

Asynchronous games 2

The true concurrency of innocence

Paul-André Melliès

Equipe Preuves Programmes Systèmes
CNRS & Université Paris 7

Abstract. In game semantics, one expresses the higher-order value passing mechanisms of the λ -calculus as sequences of atomic actions exchanged by a Player and its Opponent in the course of time. This is reminiscent of trace semantics in concurrency theory, in which a process is identified to the sequences of requests it generates. We take as working hypothesis that game semantics is, indeed, the trace semantics of the λ -calculus. This brings us to a notion of asynchronous game, inspired by Mazurkiewicz traces, which generalizes the usual notion of arena game. We then extract the true concurrency semantics of λ -terms from their interleaving semantics formulated as innocent strategies. This reveals that innocent strategies are positional strategies regulated by forward and backward interactive confluence properties. We conclude by defining a non uniform variant of the λ -calculus, whose game semantics is formulated as a trace semantics.

1 Introduction

Game semantics has taught us the art of converting the *higher-order* value passing mechanisms of the λ -calculus into sequences of *atomic* interactions exchanged by a Player and its Opponent in the course of time. This metamorphosis of higher-order syntax into interactive semantics has significantly sharpened our understanding of the simply-typed λ -calculus, either as a pure calculus, or as a calculus extended with programming features like recursion, conditional branching, local control, local states, references, non determinism, probabilistic choice, etc.

Game semantics is similar to *trace semantics* in concurrency theory. A process is commonly described as a symbolic device which interacts with its environment by emitting or receiving requests. A sequence of such requests is called a *trace*. The trace semantics of a process π is defined as the set of traces generated by this process. In many cases, this semantics characterizes the contextual behaviour of the process.

Game semantics develops quite the same story for the λ -calculus. The terminology changes obviously: requests are called *moves*, and traces are called *plays*. But everything works as in trace semantics: the semantics of a λ -term M of type A is the set of plays σ generated by the λ -term M ; and this set σ characterizes the contextual behaviour of the λ -term. One original aspect of game semantics however, not present in trace semantics, is that the type A defines a game, and that the set σ defines a *strategy* of that game.

The starting point of this work is that game semantics is *really* the trace semantics of the λ -calculus. The thesis is apparently ingenuous. But it is surprisingly subversive because it prescribes to reevaluate a large part of the technical and conceptual choices accepted in game semantics... in order to bridge the gap with concurrency theory. Three issues are raised here:

1. The treatment of duplication in mainstream game semantics (eg. in arena games) distorts the bond with trace semantics, by adding justification pointers to traces. According to our methodology, this particular treatment of duplication should be revisited. This is done in the first article of our series on asynchronous games [21]. We recall below the indexed and group-theoretic reformulation of arena games operated there.
2. Thirty years ago, a theory of *asynchronous traces* was formulated by Antoni Mazurkiewicz in order to relate the *interleaving* and *true concurrency* semantics of concurrent computations. Game semantics delivers an interleaving semantics of the λ -calculus, formulated as innocent strategies. What is the corresponding true concurrency semantics? The task of this second article on asynchronous games is to answer this question precisely.
3. Ten years ago, a series of full abstraction theorems for PCF were obtained by characterizing the interactive behaviour of λ -terms as either innocent, or history-free strategies, see [3,13,24]. We feel that the present work is another stage in the “full abstraction” program initiated by Robin Milner [23]. For the first time indeed, we do not simply characterize, but also derive the syntax of λ -terms from elementary causality principles, expressed in asynchronous transition systems. This reconstruction requires the mediation of [21] and of its indexed treatment of threads. This leads us to an *indexed* and *non-uniform* λ -calculus, from which the usual λ -calculus follows by group-theoretic principles. In this variant of the λ -calculus, the game semantics of a λ -term may be directly formulated as a trace semantics, performing the syntactic exploration or parsing of the λ -term.

The treatment of duplication. The language of traces is limited, but sufficient to interpret the *affine* fragment of the λ -calculus, in which every variable occurs at most once in a λ -term. In this fragment, every trace (=play) generated by a λ -term is an alternating sequence of received requests (=Opponent moves) and emitted requests (=Player moves). And a request appears at most once in a trace.

