A Fully Abstract Game Semantics for Parallelism with Non-Blocking Synchronization on Shared Variables

Susumu Nishimura

Dept. of Mathematics, Graduate School of Science, Kyoto University
Sakyo-ku, Kyoto 606-8502, JAPAN
`susumu@math.kyoto-u.ac.jp`

Abstract

We present a fully abstract game semantics for an Algol-like parallel language with non-blocking synchronization primitive. Elaborating on Harmer's game model for nondeterminism, we develop a game framework appropriate for modeling parallelism. The game is a sophistication of the wait-notify game proposed in a previous work, which makes the signals for thread scheduling explicit with a certain set of extra moves. The extra moves induce a Kleisli category of games, on which we develop a game semantics of the Algol-like parallel language and establish the full abstraction result with a significant use of the non-blocking synchronization operation.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages, F.1.1 Models of Computation, F.1.2 Modes of Computation

Keywords and phrases shared variable parallelism, non-blocking synchronization, full abstraction, game semantics

Digital Object Identifier 10.4230/LIPIcs.CSL.2013.578

1 Introduction

In shared memory parallel programming, parallel threads competing for shared memory cells (or shared variables) must be appropriately synchronized to avoid race conditions. A synchronization method is called *non-blocking*, if each individual thread spins over a shared resource until it acquires an exclusive access to it. In contemporary architectures including multicores, non-blocking synchronization is supported via the read-modify-write operation, most notably known as *compare-and-set (CAS)* operation. [12]

This paper concerns with game theoretical analysis of an Algol-like parallel language that supports non-blocking synchronization on shared variables. Game semantics for PCF and Idealized Algol have been well investigated and shown fully abstract [13, 3]. However, the standard methods used in the game modeling do not directly apply to the parallel extension considered in this paper:

- The models for the above deterministic languages solely concern *may-convergence*, i.e., they just observe if a program has the possibility of termination. The parallel programs, on the other hand, are inherently nondeterministic and thus may-convergence is too imprecise to give a pleasant discrimination of parallel programs: Even if two programs are judged equivalent, they can nondeterministically exhibit different convergences.
- One might expect that the parallel execution would be modeled by interleaved game plays of simultaneously running threads, but this fails to properly shuffle variable accesses, due to the parity restriction originating from the Hyland-Ong game [13], in which the opponent moves and the player moves must strictly alternate.

In a preliminary work [20], Watanabe and the present author proposed *wait-notify games* as a means to remedy the above issues. They developed the wait-notify games based on Harmer's game semantics [9, 10], in order to capture the nondeterministic nature of parallel computation more precisely. Harmer's games substantially extend Hyland-Ong's with the notion of divergence, giving the full abstraction result for a nondeterministic variant of Algol-like sequential language. It concerns both may-convergence and *must-convergence*, i.e., it also discriminates those programs which are obliged to terminate from those which are not.

The fundamental idea in wait-notify games is to have each game play interspersed by a suitable number of pairs of extra wait and notify moves, written `W` and `N`, respectively. A wait move `W` represents a delay imposed by the scheduler of the operating system, each time a single execution thread attempts to access a shared variable; A subsequent notify move `N` represents the resumption of the delayed variable access by the scheduler. However, wait-notify games are defined only for the type of parallel computation and are not well integrated with the computational structure of other types, including higher-order ones. The resulting parallel language thereby supports parallelism only under a fairly limited context: Within parallel contexts, nothing but shared variables can be parametrized.

The present paper sophisticates the idea in the wait-notify game to give a fully abstract game semantics for an Algol-like parallel language with non-blocking synchronization, in which parallelism is allowed under much wider contexts of arbitrary types, though subject to a few modest syntactic restrictions. We will develop the game model in a Kleisli category of games, induced from the monadic structure introduced by the wait and notify moves. This not only enables us to reach to the full abstraction result in a standard way but also reveals the computational structure hindered behind the parallel computation.

Here we emphasize that we do *not* intend to model parallel computation as a game between individual parallel threads. Rather, we model parallel computation as a game between the collection of simultaneously running threads and the scheduler, which is the entity invisible in the program text. The extra wait and notify moves enable the game semantical construction with the scheduler's interference explicit. The extra moves, on the other hand, should not be counted when we discuss observational behavior of programs. Thus we need to introduce a scheduler strategy that ignores these extra moves all together, later in Section 5.

The development in this paper also gives some indications on the nature of parallel computing:

- As we will discuss later, the game model in this paper has no ability to observe the termination of the entire collection of threads running in parallel. This implies that no language whose parallel running threads can join to a single sequential thread would be fully abstract with respect to the present game model or its modest extension. Due to this fact, we are driven to design our parallel language so that no parallel threads join: There is no means to merge the set of parallel threads into a single sequential thread, even after all the parallel threads have terminated.
- In the course of establishing full abstraction, we need to separate out a history-insensitive part from a given game strategy. This is usually done by the so-called *innocent factorization* [3], which does not directly apply to our parallel setting, though. The parallel threads would compete for a shared variable in which the factorized strategy keeps the history. There we make an indispensable use of the non-blocking synchronization operation CAS, as a means for mutual exclusion on the shared variable. This indicates that a language without CAS or its equivalent would be strictly less able to define strategies than the language with CAS would be.

Related work. The game model in this paper can be seen as a resumption model, which has been used for denotational modeling of concurrency [17, 19] and later examined for giving a game semantics for parallelism [1]. The present paper investigates the computational structure in the resumption-style (wait-notify) game and develops the full abstraction result for a suitable Algol-like shared variable parallel language.

There have been several approaches to full abstraction for shared variable parallel languages, e.g., [5] by Brookes and [7, 8] by Ghica, Murawski, et al. These studies concern blocking synchronization primitives, i.e., some locking mechanisms, while the present paper concerns non-blocking ones. More significantly, whereas they solely discuss may-convergence, we discuss may&must-convergence. This enables a more precise analysis of parallel computation that is inherently nondeterministic, though we need certain quotienting on the game model.

Unlike in the parallel languages they consider, parallel threads in our language never join to continue with a sequential computation, as mentioned earlier. This rigid distinction of sequential and parallel computation in our language might be more suitable for distributed computing, where a collection of parallel threads execute in concert but their computation results not necessarily need to coalesce. In a distributed environment, the indefinability without CAS might seem a reminiscent of the impossibility result for the wait-free distributed consensus [11], but one should mind their difference. Wait-freeness assumes a fairness in scheduling, while the present game model does not: A parallel computation in the latter is judged to diverge as soon as at least one out of the parallel threads does, while the former is judged irrespective of divergence of a subset of threads.

Outline. The rest of the paper is organized as follows. Section 2 introduces an Algol-like parallel language and gives its operational semantics. Reviewing Harmer's games for nondeterminism in Section 3, we develop in Section 4 a game framework, in which parallel programs are interpreted in a Kleisli category of wait-notify games. Section 5 defines the game interpretation for terms and its soundness is shown. In Section 6, we show the full abstraction, which is derived by combining two factorizations followed by a definability result. Finally, Section 7 concludes the paper with some topics for future investigations.

2 The IA_{par} Language

Let us define a programming language for shared variable parallelism, called IA_{par}, which is yet another variant of Idealized Algol [18]. IA_{par} is a strongly typed language, whose types and syntax are defined as below.

$$
\begin{aligned}
\mathtt{T} &::= \mathtt{nat} \mid \mathtt{com} \mid \mathtt{var} \mid \mathtt{par} \mid \mathtt{T} \to \mathtt{T} \\
M &::= x \mid n \mid M \star M \mid \lambda x^{\mathtt{T}}.M \mid MM \mid \mathsf{fix}_{\mathtt{T}}\, M \mid \mathsf{skip} \mid \mathsf{seq}\ M\ M \mid \mathsf{if0}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M \\
&\quad \mid \mathsf{assign}\ M\ M \mid \mathsf{deref}\ M \mid \mathsf{cas}\ M\ M\ M \mid \mathsf{mkvar}\ M\ M\ M \mid \mathsf{newvar}\ v = n\ \mathsf{in}\ M \\
&\quad \mid M\ \mathsf{or}\ M \mid M_1 \| \cdots \| M_\zeta \mid M \gg_j M
\end{aligned}
$$

The types consist of base types and arrow types built from them. Base types are either `nat` for natural numbers, `com` for sequential commands, `var` for mutable variables, or `par` for parallel commands. A *type judgment* of the form $\Gamma \vdash M : \mathtt{T}$ assigns the type τ for the term M, where Γ is a *typing context*, a finite mapping from identifiers to types. The typing rules are given in Figure 1 of Appendix A.

The terms consist of PCF terms (natural numbers, λ-terms, and general recursion, where the binary operator $\star$ on natural numbers at least includes the addition $+$ and the cut-off subtraction $-$, Algol terms (commands for mutable variables and the bad variable constructor mkvar), the erratic nondeterminism or [9, 10], and parallel constructs. The *parallel command* $M_1 \| \cdots \| M_\zeta$ executes the sequential commands M_1, ..., M_ζ simultaneously, where ζ is the fixed degree of parallelism. The *thread extension* $P \gg_j M$ $(1 \leq j \leq \zeta)$ extends the command execution of the j-th thread in the parallel command P by a sequential command M. That is, as soon as the execution in the j-th thread as specified in P has terminated, the same thread continues to execute the sequential command M. The *preamble sequencing* $\mathsf{seq}\ M\ P$, where P is of type $\mathtt{par}$, executes the sequential command M in advance of the parallel command P. Within a parallel command, every simultaneously running thread has parallel access to shared variables, which are ranged over by the identifiers v, v', ... of type $\mathtt{var}$. In addition to the primitives assign and deref for atomic read and write on mutable variables, respectively, it also provides compare-and-set (CAS) operation as a means for non-blocking synchronization on shared variables. A CAS operation $\mathsf{cas}\ v\ m\ n$ is an atomic uninterruptible operation that conditionally updates the value stored in v: If the present value stored in v is equal to m then it updates the value to n and returns n; otherwise, it leaves v as it is and returns the stored value.

The formal operational semantics for IA_{par} is given in the style of small-step operational semantics (Figure 2 of Appendix A). Each *1-step reduction* $\langle M, s\rangle \longrightarrow \langle M', s'\rangle$ corresponds to a single atomic sequential execution that may update the store s to s' as the side effect, where a *store* is a finite mapping from mutable variables to natural numbers.

We say the evaluation of a term M *may-converges* at initial state s, if $\langle M, s\rangle \longrightarrow^* \langle M', s'\rangle$ for some state s' and no reduction rules apply to $\langle M', s'\rangle$ further, where $\longrightarrow^*$ is the reflexive transitive closure of $\longrightarrow$. Also, we say the evaluation of a term M *must-converges* at initial state s, if there is no infinite reduction sequence $\langle M, s\rangle \longrightarrow \langle M', s'\rangle \longrightarrow \cdots$. In particular when M is a closed term, we write $M{\Downarrow}^{\mathrm{may}}$ for may-convergence and also write $M{\Downarrow}^{\mathrm{must}}$ for must-convergence.

A *contextual preorder* on terms is defined by means of both may- and must-convergences. We define $M \lesssim_{\mathrm{may}} N$ iff, for any context $C[-]$ of type $\mathtt{par}$, $C[M]{\Downarrow}^{\mathrm{may}}$ implies $C[N]{\Downarrow}^{\mathrm{may}}$. Also, $M \lesssim_{\mathrm{must}} N$ iff, for any context $C[-]$ of type $\mathtt{par}$, $C[M]{\Downarrow}^{\mathrm{must}}$ implies $C[N]{\Downarrow}^{\mathrm{must}}$. Overall, we define the approximation of convergence by: $M \lesssim_{\mathrm{m\&m}} N$ iff $M \lesssim_{\mathrm{may}} N$ and $M \lesssim_{\mathrm{must}} N$.

3 Harmer's game for nondeterminism

This section reviews Harmer's game model together with a little sophistication specifically needed for the development in the present paper. It is intended to make the present paper self-contained as much as possible, but some details are omitted due to page limitation. For the full details, see Harmer's thesis [10]. More general aspects on game semantics for Algol-like languages can be found in [4].

