$\mathbf{mkvar}(\lambda e . C_1; (M_2 := e))(C_1; !M_2).$$

For function types we first η-expand M_2 and then push $C_1;$ through the λ-bindings.

1.2. Operational semantics

We define the semantics of the language using a (small-step) transition relation

$$\Sigma \vdash M,\ s \longrightarrow M',\ s'.$$

Σ is a set of names of variables denoting *memory cells* and semaphores denoting *locks*; s, s' are states, i.e. functions $s, s' : \Sigma \to \mathbb{N}$, and M, M' are terms. The set of functions $\Sigma \to \mathbb{N}$ will be denoted by States(Σ).

The reduction rules are given in Fig. 1, where ✠ stands for any arithmetic operator or sequential composition. If it causes no confusion in context we may use the following abbreviations:

- $M, s \longrightarrow M', s'$ for $\Sigma \vdash M,\ s \longrightarrow M',\ s'$;
- $M \longrightarrow M'$ for $\Sigma \vdash M,\ s \longrightarrow M',\ s$;

In-context reduction rules

$$\frac{\Sigma \vdash C_1,\ s \longrightarrow C_1',\ s'}{\Sigma \vdash C_1 \,||\, C_2,\ s \longrightarrow C_1' \,||\, C_2,\ s'} \qquad \frac{\Sigma \vdash C_2,\ s \longrightarrow C_2',\ s'}{\Sigma \vdash C_1 \,||\, C_2,\ s \longrightarrow C_1 \,||\, C_2',\ s'}$$

$$\frac{\Sigma \vdash M_1,\ s \longrightarrow M_1',\ s'}{\Sigma \vdash M_1 \Yen M_2,\ s \longrightarrow M_1' \Yen M_2,\ s'} \qquad \frac{\Sigma \vdash M_2,\ s \longrightarrow M_2',\ s'}{\Sigma \vdash c \Yen M_2,\ s \longrightarrow c \Yen M_2',\ s'}$$

$$\frac{\Sigma \vdash E,\ s \longrightarrow E',\ s'}{\Sigma \vdash \mathbf{ifzero}\ E\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2,\ s \longrightarrow \mathbf{ifzero}\ E'\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2,\ s'}$$

$$\frac{\Sigma \vdash V,\ s \longrightarrow V',\ s'}{\Sigma \vdash !V,\ s \longrightarrow !V',\ s'} \qquad \frac{\Sigma \vdash F,\ s \longrightarrow F',\ s'}{\Sigma \vdash FM,\ s \longrightarrow F'M,\ s'}$$

$$\frac{\Sigma \vdash V,\ s \longrightarrow V',\ s'}{\Sigma \vdash V := n,\ s \longrightarrow V' := n,\ s'} \qquad \frac{\Sigma \vdash E,\ s \longrightarrow E',\ s'}{\Sigma \vdash V := E,\ s \longrightarrow V := E',\ s'}$$

$$\frac{\Sigma \vdash S,\ s \longrightarrow S',\ s'}{\Sigma \vdash \mathbf{grab}(S),\ s \longrightarrow \mathbf{grab}(S'),\ s'} \qquad \frac{\Sigma \vdash S,\ s \longrightarrow S',\ s'}{\Sigma \vdash \mathbf{release}(S),\ s \longrightarrow \mathbf{release}(S'),\ s'}$$

$$\frac{\Sigma, v \vdash M[v/x], s \otimes (v \mapsto n) \longrightarrow M', s' \otimes (v \mapsto n')}{\Sigma \vdash \mathbf{newvar}\, x := n\, \mathbf{in}\, M, s \longrightarrow \mathbf{newvar}\, x := n'\, \mathbf{in}\, M'[x/v], s'}$$

$$\frac{\Sigma, v \vdash M[v/x], s \otimes (v \mapsto n) \longrightarrow M', s' \otimes (v \mapsto n')}{\Sigma \vdash \mathbf{newsem}\, x := n\, \mathbf{in}\, M, s \longrightarrow \mathbf{newsem}\, x := n'\, \mathbf{in}\, M'[x/v], s'}$$

Other reductions

$$\begin{aligned}
\Sigma \vdash \mathbf{skip} \,||\, \mathbf{skip},\ s &\longrightarrow \mathbf{skip},\ s \\
\Sigma \vdash \mathbf{skip}; c,\ s &\longrightarrow c,\ s \\
\Sigma \vdash \mathbf{newvar}\, x := n\, \mathbf{in}\, c,\ s &\longrightarrow c,\ s \\
\Sigma \vdash \mathbf{newsem}\, x := n\, \mathbf{in}\, c,\ s &\longrightarrow c,\ s \\
\Sigma \vdash \{n_1\} \star n_2,\ s &\longrightarrow n,\ s, \quad \text{where } n = \{n_1\} \star n_2 \\
\Sigma \vdash \mathbf{ifzero}\, 0\, \mathbf{then}\, M_1\, \mathbf{else}\, M_2,\ s &\longrightarrow M_1,\ s \\
\Sigma \vdash \mathbf{ifzero}\, n\, \mathbf{then}\, M_1\, \mathbf{else}\, M_2,\ s &\longrightarrow M_2,\ s, \quad n \neq 0 \\
\Sigma \vdash !v,\ s \otimes (v \mapsto n) &\longrightarrow n,\ s \otimes (v \mapsto n) \\
\Sigma \vdash v := n',\ s \otimes (v \mapsto n) &\longrightarrow \mathbf{skip},\ s \otimes (v \mapsto n') \\
\Sigma \vdash \mathbf{grab}(v),\ s \otimes (v \mapsto 0) &\longrightarrow \mathbf{skip},\ s \otimes (v \mapsto 1) \\
\Sigma \vdash \mathbf{release}(v),\ s \otimes (v \mapsto n) &\longrightarrow \mathbf{skip},\ s \otimes (v \mapsto 0), \quad n \neq 0 \\
\Sigma \vdash (\lambda x.M)M',\ s \longrightarrow M[M'/x],\ s \qquad & \Sigma \vdash \mathbf{fix}\, M,\ s \longrightarrow M(\mathbf{fix}\, M),\ s \\
\Sigma \vdash (\mathbf{mkvar}\, ME) := n,\ s \longrightarrow Mn,\ s, \qquad & \Sigma \vdash !(\mathbf{mkvar}\, ME),\ s \longrightarrow E,\ s \\
\Sigma \vdash \mathbf{grab}(\mathbf{mksem}\, C_1 C_2),\ s \longrightarrow C_1,\ s, \qquad & \Sigma \vdash \mathbf{release}(\mathbf{mksem}\, C_1 C_2),\ s \longrightarrow C_2,\ s
\end{aligned}$$

Fig. 1. Reduction rules.

- $M, s \Downarrow$ if $\exists s',\ M, s \longrightarrow^* c, s'$, with $c \in \mathbb{N} \cup \{\mathbf{skip}\}$;
- $M \Downarrow$ if M is closed and $M, \emptyset \Downarrow$.

We define a *contextual approximation* relation $\Gamma \vdash M_1 \sqsubseteq_\theta M_2$ by

$$\forall \mathcal{C}[-] : \mathbf{com},\ \mathcal{C}[M_1] \Downarrow \text{ implies } \mathcal{C}[M_2] \Downarrow,$$

where $\mathcal{C}[M_i]$ are closed programs of type **com**.

Contextual equivalence ($\Gamma \vdash M_1 \cong_\theta M_2$) is defined as $\Gamma \vdash M_1 \sqsubseteq_\theta M_2$ and $\Gamma \vdash M_2 \sqsubseteq_\theta M_1$.
Note that the definition of termination $M \Downarrow$ is *angelic*. We consider a term to terminate if *there exists* a terminating evaluation. However, the evaluation is not deterministic, so it is possible that a term has both terminating and non-terminating evaluations. Moreover, we do not differentiate between the various reasons that termination might fail. In our language this can happen either because of infinite reductions (divergence, e.g. $\mathbf{fix}(\lambda x.x)$) or stuck configurations (deadlock, e.g. $\mathbf{newsem}\, s\, \mathbf{in}\, \mathbf{grab}(s); \mathbf{grab}(s)$).

2. Game semantics

2.1. Preliminaries

Game semantics models computation as a game between a Proponent (P), representing a term, and an Opponent (O), representing the environment of the term. Any *play* of the game is an interaction consisting of basic actions called *moves*, which are of two kinds: questions and answers. The fundamental rule is that questions can only be asked if they are *justified* by some previous question, and answers can be given only to relevant questions. A common metaphor is that of *polite conversation*: one must not ask irrelevant questions or provide unrequested answers. In addition, any play must obey other various rules, which are particular and intimately related to the kind of computations one is interested in modeling. P must always play according to a *strategy* that interprets the term. O does not play using some pre-determined strategy, but it still needs to behave according to the rules of play.

This game-theoretic approach, which is highly intensional and interactive, seems particularly well suited for modeling concurrent programming languages. Ironically perhaps, the greatest initial success of game semantics was in providing models for *sequential* computation. Sequentiality is a straitjacketed form of interaction, and its game models reflect this situation by being governed by a number of combinatorial rules.

The essential rule common to all sequential games is that of *alternation*: O and P must take turns. In order to model concurrency we also discard this rule. The "static" style of concurrency of our programming language requires that any process starting subprocesses must wait for the children to terminate in order to terminate itself. At the level of games, this is reflected by the following principle:

> In any prefix of a play, if a question is answered then that question and all questions justified by it are answered exactly once.

Note that this condition can be viewed as an interleaved variant of the well-bracketing condition.

It is helpful to spell out this property using two simpler and more precise rules:

Forking. Only a question that has not been answered can be used as a justifier for future moves.
Joining. A question can be answered only after all the questions justified by it have been answered.

A lot of by now standard definitions in game semantics can be adapted to the new setting. We detail the similarities and differences in what follows.

2.2. Arenas

The definition of arenas remains standard. An *arena* A is a triple $\langle M_A, \lambda_A, \vdash_A \rangle$ where

- M_A is a set of *moves*,
- $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$ is a function determining for each $m \in M_A$ whether it is an *Opponent* or a *Proponent move*, and a *question* or an *answer*. We write λ_A^{OP}, λ_A^{QA} for the composite of λ_A with respectively the first and second projections.
- $\vdash_A$ is a binary relation on M_A, called *enabling*, satisfying
  - . if $m \vdash_A n$ for no m then $\lambda_A(n) = (O, Q)$,
  - . if $m \vdash_A n$ then $\lambda_A^{OP}(m) \neq \lambda_A^{OP}(n)$,
  - . if $m \vdash_A n$ then $\lambda_A^{QA}(m) = Q$.

If $m \vdash_A n$ we say that m *enables* n. We shall write I_A for the set of all moves of A which have no enabler; such moves are called *initial*. Note that an initial move must be an Opponent question.

The *product* $(A \times B)$ and *arrow* $(A \Rightarrow B)$ arenas are defined by:

$$\begin{aligned} M_{A\times B} &= M_A + M_B \\ \lambda_{A\times B} &= [\lambda_A, \lambda_B] \\ \vdash_{A\times B} &= \vdash_A \ + \ \vdash_B \end{aligned}$$

$$\begin{aligned} M_{A\Rightarrow B} &= M_A + M_B \\ \lambda_{A\Rightarrow B} &= [\langle \lambda_A^{PO}, \lambda_A^{QA} \rangle, \lambda_B] \\ \vdash_{A\Rightarrow B} &= \vdash_A + \vdash_B + \{(b, a) \mid b \in I_B \text{ and } a \in I_A\}, \end{aligned}$$

where $\lambda_A^{PO}(m) = O$ if and only if $\lambda_A^{OP}(m) = P$.

An arena is called *flat* if its questions are all initial (consequently the P-moves can only be answers). The arenas used to interpret base types are all flat. We list them below.

Arena	O-question	possible P-answers
$[\![\mathbf{com}]\!]$	*run*	*ok*
$[\![\mathbf{exp}]\!]$	q	n
$[\![\mathbf{var}]\!]$	*read*	n
	write(n)	*ok*
$[\![\mathbf{sem}]\!]$	*grab*	*ok*
	release	*ok*

Inside the table n stands for any $n \in \mathbb{N}$. Note that $[\![\mathbf{sem}]\!]$ is isomorphic to $[\![\mathbf{com}]\!] \times [\![\mathbf{com}]\!]$ and $[\![\mathbf{var}]\!] = [\![\mathbf{exp}]\!] \times [\![\mathbf{com}]\!]^\omega$ (by $[\![\mathbf{com}]\!]^\omega$ we mean the product of countably many copies of $[\![\mathbf{com}]\!]$).

2.3. Positions

A *justified sequence* in arena A is a finite sequence of moves of A equipped with pointers. The first move is initial and has no pointer, but each subsequent move n must have a unique pointer to an earlier occurrence of a move m such that $m \vdash_A n$. We say that n is (explicitly) justified by m or, when n is an answer, that n answers m. Justified sequences, as defined above, are sometimes called *single-threaded*.

If a question does not have an answer in a justified sequence, we say that it is *pending* in that sequence. In what follows we use the letters q and a to refer to question- and answer-moves respectively, m will be used for arbitrary moves and m_A will be a move from M_A. When we write justified sequences we only indicate those justification pointers which cannot be inferred without ambiguity from the structure of the sequence.

Next we define what sequences of moves are considered "legal": (FORK) and (JOIN) are the correctness conditions for questions and answers respectively.

Definition 1. The set P_A of *positions* (or *plays*) over A consists of the justified sequences s over A which satisfy the two conditions below.

FORK: In any prefix $s' = \cdots q \overset{\frown}{\cdots} m$ of s, the question q must not be answered before m is played.
JOIN: In any prefix $s' = \cdots q \overset{\frown}{\cdots} a$ of s, all questions justified by q must be answered.

The simplest sequences of moves that *violate* FORK and JOIN respectively are:

$$q \overset{\frown}{\ } a \quad m \qquad \text{and} \qquad q \overset{\frown}{\ } q \quad a\ .$$

The notion of a play is stable with respect to various swapping operations.

Lemma 2. • *If* $sm_1m_2 \in P_A$ *and* $\lambda_A^{OP}(m_1) = \lambda_A^{OP}(m_2)$, *then* $sm_2m_1 \in P_A$.
- *If* $smq \in P_A$ *and* q *is* not *justified by* m *(e.g. when* m *is an answer), then* $sqm \in P_A$.
- *If* $sqa \in P_A$ *and* a *is not justified by (i.e. is* not *an answer to)* q, *then* $saq \in P_A$.
- *If* $sa_1a_2 \in P_A$ *and* sa_2a_1 *satisfies (JOIN) (i.e.* a_1 *is* not *an answer to a justifier of the question which* a_2 *answers), then* $sa_2a_1 \in P_A$. □

Definition 3. A play $s \in P_A$ is *complete* iff no questions in s are pending.

2.4. Strategies

Strategies describe the way programs (represented by P) interact with their environment (represented by O).