The extension from the affine fragment to the whole λ -calculus requires to handle semantically the duplication mechanisms. This is a delicate matter. Several solutions have been considered, and coexist today in the literature. By way of illustration, take the λ -term chosen by Church to interpret the natural number 2:

$$M = \lambda f. \lambda x. f f x$$

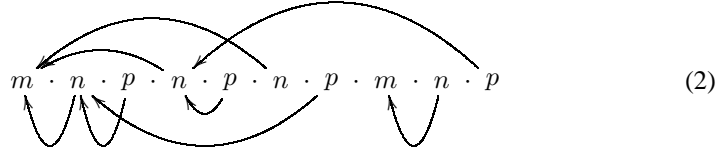
In front of two λ -terms P and Q , the λ -term M duplicates its first argument P , and applies it twice to its second argument Q . This is performed syntactically by two β -reductions:

$$MPQ \longrightarrow_{\beta} (\lambda x. PPx)Q \longrightarrow_{\beta} PPQ \quad (1)$$

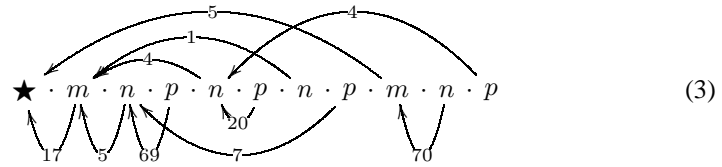
Obviously, the remainder of the computation depends on the λ -terms P and Q . The game-theoretic interpretation of the λ -term M has to anticipate all cases. This requires to manipulate several threads of the λ -term P simultaneously — and many more than two copies when the λ -term $P_{(1)}$ uses its first argument $P_{(2)}$ several times in $P_{(1)}P_{(2)}Q$.

Now, the difficulty is that each thread of P should be clearly distinguished. A compact and elegant solution has been introduced by Martin Hyland, Luke Ong and Hanno Nickau, in their *arena games* [13,24]. We recall that an *arena* is a forest, whose nodes are the *moves* of the game, and whose branches $m \vdash n$ are oriented in order to express the idea that the move m *justifies* the move n . A move n is *initial* when it is a root of the forest, or alternatively, when there is no move m such that $m \vdash n$. A *justified play* is then defined as a pair $(m_1 \cdots m_k, \varphi)$ consisting of a sequence of moves $m_1 \cdots m_k$ and a partial function $\varphi : \{1, \dots, k\} \rightarrow \{1, \dots, k\}$ providing the so-called *pointer structure*. The partial function φ associates to every occurrence i of a non-initial move m_i the occurrence $\varphi(i)$ of a move $m_{\varphi(i)}$ such that $m_{\varphi(i)} \vdash m_i$. One requires that $\varphi(i) < i$ to ensure that the justifying move $m_{\varphi(i)}$ occurs before the justified move m_i . Finally, the partial function φ is never defined on the occurrence i of any initial move m_i .

The pointer structure φ provides the necessary information to distinguish the several threads of a λ -term in the course of interaction — typically the several threads or copies of P in example (1). The pointer structure φ is conveniently represented by drawing “backward pointers” between occurrences of the sequence $m_1 \cdots m_k$. By way of illustration, consider the arena $m \vdash n \vdash p$ in which the only initial move is m . A typical justified play (s, φ) of this arena is represented graphically as:



Because adding justification pointers distorts the bond with trace semantics, in particular with Mazurkiewicz traces, we shift in [21] to another management principle based on *thread indexing*, already considered in [3,12]. The idea is to assign to each copy of the λ -term P in example (1) a natural number $k \in \mathbb{N}$ (its index) which characterizes the thread among the other copies of P . In the case of the justified play (2), this amounts to (a) adding a dumb move \star in order to justify the initial moves of the sequence, (b) indexing every justification pointer of the resulting sequence with a natural number:



then finally (c) encoding the sequence (3) as the sequence of indexed moves below:

$$m_{17} \cdot n_{17,5} \cdot p_{17,5,69} \cdot n_{17,4} \cdot p_{17,4,20} \cdot n_{17,1} \cdot p_{17,5,7} \cdot m_5 \cdot n_{5,70} \cdot p_{17,4,4}. \quad (4)$$

Obviously, the translation of a justified play (s, φ) depends on the choice of indices put on its justification pointers. Had we not taken sides with trace semantics and concur-

rency theory, we would be tempted (as most people do in fact) to retract to the notation (2) which is arguably simpler than its translation (4). But we carry on instead, and regulate the indexing by asking that two justification pointers starting from different occurrences i and j of the same move n , and ending on the same occurrence $\varphi(i) = \varphi(j)$, receive different indices k and k' . This indexing policy ensures that every indexed move occurs at most once in the sequence (4). In this way, we are back to the simplicity of the affine fragment of the λ -calculus.