Arenas. The definition of arenas is standard, except that the arena moves are ordered. An *arena* A is a triple $((M_A, <_A), \lambda_A, \vdash_A)$, where M_A is a countable set of *moves*, associated with a nonreflexive total order $<_A$ on them; $\lambda_A : M_A \to \{\mathsf{O}, \mathsf{P}\} \times \{\mathsf{Q}, \mathsf{A}\}$ is a *labeling* function that assigns each move $m \in M_A$ its attributes, either O (opponent) or P (player) and either Q (question) or A (answer); the *enabling relation* $\vdash_A$ is a binary relation over the moves satisfying: (e1) $(a \vdash_A b \wedge a \neq b) \implies \lambda_A^{\mathsf{OP}}(a) \neq \lambda_A^{\mathsf{OP}}(b)$; (e2) $b \vdash_A b \implies (\lambda_A(b) = (\mathsf{O}, \mathsf{Q}) \wedge (a \neq b \implies a \not\vdash_A b))$; (e3) $(a \vdash_A b \wedge \lambda_A^{\mathsf{QA}}(b) = \mathsf{A}) \implies \lambda_A^{\mathsf{QA}}(a) = \mathsf{Q}$, where we write $\lambda_A^{\mathsf{OP}}(b)$ (resp., $\lambda_A^{\mathsf{QA}}(b)$) for the opponent/player (resp., question/answer) attribution of the move b. When $a \vdash_A b$ $(a \neq b)$, we say *a justifies b.* A move a is called an *initial move* if $a \vdash_A a$.

The most trivial arena is $\mathbf{1} = (\emptyset, \emptyset, \emptyset)$. The arena $\mathbf{C} = ((M_{\mathbf{C}}, <_{\mathbf{C}}), \lambda_{\mathbf{C}}, \vdash_{\mathbf{C}})$ corresponding to type `com` of commands is specified, as usual, by the data $M_{\mathbf{C}} = \{\texttt{run}, \texttt{done}\}$, $\lambda_{\mathbf{C}}(\texttt{run})=(\mathsf{O}, \mathsf{Q})$, $\lambda_{\mathbf{C}}(\texttt{done}) = (\mathsf{P}, \mathsf{A})$, $\vdash_{\mathbf{C}} = \{(\texttt{run}, \texttt{run}), (\texttt{run}, \texttt{done})\}$, together with the ordering $\texttt{run} <_{\mathbf{C}} \texttt{done}$. For the arenas of other base types and type constructors, see Appendix B.

Here we notice that the arena definition does *not* preclude the possibility that question moves are justified by answer moves. We say such a question move justified by an answer move a a *subquestion* move (inferior to a). Let us write $\mathit{Subq}_A(a) = \{q \mid a \vdash_A q\}$ for the set of all subquestions inferior to the answer move a. We call an answer move a a *subquestioning answer* iff $\mathit{Subq}_A(a) \neq \emptyset$. Throughout the paper, we assume that $\mathit{Subq}_A(a)$ is a finite set for each answer move a. In this paper, subquestion moves in lifted arenas will play a significant role in modeling parallelism (Section 4.1).

Justified strings, legal plays, and strategies. Let us write ε for an empty string of arena moves and st (occasionally written $s \cdot t$ for clarity) for concatenation of strings of moves s and t. We write $s \sqsubseteq t$ to mean that s is a prefix of t and also write $s \sqsubseteq^{\text{even}} t$ in particular when s is an even-length string. Typically, strings of arena moves are ranged over by s, t, u, etc., while arena moves are ranged over by a, b, p, q, etc.

A *justified string* in an arena A is a sequence of moves in which every non-initial move p has a pointer to an earlier occurrence of a justifying move q (i.e., $q \vdash p$), written like $\cdots\ q \overset{\curvearrowleft}{\cdots}\ p\ \cdots$. In particular when p is an answer move (and hence q is a question move), we say "p answers q."

We say a justified string s is *well-opened*, if s has at most a single occurrence of initial move. We also define the *player view* of a justified string s, denoted by $\mathsf{V}(s)$, by induction on the length of s as follows: (i) $\mathsf{V}(sq) = q$ if q is initial; (ii) $\mathsf{V}(sqtp) = \mathsf{V}(s)qp$ if p is an opponent move and q justifies p; (iii) $\mathsf{V}(sq) = \mathsf{V}(s)q$ if q is a player move.

A justified string s in arena A is called a *legal play*, if s strictly alternates opponent/player moves, that is, $s = o_1 p_1 o_2 p_2 \cdots$ where o_i's are opponents and p_i's are players. We write L_A to denote the set of legal plays in the arena A and also L_A^{even} (resp., L_A^{odd}) to denote the set of even (resp., odd) length legal plays.

In order to appropriately model may&must-convergence in nondeterministic programs, Harmer defined each game strategy by a pair of execution traces and witnesses of divergence.

A *strategy* σ is a pair (T_σ, D_σ), where the trace set T_σ is an even-length prefix closed subset of L_A^{even} and the divergence set D_σ is a subset of L_A^{odd} satisfying: (d1) if $s \in T_\sigma$, $sa \in L_A$, and $sa \notin \mathit{dom}(\sigma)$, then there exists $d \in D_\sigma$ such that $d \sqsubseteq sa$; (d2) if $sa \in D_\sigma$, then $s \in T_\sigma$; If $\mathit{rng}_\sigma(sa)$ is an infinite set, then there exists $d \in D_\sigma$ such that $d \sqsubseteq sa$, where $\mathit{rng}_\sigma(sa) = \{sab \mid sab \in T_\sigma\}$ is the *range* of moves that follows sa and $\mathit{dom}(\sigma) = \{sa \mid \mathit{rng}_\sigma(sa) \neq \emptyset\}$ is the *domain* of σ.

We say a divergence $d \in D_\sigma$ *interesting*, if $d \in \mathit{dom}(\sigma)$; otherwise, it is called *uninteresting*. A strategy σ is called *deterministic* if $\mathit{rng}_\sigma(sa)$ is a singleton set for every $sa \in \mathit{dom}(\sigma)$; A strategy σ is called *reliable* if it is deterministic and further every $sa \in D_\sigma$ is uninteresting.

The composition of two strategies $\sigma : A \to B$ and $\tau : B \to C$, written $\sigma; \tau : A \to C$, is obtained by parallel composition and hiding on the pair of plays taken from each of the strategies. Given $s \in L_{A \to B}$ and $s' \in L_{B \to C}$, a parallel composition of s and s' is a play t of moves in $M_A \cup M_B \cup M_C$ such that s (resp., s') is a restriction of t to the moves $M_A \cup M_B$ (resp., $M_B \cup M_C$). Hiding the moves M_B occurring in t, we obtain a composite play.

The trace part $T_{\sigma;\tau}$ is the set of composite plays of any $s \in T_\sigma$ and $s' \in T_\tau$. The divergence $D_{\sigma;\tau}$ is the set of divergences that is a union of subsets generated by the following two ways. One subset is obtained by parallel composition and hiding on the pair of a trace from one strategy and a divergence from the other strategy. The other subset is obtained from *infinite* traces generated by a pair of traces taken from both: Let u^∞ be an infinite play of moves in $M_A \cup M_B \cup M_C$ such that only a finite number of moves from M_A or M_B are witnessed in u^∞. Then the restriction of u^∞ to $M_A \cup M_B$ moves is a divergent play, as u^∞ can be understood as exhibiting an *infinite chatter*, where two strategies make infinite (thus diverging) interaction with each other in the arena B.

We say strategy τ is more likely to converge than σ, denoted by $\sigma \leq^\natural \tau$, iff $T_\sigma \subseteq T_\tau \wedge \forall d' \in D_\tau . \exists d \in D_\sigma . d \sqsubseteq d' \wedge \forall sab.(sab \in T_\tau \wedge sab \notin T_\sigma \implies \exists d \in D_\sigma . d \sqsubseteq sab)$.

For every arena A, there is the least element $\bot_A$ subject to $\leq^\natural$, specified by $(T_{\bot_A}, D_{\bot_A}) = (\{\varepsilon\}, \{q \mid q\text{: initial}\})$. We will define $\sigma =^\natural \tau$ iff $\sigma \leq^\natural \tau$ and $\tau \leq^\natural \sigma$.

Strategy subclasses. Throughout the paper, we will only concern the class of *single-threaded* strategies [9, 10]. Intuitively, a single-threaded strategy consists of plays that are closed under arbitrary interleaving of several copies of plays. This intuition is supported by the fact that single-threaded strategies have a bijective correspondence with the so-called well-opened ones, representing a single execution of a sequential program.

We say a strategy σ is *well-opened* iff every play $s \in T_\sigma$ is well-opened and so is every interesting divergence $d \in D_\sigma$. The bijective correspondence between single-threaded strategies and well-opened ones, up to $=^\natural$, is established by a pair of mappings $\mathit{WO}(-)$ and $\mathit{ST}(-)$: for every single-threaded strategy σ and well-opened strategy υ, we have $\sigma =^\natural \mathit{ST}(\mathit{WO}(\sigma))$ and $\mathit{WO}(\mathit{ST}(\upsilon)) =^\natural \upsilon$, where $\mathit{WO}(\sigma)$ restricts the plays in σ to those well-opened ones and $\mathit{ST}(\upsilon)$ interleaves several copies of the well-opened plays in υ.

Due to this bijective correspondence, we may specify a single-threaded strategy σ by just giving the well-opened plays contained in the trace T_σ and the interesting divergences in D_σ. Every uninteresting divergence is implicitly identified by (d1), i.e., $sa \in L_A^{\mathrm{odd}}$ is identified as an uninteresting divergence whenever $s \in T_\sigma$ but $sa \notin \mathit{dom}(\sigma)$. Furthermore, the trace set can be identified by (the prefix closure of) the longest well-opened plays. Also, we may not even mention divergences, when the strategy has no interesting divergences. For example, when we say a single-threaded strategy $\sigma : \mathbf{C}$ is specified by the trace set $\{\mathtt{run} \cdot \mathtt{done}\}$ (of the longest well-opened plays), σ is formally a strategy defined by the pair $(T_\sigma, D_\sigma) = (\{(\mathtt{run}\cdot\mathtt{done})^k \mid k \geq 0\}, \{(\mathtt{run}\cdot\mathtt{done})^k \cdot \mathtt{run} \mid k \geq 1\})$.

In this paper, we will mostly concern a further limited class of strategies satisfying the following closure properties, except for the scheduler strategy to be presented in Section 5.

- A strategy σ is called *player visible*, if for every $sa \in T_\sigma$, the player move a is justified by a move in $\mathsf{V}(s)$; A strategy σ is called *player bracketing*, if for every $sa \in T_\sigma$, a is the answer to the *pending question*, i.e., the last occurrence of unanswered question in $\mathsf{V}(s)$.

Harmer gave a fully abstract semantics for Idealized Algol with erratic nondeterminism on a cartesian closed category $\mathcal{C}$ of games, whose objects are the arenas and morphisms are the ($=^\natural$-equivalence classes of) single-threaded strategies. The identity arrow assigned to an object A in $\mathcal{C}$ is the *copycat* strategy $id_A : A \to A$, which is specified by the trace set $T_{id_A} = \{s \in L^{\mathrm{even}}_{A_1 \to A_0} \mid \forall t \sqsubseteq^{\mathrm{even}} s.(t \upharpoonright A_1 = t \upharpoonright A_0)\}$.[1] Restricting morphisms in $\mathcal{C}$ to those player visible and player bracketing ones, we have a lluf subcategory, denoted by $\mathcal{C}_{vb}$.

[1] We may occasionally put subscripts or primes in order to distinguish different copies of the same arena.

In what follows, by abuse of notation, we denote each morphism in $\mathcal{C}$ (and their subcategories as well) by σ, a representative of the equivalence class containing it.

4 The Game for Parallelism

As usual, each IA_{par} term is assigned a game strategy whose arena is determined by the type of the term. The base types (except for `par`) and function types are each interpreted as $[\![\mathtt{nat}]\!] = \mathbf{N}$, $[\![\mathtt{com}]\!] = \mathbf{C}$, $[\![\mathtt{var}]\!] = \mathbf{Var}$, and $[\![\mathtt{T} \to \mathtt{T}]\!] = [\![\mathtt{T}]\!] \Rightarrow [\![\mathtt{T}]\!]$. Each typing context $\Gamma = x_1 : \mathtt{T}_1, \cdots x_k : \mathtt{T}_k$ is interpreted by a product arena, namely, $[\![\Gamma]\!] = [\![\mathtt{T}_1]\!] \times \cdots \times [\![\mathtt{T}_k]\!]$.

4.1 Interleaving game plays

While a single-threaded strategy just interleaves *independent* execution of threads, the execution of threads in a shared variable parallel program can be affected by the order of interleaved variable access operations. This possible dependency between threads can be properly modeled in lifted arenas [16, 10] within a single-threaded strategy.