Definition 4. A *strategy* σ on A (written $\sigma : A$) is a prefix-closed subset of P_A that is in addition *O-complete*, i.e. if $s \in \sigma$ and $so \in P_A$, where o is an (occurrence of an) O-move, then $so \in \sigma$.

O-completeness signifies the fact that the environment cannot be controlled during the interaction, and can make any legal move at any time. We will often define strategies using sets of sequences omitting the prefix- or O-closure. We will say that P has a response at position s (when following σ) if $sp \in \sigma$ for some P-move s.

The following notation will be useful in what follows. Note that interleavings of justified sequences are no longer justified sequences, because they might contain several occurrences of initial moves. Instead we shall call such sequences *shuffled*. For two shuffled sequences, $s_1 \amalg s_2$ will denote the set of all interleavings of s_1 and s_2. For two sets of shuffled sequences S_1 and S_2:

$$S_1 \amalg S_2 = \bigcup_{s_1 \in S_1, s_2 \in S_2} s_1 \amalg s_2.$$

Given a set X of shuffled sequences, we define $X^0 = X$, $X^{i+1} = X^i \amalg X$. Then $X^{\circledast}$, called *iterated shuffle* of X, is defined to be $\bigcup_{i \in \mathbb{N}} X^i$. The set of non-empty complete plays of a strategy σ will be denoted by $\mathsf{comp}(\sigma)$.
Two strategies $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ can be composed by considering their possible interactions in the shared arena B. Moves in B are subsequently hidden yielding a sequence of moves in A and C.

Each play in $A \Rightarrow B$ has a unique initial move, but plays in τ may use several initial B-moves. The latter corresponds to multiple uses of the argument of type B. Thus, when the strategies are interacting, σ is replicated in order to allow for any number of its copies to be "used" by τ.

More formally, let u be a sequence of moves from arenas A, B and C with justification pointers from all moves except those initial in C such that pointers from moves in C cannot point to moves in A and vice versa. Define $u \upharpoonright B, C$ to be the subsequence of u consisting of all moves from B and C (pointers between A-moves and B-moves are ignored). $u \upharpoonright A, B$ is defined analogously (pointers between B and C are then ignored). We say that u is an *interaction sequence* of A, B and C if $u \upharpoonright A, B \in P^{\circledast}_{A \Rightarrow B}$ and $u \upharpoonright B, C \in P_{B \Rightarrow C}$. The set of all such sequences is written as $int(A, B, C)$. Then the interaction sequence $\sigma \lightning \tau$ of σ and τ is defined by

$$\sigma \lightning \tau = \{u \in int(A, B, C) \mid u \upharpoonright A, B \in \sigma^{\circledast},\ u \upharpoonright B, C \in \tau\}.$$

Note that $\sigma \lightning \tau$ is prefix-closed.

Suppose $u \in int(A, B, C)$. Define $u \upharpoonright A, C$ to be the subsequence of u consisting of all moves from A and C, but where there was a pointer from a move $m_A \in M_A$ to an initial move $m_B \in M_B$ extend the pointer to the initial move in C which was pointed to from m_B. Then the composite strategy $\sigma; \tau$ is defined to be $\{u \upharpoonright A, C \mid u \in \sigma \lightning \tau\}$. That composition is well defined is proved in Section 2.6.

It is worth contrasting this definition with the alternating case, although the defining formulas are essentially the same. In the alternating case an interaction sequence of two strategies can be extended only with the help of one of the two strategies (and it is known which strategy it should be). Without alternation, the same definition makes the two strategies interact very independently, e.g. moves from A and C do not have to be separated with B moves. Besides, interaction sequences u are no longer uniquely determined by the projections $u \upharpoonright A, B$ and $u \upharpoonright B, C$ as the example below shows.

Example 5. Suppose $q_B \cdot q_A,\ q'_B \cdot q'_A \in \sigma : A \Rightarrow B, q_C \cdot q_B \cdot q'_B \cdot q'_C \in \tau : B \Rightarrow C$ (we omit pointers). Then

$$\{q_C \cdot q_A \cdot q'_A \cdot q'_C,\ q_C \cdot q_A \cdot q'_C \cdot q'_A,\ q_C \cdot q'_C \cdot q_A \cdot q'_A,\ q_C \cdot q'_A \cdot q_A \cdot q'_C,\ q_C \cdot q'_A \cdot q'_C \cdot q_A,\ \cdot q_C \cdot q'_C \cdot q'_A \cdot q_A\} \subseteq \sigma; \tau.$$

The above example demonstrates that there can be no universal identity strategy, because composition may not preserve the order of moves. We have a mismatch between the sequential definition of strategies and the nonsequential way in which they are composed. To rectify this we will move on to strategies which are stable with respect to the ambiguities arising during composition.

2.5. Saturated strategies

The original definition of strategies is inherently sequential. It relies on *sequences* of moves. Clearly, this cannot be sufficient to interpret concurrent computation. Sequences of events represent only one of possibly many *observations* of events which occur in parallel. Much of the ordering of the events present in such a sequence is arbitrary. We must consider strategies containing *all* possible such (sequential) observations of (parallel) interactions. In other words, strategies must be closed under inessential (i.e. unobservable) differences in the order of moves:

- Any action of the environment could be observed at any time between the moment when it becomes possible and the moment when it actually occurs.
- Dually, any action of the program could be observed at any time between the moment when it actually occurs and the moment it ceases to be possible.

To formalize this in terms of moves and plays, we define a preorder $\preceq$ on P_A for any arena A as the least reflexive and transitive relation satisfying $s' \preceq s$ for all $s, s' \in P_A$ such that

1. $s' = s_0 \cdot o \cdot s_1 \cdot s_2$ and $s = s_0 \cdot s_1 \cdot o \cdot s_2$, or
2. $s' = s_0 \cdot s_1 \cdot p \cdot s_2$ and $s = s_0 \cdot p \cdot s_1 \cdot s_2$,

where o is any O move and p is any P move and the justification pointers in s are "inherited" from s'. Since s, s' are legal plays by definition, it follows that no move in s_1 is justified by o (1) and p justifies no move in s_1 (2). Observe that the same preorder is obtained when s_1 consists of just one move.

Definition 6. A strategy σ is *saturated* if and only if whenever $s \in \sigma$ and $s' \preceq s$ then $s' \in \sigma$.

The two saturation conditions, in various formulations, have a long pedigree in the semantics of concurrency. For example, they have been used by Udding to describe propagation of signals across wires in delay-insensitive circuits [20] and by Josephs et al. to specify the relationship between input and output in asynchronous systems with channels [14]. Laird has been the first to propose them in game semantics, in his model of Idealized CSP [15].

For technical arguments it is convenient to use an equivalent "small-step" characterization of saturated strategies.

Lemma 7. *$\sigma : A$ is saturated if and only if the two conditions below hold.*

(1) *If $s_0 m_1 m_2 s_1 \in \sigma$ and $\lambda_A(m_1) = \lambda_A(m_2)$ then $s_0 m_2 m_1 s_1 \in \sigma$.*
(2) *If $s_0 p o s_1 \in \sigma$ and $s_0 o p s_1 \in P_A$ then $s_0 o p s_1 \in \sigma$.*

Recall that in the second clause it is necessary to stipulate $s_0 o p s_1 \in P_A$ (cf. Lemma 2).
Arenas and saturated strategies form a category $\mathcal{G}_{\mathrm{sat}}$ in which $\mathcal{G}_{\mathrm{sat}}(A, B)$ consists of saturated strategies on $A \Rightarrow B$. The identity strategy will be defined by saturating the strictly alternating copycat strategy, which is defined in the same way as identity strategies used for modelling sequential languages (but with respect to the new notion of positions).

Let P_A^{alt} be the subset of P_A consisting of alternating plays (no two consecutive moves are by the same player). The "alternating copycat strategy" id_A^{alt} is the least strategy containing

$$\{s \in P^{\mathrm{alt}}_{A_1 \Rightarrow A_2} \mid \forall\, t \sqsubseteq_{\mathrm{even}} s,\ t \upharpoonright A_1 = t \upharpoonright A_2\}.$$

In id_A^{alt} P copies O-moves as they come provided he is "fast enough" to do so before the next O-move; otherwise the strategy breaks down.

The identity strategy id_A will allow P to copy O-moves from one copy of A to the other in a "parallel" fashion: the P-copy of an O-move does not have to follow the O-move immediately and can be delayed by some other O- or P-moves.

Definition 8. Let $\mathsf{sat}(\tau)$ be the least saturated strategy containing the strategy τ. We define the *identity strategy* id_A as $\mathsf{sat}(id_A^{\mathrm{alt}})$.

Note that the saturated strategy $\mathsf{sat}(\tau)$ is obtained by closing downwards the strategy τ with respect to the preorder $\preceq$.

The product and arena constructions make $\mathcal{G}_{\mathrm{sat}}$ into a cartesian closed category. The empty arena is the terminal object, pairing amounts to taking the sum (up to the canonical embeddings in the disjoint sum). Because the arenas $A \times B \Rightarrow C$ and $A \Rightarrow (B \Rightarrow C)$ are almost identical (up to associativity of disjoint sum), currying and uncurrying essentially leave the strategies unchanged.

Proposition 9. *$\mathcal{G}_{\mathrm{sat}}$ is cartesian closed.*

Let us finish this section with a technical lemma showing that in some cases saturation is preserved by composition even though one of the strategies may not be saturated.

Lemma 10. *If $\sigma : A \Rightarrow B$, $\tau : B \Rightarrow C$ are strategies, σ is saturated and C is flat then $\sigma; \tau = \sigma; \mathsf{sat}(\tau)$. In particular, $\sigma; \tau$ is saturated.*

Proof. We use Lemma 7. Suppose $s, s' \in P_{B \Rightarrow C}$, $s \in \tau$, $s' \preceq s$ and s, s' satisfy one of the two conditions below.

(1) $s' = s_0 \cdot o \cdot m \cdot s_2$ and $s = s_0 \cdot m \cdot o \cdot s_2$;
(2) $s' = s_0 \cdot m \cdot p \cdot s_2$ and $s = s_0 \cdot p \cdot m \cdot s_2$.

We show that for any $u \in int(A, B, C)$ such that $u \upharpoonright A, B \in \sigma^{\circledast}$ and $u \upharpoonright B, C = s'$ there exists $u' \in int(A, B, C)$ such that $u' \upharpoonright B, C = s$, $u' \upharpoonright A, B \in \circledast$ and $u' \upharpoonright A, C = u \upharpoonright A, C$.

Suppose (1) holds. Because C is flat and $s, s' \in P_{B \Rightarrow C}$, both m and o must be B-moves. Let u' be u in which m and o were swapped. Note that then $u' \upharpoonright B, C = s$. If m and o come from different copies of σ in $u \upharpoonright A, B \in \circledast$, then $u' \upharpoonright A, B \in \sigma^{\circledast}$ follows from the definition of $\circledast$. If they are from the same copy, then $u' \upharpoonright A, B \in \sigma^{\circledast}$ follows, because σ is saturated (note that although o was an O-move in $B \Rightarrow C$, it is a P-move in $A \Rightarrow B$). Since u' differs from u only by B-moves, we have $u' \upharpoonright A, C = u \upharpoonright A, C$. The case of (2) is completely symmetric. □

As we shall see later, sometimes it will be convenient to use τ instead of $\mathsf{sat}(\tau)$ to simplify reasoning about composite strategies.

2.6. Categorical structure

This section contains the technical details concerning the categorical structure of $\mathcal{G}_{\mathrm{sat}}$ and may be skipped without loss of continuity.

$\mathcal{G}_{\mathrm{sat}}$ fits into the categorical pattern, pointed out in [11], shared by previous game models for programming languages without parallelism. Namely, it is an example of a *diagonal category* [7] which is isomorphic to a "lluf" subcategory of another symmetric monoidal closed category $\mathcal{G}$. Plays in $\mathcal{G}$ can contain multiple initial moves and the relevant subcategory $\mathcal{G}'$ consists of comonoid homomorphisms where the comonoid structure is given by diagonal maps $\Delta_A : A \Rightarrow A \times A$ for any A [11]. Then the arena-indexed family of diagonal maps Δ_A is a natural transformation between suitable endofunctors on $\mathcal{G}'$, which makes it a diagonal category (and thus a CCC). Below we discuss the constructions in some detail. In addition to providing a more foundational perspective on the model, the categorical view makes the proof that $\mathcal{G}'$ (and so $\mathcal{G}_{\mathrm{sat}}$) is a CCC more transparent.

2.6.1. Multi-threading

We use arenas and constructions on them as already specified in the paper. Other definitions are simply adapted to the new definition of a justified sequence.

A *multi-threaded justified sequence* in arena A is a finite sequence of moves of A equipped with pointers. Each occurrence of a non-initial move n must have a unique pointer to an earlier occurrence of a move m such that $m \vdash_A n$. Note that, contrary to the main definition of the paper, multi-threaded justified sequences allow multiple occurrences of

initial moves. Thus, multi-threaded justified sequences are simply interleavings of single-threaded justified sequences. Plays (positions) are now defined as multi-threaded justified sequences that satisfy the two conditions below.

FORK: In any prefix $s' = \cdots q \overset{\frown}{\cdots} m$ of s, the question q must not be answered before m is played.

JOIN: In any prefix $s' = \cdots q \overset{\frown}{\cdots} a$ of s, all questions justified by q must be answered.

The set of such plays in an arena A is denoted by P_A^{mt}. Because the two correctness criteria rely solely on the structure of single-threaded subsequences, each play is an interleaving of a number of single-threaded plays. Strategies are now defined in the standard way as prefix-closed subsets of P_A^{mt} subject to the *O-completeness* condition: if $s \in \sigma$ and $so \in P_A^{\mathrm{mt}}$ then $so \in \sigma$. The multi-threaded setting introduced brings symmetry into the use of initial A- and B-moves in $A \Rightarrow B$, which allows for more symmetric subsequent definitions and nicer proofs. Interaction sequences u of arenas are defined as those satisfying $u \upharpoonright A, B \in P^{\mathrm{mt}}_{A\Rightarrow B}$ and $u \upharpoonright B, C \in P^{\mathrm{mt}}_{B\Rightarrow C}$. The interactions between two strategies $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ are given by

$$\sigma \lightning \tau = \{u \in int(A, B, C) \mid u \upharpoonright A, B \in \sigma,\ u \upharpoonright B, C \in \tau\}.$$

Finally, we set

$$\sigma; \tau = \{u \upharpoonright A, C \mid u \in \sigma \lightning \tau\}.$$

Then we can prove

Lemma 11. *Composition is well-defined, i.e. $\sigma; \tau$ is a strategy.*

Proof. We should show that whenever $u \in \sigma \lightning \tau$, we have $s = u \upharpoonright A, C \in P^{\mathrm{mt}}_{A\Rightarrow C}$, i.e. (FORK) and (JOIN) are preserved.

Consider $s' = \cdots q \cdots m \sqsubseteq s$ where q justifies m. We have $s' = u' \upharpoonright A, C$ for some $u' \in \sigma \lightning \tau$ ending in m.

