Angelic Semantics of Fine-Grained Concurrency*

Dan R. Ghica and Andrzej S. Murawski

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, UK
{Dan.Ghica,Andrzej.Murawski}@comlab.ox.ac.uk

Abstract. We introduce a game model for a procedural programming language extended with primitives for parallel composition and synchronization on binary semaphores. The model uses an interleaved version of Hyland-Ong-style games, where most of the original combinatorial constraints on positions are replaced with a simple principle naturally related to static process creation. The model is fully abstract for *may*-equivalence.

1 Introduction

The two major paradigms of concurrent programming are message-passing and shared-variable. The latter style of programming 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.

Our language comes very close to Brookes's Parallel Algol (PA) [1], 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**

* Work funded by British EPSRC, Canadian NSERC and St John's College, Oxford.

and **release**. Additionally, unlike PA, our language allows side-effects in expressions. But otherwise it is very similar to PA, and both are quite faithful to Reynolds's principles of combining call-by-name λ-calculus with local-variable imperative programming.

For sequential Algol, the combination of procedures and state gives rise to difficult semantic problems [2], which were first given an adequate solution relatively recently by Abramsky and McCusker using game semantics [3]. Their game model of Algol uses Hyland-Ong-style (HO) games which had previously been used to model sequential functional computation, notably for the language PCF [4]. Since game models are strikingly *concurrent*, adapting them to the analysis of parallel computation is a natural step. 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 the 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 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. The resultant game model is fully abstract with respect to may-equivalence. Therefore it can be used to reason about safety properties, but not liveness (deadlock-freeness).

Concurrent games, using a *true concurrency* representation, have been used by Abramsky and Melliès to model multiplicative additive linear logic [5]. Abramsky also made the first attempt to model PA using resumption-style games [6], but the theoretical properties of that model have not been investigated. 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 [7] 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-down the model) 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 [8] and his relational parametric model of PA [1]. Also interesting is Röckl and Sangiorgi's process semantics of PA using the π-calculus [9]. A representation of our game model into the π-calculus seems possible, which would give a fully abstract π-calculus model of PA.

2 Syntax and Operational Semantics

The types are $\beta ::= \mathbf{exp} \mid \mathbf{com}$ and $\theta ::= \beta \mid \mathbf{var} \mid \mathbf{sem} \mid \theta \to \theta$. The type judgements are of form $\Gamma \vdash M : \theta$ where Γ maps identifiers to types. The typing rules are those of Idealized Algol (IA) with active expressions plus rules for the following new terms:

$$\frac{\Gamma \vdash C_1 : \mathbf{com} \qquad \Gamma \vdash C_2 : \mathbf{com}}{\Gamma \vdash C_1 \,||\, C_2 : \mathbf{com}} \qquad \frac{\Gamma, x : \mathbf{sem} \vdash M : \beta}{\Gamma \vdash \mathbf{newsem}\, x := n \,\mathbf{in}\, M : \beta}$$

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

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 of semaphores denoting *locks*; s, s' are states, i.e. functions $s, s' : \Sigma \to \mathbb{N}$, and M, M' are terms.

The reduction rules specific to our language are those for parallel composition, semaphore manipulation and binding and we give them below.

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

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

$$\Sigma \vdash \mathbf{grab}(v),\, s \otimes (v \mapsto 0) \longrightarrow \mathbf{skip},\, s \otimes (v \mapsto 1)$$
$$\Sigma \vdash \mathbf{release}(v),\, s \otimes (v \mapsto n) \longrightarrow \mathbf{skip},\, s \otimes (v \mapsto 0) \quad n > 0$$

Semaphores are interpreted in a standard way, using stateful *locks*, v. If v is 0 then the semaphore can be *grabbed*, which changes its state to 1; if v is non-zero then the semaphore can be *released*, which changes its state back to 0. Note that semaphore operations are *atomic*, i.e. they cannot be interrupted by other concurrent processes.

It is common to identify $\mathbf{var}$ with $(\mathbf{exp} \to \mathbf{com}) \times \mathbf{exp}$ and use a variable constructor $\mathbf{mkvar} : (\mathbf{exp} \to \mathbf{com}) \to \mathbf{exp} \to \mathbf{var}$ [2]. In the same "object-oriented" spirit, we identify $\mathbf{sem}$ with $\mathbf{com} \times \mathbf{com}$ and introduce a semaphore constructor $\mathbf{mksem} : \mathbf{com} \to \mathbf{com} \to \mathbf{sem}$.