- A *lifted arena* $A_\perp$ is a triple $((M_{A_\perp}, <_{A_\perp}), \lambda_{A_\perp}, \vdash_{A_\perp})$, where $M_{A_\perp} = \{?, \surd\} + M_A$, $\lambda_{A_\perp} = [\lambda', \lambda_A]$ with $\lambda'(?) = (\mathsf{O}, \mathsf{Q})$ and $\lambda'(\surd) = (\mathsf{P}, \mathsf{A})$, and $p \vdash_{A_\perp} q$ iff $p = q = ? \vee (p = ? \wedge q = \surd) \vee (p = \surd \wedge q \vdash_A q) \vee (p \vdash_A q \wedge p \neq q)$. The associated ordering extends $<_A$ with $? <_{A_\perp} \surd$ and also $p <_{A_\perp} q$ for every $p \in \{?, \surd\}$ and $q \in M_A$.

We interpret `par` type by $[\![\mathtt{par}]\!] = (\mathbf{C}_1 \times \cdots \times \mathbf{C}_\zeta)_\perp$. The individual threads in this arena is ordered by: $\mathtt{run}_1 <_{[\![\mathtt{par}]\!]} \mathtt{run}_2 <_{[\![\mathtt{par}]\!]} \cdots <_{[\![\mathtt{par}]\!]} \mathtt{run}_\zeta$. An IA_{par} term $\Gamma \vdash \mathtt{seq}\ M\ (M_1 \| \cdots \| M_\zeta) : \mathtt{par}$ is interpreted by a strategy whose play has the form $?\ s\ \surd\ t$, where s models the interaction of the command M with Γ and t models the execution of the subsequent parallel command $M_1 \| \cdots \| M_\zeta$. whose interleaved parallel execution is modeled by the subsequent play t. As mentioned in [10], the lifting construction gathers parallel threads of computation into a single sequence of play, allowing single-threaded strategies to express history-sensitive execution.

The arenas that interpret `par` and higher-types involving it, however, still contain some strategies that are not definable by the terms of IA_{par}: The language IA_{par} is carefully designed to force affine uses of parallel computational contents. Further, a parallel computational content cannot be converted to a sequential computational content either; it can only be either discarded or modified by means of a few parallel constructs.

A suitable (fully abstract) game model for IA_{par} is obtained by further restricting the class of strategies to *subquestion-affine* ones, which satisfy the following properties.

(sq1) For every $s{\cdot}?' \in T_\sigma$, if $?' \vdash_A \surd'$ for some subquestioning answer $\surd'$, then the occurrence of $?'$ is justified by an occurrence of $?$ in s, where $?$ is a question move satisfying $? \vdash_A \surd$ for some subquestioning answer $\surd$.

(sq2) For every $s{\cdot}\surd' \in \mathit{dom}(\sigma)$ where $\surd'$ is a subquestioning answer, $s{\cdot}\surd' \notin D_\sigma$ and $\mathit{rng}_\sigma(s{\cdot}\surd') = \{s{\cdot}\surd'{\cdot}\surd\}$ for some subquestioning answer $\surd$.

(sq3) For every $s{\cdot}\surd'{\cdot}\surd{\cdot}t{\cdot}q \in \mathit{dom}(\sigma)$ such that $\surd'$ and $\surd$ are subquestioning answers and $q \in \mathit{Subq}_A(\surd)$, it holds that $s{\cdot}\surd'{\cdot}\surd{\cdot}t{\cdot}q \notin D_\sigma$ and $\mathit{rng}_\sigma(s{\cdot}\surd'{\cdot}\surd{\cdot}t{\cdot}q) = \{s{\cdot}\surd'{\cdot}\surd{\cdot}t{\cdot}q{\cdot}\rho(q)\}$, where $\rho(q)$ is justified by $\surd'$ and $\rho : \mathit{Subq}_A(\surd) \mapsto \mathit{Subq}_A(\surd')$ is the bijection that preserves the order $<_A$.

(sq4) For every $u{\cdot}b{\cdot}\surd{\cdot}s{\cdot}q{\cdot}t \in T_\sigma$ where q is a subquestion move justified by $\surd$ which is further justified by an initial move, if $u{\cdot}b{\cdot}\surd{\cdot}s{\cdot}q{\cdot}t{\cdot}q \in L_A^{\mathrm{odd}}$ in which the both occurrences of q are justified by $\surd$ and also b is *not* a subquestioning move, then $u{\cdot}b{\cdot}\surd{\cdot}s{\cdot}q{\cdot}t{\cdot}q \notin \mathit{dom}(\sigma)$.

To see how these conditions compel the affine use of parallel computation, let $\sigma \in [\![\mathtt{T}]\!]$ be a subquestion-affine strategy for some type $\mathtt{T}$. The condition (sq4) applies to the case where $\mathtt{T}$ has the form $\cdots \to \mathtt{par}$, with q being any $\mathtt{run}_j \in [\![\mathtt{par}]\!]$ $(1 \leq j \leq \zeta)$, prohibiting any duplicated occurrences of the same move $\mathtt{run}_j$. This compels each thread of a parallel command to execute once and only once.

The conditions (sq1)–(sq3) apply to the case where $\mathtt{T}$ contains a positive occurrence of $\mathtt{par}$ and a negative occurrence of $\mathtt{par}'$ in the form $(\mathtt{T}' \to \mathtt{par}') \to \cdots \to \mathtt{par}$. Any occurrence of $?' \in M_{[\![\mathtt{par}']\!]}$ in a trace of σ must be justified $? \in M_{[\![\mathtt{par}]\!]}$ [(sq1)]; Any occurrence of $\surd' \in M_{[\![\mathtt{par}']\!]}$ in σ must be immediately followed by $\surd \in M_{[\![\mathtt{par}']\!]}$ without diverging [(sq2)]; Any occurrence of $\mathtt{run}_j \in M_{[\![\mathtt{par}]\!]}$ in σ must be immediately followed by $\mathtt{run}'_j \in M_{[\![\mathtt{par}']\!]}$ without diverging, unless the subquestioning move $\surd$ that justifies $\mathtt{run}_j$ is ever preceded by $\surd'$ [(sq3)]. Thus a typical trace of σ has the form like: $?{\cdot}s{\cdot}?'\cdots\surd'{\cdot}\surd{\cdot}\mathtt{run}_1{\cdot}\mathtt{run}'_1\cdots\mathtt{run}_2{\cdot}\mathtt{run}'_2\cdots\mathtt{done}'_1{\cdot}t_1{\cdot}\mathtt{done}_1\cdots\mathtt{done}'_2{\cdot}t_2{\cdot}\mathtt{done}_2\cdots\mathtt{run}_1{\cdot}\mathtt{run}'_1\cdots$. It is intended that the corresponding function makes just a single copy of computation of the argument type $\mathtt{par}'$, possibly augmenting it by preamble sequencing and thread extension (as denoted by subsequences s and t_i's in the trace above, respectively). Notice that, as opposed to the case of (sq4) that does not copy parallel computation, the trace can contain duplicated occurrences of the same move $\mathtt{run}_j$, each immediately followed by $\mathtt{run}'_j$. These duplicates are not harmful, since they are superficial in a sense that solely the earliest one of the duplicates can come into play, eventually when the strategy is combined with some strategy in $[\![\mathtt{T}' \to \mathtt{par}']\!]$.

In what follows, we will develop a game semantical framework on a category $\mathcal{G}$ of games, a lluf subcategory of $\mathcal{C}_{vb}$, whose morphisms of player visible and player bracketing strategies are further restricted to subquestion-affine ones. The category $\mathcal{C}$ and their subcategories $\mathcal{C}_{vb}$ and $\mathcal{G}$ are cartesian closed. For any objects A and B, $A \Rightarrow B$ is the exponential object with its associated evaluation map $ev_{A,B} : (A \Rightarrow B) \times A \to B$ being a copycat strategy between the copies of arenas A and B, respectively. The currying isomorphism is written $\Lambda_{A,B}(f) : C \to (A \Rightarrow B)$ for every $f : C \times A \to B$.

4.2 Wait-notify games for shared variable access

We model interleaved access to shared variables in wait-notify games [20], as we discussed earlier. The arena $\mathbf{WN}$ of wait and notify moves is defined as below.

- $\mathbf{WN} = ((M_{\mathbf{WN}}, <_{\mathbf{WN}}), \lambda_{\mathbf{WN}}, \vdash_{\mathbf{WN}})$ is the arena of wait-notify signals, where $M_{\mathbf{WN}} = \{\mathtt{W}, \mathtt{N}\}$, $\lambda_{\mathbf{WN}}(\mathtt{W}) = (\mathsf{O}, \mathsf{Q})$, $\lambda_{\mathbf{WN}}(\mathtt{N}) = (\mathsf{P}, \mathsf{A})$, $\vdash_{\mathbf{WN}} = \{(\mathtt{W}, \mathtt{W}), (\mathtt{W}, \mathtt{N})\}$, and and $\mathtt{W} <_{\mathbf{WN}} \mathtt{N}$.

Let $F : \mathcal{G} \to \mathcal{G}$ be the functor $\mathbf{WN} \Rightarrow (-)$ and (F, η, μ, t) be the canonical commutative strong monad, where each natural transformation is given the following game interpretation:

- The unit $\eta_A : A \to \mathbf{WN} \Rightarrow A$ is a trivial copycat between the two copies of arena A, with no witnesses of $\mathbf{WN}$ moves.
- The multiplication $\mu_A : \mathbf{WN}_1 \Rightarrow (\mathbf{WN}_2 \Rightarrow A) \to \mathbf{WN}_0 \Rightarrow A$ is a copycat in which (i) each opponent move of an arena A is immediately copied by the same move of the other A arena; (ii) each opponent $\mathtt{W}$ move of $\mathbf{WN}_1$ or $\mathbf{WN}_2$ is copied by a $\mathtt{W}$ move of $\mathbf{WN}_0$; (iii) each opponent $\mathtt{N}$ move of $\mathbf{WN}_0$ is copied by a $\mathtt{N}$ move of $\mathbf{WN}_1$ (resp., $\mathbf{WN}_2$) when the opponent $\mathtt{N}$ move is justified by a $\mathtt{W}$ move that copies a move of $\mathbf{WN}_1$ (resp., $\mathbf{WN}_2$).
- The tensorial strength $t_{A,B} : A \times (\mathbf{WN} \Rightarrow B) \to \mathbf{WN} \Rightarrow (A \times B)$ is the trivial copycat between the arenas $A \times (\mathbf{WN} \Rightarrow B)$ and $\mathbf{WN} \Rightarrow (A \times B)$.

The monadic structure in $\mathcal{G}$, in the usual way, gives rise to the Kleisli category $\mathcal{G}_F$, whose objects are the same as $\mathcal{G}$ and the homset $\mathcal{G}_F(A, B)$ is $\mathcal{G}(A, FB)$. We write $f : A \rightarrow B$ to mean f is in $\mathcal{G}_F(A, B)$.[2] The composition of two Kleisli arrows $f : A \rightarrow B$, $g : B \rightarrow C$, written $f \fatsemi g$, is the morphism $f; g^* : A \to FC$ (in $\mathcal{G}$), where $g^* : FB \to FC$ is the Kleisli extension of g. For each object A, $\mathcal{G}_F$ has the identity arrow $\mathbf{id}_A : A \rightarrow A = \eta_A$ and the terminal arrow $\mathbf{!}_A : A \rightarrow \mathbf{1} = \Lambda_{\mathbf{WN}}(!_{A\times\mathbf{WN}})$.

The category $\mathcal{G}_F$ is also cartesian closed. For objects A and B, the exponential object is $A \Rightarrow B$ (the same as that of $\mathcal{G}$) and the evaluation map is given by $\mathbf{ev}_{A,B} : (A \Rightarrow B) \times A \rightarrow B = ev_{A,B}; \eta_B$. For each $f : C \times A \rightarrow B$, the currying isomorphism is given by $\boldsymbol{\Lambda}_A(f) : C \rightarrow (A \Rightarrow B) = \Lambda_A(f); \delta^{-1}_{A,B}$, where $\delta_{A,B} : F(A \Rightarrow B) \to A \Rightarrow FB$ is the trivial copycat strategy between the arenas $\mathbf{WN} \Rightarrow (A \Rightarrow B)$ and $A \Rightarrow (\mathbf{WN} \Rightarrow B)$. Given Kleisli arrows $f_1 : A \rightarrow B_1$, ..., $f_k : A \rightarrow B_k$, we write, as usual, $\langle f_1, \ldots, f_k \rangle : A \rightarrow B_1 \times \cdots \times B_k$ for pairing operations and $\boldsymbol{\pi}_i : B_1 \times \cdots \times B_k \to B_i$ $(1 \leq i \leq k)$ for projections. $\boldsymbol{\pi}_i$ may be instead written as $\boldsymbol{\pi}_{B_i}$ when there arises no confusion.