When q and m are both in M_A, q is pending in s' before m is played if and only if the same is the case in $u' \upharpoonright A, B$. But $u' \upharpoonright A, B \in \sigma \subseteq P^{\mathrm{mt}}_{A\Rightarrow B}$ so q must be pending in $u' \upharpoonright A, B$ before m is played and thus also in $s' = u' \upharpoonright A, C$. When $q, m \in M_C$, the reasoning is analogous ($P^{\mathrm{mt}}_{B\Rightarrow C}$ is used instead of $P^{\mathrm{mt}}_{A\Rightarrow B}$).

The remaining case is that of $q \in M_C$ and $m \in M_A$. Then m must be a question, $u' = q \cdots q_B \cdots m$ where $q_B \in I_B$ and there is a pointer from m to q_B and from q_B to q. Because $u' \upharpoonright A, B \in P^{\mathrm{mt}}_{A\Rightarrow B}$ must satisfy (FORK) q_B must be pending in u' when m is played. Hence, by (JOIN) for $u' \upharpoonright B, C$, q must be pending in u', as required.

For (JOIN), consider $s' = \cdots q \cdots a \sqsubseteq s$ where a answers q. We have $s' = u' \upharpoonright A, C$ for some $u' \in \sigma \lightning \tau$ ending in a. When q, a are in A, it suffices to appeal to (JOIN) for $u' \upharpoonright A, B$ because all questions justified by q are in A. When q, a are in C and q is not initial, the same reasoning applies. If $q \in I_C$ then, by (JOIN) for $u' \upharpoonright B, C$, all questions from $B \Rightarrow C$ justified by q in u' are answered in u'. Now, by (JOIN) for $u' \upharpoonright A, B$, all questions from A which are justified by q in s' must also be answered in u', so s' satisfies (JOIN).

It is easily seen that O-completeness is preserved by composition. □

Lemma 12. *For $\sigma : A \Rightarrow B$, $\tau : B \Rightarrow C$, $\nu : C \Rightarrow D$:*

$$(\sigma; \tau); \nu = \sigma; (\tau; \nu),$$

i.e. composition is associative.

Proof. Let $s = u \upharpoonright A, D \in (\sigma; \tau); \nu$ where $u \in (\sigma; \tau) \lightning \nu \subseteq (M_A + M_C + M_D)^*$. Then $u \upharpoonright A, C = v \upharpoonright A, C \in \sigma; \tau$ where $v \in \sigma \lightning \tau \subseteq (M_A + M_B + M_C)^*$.

Hence, u and v can be combined into $w \in (M_A + M_B + M_C + M_D)^*$ (where all moves except those initial in D are equipped with pointers to moves of only the adjacent games, e.g. there are no pointers from A to D) such that $w \upharpoonright A, C, D = u$ and $w \upharpoonright A, B, C = v$. Unlike in the alternating case, there may be many ways of constructing w.

Since $w \upharpoonright B, C = v \upharpoonright B, C \in \tau$ and $w \upharpoonright C, D = u \upharpoonright C, D \in \nu$, we get $w \upharpoonright B, C, D \in \tau \lightning \nu$ and $w \upharpoonright B, D \in \tau; \nu$. Because $w \upharpoonright A, B = v \upharpoonright A, B \in \sigma$, we have $w \upharpoonright A, B, D \in \sigma \lightning (\tau; \nu)$, and finally $s = u \upharpoonright A, D = w \upharpoonright A, D \in \sigma; (\tau; \nu)$. □

We need identity strategies in order to complete the definition of a category but for this strategies need to be closed under some rearrangements of moves. For any arena A we define a preorder $\preceq$ on P_A^{mt} as the least transitive relation satisfying

$$s_0 \cdot o \cdot s_1 \cdot s_2 \preceq s_0 \cdot s_1 \cdot o \cdot s_2 \quad \text{and} \quad s_0 \cdot s_1 \cdot p \cdot s_2 \preceq s_0 \cdot p \cdot s_1 \cdot s_2$$

where o is any O move and p is any P move and the *positions* on the left are simply obtained from the positions on the right by moving the designated move (o or p) together with its outgoing and/or incoming pointers.

Definition 13. A strategy σ is *saturated* if and only if whenever $s \in \sigma$ and $s' \preceq s$ then $s' \in \sigma$.

Lemma 14. *Saturated strategies compose.*

Proof. The proof relies on Lemma 7.

- Suppose $s = s_0 m o s_1 \in \sigma; \tau$, i.e. there exists $t = t_0 m s_B o t_1 \in \sigma \between \tau$ such that $t \upharpoonright A, C = s$. Then one can move o backwards through s_B in t and still get interaction sequences. This is the case because σ and τ are saturated (if o is from C we appeal to τ, if o is from A we need σ) and the following two facts hold.
  . If $u_0 b o u_1 \in P^{\mathrm{mt}}_{A \Rightarrow B}$, o is from A and b from B, then $u_0 o b u_1 \in P^{\mathrm{mt}}_{A \Rightarrow B}$.
  . If $u_0 b o u_1 \in P^{\mathrm{mt}}_{B \Rightarrow C}$, o is from C and b from B, then $u_0 o b u_1 \in P^{\mathrm{mt}}_{B \Rightarrow C}$.

  They follow from the fact that P-moves from A (respectively C) cannot justify moves from B in $A \Rightarrow B$ (respectively $B \Rightarrow C$) and O-moves from A (respectively C) cannot be justified by moves from B in $A \Rightarrow B$ (respectively $B \Rightarrow C$).

  Thus $t_0 m o s_B t_1 \in \sigma \between \tau$. Now, if both m and o come from A (respectively C) and $s_0 o m s_1 \in P^{\mathrm{mt}}_{A \Rightarrow C}$ then by saturation of σ (respectively τ) we have $t_0 o m s_B t_1 \in \sigma \between \tau$. Otherwise, $t_0 o m s_B t_1 \in \sigma \between \tau$ by definition of $\between$. Hence, in both cases, $s_0 o m s_1 = t_0 o m s_B t_1 \upharpoonright A, C \in \sigma; \tau$.
- Suppose $s = s_0 p_1 p_2 s_1 \in \sigma; \tau$, i.e. there exists $t = t_0 p_1 s_B t_1 \in \sigma \between \tau$ such that $t \upharpoonright A, C = s$. Then one can move p_1 forward through s_B in s' and still get interaction sequences. This is the case because σ and τ are saturated (if p_1 is from C we appeal to τ, otherwise we need σ) and the following two facts hold.
  . If $u_0 p b u_1 \in P^{\mathrm{mt}}_{A \Rightarrow B}$, p is from A and b is from B, then $u_0 b p u_1 \in P^{\mathrm{mt}}_{A \Rightarrow B}$.
  . If $u_0 p b u_1 \in P^{\mathrm{mt}}_{B \Rightarrow C}$, p is from C and b is from B, then $u_0 b p u_1 \in P^{\mathrm{mt}}_{B \Rightarrow C}$.

  They follow from exactly the same principles as the previously mentioned two.

  Thus $t_0 s_B p_1 p_2 t_1 \in \sigma \between \tau$. Now if p_1, p_2 both come from A (respectively C) then by saturation of σ (respectively τ) we have $t_0 s_B p_2 p_1 t_1 \in \sigma \between \tau$. Otherwise $t_0 p_2 p_1 t_1 \in \sigma \between \tau$ by definition of $\between$. Hence, in both cases, $s_0 p_2 p_1 s_1 = t_0 s_B p_2 p_1 t_1 \upharpoonright A, C \in \sigma; \tau$. □

Let $P_A^{\mathrm{mt,alt}}$ be the subset of P_A^{mt} consisting of alternating plays (no two consecutive moves are by the same player). The set of alternating copycat traces is defined by

$$\mathit{Copycat} = \{s \in P^{\mathrm{mt,alt}}_{A_1 \Rightarrow A_2} \mid \forall\, t \sqsubseteq_{\mathrm{even}} s,\ t \upharpoonright A_1 = t \upharpoonright A_2\}.$$

Definition 15. The *identity strategy* id_A is the least *saturated* strategy containing *Copycat*.

Lemma 16. *A strategy $\sigma : A \Rightarrow B$ is saturated if and only if $\sigma; id_B = \sigma$ and $id_A; \sigma = \sigma$.*

As a consequence, we can define a category $\mathcal{G}$ whose objects are arenas and morphisms between two arenas A and B are saturated strategies on $A \Rightarrow B$. The product and arrow arenas make $\mathcal{G}$ into a symmetric monoidal closed category where the functorial action of $\times$ is defined by

$$\sigma \times \tau = \bigcup_{s \in \sigma, t \in \tau} s \amalg t$$

for $\sigma : A \Rightarrow B$ and $\tau : C \Rightarrow D$. The empty arena 1 serves as the $\times$-unit and is also a terminal object. The correspondence between $\mathcal{G}(A \times B, C)$ and $\mathcal{G}(A, B \Rightarrow C)$ is almost the identity map, because the arenas $A \times B \Rightarrow C$ and $A \Rightarrow (B \Rightarrow C)$ are identical up to associativity of disjoint sum.

The product arena construction defines weak products in $\mathcal{G}$. The two projections from $A \times B$ onto respectively A and B are the same as the appropriate identity strategies (up to the canonical embedding of moves into the disjoint sum). Pairing $\langle \sigma, \tau \rangle : A \Rightarrow B \times C$ of two strategies $\sigma : A \Rightarrow B$ and $\tau : A \Rightarrow C$ can be defined nearly in the same way as $\sigma \times \tau$ but the A-moves from both strategies are considered to come from a single copy of A. $\times$ is a weak product, because another pairing could also be defined as $\sigma \cup \tau$. Next we will identify a cartesian closed subcategory $\mathcal{G}'$ of $\mathcal{G}$ in which $\times$ is actually a product.

2.6.2. Thread-independent strategies

Given a strategy $\sigma : A$ we define $\sigma_{\mathrm{st}} \subseteq \sigma$ as its subset of single-threaded plays. We want to single out strategies which consist exactly of all possible interleavings of their single-threaded positions.

Definition 17. A strategy $\sigma : A$ is *thread-independent* if and only if $\sigma = (\sigma_{\mathrm{st}})^{\circledast}$.

Lemma 18. *Thread-independent strategies compose.*

Hence, we can define a subcategory $\mathcal{G}'$ of $\mathcal{G}$ consisting of arenas and thread-independent saturated strategies. Note that σ is saturated if and only if σ_{st} is (strictly speaking, σ_{st} is not a strategy, but it should be clear what is meant by that).

$\mathcal{G}'$ has an alternative definition. Namely, let us define a family of strategies $\Delta_A : A \Rightarrow A \times A$ such that Δ_A is almost identical to $id_A \times id_A : A \times A \Rightarrow A \times A$ except that A-moves in $A \times A$ are embedded into a single copy of A. We can then view Δ_A as generating a comonoid structure on A and consider homomorphisms between such comonoids.

Lemma 19. *Thread-independent saturated strategies are the comonoid homomorphisms in $\mathcal{G}$.*

Thus, morphisms of $\mathcal{G}'$ are precisely the comonoid homomorphisms. The maps Δ_A are then diagonal in $\mathcal{G}'$ in the sense of [7]. As proved therein, this makes the $\times$-tensor of $\mathcal{G}$ into a product on $\mathcal{G}'$ from which the Proposition below follows immediately.

Proposition 20. *$\mathcal{G}'$ is cartesian closed.*

Since thread-independent strategies are uniquely determined by their thread-independent positions, we can dispense with other positions when representing them and this is exactly how $\mathcal{G}_{\mathrm{sat}}$ from the paper is defined.

Proposition 21. *$\mathcal{G}'$ and $\mathcal{G}_{\mathrm{sat}}$ are isomorphic.*

From now on, all technical arguments will be carried out in $\mathcal{G}_{\mathrm{sat}}$, i.e. plays will be single-threaded justified sequences.

2.7. Ordering strategies

The set of strategies on a given arena A can be ordered by inclusion, which makes it into a complete lattice. The largest element $\top_A$ is P_A. The *empty strategy* $\bot_A$, in which positions are merely the initial O-moves, is the least element. Greatest lower bounds and lowest upper bounds are calculated by taking intersections and sums respectively. Saturated strategies inherit this structure because sums and intersections of saturated strategies remain saturated.

For $s \in P_A$, let us write σ_s for the smallest saturated strategy containing s. σ_s can be constructed by "O-completing" s, taking the prefix-closure and then the $\preceq$-closure. Analogously one can define σ_S, where S is a set of positions. The compact elements of $\mathcal{G}_{\mathrm{sat}}(1, A)$ (i.e. the set of saturated strategies on A) are exactly those of the shape σ_S where S is finite. It is easy to see that any element of $\mathcal{G}_{\mathrm{sat}}(A, B)$ is a supremum of a family of compact elements, so each $\mathcal{G}_{\mathrm{sat}}(A, B)$ has the structure of an ω-algebraic complete lattice. Next we present $\mathcal{G}_{\mathrm{sat}}$ as an appropriately enriched category. It is also easy to see that composition is monotone with respect to inclusion.

However, a stronger result can also be shown.

Lemma 22. *Let $\sigma, \sigma_i : A \Rightarrow B$ and $\tau, \tau_i : B \Rightarrow C$ for $i \in I$. Then we have*

$$\sigma; \left(\bigcup_{i \in I} \tau_i \right) = \bigcup_{i \in I} (\sigma; \tau_i)$$

and, provided $\{\sigma_i\}_{i\in I}$ *is directed,*

$$\left(\bigcup_{i\in I}\sigma_i\right);\tau = \bigcup_{i\in I}(\sigma_i;\tau). \quad \square$$

In particular, composition is continuous. Because pairing and currying are also continuous, we can conclude:

Theorem 23. $\mathcal{G}_{\mathrm{sat}}$ *is an* ω*CPO-enriched cartesian closed category.*

2.8. The game model

The *lambda*-calculus fragment of our language with fixed points can be modelled in a canonical way using the structure of $\mathcal{G}_{\mathrm{sat}}$ exhibited in the previous section [10]. In particular $[\![\mathbf{fix}(\lambda x^{\theta}.x)]\!] = \bot_{[\![\theta]\!]}$. We shall write Ω_θ for $\mathbf{fix}(\lambda x^{\theta}.x)$.

Next we show how to interpret the other constructs. For this purpose it is convenient to present an alternative (but equivalent) syntax of the language using applicative constants rather than term-forming combinators (θ ranges over $\{\mathbf{com}, \mathbf{exp}\}$).

arithmetic & logic : $\mathsf{op}_\star : \{\mathbf{exp}\to\}\mathbf{exp}\to\mathbf{exp}$
conditional : $\mathsf{ifzero}_\theta : \mathbf{exp}\to\theta\to\theta\to\theta$
commands : $\mathsf{seq}_\theta : \mathbf{com}\to\theta\to\theta$, $\mathsf{parc} : \mathbf{com}\to\mathbf{com}\to\mathbf{com}$
variables : $\mathsf{assg} : \mathbf{var}\to\mathbf{exp}\to\mathbf{com}$, $\mathsf{deref} : \mathbf{var}\to\mathbf{exp}$
semaphores : $\mathsf{grb} : \mathbf{sem}\to\mathbf{com}$, $\mathsf{rls} : \mathbf{sem}\to\mathbf{com}$
binders : $\mathsf{newvar}_\theta : (\mathbf{var}\to\theta)\to\theta$, $\mathsf{newsem}_\theta : (\mathbf{sem}\to\theta)\to\theta$.