We use the following abbreviations: $M, s \Downarrow$ if $\exists s',\ M, s \longrightarrow^* c, s'$, with $c \in \mathbb{N} \cup \{\mathbf{skip}\}$, and $M \Downarrow$ if M is closed and $M, \emptyset \Downarrow$. We define a *contextual approximation* relation $\Gamma \vdash M_1 \sqsubseteq_\theta M_2$ by $\forall \mathcal{C}[-] : \mathbf{com},\ \mathcal{C}[M_1] \Downarrow$ implies $\mathcal{C}[M_2] \Downarrow$, where $\mathcal{C}[M_i]$ are closed programs of $\mathbf{com}$ type. *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. $\mathbf{newsem}\, s := 0\, \mathbf{in}\, \mathbf{grab}(s); \mathbf{grab}(s)$).

3 Game Semantics

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.

The game-semantic approach, which is highly intensional and interactive, is 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 sub-processes 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.

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.

Waiting 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.

3.1 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 $\lambda_A^{OP}, \lambda_A^{QA}$ 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:

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

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:

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

Note that $[\![\mathbf{sem}]\!]$ is isomorphic to $[\![\mathbf{com}]\!] \times [\![\mathbf{com}]\!]$ and $[\![\mathbf{var}]\!] = [\![\mathbf{com}]\!]^\omega \times [\![\mathbf{exp}]\!]$, where by $[\![\mathbf{com}]\!]^\omega$ we mean the product of countably many copies of $[\![\mathbf{com}]\!]$.

3.2 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. Note that interleavings of several justified sequences may not be justified sequences; instead we shall call them *shuffled sequences*.

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":

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 $s' = \cdots\ q \cdots m$ of s, the question q must be pending before m is played.*
WAIT *: In any prefix $s' = \cdots\ q \cdots a$ of s, all questions justified by q must be answered.*

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

$$q \quad a \quad m \qquad \text{and} \qquad q \quad q \quad a$$

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

Lemma 1. – *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 then $sqm \in P_A$.*
– *If $sqa \in P_A$ and a is not justified by q, then $saq \in P_A$.*
– *If $sa_1a_2 \in P_A$ and sa_2a_1 satisfies WAIT then $sa_2a_1 \in P_A$.*

Note that the definitions of $A \times B$ and $A \Rightarrow B$ no longer imply the usual switching condition, which characterizes sequential execution.

Definition 2. *A play $s \in P_A$ is* complete *iff no questions in s are pending.*

The following notations will be useful. For two shuffled sequences s_1 and s_2, $s_1 \amalg s_2$ 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 $\bigcup_{i \in \mathbb{N}} X^i$.

3.3 Strategies

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

Definition 3. *A* strategy *σ on A (written $\sigma : A$) is a prefix-closed subset of P_A, which is* 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 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, positions of σ will be replicated in order to allow for any number of copies of σ 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 $\mathit{int}(A, B, C)$. Then the interaction sequence $\sigma \between \tau$ of σ and τ is defined by $\sigma \between \tau = \{\, u \in \mathit{int}(A, B, C) \mid u \upharpoonright A, B \in \sigma^{\circledast},\ u \upharpoonright B, C \in \tau \,\}$.

Suppose $u \in \mathit{int}(A, B, C)$. 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 $\{\, u \upharpoonright A, C \mid u \in \sigma \between \tau \,\}$.

3.4 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 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 every move in s has the same justifier as in 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).

Definition 4. *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 [10] and by Josephs *et al* to specify the relationship between input and output in asynchronous systems with channels [11]. Laird has been the first to use them in game semantics, in his model of Idealized CSP [7].

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

Lemma 2. *$\sigma : A$ is saturated if and only if the two conditions below hold.*

1. *If $sm_1m_2 \in \sigma$ and $\lambda_A(m_1) = \lambda_A(m_2)$ then $sm_2m_1 \in \sigma$.*
2. *If $spo \in \sigma$ and $sop \in P_A$ then $sop \in \sigma$.* □

Recall that in the second clause it is necessary to stipulate $sop \in P_A$ (Lemma 1).