The categories $\mathcal{G}$ and $\mathcal{G}_F$ inherit some significant properties [10] from $\mathcal{C}$: They are CPO-enriched, where the underlying CPO is an algebraic CPO with the least element $\perp_A$ for every arena A, w.r.t. the ordering $\leq^\natural$. Furthermore, a strategy σ is a *compact* element in the CPO iff $T_{WO(\sigma)}$ is a finite set.

5 The Game Model and the Soundness

The game model is given in a quite standard way except that each IA_{par} term $\Gamma \vdash M : \mathtt{T}$ is interpreted by a Kleisli arrow in $\mathcal{G}_F$, denoted by $[\![\Gamma \vdash M : \mathtt{T}]\!] : [\![\Gamma]\!] \rightarrow [\![\mathtt{T}]\!]$, as given in Fig. 3 of Appendix B. (We may instead write $[\![\Gamma \vdash M]\!]$ or $[\![M]\!]$, unless ambiguity arises.) We notice that the strategy given to every term is player visible, player bracketing, and subquestion-affine.

The $\mathtt{W}$ and $\mathtt{N}$ moves in the Kleisli arrows model the wait and notify events that come into play when accessing shared variables: Each time a program tries to access a shared variable, it is forced to yield its execution to another running thread and *wait* until it is *notified* to resume execution after an arbitrary amount of delays. Thus the terms assign, deref, and cas are modeled by strategies whose every interaction with the $\mathbf{Var}$ arena is preceded by a sequence of moves $\mathtt{W}{\cdot}\mathtt{N}$, which are kept throughout a series of Kleisli compositions. Let us consider, for instance, a term $v : \mathtt{var} \vdash \mathsf{seq}\ M_1\ M_2$ where $M_1 = \mathsf{assign}\ v\ 1$ and $M_2 = \mathsf{assign}\ v\ 2$. The term $\mathsf{seq}\ M_1\ M_2$ is interpreted by the strategy $\langle [\![M_1]\!], [\![M_2]\!] \rangle \fatsemi seq_{\mathtt{com}} = \langle [\![M_1]\!], [\![M_2]\!] \rangle; seq^*_{\mathtt{com}}$, where the Kleisli extension $seq^*_{\mathtt{com}} : F'(\mathtt{com}_1 \times \mathtt{com}_2) \to F\mathtt{com}$ has a trace $\mathtt{run}{\cdot}\mathtt{run}_1{\cdot}\mathtt{W}'{\cdot}\mathtt{W}{\cdot}\mathtt{N}{\cdot}\mathtt{N}'{\cdot}\mathtt{done}_1{\cdot}\mathtt{run}_2{\cdot}\mathtt{W}'{\cdot}\mathtt{W}{\cdot}\mathtt{N}{\cdot}\mathtt{N}'{\cdot}\mathtt{done}_2{\cdot}\mathtt{done}$ that augments the trace in $seq_{\mathtt{com}} : \mathtt{com}_1 \times \mathtt{com}_2 \to F\mathtt{com}$ with extra $\mathbf{WN}$ moves that copies each $\mathbf{WN}$ move associated to $\mathtt{com}_1$ or $\mathtt{com}_2$ to a $\mathbf{WN}$ move associated to $\mathtt{com}$. When this is composed with the strategies $[\![M_i]\!] : \mathtt{var} \to F'\mathtt{com}_i$ $(i = 1, 2)$ of subterms, each of which is specified by a trace $\mathtt{run}{\cdot}\mathtt{W}'{\cdot}\mathtt{N}'{\cdot}\mathtt{wr}_i{\cdot}\mathtt{ok}{\cdot}\mathtt{done}$, the extra moves $\mathtt{W}'$ and $\mathtt{N}'$ in both strategies are synchronized and hidden but the copies of them are kept intact in the composed trace as: $\mathtt{run}{\cdot}\mathtt{W}{\cdot}\mathtt{N}{\cdot}\mathtt{wr}_1{\cdot}\mathtt{ok}{\cdot}\mathtt{W}{\cdot}\mathtt{N}{\cdot}\mathtt{wr}_2{\cdot}\mathtt{ok}{\cdot}\mathtt{done}$.

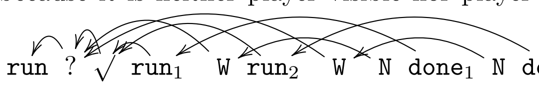
Several term constructors would worth further explanation. Thread extension $P \gg_j M$ makes use of the strategy $cont_j : (\mathbf{C}'_1 \times \cdots \times \mathbf{C}'_\zeta)_\perp \times \mathbf{C} \rightarrow (\mathbf{C}_1 \times \cdots \times \mathbf{C}_\zeta)_\perp$, which combines the strategy of the parallel command P with that of the sequential command M at the end of the j-th thread's execution. The interpretation of the parallel command $M_1 \| \cdots \| M_\zeta$ is given for its equivalent: $(\mathsf{skip} \parallel \cdots \parallel \mathsf{skip}) \gg_1 M_1 \gg_2 M_2 \cdots \gg_\zeta M_\zeta$. The

[2] Notice the uses of heavier symbols in the Kleisli category.

trivial parallel command skip $\|$ $\cdots$ $\|$ skip is interpreted by a strategy (the strategy *pskip* in Appendix B) whose well-opened trace set consist of the ζ copies of the neutral command skip in the arena $[\![\mathtt{par}]\!]$.

Since every IA_{par} term is interpreted by a Kleisli arrow in our game model, two terms that are comparable in an operational sense are not necessarily so by their corresponding strategies, due to the excess W and N moves. These excessive moves are necessary for modeling interleaved parallel execution but should be ignored when we discuss the observable behavior of programs.

Specifically for this purpose, we introduce a *scheduler* strategy $sched : F(\mathbf{C}_1 \times \cdots \times \mathbf{C}_\zeta)_\perp \to \mathbf{C}$ in $\mathcal{C}$, which is identified by the set of (well-opened longest) traces $\{\mathtt{run}\cdot ?\cdot(\mathtt{W}\cdot\mathtt{N})^k\cdot\surd\cdot s\cdot \mathtt{done} \mid k \geq 0, s \in CPP\}$, where CPP is the set of *complete parallel plays*: we say a legal play $s \in F(\mathbf{C}_1 \times \cdots \times \mathbf{C}_\zeta)$ is a complete parallel play if s arbitrarily interleaves the plays $\mathtt{run}_j\cdot(\mathtt{W}\cdot\mathtt{N})^{k_j}\cdot\mathtt{done}_j$ $(1 \leq j \leq \zeta,\ k_j \geq 0)$ and both $\mathtt{run}_j$ and $\mathtt{done}_j$ have exactly a single occurrence in s for each j. We remark that the strategy *sched* in $\mathcal{C}$ but not in $\mathcal{G}$ or $\mathcal{C}_{vb}$, because it is neither player visible nor player bracketing. For example, *sched* has a trace


$\mathtt{run}\ ?\ \surd\ \mathtt{run}_1\ \ \mathtt{W}\ \mathtt{run}_2\ \ \mathtt{W}\ \ \mathtt{N}\ \mathtt{done}_1\ \mathtt{N}\ \mathtt{done}_2\ \cdots$, where the second N is not justified by a move in its player view $\mathtt{run}\cdot ?\cdot\surd\cdot\mathtt{run}_1\cdot\mathtt{done}_1\cdot\mathtt{N}$ and also the first N does not answer to the last unanswered question in its player view, i.e., the second W. This indicates that the scheduler strategy is not definable in IA_{par} without using higher-order references and control primitives [2, 14, 15].

In what follows, the soundness property is discussed up to the so-called *intrinsic quotient*, a game model quotiented by contexts [3, 16, 10]. In most game models, the intrinsic quotient is only needed for establishing the full abstraction result, but the present paper needs it for establishing the soundness result as well, because of the above mentioned issue.

Using the scheduler strategy, we define the intrinsic quotient as follows: Given morphisms $f, g : A \longrightarrow B$ in $\mathcal{G}_F$, we define $f \precsim g$ iff $(`f\text{'} ⨾ h); sched \leq^\natural (`g\text{'} ⨾ h); sched$ for every $h : (A \Rightarrow B) \longrightarrow \mathbf{C}^\zeta{}_\perp$, where $`f\text{'} : \mathbf{1} \longrightarrow A \Rightarrow B$ is the name of $f$, namely, $`f\text{'} = \boldsymbol{\Lambda}_A(\boldsymbol{\pi}_A ⨾ f)$. We write $f \simeq g$ to mean $f \precsim g$ and $g \precsim f$. Let us write $\mathcal{G}_F/\!\simeq$ for the quotient category. $\mathcal{G}_F/\!\simeq$ is cartesian closed and is also rational, which is a sufficient condition for modeling recursions in Algol-like languages [16, 10]. We write $\mathcal{E}[\![M]\!] = [[\![M]\!]]_\simeq$, the *extensional* interpretation of the term M in $\mathcal{G}_F/\!\simeq$.

The soundness property follows from the consistency and adequacy.

▶ Proposition 1 (consistency). Suppose M is a closed term of type par. If $M{\Downarrow}^{\mathrm{may}}$, then $\mathtt{run}\cdot\mathtt{done} \in T_{[\![M]\!];sched}$; If $M{\Downarrow}^{\mathrm{must}}$, then $\mathtt{run} \notin D_{[\![M]\!];sched}$.

▶ Proposition 2 (adequacy). Suppose M is a closed term of type par. If $\mathtt{run}\cdot\mathtt{done} \in T_{[\![M]\!];sched}$, then $M{\Downarrow}^{\mathrm{may}}$; If $\mathtt{run} \notin D_{[\![M]\!];sched}$, then $M{\Downarrow}^{\mathrm{must}}$.

▶ **Theorem 3** (soundness). *If M and N are closed terms of type* T *and $\mathcal{E}[\![M]\!] \lesssim \mathcal{E}[\![N]\!]$, then $M \lesssim_{\mathrm{m\&m}} N$.*

6 Definability and Full Abstraction

As Harmer did for his nondeterministic Algol-like language [10], we can also show the full abstraction result for our parallel language IA_{par} by combining two factorizations and a definability result. However, we need to make his techniques more precise, due to extra intricacies involved in parallelism, most notably race conditions on shared variables.

6.1 Reliable factorization

We first factor out possible nondeterminism as the oracle strategy *oracle* : $1 \twoheadrightarrow \mathbf{N}' \Rightarrow \mathbf{N}$, whose (longest well-opened) traces are $\{\mathsf{qq}'0n \mid n \geq 0\} \cup \{\mathsf{q}(\mathsf{q}'m)^{n+1}n \mid m > 0 \wedge 0 \leq n \leq m\}$ and the (sole) well-opened interesting divergence is $\mathsf{qq}'0$. Given a constant 0, the *oracle* strategy nondeterministically diverges or converges to produce an arbitrary number; Given a positive constant m, it never diverges but reliably returns a number not greater than m. The strategy *oracle* is definable by the term $\lambda x^{\mathsf{nat}}.\mathsf{if0}\ x$ then $\mho$ else $\mho' x$, where $\mho = \mathsf{fix}^{\mathsf{nat}}(\lambda x.0$ or $(x+1))$ and $\mho' = \mathsf{fix}^{\mathsf{nat}\to\mathsf{nat}}(\lambda f.\lambda x.\mathsf{if0}\ x$ then 0 else (0 or $f(x-1)+1))$.

Assuming a fixed coding function code_A that assigns a unique natural number to each distinct legal play in arena A, we separate out the reliable part of a strategy σ as follows.

▶ Proposition 4. If $\sigma : 1 \twoheadrightarrow A$ is a compact strategy in $\mathcal{G}_F$, then there exists a compact, reliable strategy $det(\sigma) : (\mathbf{N}' \Rightarrow \mathbf{N}) \twoheadrightarrow A$ such that $\sigma =^{\natural}$ *oracle*$\fatsemi \det(\sigma)$.