Using the constants above we can write any imperative program in a functional form. For example, $x := !x + 1 \mid\mid x := !x + 1$ is written as

$$\mathsf{parc}\,(\mathsf{assg}\,x\,(\mathsf{plus}\,(\mathsf{deref}\,x)1))(\mathsf{assg}\,x\,(\mathsf{plus}\,(\mathsf{deref}\,x)1)).$$

The ground-type constants of the language will be interpreted as follows.

- $[\![\mathbf{skip} : \mathbf{com}]\!] : [\![\mathbf{com}]\!]$ is the unique saturated strategy with position $run\cdot ok$.
- $[\![n : \mathbf{exp}]\!] : [\![\mathbf{exp}]\!]$ is the unique saturated strategy with position $q\cdot n$.

Similarly, the strategies interpreting the functional constants can be defined by giving the set of their complete plays. We use subscripts 0, 1, 2 to indicate which instance of a type provides a move. For example, $run_2\cdot q\cdot 1\cdot run_1\cdot ok_1\cdot ok_2$ represents

$$[\![\mathbf{exp}]\!] \Rightarrow [\![\mathbf{com}]\!]_0 \Rightarrow [\![\mathbf{com}]\!]_1 \Rightarrow [\![\mathbf{com}]\!]_2$$

O				run_2
P	q			
O	1			
P			run_1	
O			ok_1	
P				ok_2

The interpretations are:

- $[\![\mathsf{ifzero}_\theta]\!] : [\![\mathbf{exp}]\!]\Rightarrow[\![\theta]\!]_0\Rightarrow[\![\theta]\!]_1\Rightarrow[\![\theta]\!]_2$ is defined by the sets of complete positions listed in Fig. 2.
- $[\![\mathsf{seq}]\!]_\theta : [\![\mathbf{com}]\!]\Rightarrow[\![\theta]\!]_0\Rightarrow[\![\theta]\!]_1$ is given by positions of the shape $q_1\cdot run\cdot ok\cdot q_0\cdot a_0\cdot a_1$, where $q\cdot a\in P_{[\![\theta]\!]}$
- $[\![\mathsf{grb}]\!], [\![\mathsf{rls}]\!] : [\![\mathbf{sem}]\!]_0\Rightarrow[\![\mathbf{com}]\!]_1$ are given respectively by the positions $run_1\cdot grab_0\cdot ok_0\cdot ok_1$ and $run_1\cdot release_0\cdot ok_0\cdot ok_1$.
- $[\![\mathsf{assg}]\!] : [\![\mathbf{var}]\!]_0\Rightarrow[\![\mathbf{exp}]\!]_1\Rightarrow[\![\mathbf{com}]\!]_2$ is defined by $run_2\cdot q_1\cdot n_1\cdot write(n)_0\cdot ok_0\cdot ok_2$.
- $[\![\mathsf{deref}]\!] : [\![\mathbf{var}]\!]_0\Rightarrow[\![\mathbf{exp}]\!]_1$ is defined by $q_1\cdot read_0\cdot n_0\cdot n_1$.

$$\begin{aligned}\theta = \mathbf{com} \quad & run_2 \cdot q \cdot 0 \cdot run_0 \cdot ok_0 \cdot ok_2,\\ & run_2 \cdot q \cdot n \cdot run_1 \cdot ok_1 \cdot ok_2\\ \theta = \mathbf{exp} \quad & q_2 \cdot q \cdot 0 \cdot q_0 \cdot m_0 \cdot m_2,\\ & q_2 \cdot q \cdot n \cdot q_1 \cdot m_1 \cdot m_2\end{aligned}$$

Fig. 2. Complete plays of strategies interpreting **if** ($m \in \mathbb{N}, n \in \mathbb{N}^+$).

$$\begin{array}{ccc}(\llbracket\mathbf{var}\rrbracket \Rightarrow \llbracket\theta\rrbracket) \Rightarrow \llbracket\theta\rrbracket & & \\ & & q\\ & q & \\ \mathit{read} & & \\ n & & \\ \mathit{write}(1) & & \\ \mathit{ok} & & \\ \mathit{read} & & \\ 1 & & \\ \vdots & & \\ & a & \\ & & a\end{array}$$

Fig. 3. Typical plays of the cell^θ_n strategy.

Note that the positions above cannot be non-trivially $\preceq$-reordered, so in each case the set of their prefixes defines a saturated strategy. For parallel composition, however, saturation must be invoked explicitly.

- $\llbracket\mathsf{parc}\rrbracket : \llbracket\mathbf{com}\rrbracket_0 \Rightarrow \llbracket\mathbf{com}\rrbracket_1 \Rightarrow \llbracket\mathbf{com}\rrbracket_2$ is the saturated strategy generated by $run_2 \cdot run_0 \cdot run_1 \cdot ok_0 \cdot ok_1 \cdot ok_2$. Thus, its complete plays are given by $run_2 \cdot (run_0 \cdot ok_0 \;\amalg\; run_1 \cdot ok_1) \cdot ok_2$.

Finally, the interpretation of variable-binding is defined by the equation

$$\llbracket\Gamma \vdash \mathbf{newvar}\, x := n \,\mathbf{in}\, M : \theta\rrbracket = \Lambda_x(\llbracket\Gamma, x : \mathbf{var} \vdash M : \theta\rrbracket); \mathsf{cell}^\theta_n,$$

where Λ_x is the currying isomorphism and the strategy

$$\mathsf{cell}^\theta_n : (\llbracket\mathbf{var}\rrbracket \Rightarrow \llbracket\theta\rrbracket) \Rightarrow \llbracket\theta\rrbracket$$

is the least strategy containing the complete plays given in Fig. 3. That is to say, as long as O does not make two consecutive moves, cell^θ_n will respond to each $write(i)$ with ok and play the most recently written value in response to $read$ (or n if no $write(i)$ has been played by O yet). However, as soon as O plays two moves in a row (e.g. $write(1) \cdot write(2)$ or $write(1) \cdot read$) cell^θ_n stops responding and no further P moves occur. cell^θ_n is thus very much like the cell strategy used for modelling local variables in Idealized Algol.

Local semaphore introduction is defined similarly:

$$\llbracket\Gamma \vdash \mathbf{newsem}\, x := n \,\mathbf{in}\, M : \theta\rrbracket = \Lambda_x(\llbracket\Gamma, x : \mathbf{sem} \vdash M : \theta\rrbracket); \mathsf{lock}^\theta_n,$$

where lock^θ_n is the least strategy containing plays of the shape $q \cdot q \cdot \square \cdot a \cdot a$, where $\square$ is a segment of alternating $grab \cdot ok$ and $release \cdot ok$ sequences. For $n = 0$ the segment must start with $grab \cdot ok$; otherwise it begins with $release \cdot ok$.

Note that cell^θ_n and lock^θ_n are not saturated. However, Lemma 10 shows that, equivalently, we could use the *saturated* strategies $\mathsf{sat}(\mathsf{cell}^\theta_n)$ and $\mathsf{sat}(\mathsf{lock}^\theta_n)$ to the same effect. Therefore, the denotations of terms defined above are always morphisms in $\mathcal{G}_{\mathrm{sat}}$, as required. Nevertheless, for technical purposes, it will turn out more convenient to work with definitions based on the simpler unsaturated strategies cell^θ_n and lock^θ_n.

2.9. Examples

A useful auxiliary function in our examples is

$$\mathbf{test} \equiv \lambda x : \mathbf{exp}.\mathbf{ifzero}\, x \,\mathbf{then}\, \mathbf{skip} \,\mathbf{else}\, \Omega_{\mathbf{com}}.$$

Example 24 (*Nondeterminism*). Let $M_0, M_1 : \mathbf{com}, \mathbf{exp}$. Define $M_0 \,\mathbf{or}\, M_1$ as

$$\mathbf{newvar}\, x := 0 \,\mathbf{in}\, (x := 0 \,||\, x := 1); \mathbf{ifzero}\, !x \,\mathbf{then}\, M_0 \,\mathbf{else}\, M_1.$$

This can be extended to **var** and **sem** using **mkvar** and **mksem** respectively, and to function types using η expansion. Then we have $[\![M_0 \,\mathbf{or}\, M_1]\!] = [\![M_0]\!] \cup [\![M_1]\!]$.

Example 25 (*Sequential Composition*). The three terms below have equal denotations in our model.

$$\begin{array}{l} c_1; c_2 \\ \mathbf{newsem}\, x := 1 \,\mathbf{in}\, c_1; \mathbf{release}(x) \,||\, \mathbf{grab}(x); c_2 \\ \quad \mathbf{newvar}\, v := 1 \,\mathbf{in}\, c_1; v := 0 \,||\, \mathbf{test}(!v); c_2. \end{array}$$

Example 26 (*Test of Linearity*). Consider a term $\Gamma \vdash M : \mathbf{com}, \mathbf{exp}$ and an identifier $s : \mathbf{sem}$. If s is initialized to 0 and not used elsewhere, then $\mathbf{grab}(s); M$ behaves exactly like M, but can be used at most once if passed as argument to a function, as in

$$p : \mathbf{com} \to \mathbf{com} \vdash \mathbf{newsem}\, s := 0 \,\mathbf{in}\, p(\mathbf{grab}(s); M).$$

Observe that instantiating p to $\lambda c : \mathbf{com}.c; c$ or $\lambda c : \mathbf{com}.c \,||\, c$ will not lead to convergence. This construction can be extended to other types in the same way as **or** (Example 24) and will play an important role in the definability argument.

Example 27 (*Test of Linear Parallelism*). The following term generates only nonalternating complete plays:

$$\begin{array}{l} p : \mathbf{com}_1 \to \mathbf{com}_2 \to \mathbf{com}_3 \vdash \\ \quad \mathbf{newsem}\, s_l, s_r, s := 0 \,\mathbf{in} \\ \quad \mathbf{newsem}\, s := 0 \,\mathbf{or}\, 1 \,\mathbf{in} \\ \qquad p\big(\mathbf{grab}(s_l); \mathbf{grab}(s); \mathbf{grab}(s)\big)\big(\mathbf{grab}(s_r); \mathbf{release}(s); \mathbf{release}(s)\big) : \mathbf{com}_4. \end{array}$$

They are generated, using saturation, by:

$$\begin{array}{cccccccc} run_4 & run_3 & run_2 & run_1 & ok_2 & ok_1 & ok_3 & ok_4, \\ O & P & O & O & P & P & O & P \end{array}$$

in $([\![\mathbf{com}]\!]_1 \Rightarrow [\![\mathbf{com}]\!]_2 \Rightarrow [\![\mathbf{com}]\!]_3) \Rightarrow [\![\mathbf{com}]\!]_4$.

Observe that instantiating p to $\lambda c_1 : \mathbf{com}, c_2 : \mathbf{com}.c_1; c_2$ leads to divergence and the corresponding strategy has no complete plays. However, we have convergence for $\lambda c_1 : \mathbf{com}, c_2 : \mathbf{com}.c_1 \,||\, c_2$.

For many programming tasks it is well known that semaphores can be programmed using shared variables only (e.g. the tie-breaker algorithms from [4]). However, such implementations have been defined with the assumption that the processes involved are distinct and can run different code. This does not seem uniform enough to program the behaviour required in Examples 26 and 27, where the competing threads are produced by the same piece of code, namely M. This apparent expressivity failure has motivated the introduction of semaphores as a primitive in our language.

2.10. Intrinsic preorder

Although $\mathcal{G}_{\mathrm{sat}}$ can be shown to be inequationally sound, it is not fully abstract. As is the case for most game models, full abstraction will be proved for the quotient of $\mathcal{G}_{\mathrm{sat}}$ with respect to the so-called *intrinsic preorder*. Fortunately, in our case the quotient turns out to have a more explicit representation based on complete plays (like for Idealized Algol, but not PCF), which makes it easy to apply our model to reasoning about program approximation and equivalence.

Let Σ be the game with a single question q and one answer a such that $q \vdash_\Sigma a$ (note that Σ is the same as $[\![\mathbf{com}]\!]$). There are two strategies for Σ: the bottom strategy $\bot_\Sigma$ and the top strategy $\top_\Sigma = \{\epsilon, q, q \cdot a\}$. The intrinsic preorder for *saturated* strategies on A is defined as follows.

$$\tau_1 \lesssim \tau_2 \text{ iff } \forall \alpha \in \mathcal{G}_{\mathrm{sat}}(A, \Sigma) \text{ if } \tau_1; \alpha = \top_\Sigma \text{ then } \tau_2; \alpha = \top_\Sigma.$$

(For composition the strategies $\tau_i : A$ are regarded as ones between 1 and A.)

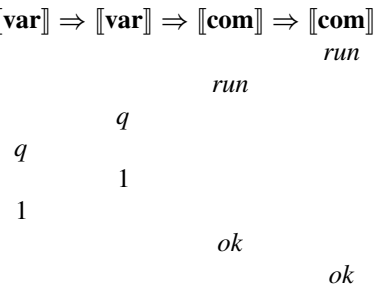


Fig. 4. Plays not occurring in $[\![s]\!]_\theta$.

Let us denote by $\mathsf{comp}(\tau)$ the set of *complete* plays in the strategy τ, that is, those plays in which the opening question is answered. Given a play u, we shall write σ_u for the least saturated strategy containing u.

Theorem 28 (*Characterization*). *Let τ_1, τ_2 be saturated strategies on A. $\tau_1 \lesssim \tau_2$ if and only if $\mathsf{comp}(\tau_1) \subseteq \mathsf{comp}(\tau_2)$.*

Proof. Assume $\tau_1 \lesssim \tau_2$ and $s \in \mathsf{comp}(\tau_1)$. Then we have $\tau_1; \sigma_{q\cdot s\cdot a} = \top$, so also $\tau_2; \sigma_{q\cdot s\cdot a} = \top$. Hence, there exists s' such that $q \cdot s' \cdot a \in \tau_2 \not\perp \sigma_{q\cdot s\cdot a}$. Because $q \cdot s' \cdot a$ is complete and $\sigma_{q\cdot s\cdot a}$ is the smallest strategy containing $q \cdot s \cdot a$ we must have $q \cdot s' \cdot a \preceq q \cdot s \cdot a$. Since s contains only one initial A-move, so does s' and we have $s' \in \mathsf{comp}(\tau_2)$. Because $q \cdot s' \cdot a \preceq q \cdot s \cdot a$ and $s, s' \in P_A$, we have $s \preceq s'$. But τ_2 is saturated, so $s' \in \mathsf{comp}(\tau_2)$ implies $s \in \mathsf{comp}(\tau_2)$.