Arenas and saturated strategies form a category $\mathcal{G}_{\text{sat}}$ in which $\mathcal{G}_{\text{sat}}(A, B)$ consists of saturated strategies on $A \Rightarrow B$. The identity strategy will be defined by saturating the strictly alternating copy-cat strategy, which is turn defined in the same way as identity strategies used for modeling 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 copy-cat strategy" id_A^{alt} is the least strategy containing $\{\, s \in P_{A_1 \Rightarrow A_2}^{\text{alt}} \mid \forall\, t \sqsubseteq_{\text{even}} s,\ t \upharpoonright A_1 = t \upharpoonright A_2 \,\}$. 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.

Recall the lack of switching conditions for $A_1 \Rightarrow A_2$. Consequently, id_A^{alt} also admits plays of the shape $d_2d_1e_1e_2f_1f_2$, which are illegal in the alternating setting. We used subscripts $1, 2$ to indicate which instance of a type provides a move.

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

The product and arena constructions make $\mathcal{G}_{\text{sat}}$ 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 un-currying essentially leave the strategies unchanged.

Proposition 1. *$\mathcal{G}_{\text{sat}}$ is Cartesian closed.*

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* $\perp_A$, 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.

Theorem 1. *$\mathcal{G}_{\text{sat}}$ is an ωCPO-enriched Cartesian closed category.*

We finish with a technical lemma which shows that in some cases saturation is preserved by composition even though one of the strategies may not be saturated.

Lemma 3. *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, $\sigma;\tau : A \Rightarrow C$ is saturated.*

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

3.5 The Game Model

The lambda-calculus fragment of our language with fixed points can be modelled in a canonical way using the structure of $\mathcal{G}_{\text{sat}}$ exhibited in the previous section. 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. It is convenient to present an alternative, but equivalent syntax of the language using functions rather than term-forming combinators:

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

The strategies interpreting the functional constants of the language can be defined by giving the set of their complete plays.

Those inherited from IA are interpreted exactly as in [3]. For instance $[\![\mathsf{seq}]\!] : [\![\mathbf{com}]\!] \Rightarrow [\![\beta]\!]_0 \Rightarrow [\![\beta]\!]_1$ is given by positions of the shape $q_1 \cdot run \cdot ok \cdot q_0 \cdot a_0 \cdot a_1$ and $[\![\mathsf{assg}]\!] : [\![\mathbf{var}]\!]_0 \Rightarrow [\![\mathbf{exp}]\!]_1 \Rightarrow [\![\mathbf{com}]\!]_2$ is defined by $run_2 \cdot q_1 \cdot n_1 \cdot write(n)_0 \cdot ok_0 \cdot ok_2$.

The interpretations for $[\![\mathsf{grb}]\!], [\![\mathsf{rls}]\!] : [\![\mathbf{sem}]\!]_0 \Rightarrow [\![\mathbf{com}]\!]_1$ are given respectively by the positions $run_1 \cdot grab_0 \cdot ok_0 \cdot ok_1$ and $run_1 \cdot release_0 \cdot ok_0 \cdot ok_1$.

For parallel composition, $[\![\mathsf{parc}]\!] : [\![\mathbf{com}]\!]_0 \Rightarrow [\![\mathbf{com}]\!]_1 \Rightarrow [\![\mathbf{com}]\!]_2$ 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 exactly those of $run_2 \cdot (run_0 \cdot ok_0 \amalg run_1 \cdot ok_1) \cdot ok_2$. Note that this is the only language constant interpreted by a strategy with non-alternating plays.

3.6 Stateful Behaviour: Cells and Locks

The interpretation of a local variable is defined as the composition of the following *strategies*:

$$[\![\Gamma \vdash \mathbf{newvar}\, x := n\, \mathbf{in}\, M : \beta]\!] = \Lambda_x([\![\Gamma, x : \mathbf{var} \vdash M : \beta]\!]); \mathsf{cell}_n^\beta$$

where Λ_x is the currying isomorphism and the strategy $\mathsf{cell}_n^\beta : ([\![\mathbf{var}]\!] \Rightarrow [\![\beta]\!]) \Rightarrow [\![\beta]\!]$ is the least strategy containing the alternating plays of the shape: $q \cdot q \cdot read \cdot$

$n \cdot \mathit{write}(i) \cdot \mathit{ok} \cdot \mathit{read} \cdot i \cdots a \cdot a$, where P responds to each $\mathit{write}(i)$ with ok and plays the most recently written value in response to read (or n if no $\mathit{write}(i)$ has been played by O yet).