Proof. The proof basically follows that of Proposition 4.6.2 in [10], but we must be careful that our oracle strategy is different from the one employed in Harmer's original proof, in that Harmer's oracle strategy makes use of local variables whereas ours doesn't. The effect is that ours makes duplicated copies of the argument, witnessed as the subsequence $(\mathsf{q}'m)^{n+1}$ in the trace. This does not matter for factorization, though. Wherever Harmer factors out a finitely branching nondeterminism at $sa \in dom(\sigma)$ by a trace $\cdots a \cdot \mathsf{q} \cdot \mathsf{q}'m \cdot j \cdot b$ in $det(\sigma)$, we do it by a bit longer trace $\cdots a \cdot \mathsf{q} \cdot (\mathsf{q}' \cdot m)^{j+1} \cdot j \cdot b$. Factorization at diverging points is similarly done.

Further, in order to have the factorization process closed under the subquestion-affine property, we have to make Harmer's factorization more precise so that any oracle moves are not inserted where the subsequent player move is uniquely determined. ◀

Our preference to the oracle that is definable without local variables is due to the extra **WN** moves to be introduced otherwise. Harmer's oracle strategy would also work in our game model modulo $\simeq$. We will deal with $\simeq$-quotients in the next step, where we can work with reliable strategies, without being bothered with divergence or nondeterminism.

6.2 Innocent factorization

The second step toward full abstraction is *innocent factorization* [3], which separates an innocent (history-free, in other words) strategy from history-sensitive one. A strategy σ is called *innocent* if σ is reliable and player visible and for every $sab \in T_\sigma$ and $t \in L_\sigma^{\mathrm{odd}}$ such that $\mathsf{V}(sa) = \mathsf{V}(t)$, $tb \in T_\sigma$ and $\mathsf{V}(sa)b = \mathsf{V}(t)b$, where b is justified by the matching move in the player view. An innocent strategy σ is uniquely identified by its *view function*, $\mathsf{fun}(\sigma) = \{\mathsf{V}(s) \mid s \in T_\sigma\}$.

The idea in innocent factorization of a strategy is to determine its behavior up to, instead of the trace that it has played so far, a record of execution history kept in a variable. The standard innocent factorization procedure builds an innocent strategy, whose view function contains a player view of the form $s' \cdot a \cdot \mathtt{rd} \cdot \mathsf{code}_A(s) \cdot \mathtt{wr}_{\mathsf{code}_A(s \cdot a \cdot b)} \cdot \mathtt{ok} \cdot b$, for every $sab \in T_\sigma$ and the player view s' of the factorization of s. This construction violates, however, the subquestion-affine conditions (sq2) and (sq3). Thus we need a factorization procedure with improved precision.

Let σ be a reliable and player visible strategy. A player view $\mathsf{V}(sab)$ at an opponent move a of σ is called *locally innocent*, if it holds that $\mathsf{V}(sab) = \mathsf{V}(s'ab')$ for every $s'ab' \in T_\sigma$ satisfying $\mathsf{V}(sa) = \mathsf{V}(s'a)$. Wherever a player view at an opponent move is locally innocent, as the next player move that follows is uniquely determined by the player view, we can skip and postpone the history update until we reach a point that is not locally innocent.

More fundamentally, we need another change in the factorization procedure, in order to avoid possible race conditions caused by parallel accesses to the shared variable. Suppose we have a strategy in arena $F[\![\mathtt{par}]\!]$ that has a factorized view function containing plays like $?\cdot\surd\cdot\mathtt{run}_j\cdot s_j\cdot\mathtt{done}_j$ $(1 \leq j \leq \zeta)$, where each s_j contains successive moves $a_j\cdot\mathtt{W}\cdot\mathtt{N}\cdot\mathtt{rd}\cdot m_j\cdot\mathtt{W}\cdot\mathtt{N}\cdot\mathtt{wr}_{n_j}\cdot\mathtt{ok}\cdot p_j$. (Remember that every move representing a variable access operation is preceded by $\mathtt{W}\cdot\mathtt{N}$ in our game modeling.) Then the factorized strategy, which arbitrarily interleaves the plays in the view function, contains a play that competes for the shared variable, e.g., $?\cdot\surd\cdots a_1\cdot\mathtt{W}\cdot\mathtt{N}\cdot\mathtt{rd}\cdot m_1\cdot\mathtt{W}\cdot a_2\cdot\mathtt{W}\cdot\mathtt{N}\cdot\mathtt{rd}\cdot m_2\cdot\mathtt{W}\cdot\mathtt{N}\cdot\mathtt{wr}_{n_1}\cdot\mathtt{ok}\cdot p_1\cdot\mathtt{N}\cdot\mathtt{wr}_{n_2}\cdot\mathtt{ok}\cdot p_2$. At the end of the play, thread 1 has already reached its player move p_1 but it is not recorded in the storage, overwritten by the subsequent moves of thread 2.

Here we achieve innocent factorization by a *thread-safe programming* on game plays. We make use of the atomic read-modify-write ability provided by the CAS operation in order to avoid race conditions, making each history update inseparable from the completion of a single computation step. To do this, we may just spin over the variable storing the history, repeatedly trying to atomically update the variable by CAS until successful. However, the spin lock mechanism is too naïve as a means for factorization, as it may introduce an undesired divergence when the update fails forever. Instead, we repeatedly try to update, but bounded by a sufficiently large number of times. Such a bound exists, if we assume a compact strategy, i.e., a strategy whose well-opened traces are finite, because there can be at most a bounded number of successful writes throughout the entire traces, meaning that each repeated execution of update by CAS is guaranteed to succeed within the bound.

To sum up, given a compact, player visible, and reliable strategy σ on an arena $A \to FB$ and also a positive number d, we construct an innocent strategy $\mathsf{inn}_d(\sigma)$, identified by the view function $\mathsf{fun}(\mathsf{inn}_d(\sigma))$ that contains the following player views for every $s\cdot a\cdot b \in T_\sigma$:

- When the player view $\mathsf{V}(sa)$ is locally innocent, we just add player views of the form $s'\cdot a\cdot b$ to the view function, where s' is a factorization obtained from s;
- Otherwise, for any partial function ν on natural numbers such that $\nu(n) = m$ iff $n = \mathsf{code}(s')$, $m = \mathsf{code}(s'ab)$ for some $s'ab \in T_\sigma$ satisfying $\mathsf{V}(s'a) = \mathsf{V}(sa)$, where s' is a factorization obtained from s. Then, we add (every even-length prefixes of) player views of the following form $s'\cdot a\cdot(\mathtt{W}\cdot\mathtt{N})^d\cdot\mathtt{rd}\cdot\mathsf{code}(s)\cdot u_{c_0,c_1}\cdot u_{c_1,c_2}\cdots u_{c_i,c_{i+1}}\cdots u_{c_k,\nu(c_k)}\cdot(\mathtt{W}\cdot\mathtt{N})^d\cdot b$, where $u_{m,m'}$ is a sequence of moves $(\mathtt{W}\cdot\mathtt{N})^d\cdot\mathsf{cas}_{m,\nu(m)}\cdot m'$, $c_0 = \mathsf{code}(s)$, and for every i $(1 \leq i \leq k)$, $c_i \neq \nu(c_{i-1})$ and there exists $ta'b' \in T_{WO(\sigma)}$ satisfying $\mathsf{code}(t) = c_{i-1}$ and $\mathsf{code}(ta'b') = c_i$. k is bounded by a number determined by each given strategy, i.e., the length of the longest well-opened trace.

We call the strategy $\mathsf{inn}_d(\sigma)$ a *d-delayed* innocent strategy, as every $sa \in dom(\sigma)$ must be followed by at least d successive sequences of $\mathtt{W}\cdot\mathtt{N}$ moves, unless the player view $\mathsf{V}(sa)$ uniquely determines the next player move. Formally, an innocent strategy σ is called *d-delayed* iff for every trace $sab \in T_\sigma$, either b is $\mathtt{W}$, the player view $\mathsf{V}(sa)$ is locally innocent, or $\mathsf{V}(sab) = t\cdot(\mathtt{W}\cdot\mathtt{N})^d\cdot b$ for some t. The extra d-delays, not just a single delay, are needed for obtaining the definability result (Section 6.3).

▶ Proposition 5. Let $f : 1 \to A$ be a compact reliable strategy in $\mathcal{G}_F$. Then, for every $d \geq 1$, there is a compact d-delayed innocent strategy $\mathsf{inn}_d(f) : \mathbf{Var} \to A$ in $\mathcal{G}_F$ such that $(cell_{\mathsf{code}(\varepsilon)}; \eta_{\mathbf{Var}})⨾\, \mathsf{inn}_d(f) \simeq f$.

We notice that factorization is modulo $\simeq$, which ignores the extra **WN** moves introduced during the factorization procedure.

6.3 Definability

The last step toward the full abstraction is the definability: We are obliged to show that every strategy in a suitable strategy subclass is definable by an IA_{par} term.

In the construction of IA_{par} terms below, we need a general conditional expression $\mathsf{case}_{\ell,\mathtt{T}} : \mathtt{nat} \times \mathtt{T}^{\ell} \to \mathtt{T}$, where $\ell \geq 0$ and $\mathtt{T}$ is either $\mathtt{nat}$, $\mathtt{com}$, or $\mathtt{par}$. This conditional expression is an operationally conservative extension to the language, as it is defined by the following IA_{par} term:

$$\begin{aligned}&\lambda x x_1 \cdots x_\ell.\big(\mathsf{newvar}\ v = 0\ \mathsf{in}\\ &\quad \mathsf{seq}\ (\mathsf{assign}\ v\ x)\ \ (\mathsf{if0}\ (\mathsf{deref}\ v)\ \mathsf{then}\ x_1\ \mathsf{else}\\ &\qquad\qquad (\mathsf{if0}\ (\mathsf{deref}\ v) - 1\ \mathsf{then}\ x_2\ \mathsf{else}\ \cdots\ (\mathsf{if0}\ (\mathsf{deref}\ v) - \ell\ \mathsf{then}\ x_\ell\ \mathsf{else}\ \Omega)\cdots))\big)\end{aligned}$$

where Ω is the divergence at type $\mathtt{T}$.

Due to the variable access operations being involved, however, the game interpretation of $\mathsf{case}_{\ell,\mathtt{T}}$ contains extra $\mathbf{WN}$ moves in its traces: Given a natural number m less than ℓ as its first argument, $\mathsf{case}_{k,\mathtt{T}}$ has $m+1$ accesses to the local variable v, leaving $(\mathtt{W}\cdot\mathtt{N})^{m+1}$ in its trace as the footprint. The extra $\mathbf{WN}$ moves are canceled by the aforementioned d-delayed innocent strategy $\mathsf{inn}_d(\sigma)$, where d is a number larger than the maximum number of possible opponent moves that immediately follow the same player view, i.e., $\max\{\#\{ta'b' \mid ta'b' \in \mathsf{fun}(\sigma)\} \mid tab \in \mathsf{fun}(\sigma)\}$.

In case some trace in the innocent strategy contains excess $\mathtt{W}\cdot\mathtt{N}$ sequences than those to be canceled out, we let the term $\mathsf{newvar}\ v = 0\ \mathsf{in}\ \mathsf{assign}\ v\ 0$, denoted by touch hereafter, cancel out the remaining ones. The term touch has the trace $\mathtt{run}\cdot\mathtt{W}\cdot\mathtt{N}\cdot\mathtt{done}$, which witnesses a single $\mathtt{W}\cdot\mathtt{N}$ sequence as the footprint of a single access to the local variable v. Further, for brevity, we will write touch^k for the command $\overbrace{\mathsf{seq\ touch}\ (\mathsf{seq\ touch}\ \cdots (\mathsf{seq\ touch}}^{k\text{ times}}\ \mathsf{skip})\cdots)$.

Let us show that any compact, d-delayed innocent, player bracketing strategy $\sigma : [\![\mathtt{T}_1]\!] \times \cdots \times [\![\mathtt{T}_n]\!] \to [\![\mathtt{T}]\!]$, where d is a sufficiently large number as analyzed above, is definable by an IA_{par} term of the form $x_1 : \mathtt{T}_1, \cdots, x_n : \mathtt{T}_n \vdash M : \mathtt{T}$. We construct such a term by induction on the size of view function $\mathsf{fun}(\sigma)$, where the base case is $\mathsf{fun}(\sigma) = \{\varepsilon\}$ that is trivially definable by Ω. Below we will mostly concern $\mathtt{par}$ types and the extra $\mathbf{WN}$ moves. (We will omit some details not concerning these extra complications. See [10] for the missing details.)