An interesting point remains to be understood: what can be said about two different encodings of the same justified play? The first article of our series [21] clarifies this point. Every game is equipped with a left and right group actions on moves:

$$G \times M \times H \longrightarrow M \quad (g, m, h) \mapsto g \cdot m \cdot h \quad (5)$$

where M denotes the set of (indexed) moves, and G and H the two groups acting on M . Intuitively, the right (resp. left) group action operates on an indexed move m_{k_0, \dots, k_j} by altering the indices k_{2i} assigned by Opponent (resp. the indices k_{2i+1} assigned by Player). The *orbit* of an (indexed) move m_{k_0, \dots, k_j} , is precisely the set of all (indexed) moves of the form $m_{k'_0, \dots, k'_j}$. Now, the action of $g \in G$ and $h \in H$ on (indexed) moves induces a left and right action on plays, defined pointwise:

$$g \cdot (m_1 \cdots m_k) \cdot h = (g \cdot m_1 \cdot h) \cdots (g \cdot m_k \cdot h) \quad (6)$$

It appears that the justified plays of the original arena game coincide with the orbits of plays modulo the left and right group actions. Typically, the justified play (2) is just the play (4) modulo pointwise group action (6). One nice contribution of this second article on asynchronous games, is to explain the *syntactic* meaning of the group action (5). This is done in a non-uniform variant of the λ -calculus introduced in Section 6.

Asynchronous traces. After these necessary preliminaries on thread indexing, we shift to the core of this article: true concurrency vs. interleaving in game semantics. Two requests a and b are called *independent* in a process π when they can be emitted or received by π in any order, without interference. Independence of a and b is represented graphically by *tiling* the two sequences $a \cdot b$ and $b \cdot a$ in the 2-dimensional diagram:

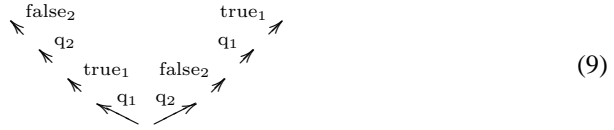
$$\begin{array}{ccc}
 & & \pi' \\
 & \nearrow^b & \nwarrow^a \\
 \pi_1 & & \pi_2 \\
 & \nwarrow^a & \nearrow^b \\
 & & \pi
 \end{array}
 \quad \sim \quad
 \begin{array}{ccc}
 & & \pi \\
 & \nearrow^a & \nwarrow^b \\
 \pi_1 & & \pi_2 \\
 & \nwarrow^b & \nearrow^a \\
 & & \pi
 \end{array}
 \quad (7)$$

The *true concurrency* semantics of a process π is then extracted from its *interleaving* semantics, by quotienting the traces of π modulo the *homotopy equivalence* \sim obtained by permuting independent requests. Expressing concurrency by permuting events is a pervading idea in concurrency theory. It originates from the work of Antoni Mazurkiewicz on asynchronous traces over a partially ordered alphabet [18,19] and appears in the theory of asynchronous transition systems [25,15,27] as well as in rewriting theory [20]. The n -dimensional presentation of the idea, and the connection to (directed) homotopy in cubical sets, is formulated in [26,10].

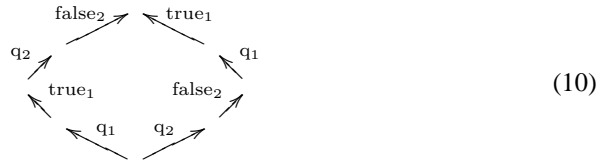
In comparison, mainstream game semantics is still very much 1-dimensional. By way of illustration, take the sequential boolean game \mathbb{B} , starting by an Opponent question q followed by a Player answer true or false:



The plays of the tensor product $\mathbb{B} \otimes \mathbb{B}$ are obtained by interleaving the plays of the two instances \mathbb{B}_1 and \mathbb{B}_2 of \mathbb{B} . Thus, (a fragment of) the game $\mathbb{B} \otimes \mathbb{B}$ looks like this:

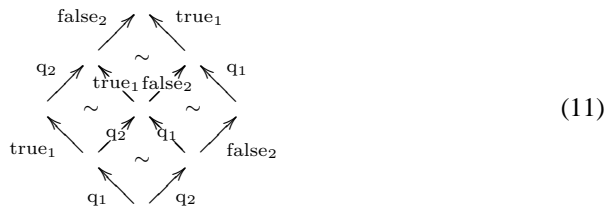


We point out in [22] that the two plays in (9) are different from a *procedural* point of view, but equivalent from an *extensional* point of view, since both of them realize the “extensional value” (*true, false*). We thus bend the two paths, and tile the resulting 2-dimensional octagon as follows:



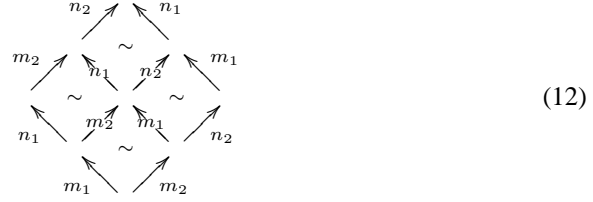
By doing so, we shift from usual sequential games played on trees, to sequential games played on *directed acyclic graphs* (dags). This enables us to analyze the extensional content of sequential games, and to obtain a game-theoretic proof of Ehrhard’s collapse theorem [9].

However instructive, the framework developed in [22] is not entirely satisfactory, because the permutation tiles are *global* — that is, they involve more than two moves in general. In contrast, the asynchronous game model presented here admits only *local* permutations tiles, similar to tile (7). By way of illustration, this decomposes the global tile (10) into four local tiles:



It is interesting that by shifting from (10) to (11), concurrent plays like $q_1 \cdot q_2$ appear in the model. From our point of view, this means that a satisfactory theory of sequentiality requires a concurrent background.

The non-uniform λ -calculus. Here comes the most surprising, most difficult, and maybe most controversial, part of the paper. In Section 2, we define an asynchronous game as an event structure whose events are polarized $+1$ for Player moves and -1 for Opponent moves. This polarization of events gives rise to a new class of events $m \cdot n$ consisting of an Opponent move m followed by a Player move n . We call *OP-moves* any such pair of moves. Just like ordinary moves, two *OP-moves* $m_1 \cdot n_1$ and $m_2 \cdot n_2$ may be permuted in a play, in the following way:



The permutation diagram (12) induces an homotopy relation \sim_{OP} between plays. The dual relation \sim_{PO} is defined symmetrically, by permuting *PO-moves* $n \cdot m$ instead of *OP-moves*, where by *PO-move* $n \cdot m$ we mean an Opponent move n followed by a Player move m . Note that both \sim_{OP} and \sim_{PO} preserve *alternation* of plays.

Now, there is a well-established theory of *stable* asynchronous transition systems, see for instance [25,15,20], in which every sequence of transitions s is characterized (modulo homotopy) as a directed acyclic graph of so-called *canonical forms*. The canonical form of a transition a in a sequence $s \cdot a$ of transitions, expresses the cascade of transitions necessary for the enabling of the transition a . Formally, a sequence of transitions $t \cdot a$ is a canonical form of $s \cdot a$ when (1) $s \cdot a \sim t \cdot a \cdot t'$ for some t' , and (2) whenever $t \sim t' \cdot b$, then a cannot be permuted before b . The stability property ensures that this canonical form $s \cdot a$ is *unique*.

The theory may be applied to the asynchronous transition system with *OP-moves* as transitions, which happens to be stable. From this follows that every *OP-move* $m \cdot n$ in an alternating play $s \cdot m \cdot n$ has a unique *canonical form*. Strikingly, this canonical form is precisely the so-called *Player view* $\lceil s \cdot m \cdot n \rceil$ of the play $s \cdot m \cdot n$, introduced by Martin Hyland *et al.* in arena games [13,24] and adapted to asynchronous games in Section 3.

We claim that here lies the essence of the syntax of the λ -calculus. It has been already noted in [7] that every Player view of a justified play (s, ϕ) corresponds to the branch of an η -long Böhm tree. When adapted to the indexed treatment of threads described in [21] and recalled above, the correspondence defines the branch of a *non-uniform* η -long Böhm tree. The definition of the non-uniform λ -calculus is given in Section 6. A nice feature of the calculus is that the strategy σ associated to a non-uniform λ -term may be defined in the same way as a trace semantics. This is also done in Section 6.

Related works. The idea of relating a dynamic and a static semantics of interaction is formulated for the first time by Patrick Baillot *et al.* in [6]. The idea reappears implicitly in the concurrent game semantics introduced by Samson Abramsky and the author [5],

in which games are complete lattices of positions, and strategies are closure operators. As a closure operator, every strategy is at the same time an increasing function on positions (the dynamic point of view) and a set of positions (the static point of view). The present paper is the result of a long journey (five years!) to connect this concurrent game semantics to mainstream sequential game semantics. See also [2].