Local semaphore introduction is defined similarly:

$$[\![\Gamma \vdash \mathbf{newsem}\, x := n \,\mathbf{in}\, M : \beta]\!] = \Lambda_x([\![\Gamma, x : \mathbf{sem} \vdash M : \beta]\!]); \mathsf{lock}_n^\beta,$$

where $\mathsf{lock}_n^\beta : ([\![\mathbf{sem}]\!] \Rightarrow [\![\beta]\!]) \Rightarrow [\![\beta]\!]$ is the least strategy containing plays of the shape $q \cdot q \cdot \Box \cdot a \cdot a$ where $\Box$ is a segment of alternating $\mathit{grab} \cdot \mathit{ok}$ and $\mathit{release} \cdot \mathit{ok}$ sequences.

Equivalently, by Lemma 3, instead of cell_n^β and lock_n^β one can use the *saturated* strategies $\mathsf{sat}(\mathsf{cell}_n^\beta)$ and $\mathsf{sat}(\mathsf{lock}_n^\beta)$. Therefore, the above definitions always lead to saturated strategies, i.e. morphisms of $\mathcal{G}_{\mathrm{sat}}$.

3.7 Examples

Example 1 (Nondeterminism). Let $M_0, M_1 : \beta$. Define $M_0 \,\mathbf{or}\, M_1 : \beta$ as

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

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

Example 2 (Test of linearity). Consider a term $\Gamma \vdash M : \beta$ 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 $p : \beta \to \beta' \vdash \mathbf{newsem}\, s := 0 \,\mathbf{in}\, p(\mathbf{grab}(s); M)$. Observe that instantiating p to $\lambda c\!:\!\mathbf{com}.c; c$ or $\lambda c\!:\!\mathbf{com}.c \,||\, c$ will not lead to convergence and that the corresponding strategy has no complete plays.

This construction can be extended to other types as in the case of $\mathbf{or}$ (Example 1) and plays an important role in the definability argument.

Example 3 (Test of linear parallelism). The following term generates only non-alternating complete plays:

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

They are generated, using saturation, by: $\mathit{run}_4 \cdot \mathit{run}_3 \cdot \mathit{run}_2 \cdot \mathit{run}_1 \cdot \mathit{ok}_2 \cdot \mathit{ok}_1 \cdot \mathit{ok}_3 \cdot \mathit{ok}_4$ in $([\![\mathbf{com}]\!]_1 \Rightarrow [\![\mathbf{com}]\!]_2 \Rightarrow [\![\mathbf{com}]\!]_3) \to [\![\mathbf{com}]\!]_4$. 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 [12]). 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 2 and 3, where the competing threads are produced by the same piece of code. This apparent expressivity failure has motivated the introduction of semaphores as a primitive in our language.

4 Soundness and Adequacy

Although $\mathcal{G}_{\text{sat}}$ can be shown to be inequationally sound, it is not fully abstract. As is the case for all game models, full abstraction will be proved for the quotient of $\mathcal{G}_{\text{sat}}$ 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 IA, 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 essentially 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 by $\tau_1 \lesssim \tau_2$ iff $\forall \alpha \in \mathcal{G}_{\text{sat}}(A, \Sigma)$ if $\tau_1 ; \alpha = \top_\Sigma$ then $\tau_2 ; \alpha = \top_\Sigma$. For composition the strategies $\tau_i : A$ are regarded as ones between 1 and A.