Suppose $\mathsf{comp}(\tau_1) \subseteq \mathsf{comp}(\tau_2)$ and $\tau_1; \alpha = \top$. Thus there exists an interaction sequence $q \cdot s \cdot a$ of τ_1 and α. Then, by (JOIN) s must be an interleaving of several complete positions from τ_1. By assumption these positions are also in τ_2, so $\tau_2; \alpha = \top$. □

Because the quotient $\mathcal{G}_{\mathrm{qsat}} = \mathcal{G}_{\mathrm{sat}}/\lesssim$ has such a direct representation based on inclusion of complete plays, it is easy to see that it is also a ωCPO-enriched category. The compact elements of $\mathcal{G}_{\mathrm{qsat}}$ are precisely the equivalence classes $[\sigma]_\lesssim$ such that $\mathsf{comp}(\sigma)$ is finite. The next sections are devoted to showing that $\mathcal{G}_{\mathrm{qsat}}$ is sound, adequate and, finally, fully abstract. Note however that for convenience we will still write $[\![M]\!]$ for the interpretation in $\mathcal{G}_{\mathrm{qsat}}$.

3. Soundness and adequacy

3.1. Soundness

For the purpose of relating our model with the operational semantics we will represent a state $s : \Sigma \to \mathbb{N}$ by a *non-saturated* strategy

$$[\![s]\!]_\beta : \big([\![\theta_1]\!] \Rightarrow \cdots \Rightarrow [\![\theta_m]\!] \Rightarrow [\![\beta]\!]\big) \Rightarrow [\![\beta]\!]\,.$$

$[\![s]\!]_\beta$ first copies the initial O-question q to the other $[\![\beta]\!]$ subgame. Then, as long as O does not make two consecutive moves, it behaves in each θ_i component like suitably initialized $\mathsf{cell}^\theta_{n_i}$ and $\mathsf{lock}^\theta_{n_i}$ strategies (i.e. $n_i = s(l_i)$). When two consecutive O-moves do occur, $[\![s]\!]_\beta$ stops playing and no P-moves are played from then on. Finally, when the copy of the initial question is answered by O, $[\![s]\!]_\beta$ answers the initial question with the same answer.

Thus, all complete plays of $[\![s]\!]_\beta$ are of the shape $q \cdot q' \cdot \square \cdot a' \cdot a$ where $\square$ stands for a (possibly empty) sequence of segments of one of the following shapes: $\mathit{read} \cdot n$, $\mathit{write}(n) \cdot \mathit{ok}$, $\mathit{grab} \cdot \mathit{ok}$ or $\mathit{release} \cdot \mathit{ok}$. Such two-move segments will be referred to as *atomic state operations*. The whole sequence $\square$ will be called a *state operation*.

We can think of $[\![s]\!]_\beta$ as a "supersequentialized" store, where individual cells and locks are accessed sequentially both individually and as a group. For example, $[\![(x \mapsto 1 \mid y \mapsto 1)]\!]_{\mathbf{com}}$ has no plays of the form illustrated in Fig. 4.

In what follows, given $\Sigma \vdash M : \beta$ where Σ contains identifiers of type **var** or **sem**, the saturated strategy $[\![\Sigma \vdash M : \beta]\!]\,; [\![s]\!]_\beta$ will be the interpretation of $\Sigma \vdash M : \beta$ at state s. Note that, by Lemma 10, $[\![\Sigma \vdash M]\!]\,; [\![s]\!]_\beta$ will be saturated, though $[\![s]\!]_\beta$ is not. If clear from the context we shall omit the β subscript.

Let Λ^m and Λ^{-m} stand for m-fold currying and uncurrying respectively, so that for a strategy $\sigma : [\![\theta_1]\!] \Rightarrow \cdots \Rightarrow [\![\theta_m]\!] \Rightarrow [\![\mathbf{var}]\!] \Rightarrow [\![\beta]\!]$, we have $\Lambda^{-m}(\sigma) : [\![\theta_1]\!] \times \cdots \times [\![\theta_m]\!] \Rightarrow [\![\mathbf{var}]\!] \Rightarrow [\![\beta]\!]$, a strategy with essentially the same plays (up to trivial relabelling required by associativity of disjoint sum).

We can use the convenient "supersequentialized" model of store because it induces the same meaning of terms as the store obtained from repeated compositions with strategies cell or lock in a sense made precise by the following proposition.

Proposition 29. *For any states $s : \Sigma \to \mathbb{N}$, $s' : (\Sigma \cup \{x_{m+1} : \mathbf{var}\}) \to \mathbb{N}$ such that $s'(x_i) = s(x_i)$ for $i = 1, \ldots, m$ and for any saturated strategy*

$$\sigma : [\![\theta_1]\!] \Rightarrow \cdots \Rightarrow [\![\theta_m]\!] \Rightarrow [\![\mathbf{var}]\!] \Rightarrow [\![\beta]\!],$$

with $\theta_i \in \{\mathbf{var}, \mathbf{sem}\}$ we have

$$\Lambda^m(\Lambda^{-m}(\sigma); \mathsf{cell}^{\beta}_{s'(x_{m+1})}); [\![s]\!]_\beta = \sigma; [\![s']\!]_\beta.$$

If $x_{m+1} : \mathbf{sem}$ then

$$\Lambda^m(\Lambda^{-m}(\sigma); \mathsf{lock}^{\beta}_{s'(x_{m+1})}); [\![s]\!]_\beta = \sigma; [\![s']\!]_\beta.$$

Proof. The $\supseteq$ inclusion is easy. For $\subseteq$ observe that, due to saturation of σ, each interaction sequence giving rise to a complete play on the left can be rearranged in such a way that σ behaves "supersequentially" in the θ_i subgames. □

To prove soundness we note a simple property of reduction.

Proposition 30. *If $\Sigma \vdash M,\ s \longrightarrow M',\ s'$ then there exist $v \in \Sigma, n \in \mathbb{N}$ such that $s' = (s \mid v \mapsto n)$.*

To wit, a one-step reduction can only change the value of (at most) one location or lock in the state.

Lemma 31. *Let $\Sigma = \{x_1 : \theta_1, \ldots, x_m : \theta_m\}$. For any states $s_1, s_2 : \Sigma \to \mathbb{N}$, any terms $\Sigma \vdash M_1, M_2 : \theta$ such that $\Sigma \vdash M_1, s_1 \longrightarrow M_2, s_2$ and any $q \cdot t \cdot a \in \mathsf{comp}([\![\Sigma \vdash M_2]\!])$ such that t is a state operation there exists an empty or atomic state operation t_a such that $q \cdot t_a \cdot t \cdot a \in \mathsf{comp}([\![\Sigma \vdash M_1]\!])$ and:*

- *if $s_1 = s_2$ then either t_a is empty or, for some $1 \le i \le m$ such that $x_i : \mathbf{var}$, we have $t_a = read_i \cdot n_i$ or $t_a = write(n)_i \cdot ok_i$, where $n = s_1(x_i)$*
- *if $s_1(x_i) \neq s_2(x_i)$ and $x_i : \mathbf{var}$ then $t_a = write(n)_i \cdot ok_i$ where $n = s_2(x_i)$*
- *if $s_1(x_i) = 0$ and $s_2(x_i) \neq 0$ and $x_i : \mathbf{sem}$ then $t_a = grab_i \cdot ok_i$*
- *if $s_1(x_i) \neq 0$ and $s_2(x_i) = 0$ and $x_i : \mathbf{sem}$ then $t_a = release_i \cdot ok_i$.*

Note that Proposition 30 ensures that the four cases above are disjoint and exhaustive.

Proof. The proof is by induction on the derivation of $\Sigma \vdash M_1, s_1 \longrightarrow M_2, s_2$.

For most base reduction rules t_a can be taken to be empty except the following four cases.

- $\Sigma \vdash x_i := n, s \longrightarrow \mathbf{skip}, (s \mid x_i \mapsto n)$. We take $t_a = write(n)_i \cdot ok_i$.
- For $\Sigma \vdash !x_i, s \longrightarrow n, s$ where $s(x_i) = n$ we can choose $t_a = read_i \cdot n_i$.
- For $\Sigma \vdash \mathbf{grab}(x_i), s \longrightarrow \mathbf{skip}, (s \mid x_i \mapsto 1)$ such that $s(x) = 0$, we take $t_a = grab_i \cdot ok_i$.
- For $\Sigma \vdash \mathbf{release}(x_i), s \longrightarrow \mathbf{skip}, (s \mid x_i \mapsto 0)$ such that $s(x) \neq 0$, we take $t_a = release_i \cdot ok_i$.

The inductive step is largely a straightforward application of the inductive hypothesis. We analyze the more interesting cases, namely **newvar**,**newsem** and $||$, below.

Consider

$$\frac{\Sigma, v \vdash M_1[v/x], s_1 \otimes (v \mapsto n) \longrightarrow M_2, s_2 \otimes (v \mapsto n')}{\Sigma \vdash \mathbf{newvar}\, x := n \,\mathbf{in}\, M_1, s_1 \longrightarrow \mathbf{newvar}\, x := n' \,\mathbf{in}\, M_2[x/v], s_2}\ .$$

Let $q \cdot t \cdot a \in \mathsf{comp}([\![\Sigma \vdash \mathbf{newvar}\, x := n' \,\mathbf{in}\, M_2[x/v]]\!])$, where t is a state operation. Then, thanks to saturation, $q \cdot t^+ \cdot a \in \mathsf{comp}([\![\Sigma, v : \mathbf{var} \vdash M_2]\!])$, where t^+ is a state operation which restricted to moves related to Σ is the same as t but may also contain atomic state operations related to v.

- Suppose $s_1 = s_2$. Then, by IH, $q \cdot t'_a \cdot t^+ \cdot a \in \mathsf{comp}([\![\Sigma, v : \mathbf{var} \vdash M_1[v/x]]\!])$ where either t'_a is empty or $t'_a = read_i \cdot (s_1(x_i))_i$ or $t'_a = read_v \cdot n_v$ or $t'_a = write(s_1(x_i))_i \cdot ok_i$ or $t'_a = write(n)_v \cdot ok_v$. Then we can take t_a to be respectively empty, $read_i \cdot (s_1(x_i))_i$, empty, $write(s_1(x_i))_i \cdot ok_i$, empty to get $q \cdot t_a \cdot t \cdot a \in \mathsf{comp}([\![\Sigma \vdash \mathbf{newvar}\, x := n \,\mathbf{in}\, M_1]\!])$.
- Suppose $s_1 \neq s_2$. Then, by Proposition 30, $n = n'$ and there exists a unique i such that $s_1(x_i) \neq s_2(x_i)$. By IH, $q \cdot t_a \cdot t^+ \cdot a \in \mathsf{comp}([\![\Sigma, v : \mathbf{var} \vdash M_1[v/x]]\!])$, where t_a contains moves related to x_i. This implies $q \cdot t_a \cdot t \cdot a \in \mathsf{comp}([\![\Sigma \vdash \mathbf{newvar}\, x := n \,\mathbf{in}\, M_1]\!])$.

The case of **newsem** is proved similarly so, finally, we consider $||$ and, for example, the rule

$$\frac{\Sigma \vdash C_1,\ s \longrightarrow C'_1,\ s'}{\Sigma \vdash C_1 \,||\, C_2,\ s \longrightarrow C'_1 \,||\, C_2,\ s'}\ .$$

Let $q \cdot t \cdot a \in \mathsf{comp}([\![\Sigma \vdash C'_1 \,||\, C_2]\!])$, where t is a state operation. Then $q \cdot t^- \cdot a \in \mathsf{comp}([\![\Sigma \vdash C'_1]\!])$, where t^- is a subsequence of t in which only the atomic state operations related to C'_1 are present. Then, by IH, $q \cdot t_a \cdot t^- \cdot a \in \mathsf{comp}([\![\Sigma \vdash C'_1]\!])$ for suitable t_a. But note that then $q \cdot t_a \cdot t \cdot a \in \mathsf{comp}([\![\Sigma \vdash C'_1 \,||\, C_2]\!])$ for the same t_a and we are done. □

The following corollary expresses the sense in which our angelic semantics has a sound model.

Corollary 32 (*Soundness*).

(1) *For any term* $\Sigma \vdash M : \beta$ *and any state* s, *if* $\Sigma \vdash M, s \longrightarrow M', s'$ *then* $[\![\lambda\vec{x}.M']\!]\,;[\![s']\!]_\beta \subseteq [\![\lambda\vec{x}.M]\!]\,;[\![s]\!]_\beta$.
(2) *For any term* $\Sigma \vdash M : \beta$ *and any state* s, *if* $\Sigma \vdash M, s \longrightarrow^* M', s'$ *then* $[\![\lambda\vec{x}.M']\!]\,;[\![s']\!]_\beta \subseteq [\![\lambda\vec{x}.M]\!]\,;[\![s]\!]_\beta$.
(3) *For any* $\Sigma \vdash M : \beta$ *and any state* s, *if* $\Sigma \vdash M, s \Downarrow$ *then* $[\![\lambda\vec{x}.M]\!]\,;[\![s]\!]_\beta \neq \bot$.

Proof. Note that, because β is a base type, $[\![\lambda\vec{x}.M']\!]\,;[\![s']\!]_\beta \subseteq [\![\lambda\vec{x}.M]\!]\,;[\![s]\!]_\beta$ is equivalent to $\mathsf{comp}([\![\lambda\vec{x}.M']\!]\,;[\![s']\!]_\beta) \subseteq \mathsf{comp}([\![\lambda\vec{x}.M]\!]\,;[\![s]\!]_\beta)$.

(1) follows from Lemma 31.
(2) follows from (1).
(3) is an immediate consequence of (2). □

3.2. Adequacy

In order to use strategies to reason about termination we use logical relations.

If t is a complete play on $[\![\Sigma]\!] \Rightarrow [\![\theta]\!]$ and $t \upharpoonright \Sigma$ is a state operation then t will be called *supersequential*. The set of non-empty supersequential complete plays of $[\![\Sigma \vdash M : \theta]\!]$ will be denoted by $\mathsf{scomp}([\![\Sigma \vdash M : \theta]\!])$. By saturation, $\mathsf{comp}([\![\Sigma \vdash M : \theta]\!])$ is empty iff $\mathsf{scomp}([\![\Sigma \vdash M : \theta]\!])$ is.

The logical relation will relate supersequential complete plays with terms. Intuitively, $t \mathrel{\triangleleft}_{\Sigma,\theta} M$ means that every suitably defined state makes play t "operationally realizable" by M. If σ is a strategy, $\sigma \mathrel{\triangleleft}_{\Sigma,\theta} M$ means that all supersequential complete plays of σ are "operationally realizable" by M. The definition of the logical relation is inductive on the structure of supersequential complete plays for **com** and **exp** and otherwise on the structure of types.