We may assume that $\mathtt{T}$ is a base type. If otherwise, say, $\mathtt{T} = \mathtt{T}'_1 \to \cdots \to \mathtt{T}'_m \to \mathtt{T}'$ with $\mathtt{T}'$ being a base type, we instead work on the isomorphic strategy on $[\![\mathtt{T}_1]\!] \times \cdots \times [\![\mathtt{T}_n]\!] \times [\![\mathtt{T}'_1]\!] \times \cdots \times [\![\mathtt{T}'_m]\!] \to [\![\mathtt{T}']\!]$.

Here we consider solely the case $\mathtt{T} = \mathtt{par}$. The remaining cases $\mathtt{T} = \mathtt{nat}$, $\mathtt{T} = \mathtt{com}$, and $\mathtt{T} = \mathtt{var}$ are shown in almost the same way, except that, when $\mathtt{T} = \mathtt{var}$, we need to construct a *bad variable* $\mathsf{mkvar}\ M\ N\ L$ with L being the term that simulates a CAS-like operation.

Suppose that $?\cdot(\mathtt{W}\cdot\mathtt{N})^d\cdot\mathtt{q}_j \in \mathsf{fun}(\sigma)$, where $\mathtt{q}_j$ is a move from the arena $\mathbf{N}$ in particular when $\mathtt{T}_j$ has the form $\mathtt{T}'_1 \to \cdots \to \mathtt{T}'_m \to \mathtt{nat}$, with d being the sufficiently large number. We derive a class of substrategies from σ that separate out the threads of play in $[\![\mathtt{T}_j]\!]$ induced by the move $\mathtt{q}_j$, as follows. Let $\sigma_i : [\![\mathtt{T}_1]\!] \times \cdots \times [\![\mathtt{T}_n]\!] \to [\![\mathtt{par}]\!]$ $(i \geq 0)$ and $\sigma'_h : [\![\mathtt{T}_1]\!] \times \cdots \times [\![\mathtt{T}_n]\!] \to [\![\mathtt{T}'_h]\!]$ $(1 \leq h \leq m)$ be substrategies identified by view functions $\mathsf{fun}(\sigma_i) = \{?\cdot s \mid ?\cdot(\mathtt{W}\cdot\mathtt{N})^d\cdot\mathtt{q}_j\cdot i\cdot s \in \mathsf{fun}(\sigma)\}$ and $\mathsf{fun}(\langle\sigma'_1, \cdots, \sigma'_m\rangle) = \{s \mid ?\cdot(\mathtt{W}\cdot\mathtt{N})^d\cdot\mathtt{q}_j\cdot s \in \mathsf{fun}(\sigma),\ s \text{ contains no answer to } \mathtt{q}_j\}$, respectively. By the compactness, there exists a natural number ℓ such that $\mathsf{fun}(\sigma_i) = \{\varepsilon\}$ for every $i \geq \ell$. Since each substrategy is again a compact, d-delayed innocent, player bracketing strategy that is strictly smaller than $\mathsf{fun}(\sigma)$, by induction hypothesis, we have terms M_i and M'_h defining σ_i and σ'_h, respectively, for each i and h. Then the strategy σ is definable by the term:

$$\mathsf{case}_{\ell,\mathtt{par}}\ (x_j M'_1 \cdots M'_m)\ \ (\mathsf{seq\ touch}^{d-1}\ M_0)\ (\mathsf{seq\ touch}^{d-2}\ M_1)\ \cdots\ (\mathsf{seq\ touch}^{d-\ell}\ M_{\ell-1}).$$

The case the right-most base type in $\mathtt{T}_j$ being `com` or `par` is similarly defined by using `seq` in place of case; The case for `var` is also similar, except that we need to additionally deal with CAS operations.

When $?{\cdot}\mathtt{W}{\cdot}\mathtt{N}{\cdot}p \in \mathsf{fun}(\sigma)$ with p not being a move from $[\![T_j]\!]$'s, the strategy σ is simply definable by the term $\mathsf{seq\ touch}\ M$, where M is the defining term of the substrategy σ' specified by $\mathsf{fun}(\sigma_i) = \{?{\cdot}p{\cdot}s \mid ?{\cdot}\mathtt{W}{\cdot}\mathtt{N}{\cdot}p{\cdot}s \in \mathsf{fun}(\sigma)\}$. When $?{\cdot}\surd \in \mathsf{fun}(\sigma)$, we define substrategies $\sigma_j : [\![\mathtt{T}_1]\!] \times \cdots \times [\![\mathtt{T}_n]\!] \to \mathbf{C}_j$ $(1 \le j \le \zeta)$ by view functions $\mathsf{fun}(\sigma_j) = \{\mathtt{run}_j{\cdot}t \mid ?{\cdot}\surd{\cdot}\mathtt{run}_j{\cdot}t \in \mathsf{fun}(\sigma)\}$. By induction hypothesis, we have a term M_i that defines σ_i for each i. Then the strategy σ is definable by the term $M_1 \| \cdots \| M_\zeta$.

The remaining case is that there exists $?{\cdot}?'{\cdot}\surd'{\cdot}\surd \in \mathsf{fun}(\sigma)$ where $\surd'$ is a subquestioning answer, which is derived from the $\mathtt{par}'$ type in $\mathtt{T}_j$ of the form $\mathtt{T}'_1 \to \cdots \to \mathtt{T}'_m \to \mathtt{par}'$. Likewise above, we derive substrategies from σ. Let $\sigma_i : [\![\mathtt{T}_1]\!] \times \cdots \times [\![\mathtt{T}_n]\!] \to \mathbf{C}$ $(1 \le i \le \zeta)$ and $\sigma'_h : [\![\mathtt{T}_1]\!] \times \cdots \times [\![\mathtt{T}_n]\!] \to [\![\mathtt{T}'_h]\!]$ $(1 \le h \le m)$ be substrategies identified by view functions $\mathsf{fun}(\sigma_i) = \{\mathtt{run}_i{\cdot}s' \mid ?{\cdot}?'{\cdot}\surd'{\cdot}\surd{\cdot}\mathtt{run}_i{\cdot}\mathtt{run}'_i{\cdot}\mathtt{done}'_i{\cdot}s' \in \mathsf{fun}(\sigma)\}$ and $\mathsf{fun}(\langle\sigma'_1, \cdots, \sigma'_m\rangle) = \{s' \mid ?{\cdot}?'{\cdot}\surd'{\cdot}\surd{\cdot}\mathtt{run}_j{\cdot}\mathtt{run}'_j{\cdot}s' \in \mathsf{fun}(\sigma),\ s'$ contains no answer to $\mathtt{run}'_j\}$, respectively. Again, by induction hypothesis, we have terms M_i, and M'_j defining σ_i, and σ'_j, respectively, for each i and j. Then σ is definable by the term $(x_j M'_1 \cdots M'_m) \gg_1 M_1 \gg_2 M_2 \cdots \gg_\zeta M_\zeta$.

▶ **Theorem 6** (definability). *Let $\sigma : [\![\Gamma]\!] \to [\![\mathtt{T}]\!]$ be a compact and player bracketing strategy in $\mathcal{G}_F$ (henceforth, it is also a player visible, subquestion-affine strategy.) Then, σ is definable in IA_{par}.*

▶ **Theorem 7** (Full abstraction). *Suppose M and N are closed terms of type $\mathtt{T}$. Then, $\mathcal{E}[\![M]\!] \lesssim \mathcal{E}[\![N]\!]$ iff $M \lesssim_{\mathrm{m\&m}} N$.*

7 Conclusion and Future Work

We have developed a full abstract game semantics for an Algol-like parallel language with a non-blocking synchronization primitive CAS. Elaborating on the wait-notify game [20] in the framework of Harmer's game model for nondeterminism [10], we exploited the computational structure of the Kleisli category induced by the extra `W` and `N` moves and thereby established the full abstraction, in which we made a significant use of CAS operations.

Based on the full abstraction result, it would be beneficial to find a subset language whose observational equality is decidable, as it would provide a mechanized method for equivalence checking, as done in [6]. Further, though the present paper is limited to an unfair thread scheduling policy specified by the particular strategy *sched*, more flexible variants of scheduling policy (say, the Round-robin scheduling) would worth investigating. This requires a game model for fair nondeterminism, which would provide a deeper insight on parallel computation.

Acknowledgment. I would like to thank Shin-ya Katsumata for his valuable comments and suggestions on an early draft of the paper. I am also grateful for reviewers for their helpful comments. This work was supported by KAKENHI 24500014.

References

1 Samson Abramsky. Game semantics of idealized parallel Algol. Lecture given at the Newton Institute, 1995.

2 Samson Abramsky, Kohei Honda, and Guy McCusker. A fully abstract game semantics for general references. In *13th Annual IEEE Symposium on Logic in Computer Science*, pages 334–344, 1998.

3 Samson Abramsky and Guy McCusker. Linearity, sharing and state: A fully abstract game semantics for idealized Algol with active expressions. In P. W. O'Hearn and R. D. Tennent, editors, *Algol-like Languages*, volume 2 of *Progress in Theoretical Computer Science*, pages 297–329. Birkhäuser, 1997.

4 Samson Abramsky and Guy McCusker. Game semantics. In H. Schwichtenberg and U. Berger, editors, *Computational Logic: Proceedings of the 1997 Marktoberdorf Summer School*, pages 1–56. Springer-Verlag, 1999.

5 Stephen Brookes. Full abstraction for a shared variable parallel language. In *Proceedings of 8th Annual IEEE Symposium on Logic in Computer Science*, pages 98–109, 1993.

6 Dan R. Ghica and Guy McCusker. The regular-language semantics of second-order idealized ALGOL. *Theoretical Computer Science*, 309(1–3):469–502, 2003.

7 Dan R. Ghica and Andrzej S. Murawski. Angelic semantics of fine-grained concurrency. *Annals of Pure and Applied Logic*, 151(2-3):89–114, 2008.

8 Dan R. Ghica, Andrzej S. Murawski, and C.-H. Luke Ong. Syntactic control of concurrency. *Theoretical Computer Science*, 350(2-3):234–251, 2006.

9 Russel Harmer and Guy McCusker. A fully abstract game semantics for finite nondeterminism. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*, pages 422–430, 1999.

10 Russell Harmer. *Games and Full Abstraction for Nondeterministic Languages.* PhD thesis, University of London, 1999.

11 Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1), 1991.

12 Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming.* Morgan Kaufmann, 2008.

13 J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Information and Computation*, 163(2):285–408, 2000.

14 James Laird. Full abstraction for functional languages with control. In *12th Annual IEEE Symposium on Logic in Computer Science*, pages 58–67, 1997.

15 James Laird. A fully abstract game semantics of local exceptions. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 105–114, 2001.

16 Guy McCusker. *Games and Full Abstraction for a Functional Metalanguage with Recursive Types.* Distinguished Dissertations. Springer, 1998.

17 Gordon D. Plotkin. A powerdomain construction. In *SIAM J. Comput.*, number 5 in 3, pages 452–487, 1976.

18 John C. Reynolds. The essence of Algol. In *Proceedings of the 1981 International Symposium on Algorithmic Languages*, pages 345–372. North-Holland, 1981.