Theorem 2 (Characterization). *Let τ_1, τ_2 be saturated strategies on A. $\tau_1 \lesssim \tau_2$ if and only if $\mathsf{comp}(\tau_1) \subseteq \mathsf{comp}(\tau_2)$.*

Because the quotient $\mathcal{G}_{\text{qsat}} = \mathcal{G}_{\text{sat}} / \lesssim$ 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 $\mathcal{G}_{\text{qsat}}$ are precisely the equivalence classes $[\sigma]_\lesssim$ such that $\mathsf{comp}(\sigma)$ is finite. Next we examine the theoretical properties of $\mathcal{G}_{\text{qsat}}$: soundness, adequacy and, finally, full-abstraction.

For the purpose of relating our model with the operational semantics we will represent a state $s : \Sigma \to \mathbb{N}$ by the strategy $[\![s]\!]_\beta : \big([\![\theta_1]\!] \Rightarrow \cdots \Rightarrow [\![\theta_m]\!] \Rightarrow [\![\beta]\!]\big) \Rightarrow [\![\beta]\!]$ generated from complete plays of the shape $q \cdot q \cdot \Box \cdot a \cdot a$ where $\Box$ 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 that the projections onto θ_i are of the same shape as those of suitably initialized (i.e. $n_i = s(l_i)$) $\mathsf{cell}^\beta_{n_i}$ or $\mathsf{lock}^\beta_{n_i}$ strategies. We can think of $[\![s]\!]_\beta$ as a "super-sequentialized" store, where individual cells and locks are accessed sequentially both individually and as a group. In what follows, $[\![\Sigma \vdash M : \beta]\!] ; [\![s]\!]_\beta$ will be the interpretation of $\Sigma \vdash M : \beta$ at state s. Recall that, by Lemma 3, the same result would be achieved by using $\mathsf{sat}([\![s]\!]_\beta)$.

Lemma 4. *For any term $\Sigma \vdash M : \beta$ and any any state s, if $\Sigma \vdash M, s \longrightarrow M', s'$ then $[\![\lambda \vec{x}.M']\!] ; [\![s']\!]_\beta \subseteq [\![\lambda \vec{x}.M]\!] ; [\![s]\!]_\beta$.*

Soundness then follows.

Proposition 2 (Soundness). *For any $\Sigma \vdash M : \beta$ and any state s, if $\Sigma \vdash M, s \Downarrow$ then $[\![\lambda \vec{x}.M]\!] ; [\![s]\!]_\beta \neq \bot$.*

Our semantic model is adequate in the usual sense. The proof uses logical relations, adapted to small-step operational semantics.

Proposition 3 (Computational adequacy). *For any program P, $P \Downarrow$ if and only if $[\![P]\!] \neq \bot$.*

Together, the two propositions imply:

Theorem 3 (Inequational Soundness). *Let $\Gamma \vdash M_i : \theta$ for $i = 1, 2$. If $[\![M_1]\!] \lesssim [\![M_2]\!]$ then $M_1 \sqsubseteq_\theta M_2$. Equivalently, $\mathsf{comp}([\![M_1]\!]) \subseteq \mathsf{comp}([\![M_2]\!])$ implies $M_1 \sqsubseteq_\theta M_2$.*

5 Full Abstraction

We give a direct recursive algorithm, called $PROC^+$, which, given a position s of $[\![\theta]\!]$, returns a term of type θ whose denotation is the smallest saturated strategy containing s.

The basic idea of the construction is to use justification pointers to identify potential threads. 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 compose all these threads in parallel. Then we use specially designated side-effects as time-stamps to enforce the particular order of moves that happens in the position. Of course, we can only try to achieve this up to the saturation conditions.

In order to generate the desired positions 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 synchronization policies, restricting the order of moves present in the original sequence up to the reorderings dictated by the saturation conditions. Each O-move s_j produces an associated time-stamp which is stored in a variable x_j, bound by **new** at the top level and initialized to 0. We "time-stamp" the variable by assigning 1 to it. For $1 \leq j \leq |s| - 1$, let $O_j = \{\, i \in \mathbb{N} \mid 0 \leq i < j,\ s_i \text{ is an O-move} \,\}$. Let $\mathbf{test} \equiv \lambda x : \mathbf{exp}.\mathbf{ifzero}\, x\, \mathbf{then}\, \mathbf{skip}\, \mathbf{else}\, \Omega_{\mathbf{com}}$. We define $WAIT_j$ as the guard which checks for time-stamps originating from all the O-moves with indices smaller than j: $WAIT_j \equiv \mathbf{test}(1 - !x_{g_1}); \cdots ; \mathbf{test}(1 - !x_{g_k})$, with $O_j = \{g_1, \ldots, g_k\}$.