Definition 33. (1) $\sigma \mathrel{\triangleleft}_{\Sigma,\theta} M$ iff $\forall t \in \mathsf{scomp}(\sigma).\ t \mathrel{\triangleleft}_{\Sigma,\theta} M$
(2) For $\beta \in \{\mathbf{exp}, \mathbf{com}\}$:
  (a) $q \cdot n \mathrel{\triangleleft}_{\Sigma,\mathbf{exp}} M$ iff $\forall s \in \mathrm{States}(\Sigma).\ M, s \longrightarrow^* n, s$
  (b) $run \cdot ok \mathrel{\triangleleft}_{\Sigma,\mathbf{com}} M$ iff $\forall s \in \mathrm{States}(\Sigma).\ M, s \longrightarrow^* \mathbf{skip}, s$
  (c) $q \cdot t_a \cdot t \cdot a \mathrel{\triangleleft}_{\Sigma,\beta} M$ iff there exists M' such that $q \cdot t \cdot a \mathrel{\triangleleft}_{\Sigma,\beta} M'$ and

(i) if $t_a = read_i \cdot n_i$ then $M, s \longrightarrow^* M', s$ for all s such that $s(x_i) = n$;
(ii) if $t_a = write(n)_i \cdot ok_i$ then $M, s \longrightarrow^* M', (s \mid x_i \mapsto n)$ for all s;
(iii) if $t_a = grab_i \cdot ok_i$ then $M, s \longrightarrow^* M', (s \mid x_i \mapsto 1)$ for all s such that $s(x_i) = 0$;
(iv) if $t_a = release_i \cdot ok_i$ then $M, s \longrightarrow^* M', (s \mid x_i \mapsto 0)$ for all s such that $s(x_i) \neq 0$.

(3) $read \cdot t \cdot n \triangleleft_{\Sigma,\mathbf{var}} M$ iff $q \cdot t \cdot n \triangleleft_{\Sigma,\mathbf{exp}} !M$
(4) $write(n) \cdot t \cdot ok \triangleleft_{\Sigma,\mathbf{var}} M$ iff $run \cdot t \cdot ok \triangleleft_{\Sigma,\mathbf{com}} M := n$
(5) $grab \cdot t \cdot ok \triangleleft_{\Sigma,\mathbf{sem}} M$ iff $run \cdot t \cdot ok \triangleleft_{\Sigma,\mathbf{com}} \mathbf{grab}(M)$
(6) $release \cdot t \cdot ok \triangleleft_{\Sigma,\mathbf{sem}} M$ iff $run \cdot t \cdot ok \triangleleft_{\Sigma,\mathbf{com}} \mathbf{release}(M)$
(7) $t \triangleleft_{\Sigma,\theta_1\to\theta_2} M$ iff for any τ and $\Sigma \vdash N : \theta_1$ such that $\tau \triangleleft_{\Sigma,\theta_1} N$ we have $\langle \sigma_t; \tau \rangle; \mathsf{ev} \triangleleft_{\Sigma,\theta_2} MN$, where $\mathsf{ev}_{\theta_1,\theta_2} : (\llbracket\theta_1\rrbracket \Rightarrow \llbracket\theta_2\rrbracket) \times \llbracket\theta_1\rrbracket \Rightarrow \llbracket\theta_2\rrbracket$ is the canonical evaluation morphism.

The following properties of $\triangleleft$ are essential. Recall that σ_u denotes the least saturated strategy containing the play u.

Proposition 34. *For $\beta \in \{\mathbf{com}, \mathbf{exp}\}$:*

(1) *if $run \cdot t_i \cdot ok \triangleleft_{\Sigma,\mathbf{com}} M_i$ $(i = 1, 2)$ and $t \in t_1 \sqcup t_2$ is a state operation then $run \cdot t \cdot ok \triangleleft_{\Sigma,\mathbf{com}} M_1 \,||\, M_2$,*
(2) *if $(x : \mathbf{var}) \in \Sigma$, $q \cdot t \cdot a \triangleleft_{\Sigma,\beta} M$ and $\Lambda_x(\sigma_{q\cdot t\cdot a}); \mathsf{cell}^\beta_n \neq \bot$ then $q \cdot t^- \cdot a \triangleleft_{\Sigma,\beta} \mathbf{newvar}\, x := n \,\mathbf{in}\, M$,*
(3) *if $(x : \mathbf{sem}) \in \Sigma$, $q \cdot t \cdot a \triangleleft_{\Sigma,\beta} M$ and $\Lambda_x(\sigma_{q\cdot t\cdot a}); \mathsf{lock}^\beta_n \neq \bot$ then $q \cdot t^- \cdot a \triangleleft_{\Sigma,\beta} \mathbf{newsem}\, x := n \,\mathbf{in}\, M$,*

where t^- denotes t from which all moves related to x have been removed and Λ_x stands for currying with respect to the component corresponding to x.

Proof. (1) The proof is by induction on $l = |t_1| + |t_2|$.

$l = 0$: Then t_1, t_2, t are all empty, so $run \cdot t_i \cdot ok \triangleleft M_i$ implies $M_i, s \longrightarrow^* \mathbf{skip}, s$ for all s and $i = 1, 2$. Hence, $M_1 \,||\, M_2, s \longrightarrow^* \mathbf{skip}, s$ and we get $run \cdot t \cdot ok \triangleleft M_1 \,||\, M_2$.

$l = l' + 1$: Then $t = t_a \cdot t'$, where $t' \in (t'_1 \sqcup t'_2)$ and t_a is an atomic state operation such that either ($t_1 = t_a \cdot t'_1$ and $t_2 = t'_2$) or ($t_1 = t'_1$ and $t_2 = t_a \cdot t'_2$). Because the two cases are perfectly symmetric, we consider only the first one. Note that since t is a state operation so is t'.

Then, because $run \cdot t_a \cdot t'_1 \cdot ok \triangleleft M_1$, there exists M'_1 such that $run \cdot t'_1 \cdot ok \triangleleft M'_1$ and $M_1, s \longrightarrow^* M'_1, s'$ for suitable s, s' (depending on t_a). Hence, $M_1 \,||\, M_2, s \longrightarrow^* M'_1 \,||\, M_2, s'$ for the same s, s' as above.

Because we also have $run \cdot t'_2 \cdot ok \triangleleft M_1$ as an assumption and $|t'_1| + |t'_2| < |t_1| + |t_2|$, by IH, we get $run \cdot t' \cdot ok \triangleleft M'_1 \,||\, M_2$.

Thus $run \cdot t \cdot ok \triangleleft M_1 \,||\, M_2$.

(2) The proof is by induction on $|t^-|$. We consider the case $\beta = \mathbf{com}$ for illustration.

Suppose t^- is empty. Then $t = t_{a(1)} \cdot \cdots \cdot t_{a(k)}$ and all the atomic state operations $t_{a(i)}$ are associated with x. Since $q \cdot t \cdot a \triangleleft M$, we can unfold the definition of $\triangleleft$ k times to get terms $M_0, M_1, \ldots, M_k$ such that $M_0 \equiv M$ and $M_i, s'_i \longrightarrow^* M_{i+1}, s_{i+1}$ $(i = 0, \ldots, k-1)$ for all s'_i, s_{i+1} satisfying the constraints specified in the definition of $\triangleleft$ (depending on $t_{a(i+1)}$). In particular, $s_{i+1}(y) = s'_i(y)$ for $y \not\equiv x$ and $run \cdot ok \triangleleft M_k$.

Because $\Lambda_x(\sigma_{q\cdot t\cdot a}); \mathsf{cell}^\beta_n \neq \bot$, the constraints stipulated by the definition of $\triangleleft$ must be compatible with the behaviour of a storage cell initialized with n. Hence, we can chain the above reductions to obtain: for each s'_0 such that $s'_0(x) = n$, $M, s'_0 \longrightarrow^* M_k, s_k$ and $s'_0(y) = s_k(y)$ for $y \not\equiv x$. Because $run \cdot ok \triangleleft M_k$, we also have $M_k, s \longrightarrow^* \mathbf{skip}, s$ for all s. Consequently, for any s, $\mathbf{newvar}\, x := n \,\mathbf{in}\, M, s \longrightarrow^* \mathbf{skip}, s$, so $q \cdot t^- \cdot a \triangleleft \mathbf{newvar}\, x := n \,\mathbf{in}\, M$.

Now suppose $t^- = t_a \cdot u$. Then $t = t_{a(1)} \cdot \cdots \cdot t_{a(k)} \cdot t_a \cdot t_1$ and $u = t_1^-$, where $t_{a(i)}$ are atomic state operations all related to x. By definition of $\triangleleft$, there exist terms $M_0, M_1, \ldots, M_k$ such that $M_i, s'_i \longrightarrow^* M_{i+1}, s_{i+1}$ $(i = 0, \ldots, k-1)$, where $M_0 \equiv M$ and s'_i, s_{i+1} are any states satisfying the restrictions in the definition of $\triangleleft$. In particular $s_{i+1}(y) = s'_i(y)$ for $y \not\equiv x$. Moreover, $run \cdot t_a \cdot t_1 \cdot ok \triangleleft M_k$.

Because $\Lambda_x(\sigma_{q\cdot t\cdot a}); \mathsf{cell}^\beta_n \neq \bot$, for all s'_0 with $s'_0(x) = n$ we have $M, s'_0 \longrightarrow^* M_k, s_k$ and $s'_0(y) = s_k(y)$ for $y \not\equiv x$. Because $run \cdot t_a \cdot t_1 \cdot ok \triangleleft M_k$ there exists M_{k+1} such that $M_k, s_k \to^* M_{k+1}, s_{k+1}$ for all s_k, s_{k+1} (depending on t_a, in particular $s_k(x) = s_{k+1}(x)$) and $q \cdot t_1 \cdot ok \triangleleft M_{k+1}$. Consequently, $\mathbf{newvar}\, x := n \,\mathbf{in}\, M, s'^-_0 \longrightarrow^* \mathbf{newvar}\, x := s_{k+1}(x) \,\mathbf{in}\, M_{k+1}, s^-_{k+1}$, where $s^- = s \restriction (\Sigma \setminus \{x\})$. Note that then s'^-_0 and s^-_{k+1} depend on t_a in the same way as s_k, s_{k+1}, because $s'_0(y) = s_k(y)$ for $y \not\equiv x$.

Because $\Lambda_x(\sigma_{q\cdot t\cdot a});\mathsf{cell}^\beta_n \neq \bot$ we have $\Lambda_x(\sigma_{q\cdot t_1\cdot a});\mathsf{cell}^\beta_{s_{k+1}(x)} \neq \bot$. Thus, by using IH for t_1 (note that $|t_1^-| < |t^-|$), we obtain

$$q\cdot t_1^-\cdot a \lhd \mathbf{newvar}\, x := s_{k+1}(x)\, \mathbf{in}\, M_{k+1}.$$

To sum up, we have shown the above and $\mathbf{newvar}\, x := n\, \mathbf{in}\, M, s \longrightarrow^* \mathbf{newvar}\, x := s_{k+1}(x)\, \mathbf{in}\, M_{k+1}, s'$, where s, s' are determined by the definition of $\lhd$. Hence, we can conclude with $q\cdot t^-\cdot a \lhd \mathbf{newvar}\, x := n\, \mathbf{in}\, M$.

(3) This case is analogous to the previous one. □

A Plotkin-style computability result will be a consequence of the following key lemma.

Lemma 35. *Let $\Sigma, \Gamma \vdash M : \theta$, with $\Gamma = \{y_i : \theta_i \mid i = 1, n\}$. Let strategies σ_i, and terms N_i be such that $\sigma_i \lhd_{\Sigma,\theta_i} N_i$ for $i = 1, \ldots, n$. Then $\langle id, \vec{\sigma}\rangle; [\![M]\!] \lhd_{\Sigma,\theta} M[\vec{N}/\vec{y}]$. In particular, if M is closed then $[\![M]\!] \lhd_\theta M$.*

Proof. The proof is by induction on the syntax of M, where M cannot contain any occurrences of **fix** except for Ω. The difficult cases are parallel composition and local variable and semaphore definition, which follow from Proposition 34. Finally, the result can be lifted to terms with other occurrences of **fix** in the standard way [11]. □

We are now in a position to formulate the computability lemma.

Lemma 36 (*Plotkin-style Computability*). *For any term $\Sigma \vdash M : \beta$, where $\beta \in \{\mathbf{com}, \mathbf{exp}\}$ and any $s \in \mathrm{States}(\Sigma)$ if $[\![\lambda\vec{x}.M]\!]; [\![s]\!]_\beta \neq \bot$ then $\Sigma \vdash M, s \Downarrow$.*

Proof. If $[\![\lambda\vec{x}.M]\!]; [\![s]\!]_\beta \neq \bot$ then by Proposition 29 $[\![\mathbf{newvar}\, \vec{x} := s(\vec{x})\, \mathbf{in}\, M]\!] \neq \bot$. By the previous lemma $[\![\mathbf{newvar}\, \vec{x} := s(\vec{x})\, \mathbf{in}\, M]\!] \lhd \mathbf{newvar}\, \vec{x} := s(\vec{x})\, \mathbf{in}\, M$, so by definition of $\lhd$ we get $\mathbf{newvar}\, \vec{x} := s(\vec{x})\, \mathbf{in}\, M \Downarrow$, which implies $\Sigma \vdash M, s \Downarrow$. □

Corollary 32 and Lemma 36 immediately imply the following.

Proposition 37 (*Computational Adequacy*). *For any program P, $P \Downarrow$ if and only if $[\![P]\!] \neq \bot$.*

3.3. Inequational soundness

We prove inequational soundness both for $\mathcal{G}_{\mathrm{sat}}$ and $\mathcal{G}_{\mathrm{qsat}}$.