19 David A. Schmidt. *Denotational Semantics: A Methodology for Language Development.* 1986.

20 Keisuke Watanabe and Susumu Nishimura. May&must-equivalence of shared variable parallel programs in game semantics. *Information Processing Society of Japan Transactions on Programming (PRO)*, 5(4):17–26, 2012. `http://jlc.jst.go.jp/DN/JST.JSTAGE/ipsjtrans/5.167`.

$$\frac{}{\Gamma, x:\mathtt{T} \vdash x:\mathtt{T}} \qquad \frac{}{\Gamma \vdash n:\mathtt{nat}} \qquad \frac{\Gamma \vdash M:\mathtt{nat} \quad \Gamma \vdash N:\mathtt{nat}}{\Gamma \vdash M \star N:\mathtt{nat}}$$

$$\frac{\Gamma, x:\mathtt{T}' \vdash M:\mathtt{T}}{\Gamma \vdash \lambda x^{\mathtt{T}'}.M:\mathtt{T}' \to \mathtt{T}} \qquad \frac{\Gamma \vdash M_1:\mathtt{T}' \to \mathtt{T} \quad \Gamma \vdash M_2:\mathtt{T}'}{\Gamma \vdash M_1 M_2:\mathtt{T}} \qquad \frac{\Gamma \vdash M:\mathtt{T} \to \mathtt{T}}{\Gamma \vdash \mathsf{fix}_{\mathtt{T}}\ M:\mathtt{T}}$$

$$\frac{}{\Gamma \vdash \mathsf{skip}:\mathtt{com}} \qquad \frac{\Gamma \vdash M_1:\mathtt{com} \quad \Gamma \vdash M_2:\mathtt{T} \quad \mathtt{T} \in \{\mathtt{nat}, \mathtt{com}, \mathtt{par}\}}{\Gamma \vdash \mathsf{seq}\ M_1\ M_2:\mathtt{T}}$$

$$\frac{\Gamma \vdash M:\mathtt{nat} \quad \Gamma \vdash M_1:\mathtt{T} \quad \Gamma \vdash M_2:\mathtt{T} \quad \mathtt{T} \in \{\mathtt{nat}, \mathtt{com}, \mathtt{par}\}}{\Gamma \vdash \mathsf{if0}\ M\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2:\mathtt{T}}$$

$$\frac{\Gamma \vdash M:\mathtt{var} \quad \Gamma \vdash N:\mathtt{nat}}{\Gamma \vdash \mathsf{assign}\ M\ N:\mathtt{com}} \qquad \frac{\Gamma \vdash M:\mathtt{var}}{\Gamma \vdash \mathsf{deref}\ M:\mathtt{nat}}$$

$$\frac{\Gamma \vdash L:\mathtt{var} \quad \Gamma \vdash M:\mathtt{nat} \quad \Gamma \vdash N:\mathtt{nat}}{\Gamma \vdash \mathsf{cas}\ L\ M\ N:\mathtt{nat}}$$

$$\frac{\Gamma \vdash M:\mathtt{nat} \to \mathtt{com} \quad \Gamma \vdash N:\mathtt{nat} \quad \Gamma \vdash L:\mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}}{\Gamma \vdash \mathsf{mkvar}\ M\ N\ L:\mathtt{var}}$$

$$\frac{\Gamma, v:\mathtt{var} \vdash M:\mathtt{T} \quad \mathtt{T} \in \{\mathtt{com}, \mathtt{par}\}}{\Gamma \vdash \mathsf{newvar}\ v = n\ \mathsf{in}\ M:\mathtt{T}} \qquad \frac{\Gamma \vdash M_1:\mathtt{nat} \quad \Gamma \vdash M_2:\mathtt{nat}}{\Gamma \vdash M_1\ \mathsf{or}\ M_2:\mathtt{nat}}$$

$$\frac{\Gamma \vdash M_1:\mathtt{com} \quad \cdots \quad \Gamma \vdash M_\zeta:\mathtt{com}}{\Gamma \vdash M_1 \| \cdots \| M_\zeta:\mathtt{par}} \qquad \frac{\Gamma \vdash P:\mathtt{par} \quad \Gamma \vdash M:\mathtt{com}}{\Gamma \vdash P \gg_j M:\mathtt{par}}$$

Figure 1 Typing rules.

A Typing Rules and Operational Semantics for IA$_{par}$

The typing rules of IA$_{par}$ are given in Figure 1.

Let us write $\langle s \mid v \mapsto m \rangle$ for a store that updates s to give the natural number m at v. Let us also write $M[N/x]$ for the term substitution, which replaces every free occurrence of variable x in M with N, assuming the usual variable convention.

The reduction rules are given in Figure 2, relative to *evaluation context*. An evaluation context E is a term with a single hole $[\,]$ defined by the following grammar:

$$\begin{aligned} E ::= & [\,] \mid E \star M \mid n \star E \mid EM \mid \mathsf{seq}\ E\ M \mid \mathsf{if0}\ E\ \mathsf{then}\ M\ \mathsf{else}\ N \mid E \gg_j M \\ & \mid M_1 \| \cdots \| M_{j-1} \| E \| M_{j+1} \| \cdots \| M_\zeta . \end{aligned}$$

We write $E[M]$ for the term obtained by filling the hole in E with term M.

Here we notice that there are three term formations whose evaluation can be nondeterministic: M_1 or M_2, the erratic binary choice on natural numbers; $M_1 \parallel \cdots \parallel M_\zeta$, the choice of a single command out of ζ simultaneously running sequential commands; $(M_1 \parallel \cdots \parallel M_\zeta) \gg_j M$, which either extends the j-the thread's execution by the command M or executes just one out of ζ parallel threads a single step forward. The last nondeterminism, nevertheless, is neutral to the observational property, that is, the different reduced terms can conflue even after further reductions. Indeed, they will be given the identical game interpretation.

B Game Interpretation of Terms

The arenas corresponding to common base types found in Algol-like languages are given below. (For notational convenience, let us we write $\overline{\lambda}_A$ for the labeling function whose

$$\langle E[m \star n], s\rangle \longrightarrow \langle E[n'], s\rangle, \text{ where } n' = m \star n \qquad \langle E[(\lambda x^{\mathtt{T}}.M)\, N], s\rangle \longrightarrow \langle E[M[N/x]], s\rangle$$
$$\langle E[\mathsf{if0}\ 0\ \mathsf{then}\ M\ \mathsf{else}\ N], s\rangle \longrightarrow \langle E[M], s\rangle \qquad \langle E[\mathsf{if0}\ n+1\ \mathsf{then}\ M\ \mathsf{else}\ N], s\rangle \longrightarrow \langle E[N], s\rangle$$
$$\langle E[\mathsf{fix}\ M], s\rangle \longrightarrow \langle E[M(\mathsf{fix}\ M)], s\rangle \qquad \langle E[\mathsf{seq}\ \mathsf{skip}\ M], s\rangle \longrightarrow \langle E[M], s\rangle$$
$$\langle E[\mathsf{assign}\ v\ n, s\rangle] \longrightarrow \langle E[\mathsf{skip}], \langle s \mid v \mapsto n\rangle\rangle \qquad \langle E[\mathsf{deref}\ v], s\rangle \longrightarrow \langle E[s(v)], s\rangle$$
$$\langle E[\mathsf{cas}\ v\ m\ n], s\rangle \longrightarrow \langle E[n], \langle s \mid v \mapsto n\rangle\rangle, \text{ where } s(v) = m$$
$$\langle E[\mathsf{cas}\ v\ m\ n], s\rangle \longrightarrow \langle E[s(v)], s\rangle, \text{ where } s(v) \neq m$$
$$\langle E[\mathsf{assign}\ (\mathsf{mkvar}\ M\ N\ L)\ n], s\rangle \longrightarrow \langle E[Mn], s\rangle \qquad \langle E[\mathsf{deref}\ (\mathsf{mkvar}\ M\ N\ L)], s\rangle \longrightarrow \langle E[N], s\rangle$$
$$\langle E[\mathsf{cas}\ (\mathsf{mkvar}\ M\ N\ L)\ m\ n], s\rangle \longrightarrow \langle E[Lmn], s\rangle$$
$$\langle E[M\ \mathsf{or}\ N], s\rangle \longrightarrow \langle E[M], s\rangle \qquad \langle E[M\ \mathsf{or}\ N], s\rangle \longrightarrow \langle E[N], s\rangle$$
$$\langle E[\mathsf{newvar}\ v = n\ \mathsf{in}\ \mathsf{skip}], s\rangle \longrightarrow \langle E[\mathsf{skip}], s\rangle$$
$$\langle E[\mathsf{newvar}\ v = n\ \mathsf{in}\ (\mathsf{skip}\|\cdots\|\mathsf{skip})], s\rangle \longrightarrow \langle E[\mathsf{skip}\|\cdots\|\mathsf{skip}], s\rangle$$
$$\frac{\langle M, \langle s \mid v \mapsto n\rangle\rangle \longrightarrow \langle M', s'\rangle}{\langle E[\mathsf{newvar}\ v = n\ \mathsf{in}\ M], s\rangle \longrightarrow \langle E[\mathsf{newvar}\ v = s'(v)\ \mathsf{in}\ M'], \langle s' \mid v \mapsto s(v)\rangle\rangle}$$
$$\langle E[(\mathsf{newvar}\ v = n\ \mathsf{in}\ P) \gg_j M], s\rangle \longrightarrow \langle E[\mathsf{newvar}\ v = n\ \mathsf{in}\ (P \gg_j M)], s\rangle$$
$$\langle E[(M_1\|\cdots\|M_{j-1}\|M_j\|M_{j+1}\|\cdots\|M_\varsigma) \gg_j M], s\rangle$$
$$\longrightarrow \langle E[M_1\|\cdots\|M_{j-1}\|\mathsf{seq}\ M_j\ M\|M_{j+1}\|\cdots\|M_\varsigma], s\rangle$$

Figure 2 Reduction rules.

opponent/player attribution is swapped, i.e., $\overline{\lambda}_A^{\mathsf{OP}}(b) = \mathsf{O}$ iff $\lambda_A^{\mathsf{OP}}(b) = \mathsf{P}$ for every move b.)

- The arena $\mathbf{N} = ((M_\mathbf{N}, <_\mathbf{N}), \lambda_\mathbf{N}, \vdash_\mathbf{N})$ of natural numbers, where $M_\mathbf{N} = \{\mathsf{q}\} \cup \{n \mid n \geq 0\}$, $\lambda_\mathbf{N}(\mathsf{q}) = (\mathsf{O}, \mathsf{Q})$, $\lambda_\mathbf{N}(n) = (\mathsf{P}, \mathsf{A})$ for every n, $\vdash_\mathbf{N} = \{(\mathsf{q}, \mathsf{q})\} \cup \{(\mathsf{q}, n) \mid n \geq 0\}$.
- The arena $\mathbf{C} = ((M_\mathbf{C}, <_\mathbf{C}), \lambda_\mathbf{C}, \vdash_\mathbf{C})$ of commands, where $M_\mathbf{C} = \{\mathtt{run}, \mathtt{done}\}$, $\lambda_\mathbf{C}(\mathtt{run}) = (\mathsf{O}, \mathsf{Q})$, $\lambda_\mathbf{C}(\mathtt{done}) = (\mathsf{P}, \mathsf{A})$, $\vdash_\mathbf{C} = \{(\mathtt{run}, \mathtt{run}), (\mathtt{run}, \mathtt{done})\}$.
- The arena $\mathbf{Var} = ((M_\mathbf{Var}, <_\mathbf{Var}), \lambda_\mathbf{Var}, \vdash_\mathbf{Var})$ of mutable variables, where $M_\mathbf{Var} = \{\mathtt{rd}, \mathtt{ok}\} \cup \{n \mid n \geq 0\} \cup \{\mathtt{wr}_n \mid n \geq 0\} \cup \{\mathtt{cas}_{m,n} \mid m, n \geq 0\}$, $\lambda_\mathbf{Var}(\mathtt{rd}) = \lambda_\mathbf{Var}(\mathtt{wr}_n) = \lambda_\mathbf{Var}(\mathtt{cas}_{m,n}) = (\mathsf{O}, \mathsf{Q})$ and $\lambda_\mathbf{Var}(\mathtt{ok}) = \lambda_\mathbf{Var}(n) = (\mathsf{P}, \mathsf{A})$ for every $m, n \geq 0$, and $\vdash_\mathbf{Var} = \{(\mathtt{rd}, \mathtt{rd})\} \cup \{(\mathtt{rd}, n) \mid n \geq 0\} \cup \{(\mathtt{wr}_n, \mathtt{wr}_n), (\mathtt{wr}_n, \mathtt{ok}) \mid n \geq 0\} \cup \{(\mathtt{cas}_{m,n}, \mathtt{cas}_{m,n}) \mid m, n \geq 0\} \cup \{(\mathtt{cas}_{m,n}, l) \mid l, m, n \geq 0\}$.

We associate an arbitrary (but fixed) ordering to each of the arena moves above, leaving its explicit definition unspecified.

The compound arenas $A \times B$ and $A \Rightarrow B$ are defined as follows. (Below, for any pairs of functions $f : S \to U$ and $g : T \to U$, we write $[f, g] : S + T \to U$ for the coproduct function, where $S + T$ stands for the disjoint sum of S and T.)