Below we give the definition of $PROC^+(s : \theta)$ for the case where θ is generated from **com** only (see also the example). The complete definition is available in [13].

$PROC^+$ first calls $PROC$ and then adds bindings to the term returned by $PROC$. 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. $PROC$ uses indices relative to the original s; we write s_i for the ith move of s, assuming s_0 initial. $PROC(t : \theta)$ where $\theta = \theta_1 \to \ldots \to \theta_h \to \mathbf{com}$ is defined 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).

$$\begin{array}{ccccccccccccc} s_i & \cdots & s_{i_1} & \cdots & s_{i_j} & \cdots & s_{j_{k,1}} & \cdots & s_{i_m} & \cdots & s_{j_{k,m_k}} & \cdots \\ o & & p_- & & p_x & & o_k & & p_- & & o_k & \end{array}$$

Fig. 1. Questions and justification pointers

---

1. Let $p_1, \cdots, p_h$ be all the P-questions enabled by o (corresponding respectively to $\theta_1, \cdots, \theta_h$). Let $i_1 < \cdots < i_m$ be the s-indices of all occurrences of $p_1, \cdots, p_h$ in t which are explicitly justified by s_i (see Figure 1). Then $PROC$ returns $\lambda p_1 \cdots p_h.(x_i := 1); (P_1 \,||\, \cdots \,||\, P_m); PANS_i^{\mathbf{com}}$ where $P_1, \cdots, P_m$ are defined in 2. and $PANS_i^{\mathbf{com}}$ is either $\Omega_{\mathbf{com}}$ (if s_i is unanswered in t) or $WAIT_{i'}$ (if $s_{i'}$ answers s_i in t). By convention, $(P_1 \,||\, \cdots \,||\, P_m)$ degenerates to **skip** for $m = 0$.
2. Here we show how to define he terms P_j for $1 \leq j \leq m$. Let us fix j and suppose that $s_{i_j} = p_x$ $(1 \leq x \leq h)$ and $\theta_x = \theta'_1 \to \ldots \to \theta'_n \to \mathbf{com}$. Let $o_1, \cdots, o_n$ be all the O-questions enabled by p_x (corresponding to $\theta'_1, \cdots, \theta'_n$ respectively).

   For each k $(1 \leq k \leq n)$ 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 Figure 1).

   If $m_k = 0$, then $P_j^k \equiv \Omega_{\theta'_k}$. Otherwise, for all $l = 1, \cdots, m_k$ 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 1, and $ONCE_w[M] = \mathbf{grab}(w); M$ as in Example 2.

   $P_j \equiv WAIT_{i_j}; (p_x P_j^1 \cdots P_j^n); OANS_{i_j}^{\mathbf{com}}$ where $OANS_c^{\mathbf{com}}$ is **skip** (if s_c is unanswered in t) or $x_{c'} := 1$ (when $s_{c'}$ answers s_c in t).

After $PROC(s : \theta)$ returns $\lambda p_1 \cdots p_k.M$, all variables and semaphores (x_-, 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) by taking

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

We denote the final term by $PROC^+(s : \theta)$.

Example 4. Consider the play

$$\begin{array}{ccccccccc} s = & run_4 & run_2 & run_1 & run_3 & ok_3 & ok_1 & ok_2 & ok_4 \\ & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}$$

in arena $[\![(\mathbf{com}_1 \to \mathbf{com}_2) \to \mathbf{com}_3 \to \mathbf{com}_4]\!]$. The term $\lambda f.\lambda x.fx$ has this play among its complete positions. The term $PROC^{+}(s)$ is:

$$\begin{array}{l}\lambda f.\lambda a.\mathbf{newvar}\, x_0, x_2, x_4, x_6 := 0\,\mathbf{in}\,\mathbf{newsem}\, w_2 := 0\,\mathbf{in}\\ \quad x_0 := 1;\\ \quad\quad ((\mathit{WAIT}_1; f(\mathit{ONCE}_{w_2}[x_2 := 1; \mathit{WAIT}_5]); x_6 := 1) \,||\, (\mathit{WAIT}_3; a; x_4 := 1));\\ \quad \mathit{WAIT}_7\end{array}$$

Notice that the second argument a can be evaluated only after the first one (f) is, because of WAIT_3. On the other hand, a must be evaluated before f's argument because of WAIT_5. The resulting temporal ordering of the moves is, consequently, the same as in $f(a)$.