Theorem 38 (*Inequational Soundness for $\mathcal{G}_{\mathrm{sat}}$*). *Let $\Gamma \vdash M_i : \theta$ for $i = 1, 2$. If $[\![\Gamma \vdash M_1]\!] \subseteq [\![\Gamma \vdash M_2]\!]$ then $\Gamma \vdash M_1 \sqsubseteq_\theta M_2$.*

Proof. Suppose $[\![\Gamma \vdash M_1]\!] \subseteq [\![\Gamma \vdash M_2]\!]$ and for some context $\mathcal{C}[-]$ we have $\mathcal{C}[M_1] \Downarrow$. By the preceding proposition, $[\![\mathcal{C}[M_1]]\!] \neq \bot$. By monotonicity of our semantics $[\![\mathcal{C}[M_1]]\!] \subseteq [\![\mathcal{C}[M_2]]\!]$, so $[\![\mathcal{C}[M_2]]\!] \neq \bot$. Therefore, by Adequacy, $\mathcal{C}[M_2] \Downarrow$, i.e. $\Gamma \vdash M_1 \sqsubseteq_\theta M_2$. □

Proposition 39 (*Inequational Soundness for $\mathcal{G}_{\mathrm{qsat}}$*). *Let $\Gamma \vdash M_i : \theta$ for $i = 1, 2$. If $[\![\Gamma \vdash M_1]\!] \lesssim [\![\Gamma \vdash M_2]\!]$ then $\Gamma \vdash M_1 \sqsubseteq_\theta M_2$. Equivalently, $\mathsf{comp}([\![\Gamma \vdash M_1]\!]) \subseteq \mathsf{comp}([\![\Gamma \vdash M_2]\!])$ implies $\Gamma \vdash M_1 \sqsubseteq_\theta M_2$.*

Proof. Suppose $[\![\Gamma \vdash M_1]\!] \lesssim [\![\Gamma \vdash M_2]\!]$ and $\mathcal{C}[M_1] \Downarrow$ for some context $\mathcal{C}[-]$. By Adequacy $[\![\mathcal{C}[M_1]]\!] \neq \bot$. Let $\Gamma = \{x_1, \ldots, x_k\}$. Then we have $[\![\mathcal{C}[(\lambda\Gamma.M_1)x_1\cdots x_k]]\!] = [\![\mathcal{C}[M_1]]\!]$. Because $[\![\mathcal{C}'[M]]\!] = [\![M]\!]; [\![x \vdash \mathcal{C}'[x]]\!]$ for closed M, by taking $\mathcal{C}'[x] = \mathcal{C}[xx_1\cdots x_k]$, we have $[\![\lambda\Gamma.M_1]\!]; [\![x \vdash \mathcal{C}'[x]]\!] \neq \bot$. Since $[\![\Gamma \vdash M_1]\!] \lesssim [\![\Gamma \vdash M_2]\!]$, we also have $[\![\lambda\Gamma.M_2]\!]; [\![x \vdash \mathcal{C}'[x]]\!] \neq \bot$. Hence,

$$[\![\mathcal{C}[M_2]]\!] = [\![\mathcal{C}[(\lambda\Gamma.M_2)x_1\cdots x_k]]\!] = [\![\lambda\Gamma.M_2]\!]; [\![x \vdash \mathcal{C}'[x]]\!] \neq \bot.$$

By Adequacy $\mathcal{C}[M_2] \Downarrow$. □

4. Definability

We will show how, given a position s of $[\![\theta]\!]$, one can construct a term of type θ whose denotation is the smallest saturated strategy containing s. The basic idea of the construction is to use the justification pointers to identify potential threads of computation. If two moves are justified by the same move, we can think of them as occurring in parallel threads spawned by the thread corresponding to the justifier. When constructing the term for the position we will

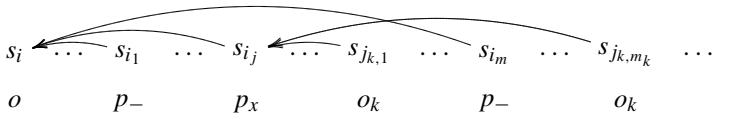


Fig. 5. Questions and justification pointers.

compose all these threads in parallel. Then we use specially designated side-effects as time-stamps to enforce the particular order of moves which happens in the position. Of course, we can only try to achieve this up to the saturation conditions.

The terms resulting from traces using our method are rather peculiar, because more threads are created than strictly necessary. For example, plays produced by terms with sequential composition $M; N$ are reconstructed back into terms with parallel composition which simulate sequential composition, much like in Example 25. Another example is given at the end of this section.

Below we define a recursive algorithm, called *PROC*, which takes a position t in $[\![\theta]\!]$ and returns a term $P : \theta$. The initial argument to *PROC* is the original position s. In the recursive invocations, the argument is a subsequence of the form $s \upharpoonright m$, where $t \upharpoonright m$ is the subsequence of t consisting of m and all moves hereditarily justified by m, always an O-question. Note that consequently a move in t is answered in t if and only if it is answered in s.

Throughout the execution of *PROC* it is convenient to use indices relative to the original s; we write s_i for the ith move of s, assuming s_0 initial. In order to generate the desired position we need to control the way in which both P and O move. We control P-moves using guards that wait for special side-effects (time-stamps) caused by O-moves. The effects take place only if a correct O-move is played and we make sure that they occur only once by using a fresh semaphore for each O-move. This allows us to enforce arbitrary synchronizations policies, restricting the order of moves present in the original sequence up to the reorderings dictated by the saturation conditions.

To that effect, a *global* variable x_j, i.e. a variable which is bound by **new** at the top level and initialized to 0, is associated which each index of an O move in s. The time-stamp consists of assigning 1 to the variable, $x_j := 1$.

For $1 \leq j \leq |s| - 1$, let us define:

$$O_j = \{i \in \mathbb{N} \mid 0 \leq i < j,\ s_i \text{ is an O-move}\}.$$

We define $JOIN_j$ as the term which checks for time-stamps originating from all the O-moves with indices smaller than j (**test** has been defined in Section 2.9):

$$JOIN_j \equiv \mathbf{test}(1-!x_{g_1}); \ldots; \mathbf{test}(1-!x_{g_k}), \quad O_j = \{g_1, \ldots, g_k\}.$$

Note that we could avoid generating divergences by using semaphores instead of variables: $JOIN_j$ could be replaced by $||_{i=1}^{k}(\mathbf{grab}(x_{g_i}); \mathbf{release}(x_{g_i}))$ and $x_i := 1$ by $\mathbf{release}(x_i)$; the variables x_i would have to be initialized to 1 then.

The construction below uses some insights from the definability proof for the HO game model of PCF [13] (nicely explained in [12, Sec. 1.1]) as to the correspondence between moves and subterms.

PROC$(t : \theta)$ where $\theta = \theta_1 \to \cdots \to \theta_h \to \beta$ is defined as follows in two stages which manage O-questions and P-answers, and respectively P-questions and O-answers.

If t is empty, $\lambda p_1 \cdots p_h.\Omega_{\theta_0}$ is returned. Otherwise, let $o = s_i$ be the initial move of t (which is always an O-question).

(1) Let $p_1, \ldots, p_h$ be all the P-questions enabled by o (corresponding respectively to $\theta_1, \ldots, \theta_h$). Let $i_1 < \cdots < i_m$ be the s-indices of all occurrences of $p_1, \ldots, p_h$ in t which are explicitly justified by s_i (see Fig. 5).

PROC returns the following results, depending on β.

- $\beta = \mathbf{com}$:

$$\lambda p_1 \cdots p_h.(x_i := 1); (P_1 \,||\, \cdots \,||\, P_m); PANS_i^{\mathbf{com}}$$

where the terms $P_1, \ldots, P_m : \mathbf{com}$ will be defined later and

$$PANS_i^{\mathbf{com}} \equiv \begin{cases} \Omega_{\mathbf{com}} & s_i \text{ is unanswered in } t \\ JOIN_{i'} & s_{i'} \text{ answers } s_i \text{ in } t. \end{cases}$$

By convention, $(P_1 \,||\, \cdots \,||\, P_m)$ degenerates to **skip** for $m = 0$.

- $\beta = \mathbf{exp}$: Same as for **com** except that $PANS_i^{\mathbf{com}}$ is replaced with $PANS_i^{\mathbf{exp}}$ defined below.

$$PANS_i^{\mathbf{exp}} \equiv \begin{cases} \Omega_{\mathbf{exp}} & s_i \text{ is unanswered in } t \\ JOIN_{i'}; s_{i'} & s_{i'} \text{ answers } s_i \text{ in } t \end{cases}$$

- $\beta = \mathbf{var}$:
  . If $s_i = read$:
  $$\mathbf{mkvar}(\lambda x.\Omega_{\mathbf{com}}, (x_i := 1); (P_1 \,||\, \cdots \,||\, P_m); PANS_i^{\mathbf{exp}}).$$
  . If $s_i = write(v)$:
  $$\mathbf{mkvar}(\lambda x.\mathbf{if}\,(x = v)\,\mathbf{then}\,x_i := 1\,\mathbf{else}\,\mathbf{skip}; (P_1 \,||\, \cdots \,||\, P_m); PANS_i^{\mathbf{com}}, \Omega_{\mathbf{exp}}).$$
  The presence of the $x = v$ test serves to ensure that the only acceptable move by O is only that which writes v, and no other value.
- $\beta = \mathbf{sem}$ is analogous to **var**:
  . If $s_i = grab$:
  $$\mathbf{mksem}((x_i := 1); (P_1 \,||\, \cdots \,||\, P_m); PANS_i^{\mathbf{com}},\ \Omega_{\mathbf{com}}).$$
  . If $s_i = release$:
  $$\mathbf{mksem}(\Omega_{\mathbf{com}}, (x_i := 1); (P_1 \,||\, \cdots \,||\, P_m); PANS_i^{\mathbf{com}}).$$

1. Finally we show how to define the terms P_j for $1 \le j \le m$. Let us fix j and suppose that $s_{i_j} = p_x$ $(1 \le x \le h)$ and $\theta_x = \theta'_1 \to \cdots \to \theta'_n \to \beta'$. Let $o_1, \ldots, o_n$ be all the O-questions enabled by p_x (corresponding to $\theta'_1, \ldots, \theta'_n$ respectively).

   For each k $(1 \le k \le n)$ let $j_{k,1} < \cdots < j_{k,m_k}$ be the s-indices of all occurrences of o_k in t which are explicitly justified by s_{i_j} (see Fig. 5).

   If $m_k = 0$, then $P_j^k \equiv \Omega_{\theta'_k}$. Otherwise, for all $l = 1, \ldots, m_k$ we make the following definitions: $P_j^{k,l} \equiv PROC(t \upharpoonright s_{j_{k,l}} : \theta'_k)$ and

$$P_j^k \equiv ONCE_{w_{j_{k,1}}}[P_j^{k,1}]\ \mathbf{or}\ \cdots\ \mathbf{or}\ ONCE_{w_{j_{k,m_k}}}[P_j^{k,m_k}],$$

   where $w_{j_{k,1}}, \ldots, w_{j_{k,m_k}}$ are fresh semaphore names. The construction **or** is defined as in Example 24, and $ONCE_w(M) = \mathbf{grab}(w); M$ as in Example 26.

   Finally, we define the terms P_j, depending on β'. The fresh variables z_c are used to "store" O-answers for future tests.

   First, it is useful to define the following macros:

$$OANS_c^{\mathbf{com}} \equiv \begin{cases} \mathbf{skip} & s_c \text{ is unanswered in } t \\ x_{c'} := 1 & s_{c'} \text{ answers } s_c \text{ in } t \end{cases}$$

$$OANS_c^{\mathbf{exp}} \equiv \begin{cases} \mathbf{skip} & s_c \text{ is unanswered in } t \\ \mathbf{if}\,(!z_c = s_{c'})\,\mathbf{then}\,x_{c'} := 1\,\mathbf{else}\,\mathbf{skip} & s_{c'} \text{ answers } s_c \text{ in } t. \end{cases}$$

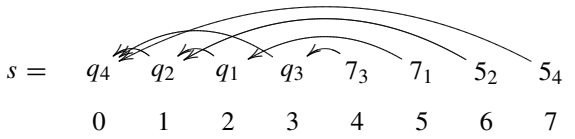
   - For $\beta' = \mathbf{com}$, $P_j \equiv JOIN_{i_j}; (p_x P_j^1 \cdots P_j^n); OANS_{i_j}^{\mathbf{com}}$
   - For $\beta' = \mathbf{exp}$, $P_j \equiv JOIN_{i_j}; z_{i_j} := (p_x P_j^1 \cdots P_j^n); OANS_{i_j}^{\mathbf{exp}}$.
   - For $\beta' = \mathbf{var}$ there are two subcases:
     . If $s_{i_j} = read$, $P_j \equiv JOIN_{i_j}; z_{i_j} := !(p_x P_j^1 \cdots P_j^n); OANS_{i_j}^{\mathbf{exp}}$.
     . If $s_{i_j} = write(v)$, $P_j \equiv JOIN_{i_j}; (p_x P_j^1 \cdots P_j^n) := v; OANS_{i_j}^{\mathbf{com}}$.
   - For $\beta' = \mathbf{sem}$, P_j there are two subcases:
     . If s_{i_j} is $grab$, $P_j \equiv JOIN_{i_j}; \mathbf{grab}(p_x P_j^1 \cdots P_j^n); OANS_{i_j}^{\mathbf{com}}$
     . If s_{i_j} is $release$, $P_j \equiv JOIN_{i_j}; \mathbf{release}(p_x P_j^1 \cdots P_j^n); OANS_{i_j}^{\mathbf{com}}$

After $PROC(s : \theta)$ returns $\lambda p_1 \cdots p_k.M$, all variables and semaphores (x_-, z_-, w_-) used in the construction of M must be bound at the topmost level (the variables x_- must be initialized to 0, the semaphores w_- to 0, the initial values of z_- are irrelevant). For $\beta = \mathbf{com}$, **exp** this is done by taking

$$\lambda p_1 \cdots p_k.\mathbf{newvar}\,\vec{x}, \vec{z} := \vec{0}\,\mathbf{in}\,(\mathbf{newsem}\,\vec{w} := \vec{0}\,\mathbf{in}\,M).$$

For $\beta = \mathbf{var}$, **com** the binders have to be pushed inside **mkvar** or **mksem**. We denote the final term by $PROC^+(s : \theta)$.

Example 40. Consider the play



in the arena $[\![(\mathbf{exp}_1 \to \mathbf{exp}_2) \to \mathbf{exp}_3 \to \mathbf{exp}_4]\!]$. Observe that, for instance, the term $\lambda f.\lambda x.fx$ has this play among its complete positions. The term $PROC^+(s)$ has the following shape:

$$
\begin{array}{l}
\lambda f.\lambda a. \\
\quad \mathbf{newvar}\, x_0, x_2, x_4, x_6, z_1, z_3 := 0\, \mathbf{in} \\
\quad \mathbf{newsem}\, w_2 := 0\, \mathbf{in} \\
\quad\quad x_0 := 1; \\
\quad\quad\quad (JOIN_1; z_1 := f(ONCE_{w_2}[x_2 := 1; \mathbf{skip}; JOIN_5; 7]); \mathbf{if}\, (!z_1 = 5)\, \mathbf{then}\, x_6 := 1\, \mathbf{else}\, \mathbf{skip} \\
\quad\quad\quad || \\
\quad\quad\quad JOIN_3; z_3 := a; \mathbf{if}\, (!z_3 = 7)\, \mathbf{then}\, x_4 := 1\, \mathbf{else}\, \mathbf{skip}); \\
\quad\quad JOIN_7; 5
\end{array}
$$

Notice that the second argument a can be evaluated only after the first one (f) is, because of $JOIN_3$. On the other hand, a must be evaluated before f's argument because of $JOIN_5$. The resulting temporal ordering of the moves is, consequently, the same as in $f(a)$.

We can now state the main result of this section.

Theorem 41 (*Definability*). $[\![PROC^+(s : \theta)]\!] = \sigma_s$ *for any* $s \in P_{[\![\theta]\!]}$.

Proof. For illustration we only consider types that are generated from **com**. We prove by induction that for each $0 \le i < |s|$ $[\![M_i]\!] = \sigma_{s_{\le i}}$, where $M_i = [\![PROC^+(s_{\le i} : \theta)]\!]$ and $s_{\le i} = s_0 \cdots s_i$.

For the base case observe that $M_0 = \lambda p_1 \cdots p_h.(x_0 := 1); \mathbf{skip}; \Omega_{\mathbf{com}}$, so $[\![M_0]\!] = \{\epsilon, s_0\} = \sigma_{s_0}$.

For the inductive step assume $[\![M_i]\!] = \sigma_{s_0 \cdots s_i}$ and $0 \le i < |s| - 1$.