- A *product arena* $A \times B$ is a triple $((M_{A\times B}, <_{A\times B}), \lambda_{A\times B}, \vdash_{A\times B})$, where $M_{A\times B} = M_A + M_B$, $\lambda_{A\times B} = [\lambda_A, \lambda_B]$, and $n \vdash_{A\times B} m$ iff $n \vdash_A m \vee n \vdash_B m$. The associated order extends the union of orders $<_A \cup <_B$ with additional orderings $a <_{A\times B} b$ for every $a \in M_A$ and $b \in M_B$.
- An *arrow arena* $A \Rightarrow B$ is a triple $((M_{A\Rightarrow B}, <_{A\Rightarrow B}), \lambda_{A\Rightarrow B}, \vdash_{A\Rightarrow B})$, where $M_{A\Rightarrow B} = M_A + M_B$, $\lambda_{A\Rightarrow B} = [\overline{\lambda}_A, \lambda_B]$, and $n \vdash_{A\Rightarrow B} m$ iff $n \vdash_B m \vee (n \neq m \wedge n \vdash_A m) \vee (n \vdash_B n \wedge m \vdash_A m)$. The associated order $<_{A\Rightarrow B}$ is the same as $<_{A\times B}$.

Fig. 3 gives the game interpretation of each IA_{par} term $\Gamma \vdash M : \mathtt{T}$, specified as a Kleisli arrow $[\![\Gamma \vdash M : \mathtt{T}]\!] : [\![\Gamma]\!] \rightarrow [\![\mathtt{T}]\!]$ in $\mathcal{G}_F$.

$$
\begin{aligned}
&[\![\Gamma, x : \mathtt{T} \vdash x : \mathtt{T}]\!] = \boldsymbol{\pi}_{[\![\mathtt{T}]\!]}\\
&[\![\Gamma \vdash \lambda x :^{\mathtt{T}} .M : \mathtt{T}']\!] = \boldsymbol{\Lambda}_{[\![\mathtt{T}]\!]}([\![\Gamma, x : \mathtt{T} \vdash M : \mathtt{T}']\!])\\
&[\![\Gamma \vdash MN : \mathtt{T}]\!] = \langle [\![\Gamma \vdash M : \mathtt{T}' \to \mathtt{T}]\!], [\![\Gamma \vdash N : \mathtt{T}']\!] \rangle \mathbin{;} \mathbf{ev}_{[\![\mathtt{T}']\!],[\![\mathtt{T}]\!]}\\
&[\![\Gamma \vdash \mathsf{fix}_{\mathtt{T}} M : \mathtt{T}]\!] = \bigsqcup_i \sigma_i, \text{ where } \sigma_0 = \bot \text{ and } \sigma_{i+1} = \langle [\![\Gamma \vdash M : \mathtt{T} \to \mathtt{T}]\!], \sigma_i \rangle \mathbin{;} \mathbf{ev}_{[\![\mathtt{T}]\!],[\![\mathtt{T}]\!]}.\\
&[\![\Gamma \vdash n : \mathtt{nat}]\!] = \mathbf{!}_{[\![\Gamma]\!]} \mathbin{;} \mathit{cnst}_n\\
&[\![\Gamma \vdash M \star N : \mathtt{nat}]\!] = \langle [\![\Gamma \vdash M]\!], [\![\Gamma \vdash N]\!] \rangle \mathbin{;} \mathit{binop}_\star\\
&[\![\Gamma \vdash \mathsf{skip} : \mathtt{com}]\!] = \mathbf{!}_{[\![\Gamma]\!]} \mathbin{;} \mathit{skip}\\
&[\![\Gamma \vdash \mathsf{seq}\ M_1\ M_2 : \mathtt{T}]\!] = \langle [\![\Gamma \vdash M_1 : \mathtt{com}]\!], [\![\Gamma \vdash M_2 : \mathtt{T}]\!] \rangle \mathbin{;} \mathit{seq}_{[\![\mathtt{T}]\!]}\\
&[\![\Gamma \vdash \mathsf{if0}\ M\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2 : \mathtt{T}]\!] = \langle [\![\Gamma \vdash M : \mathtt{nat}]\!], [\![\Gamma \vdash M_1 : \mathtt{T}]\!], [\![\Gamma \vdash M_2 : \mathtt{T}]\!] \rangle \mathbin{;} \mathit{cond}_{[\![\mathtt{T}]\!]}\\
&[\![\Gamma \vdash \mathsf{assign}\ M\ N : \mathtt{com}]\!] = \langle [\![\Gamma \vdash M]\!], [\![\Gamma \vdash N]\!] \rangle \mathbin{;} \mathit{asgn}\\
&[\![\Gamma \vdash \mathsf{deref}\ M : \mathtt{nat}]\!] = [\![\Gamma \vdash M]\!] \mathbin{;} \mathit{deref}\\
&[\![\Gamma \vdash \mathsf{cas}\ L\ M\ N : \mathtt{com}]\!] = \langle [\![\Gamma \vdash L]\!], [\![\Gamma \vdash M]\!], [\![\Gamma \vdash N]\!] \rangle \mathbin{;} \mathit{cas}\\
&[\![\Gamma \vdash \mathsf{mkvar}\ M\ N\ L : \mathtt{var}]\!] = \langle [\![\Gamma \vdash M]\!] \mathbin{;} \mathit{acpt}, [\![\Gamma \vdash N]\!], [\![\Gamma \vdash L]\!] \mathbin{;} \mathit{cacpt} \rangle\\
&[\![\Gamma \vdash \mathsf{newvar}\ v = n\ \mathsf{in}\ M : \mathtt{T}]\!] = \mathit{ST}\left(\langle \mathit{id}_{[\![\Gamma]\!]}, !_{[\![\Gamma]\!]} ; \mathit{cell}_n \rangle ; \mathit{WO}\left([\![\Gamma, v : \mathtt{var} \vdash M : \mathtt{T}]\!]\right)\right)\\
&[\![\Gamma \vdash M_1\ \mathsf{or}\ M_2 : \mathtt{nat}]\!] = \langle [\![\Gamma \vdash M_1]\!], [\![\Gamma \vdash M_2]\!] \rangle \mathbin{;} \mathit{choice}\\
&[\![\Gamma \vdash M_1 \| \cdots \| M_\zeta : \mathtt{par}]\!] = \sigma_\zeta, \text{ where } \sigma_0 = \mathbf{!}_{[\![\Gamma]\!]} \mathbin{;} \mathit{pskip} \text{ and } \sigma_{i+1} = \langle \sigma_i, [\![\Gamma \vdash M_{i+1} : \mathtt{com}]\!] \rangle \mathbin{;} \mathit{cont}_{i+1}.\\
&[\![\Gamma \vdash P \gg_j M : \mathtt{par}]\!] = \langle [\![\Gamma \vdash P : \mathtt{par}]\!], [\![\Gamma \vdash M : \mathtt{com}]\!] \rangle \mathbin{;} \mathit{cont}_j
\end{aligned}
$$

Figure 3 The game interpretation of IA_{par} terms.

The strategies printed in *italic* fonts in the figure are defined (by their longest well-opened traces) as below.

- $\mathit{cnst}_n : \mathbf{1} \to \mathbf{N}$ is defined by the trace set $\{\mathtt{q}n\}$.
- $\mathit{binop}_\star : \mathbf{N}' \times \mathbf{N}'' \to \mathbf{N}$ is defined by the trace set $\{\mathtt{q}\mathtt{q}'m'\mathtt{q}''n''k \mid m \star n = k\}$.
- $\mathit{skip} : \mathbf{1} \to \mathbf{C}$ is defined by the trace set $\{\mathtt{run}{\cdot}\mathtt{done}\}$.
- $\mathit{seq}_A : \mathbf{C} \times A \to A$ is defined by the trace set $\{q{\cdot}\mathtt{run}{\cdot}\mathtt{done}{\cdot}t \mid q{\cdot}t \in T_{\mathit{WO}(\mathit{id}_A)}\}$.
- $\mathit{deref} : \mathbf{Var} \to \mathbf{N}$ is defined by the trace set $\{\mathtt{q}{\cdot}\mathtt{W}{\cdot}\mathtt{N}{\cdot}\mathtt{rd}{\cdot}n{\cdot}n \mid n \geq 0\}$.
- $\mathit{cas} : \mathbf{Var} \times \mathbf{N}_1 \times \mathbf{N}_2 \to \mathbf{N}$ is defined by the trace set $\{\mathtt{q}{\cdot}\mathtt{q}_1{\cdot}m{\cdot}\mathtt{q}_2{\cdot}n{\cdot}\mathtt{W}{\cdot}\mathtt{N}{\cdot}\mathtt{cas}_{m,n}{\cdot}k{\cdot}k \mid m, n, k \geq 0\}$.
- $\mathit{acpt} : (\mathbf{N} \Rightarrow \mathbf{C}) \to \mathbf{Var}$ is defined by the trace set $\{\mathtt{wr}_n{\cdot}\mathtt{run}{\cdot}\mathtt{q}{\cdot}n{\cdot}\mathtt{done}{\cdot}\mathtt{ok} \mid n \geq 0\}$.
- $\mathit{cacpt} : (\mathbf{N}_1 \Rightarrow \mathbf{N}_2 \Rightarrow \mathbf{N}) \to \mathbf{Var}$ is defined by the trace set $\{\mathtt{cas}_{m,n}{\cdot}\mathtt{q}{\cdot}\mathtt{q}_1{\cdot}m{\cdot}\mathtt{q}_2{\cdot}n{\cdot}k{\cdot}k \mid m, n, k \geq 0\}$.
- $\mathit{choice} : \mathbf{N}_1 \times \mathbf{N}_2 \to \mathbf{N}$ is defined by the trace set $\{\mathtt{q}{\cdot}\mathtt{q}_i{\cdot}n{\cdot}n \mid i \in \{1, 2\}, n \geq 0\}$.
- $\mathit{pskip} : \mathbf{1} \to (\mathbf{C}_1 \times \cdots \times \mathbf{C}_\zeta)_\bot$ is defined by the trace set $\{?\surd{\cdot}\mathtt{run}_{\rho(1)}{\cdot}\mathtt{done}_{\rho(1)} \cdots \mathtt{run}_{\rho(\zeta)}{\cdot}\mathtt{done}_{\rho(\zeta)} \mid \rho : \{1, \ldots, \zeta\} \mapsto \{1, \ldots, \zeta\}$ is a bijection $\}$.
- $\mathit{cont}_i : (\mathbf{C}'_1 \times \cdots \times \mathbf{C}'_\zeta)_\bot \times \mathbf{C} \to (\mathbf{C}_1 \times \cdots \times \mathbf{C}_\zeta)_\bot$ is specified by the traces $\{??'\surd'\surd s \mid s \in T_{\langle \sigma_1, \cdots, \sigma_\zeta \rangle}\}$, where the trace of each $\sigma_j : \mathbf{C}'_j \times \mathbf{C} \to \mathbf{C}_j$ is defined by $\{\mathtt{run}_j{\cdot}\mathtt{run}'_j{\cdot}\mathtt{done}'_j{\cdot}\mathtt{done}_j\}$ if $j \neq i$ and σ_i is defined by $\{\mathtt{run}_i{\cdot}\mathtt{run}'_i{\cdot}\mathtt{done}'_i{\cdot}\mathtt{run}{\cdot}\mathtt{done}{\cdot}\mathtt{done}_i\}$.
- $\mathit{cell}_n : \mathbf{1} \to \mathbf{Var}$ is the strategy defined by the set of *causal* traces whose initial value is n. A trace in $\mathbf{Var}$ is called a *causal* trace, if every $\mathtt{wr}_n$ move is immediately followed by a move $\mathtt{ok}$, and each $\mathtt{rd}$ or $\mathtt{cas}_{m,n}$ move is followed by a natural number stored in the variable just after the corresponding operation is finished. The value stored in the variable is determined by the last successful write: A *successful write* by n is identified

by a pair of successive moves, either $\mathtt{wr}_n{\cdot}\mathtt{ok}$ of a write operation or $\mathtt{cas}_{m,n}{\cdot}n$ of a CAS operation.

The newvar $v = n$ in M construct delimits the scope of variable v local to M and further forces the causality induced by the order of accesses to v. Unless bound in newvar, v behaves like a variable allocated in a volatile memory cell. We notice that, since every term M of type `com` or `par` under a scope of a local variable is executed exactly once, it must be interpreted, as done in [10], by a composition of well-opened strategy in $\mathit{WO}\,(\llbracket M \rrbracket)$ with a cell strategy in $\mathcal{G}$, written $\mathit{cell}_n : \mathbf{1} \rightarrow \mathbf{Var}$, which comprises of causal traces whose initial value is n.