Using $PROC^{+}$ and **or** we can show the following result:

Theorem 4 (Compact Definability). *Any compact saturated strategy σ, i.e. one generated by a finite set of positions, is definable.*

With adequacy and definability established, full abstraction follows routinely.

Theorem 5 (Full abstraction). *Let $\Gamma \vdash M, N : \theta$. Then $[\![M]\!] \lesssim [\![N]\!]$ if and only if $M \sqsubseteq_{\theta} N$. Equivalently, by the Characterization Theorem (Thm. 2), $\mathsf{comp}([\![M]\!]) \subseteq \mathsf{comp}([\![N]\!])$ if and only if $M \sqsubseteq_{\theta} N$.*

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 for modeling sequential computation can be avoided. Therefore, we can give a direct definability construction, as opposed to the usual factorization method.

In addition to its theoretical interest, our fully abstract model can be used to reason about program may-equivalence. We can make straightforward arguments about ground-type equivalences such as Brookes's *laws of parallel programming* [1], 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 [14]. For this purpose, the most convenient representations are those which are finite-state, such as regular expressions, regular languages, labelled transitions systems, etc. Such a representation can be easily integrated in our ongoing research effort in game-based software model checking [15]. However, identifying a non-trivial fragment of this language for which the strategies are finitary is not straightforward.

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 nondeterminism by Harmer and McCusker [16], and some of their techniques may be applicable in our setting.

Acknowledgements

We are grateful to Samson Abramsky and Guy McCusker for stimulating discussions on the subject matter of the paper.

References

1. Brookes, S.: The essence of Parallel Algol. Chapter 21 of [2].
2. O'Hearn, P.W., Tennent, R.D., eds.: ALGOL-like Languages. Progress in Theoretical Computer Science. Birkhäuser (1997)
3. Abramsky, S., McCusker, G.: Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. Chapter 20 of [2].
4. Hyland, J. M. E., Ong, C.-H. L.: On full abstraction for PCF: I, II and III. Information and Computation **163(2)** (2000) 285–408
5. Abramsky, S., Melliès, P. A.: Concurrent games and full completeness. In: Proceedings of LICS (1999) 431–442
6. Abramsky, S.: Game semantics of Idealized Parallel Algol. Lecture given at the Newton Institute (1995)
7. Laird, J.: A games semantics of Idealized CSP. In: Volume 45 of Electronic Notes in Theoretical Computer Science (2001) 157–176
8. Brookes, S.: Full abstraction for a shared variable parallel language. Chapter 21 of [2].
9. Röckl, C., Sangiorgi, D.: A π-calculus process semantics of Concurrent Idealized Algol. In: Proceedings of FOSSACS, Volume 1578 of LNCS (1999) 306–322
10. Udding, J.T.: A formal model for defining and classifying delay-insensitive circuits and systems. Distributed Computing **1(4)** (1986) 197–204
11. Jifeng, H., Josephs, M.B., Hoare, C.A.R.: A theory of synchrony and asynchrony. In: Programming Concepts and Methods. Elsevier (1990) 459–473
12. Andrews, G.: Concurrent Programming: principles and practice. Addison-Wesley Publishing Company (1991)
13. Ghica, D. R., Murawski, A. S.: Angelic semantics of fine-grained concurrency. Technical Report PRG-RR-03-20. Oxford University Computing Laboratory (2003)
14. Ghica, D. R., McCusker, G.: Reasoning about Idealized ALGOL using regular languages. In: Proceedings of ICALP. Volume 1853 of LNCS (2000) 103–116
15. Abramsky, S., Ghica, D. R., Murawski, A. S., Ong, C.-H. L.: Algorithmic game semantics and component-based verification. In: Proceedings of TACAS. LNCS (2004)
16. Harmer, R., McCusker, G.: A fully abstract game semantics for finite nondeterminism. In: Proceedings of LICS (1999) 422–430