- If s_{i+1} is an O-move, then $\sigma_{s_{\le (i+1)}} = \sigma_{s_{\le i}}$.
  . If s_{i+1} is a question then there exist contexts $\mathcal{C}[-], \mathcal{C}'[-]$ such that

$$
\begin{aligned}
M_i &\equiv \mathcal{C}'[p_x P_j^1 \cdots P \cdots P_j^n] \\
M_{i+1} &\equiv \mathcal{C}[p_x P_j^1 \cdots (P\ \mathbf{or}\ ONCE_{w_{i+1}}[\lambda p_1 \cdots p_n.(x_{i+1} := 1); \Omega_{\mathbf{com}}]) \cdots P_j^n]
\end{aligned}
$$

  $\mathcal{C}$ is almost the same as $\mathcal{C}'$ except that it has two more top-level bindings: respectively for x_{i+1} and w_{i+1}. Note that p_x was introduced in Part 2 of *PROC* in order to correspond to the justifier of s_{i+1}.

  Thus the only difference between M_i and M_{i+1} is the side-effect $x_{i+1} := 1$ which is interpreted with $write(1), ok$. In $[\![M_{i+1}]\!]$ these moves will be hidden due to composition with cell (they will appear in some interaction sequences defining $[\![M_{i+1}]\!]$ but in each such sequence $write(1)$ and ok can occur only once because $x_{i+1} := 1$ is wrapped in *ONCE*). Hence, $[\![M_i]\!] = [\![M_{i+1}]\!]$ as required.
  . If s_{i+1} is an answer then there exist contexts $\mathcal{C}[-], \mathcal{C}'[-]$ such that

$$
\begin{aligned}
M_i &\equiv \mathcal{C}[JOIN_{i_j}; (p_x P_j^1 \cdots P_j^n); \mathbf{skip}] \\
M_{i+1} &\equiv \mathcal{C}'[JOIN_{i_j}; (p_x P_j^1 \cdots P_j^n); (x_{i+1} := 1)]
\end{aligned}
$$

  and $\mathcal{C}'$ has one more binding than $\mathcal{C}$ (where p_x has been introduced in Part 2 of *PROC* in relation to the justifier of s_{i+1}). As before, the only difference is the side-effect but it is hidden so we have $[\![M_i]\!] = [\![M_{i+1}]\!]$. Note that the side-effect can be interpreted only once, because each question can be answered only once.
- If s_{i+1} is a P-move we also have two cases.

. If s_{i+1} is a question there exists a context $\mathcal{C}[-]$ such that

$$M_i \equiv \mathcal{C}[\lambda p_1 \cdots p_h.(x_j := 1); (P_1 \,||\, \cdots \,||\, P_n); PANS_j^{\mathbf{com}}],$$
$$M_{i+1} \equiv \mathcal{C}[\lambda p_1 \cdots p_h.(x_j := 1); (P_1 \,||\, \cdots \,||\, P_n \,||\, P_{n+1}); PANS_j^{\mathbf{com}}],$$

and $P_{n+1} \equiv JOIN_{i+1}; (p_x\, \Omega \cdots \Omega); \Omega_{\mathbf{com}}$, where j is the s-index of the justifier of s_{i+1} and p_x corresponds to s_{i+1} (see Part 2).

Note that there exists an interaction sequence yielding $s_{\leq i} \in [\![M_i]\!]$ in which all the side-effects corresponding to O-moves have been played out. Thus we have $s_{\leq(i+1)} \in [\![M_{i+1}]\!]$ because the guard $JOIN_{i+1}$ can be satisfied. Hence, $\sigma_{s_{\leq(i+1)}} \subseteq [\![M_{i+1}]\!]$.

. If s_{i+1} is an answer then

$$M_i \equiv C[\lambda p_1 \cdots p_h.(x_j := 1); (P_1 \,||\, \cdots \,||\, P_n); \Omega_{\mathbf{com}}],$$
$$M_{i+1} \equiv C[\lambda p_1 \cdots p_h.(x_j := 1); (P_1 \,||\, \cdots \,||\, P_n); JOIN_{i+1}].$$

As before, $s_{\leq(i+1)} \in [\![M_{i+1}]\!]$, because the removal of $\Omega_{\mathbf{com}}$ makes s_{i+1} possible after $s_{\leq i}$ is played. Hence, $\sigma_{s_{\leq(i+1)}} \subseteq [\![M_{i+1}]\!]$.

Before we show the converse let us make some auxiliary definitions. We will say that an occurrence of a move m in a position $t_1 m t_2$ is *free* iff $t_1 t_2$ is also a position. Free occurrences of moves m in s can be characterized in a direct way:

. either m is a question which does not justify any future moves in s,
. or m is an answer to a question whose justifier remains unanswered in s.

Note that if m is free in $s = t_1 m t_2$, then $t_1 t_2 m \in P_A$. Therefore, if $s \in \sigma$ and σ is saturated then $t_1 t_2 p \in \sigma$. For instance, we have $s'_{\leq j} \in \sigma_{s_{\leq j}}$ where $s'_{\leq j}$ arises from $s_{\leq j}$ by removing some free occurrences of P-moves occurring in $s_{\leq j}$.

Now to show that $[\![M_{i+1}]\!] \subseteq \sigma_{s_{\leq(i+1)}}$ let us assume $u \in [\![M_{i+1}]\!]$. If $u \in [\![M_i]\!]$ then by IH $u \in \sigma_{s_{\leq i}} \subseteq \sigma_{s_{\leq(i+1)}}$, so suppose $u \in [\![M_{i+1}]\!] \setminus [\![M_i]\!]$, i.e. u must contain the P-move corresponding to the extra subterm of M_{i+1}. By construction that P-move is the same as s_{i+1}, so $u = u_1 s_{i+1} u_2$.

Because s_{i+1} was played and $u \notin [\![M_i]\!]$, $JOIN_{i+1}$ must have been satisfied after the moves from u_1 were played. By looking at the relevant interaction sequence with the cell and lock strategies we can attribute the time-stamps to distinct occurrences of O-moves (as pointed out before, each side-effect generates at most one pair $write(1)$, ok in the interaction sequence). Then all the occurrences of O-moves associated with time-stamps must be in u_1. We shall call them *active*. All occurrences of O-moves in u_2 are thus inactive and there can also be inactive occurrences in u_1.

Suppose o is a inactive occurrence in u (either in u_1 and u_2). Then o is free in u and we can move it to the end of u. Moreover, the resulting position will also be in $[\![M_{i+1}]\!]$, because inactive occurrences of O-moves do not "enable" any P-moves in u, neither by justification nor by side-effects.

Consequently, we can move all the inactive occurrence of O-moves to the end of u and get another position $u' = u'_1 s_{i+1} u'_2 O \in [\![M_{i+1}]\!]$, where u'_2 consists of P-moves and O is a segment consisting of the relocated O-moves. By saturation $u'_1 u'_2 s_{i+1} O \in [\![M_{i+1}]\!]$. Then $u'_1 u'_2 \in [\![M_i]\!]$, so by IH, $u'_1 u'_2 \in \sigma_{s_{\leq i}}$. Since each O-move in $u'_1 u'_2$ is active, it has a unique counterpart in $s_{\leq i}$. Thus from $u'_1 u'_2 \in \sigma_{s_{\leq i}}$ we can deduce $u'_1 u'_2 \preceq s'_{\leq i}$ where $s'_{\leq i}$ is same as $s_{\leq i}$ without some free occurrences of P-moves. Then we have

$$u \preceq u' = u'_1 s_{i+1} u'_2 O \preceq u'_1 u'_2 s_{i+1} O \preceq s'_{\leq i} s_{i+1} O.$$

Because $s'_{\leq i} s_{i+1} \in \sigma_{s_{\leq(i+1)}}$ we can conclude, by saturation, that $u \in \sigma_{s_{\leq(i+1)}}$. □

Using **or** (Example 24) we can combine multiple uses of the above theorem to show the following.

Theorem 42 (*Compact Definability*). *Any compact saturated strategy σ, i.e. one generated by a finite set of positions, is definable.*

5. Full abstraction

With soundness, adequacy and definability established, the full abstraction result follows routinely.

Theorem 43 (*Full Abstraction*). *Let $\Gamma \vdash M, N : \theta$. Then $[\![M]\!] \lesssim [\![N]\!]$ if and only if $M \sqsubseteq_\theta N$.*

Proof. The left-to-right direction is inequational soundness (Theorem 38).

For the right-to-left direction, we can assume w.l.o.g. that M and N are closed terms. Suppose $[\![M]\!]; \alpha = \top$ for a saturated strategy $\alpha : [\![\theta]\!] \to [\![\mathbf{com}]\!]$. Then for some $s \in \mathsf{comp}(\alpha)$ we have $[\![M]\!]; \sigma_s = \top$. By Theorem 41 there exists a term $x : \theta \vdash C[x] : \mathbf{com}$ such that $[\![C[x]]\!] = \sigma_s$. Since $[\![C[M]]\!] = [\![M]\!]; \sigma_s$, Computational Adequacy implies $C[M] \Downarrow$. Because $M \sqsubseteq_\theta N$, we have $C[N] \Downarrow$ and, by Adequacy again, $[\![C[N]]\!] = \top$. Hence $[\![N]\!]; \alpha \supseteq [\![N]\!]; \sigma_s = [\![C[N]]\!] = \top$. □

6. Conclusion

We have presented a fully abstract game model for a programming language with fine-grained shared-variable concurrency. We found that HO-style games are naturally suited to interpreting concurrency, and most of the technical complexity required in interpreting sequential computation can be avoided.

In addition to theoretical interest, our fully abstract model can be used to reasoning about program equivalence. In addition to the examples described in Section 2.9 we can also make straightforward arguments about ground-type equivalences such as Brookes's *laws of parallel programming* [6], or other typical second-order equivalences. In order for such arguments to be formalized, and even automated, it is necessary to find a concrete representation of strategies, along the lines of [8]. For this purpose, the most convenient representations are those which are finite-state, such as regular expressions, regular languages, labelled transitions systems, etc.

However, identifying a non-trivial fragment of this language for which the strategies are finitary is not straightforward. If the Opponent is allowed to ask a question, then it can ask an arbitrary number of questions. This implies that even second-order terms such as $f : \mathbf{com} \to \mathbf{com}, c : \mathbf{com} \vdash f(c) : \mathbf{com}$ do not have a finite-state model, because sets of complete plays can be immediately seen not to satisfy the pumping property of regular languages. This reflects the fact that f can use c in an arbitrarily large number of concurrent threads. One way of approaching this problem has been presented in [9], where we have introduced a type system to control concurrency effects by identifying a bound on the number of concurrent threads that a function is allowed to create.

The main theoretical development which is required is adapting our model to dealing with *must*-equivalence, i.e. a notion of equivalence which considers not just termination but the full spectrum of observable behaviour: termination, failure and divergence. Must-equivalence has been studied using game semantics in the simpler setting of bounded determinism by Harmer and McCusker [11], and some of their techniques may be applicable in our setting.

Acknowledgement

We gratefully acknowledge financial support from the UK EPSRC (GR/R88861).

References

[1] S. Abramsky, Game semantics of idealized Parallel Algol, Lecture given at the Newton Institute, 1995.
[2] S. Abramsky, G. McCusker, Linearity, sharing and state: A fully abstract game semantics for Idealized Algol with active expressions, in: Proceedings of the Linear Logic Tokyo Meeting, in: ENTCS, vol. 3, 1996, pp. 2–14. Also published as Chapter 20 of [17].
[3] S. Abramsky, P.-A. Melliès, Concurrent games and full completeness, in: Proceedings of 14th Symposium on Logic in Computer Science, IEEE Computer Society Press, 1999, pp. 431–442.
[4] G. Andrews, Concurrent Programming: Principles and practice, Addison-Wesley Publishing Company, 1991.
[5] S. Brookes, Full abstraction for a shared variable parallel language, in: Proceedings of 8th Symposium on Logic in Computer Science, IEEE Computer Society Press, 1993, pp. 98–109.
[6] S. Brookes, The essence of Parallel Algol, in: Proceedings of Symposium of 11th Symposium on Logic in Computer Science, IEEE Computer Society Press, 1996, pp. 164–173. Also published as Chapter 21 of [17].
[7] P.J. Freyd, P.W. O'Hearn, A.J. Power, M. Takeyama, R. Street, R.D. Tennent, Bireflectivity, Theoretical Computer Science 228 (1–2) (1999) 49–76.
[8] D.R. Ghica, G. McCusker, Reasoning about Idealized ALGOL using regular languages, in: Proceedings of 27th International Colloquium on Automata, Languages and Programming, in: LNCS, vol. 1853, Springer-Verlag, 2000, pp. 103–116.
[9] D.R. Ghica, A.S. Murawski, C.-H.L. Ong, Syntactic control of concurrency, in: Proceedings of 31st International Colloquium on Automata, Languages and Programming, in: Lecture Notes in Computer Science, vol. 3142, Springer-Verlag, 2004, pp. 683–694.
[10] C.A. Gunter, Semantics of Programming Languages: Structures and Techniques, MIT Press, 1992.
[11] R. Harmer, G. McCusker, A fully abstract game semantics for finite nondeterminism, in: Proceedings of 14th Symposium on Logic in Computer Science, IEEE Computer Society Press, 1999, pp. 422–430.
[12] D. Hughes, Hypergame semantics: Full completeness for system F, Ph.D. Thesis, Oxford University Computing Laboratory, 2000.

[13] J.M.E. Hyland, C.-H.L. Ong, On full abstraction for PCF: I, II and III, Information and Computation 163 (2) (2000) 285–408.
[14] J. He, M.B. Josephs, C.A.R. Hoare, A theory of synchrony and asynchrony, in: Programming Concepts and Methods, Elsevier, 1990, pp. 459–473.
[15] J. Laird, A games semantics for Idealized CSP, in: Proceedings of 17th Conference on Mathematical Foundations of Programming Semantics, in: ENTCS, vol. 45, Elsevier, 2001, pp. 1–26.
[16] P.-A. Melliès, Asynchronous games 2: The true concurrency of innocence, in: Proceedings of 15th International Conference on Concurrency Theory, in: LNCS, vol. 3170, Springer, 2004, pp. 448–465.
[17] P.W. O'Hearn, R.D. Tennent (Eds.), ALGOL-like Languages, in: Progress in Theoretical Computer Science, Birkhäuser, 1997, two volumes.
[18] J.C. Reynolds, The essence of Algol, in: J.W. de Bakker, J. van Vliet (Eds.), Algorithmic Languages, North Holland, 1978, pp. 345–372.
[19] C. Röckl, D. Sangiorgi, A π-calculus process semantics of Concurrent Idealised ALGOL, in: Proceedings of the 2nd Conference on the Foundations of Software Science and Computation Structures, in: LNCS, vol. 1578, Springer, 1999, pp. 306–321.
[20] J.T. Udding, A formal model for defining and classifying delay-insensitive circuits and systems, Distributed Computing 1 (4) (1986) 197–204.