Martin Hyland and Andrea Schalk develop in [14] a notion of games on graphs quite similar to the constructions presented here and in [22]. One difference is the treatment of duplication: backtracking in [14,22], repetitive and indexed here. From this choice follows that the permutation tilings are global in [14,22] whereas they are local here. Another difference is that our positions are defined as *ideals* of moves.

Outline. In the remainder of the article, we define our notion of asynchronous game (Section 2) and adapt the usual definition of innocent strategy to our setting (Section 3). We then characterize the innocent strategies in two ways: diagrammatically (Section 4) and positionally (Section 5). This leads to a non-uniform variant of the λ -calculus, for which we define a trace semantics, and which we relate to the usual λ -calculus (Section 6). Finally, we deliver a series of refinements of asynchronous games (Section 7).

2 Asynchronous games

We choose the simplest possible definition of *asynchronous game*, in which the only relation between moves is an order relation \leq which reformulates the *justification* structure of arena games. This is enough to describe the language PCF, a simply-typed λ -calculus enriched with arithmetic, conditional branching, and recursion. Other more expressive variants are discussed in section 7.

Event structures. An *event structure* is an ordered set (M, \leq) such that every element $m \in M$ defines a *finite* downward-closed subset $m \downarrow = \{n \in M \mid n \leq m\}$.

Asynchronous games. An *asynchronous game* is a triple $A = (M_A, \leq_A, \lambda_A)$ consisting of:

- an event structure (M_A, \leq_A) whose elements are called the *moves* of the game,
- a function $\lambda_A : M_A \longrightarrow \{-1, +1\}$ which associates to every move a *polarity* $+1$ (for the Player moves) or -1 (for the Opponent moves).

Positions. A *position* of an asynchronous game A is any *finite* downward closed subset of (M_A, \leq_A) .

The positional lattice. The set of positions of A is denoted $\mathcal{D}(A)$. Since positions are ordered by inclusion, and closed under finite union, the partial order $(\mathcal{D}(A), \subseteq)$ defines a sup-lattice. The empty position, which is the least element of $(\mathcal{D}(A), \subseteq)$, is denoted $*_A$. Positions are also closed under arbitrary *nonempty* intersection. Adding a top element \top to $(\mathcal{D}(A), \subseteq)$ provides a neutral element to intersection, and induces a

complete lattice $\mathcal{D}(A)^\top = (\mathcal{D}(A), \subseteq)^\top$. The greatest least bound and least upper bound of a family $(x_i)_{i \in I}$ of positions in $\mathcal{D}(A)$ are computed respectively as:

$$\bigwedge_{i \in I} x_i = \begin{cases} \top & \text{if } I \text{ is empty,} \\ \bigcap_{i \in I} x_i & \text{otherwise,} \end{cases}$$

$$\bigvee_{i \in I} x_i = \begin{cases} \top & \text{if } \bigcup_{i \in I} x_i \text{ is infinite,} \\ \bigcup_{i \in I} x_i & \text{if } \bigcup_{i \in I} x_i \text{ is finite.} \end{cases}$$

We call $\mathcal{D}(A)^\top$ the *positional lattice* associated to the game A .

The positional graph. Every asynchronous game A induces a graph $\mathcal{G}(A)$:

- whose nodes are the positions $x, y \in \mathcal{D}(A)$,
- whose edges $m : x \rightarrow y$ are the moves verifying $y = x + \{m\}$, where $+$ denotes disjoint union, or equivalently, that $y = x \cup \{m\}$ and that the move m is not element of x .

We call this graph $\mathcal{G}(A)$ the *positional graph* of the game A . We write $s : x \rightarrow y$ for a path

$$x \xrightarrow{m_1} x_1 \xrightarrow{m_2} \dots \xrightarrow{m_{k-1}} x_{k-1} \xrightarrow{m_k} y$$

between two positions x and y . Note that there is no repetition of move in the sequence:

$$\forall i, j \in \{1, \dots, k\}, \quad i \neq j \Rightarrow m_i \neq m_j.$$

The target y of the path $s : x \rightarrow y$ may be deduced from the source x and the sequence of moves m_1, \dots, m_k , using the equation:

$$y = x + \bigcup_{1 \leq i \leq k} \{m_i\}.$$

A path of $\mathcal{G}(A)$ is thus characterized by its source (or alternatively, its target) and the sequence of moves $m_1 \cdot \dots \cdot m_k$.

Homotopy. Given two paths $s, s' : x \rightarrow y$ in $\mathcal{G}(A)$, we write $s \sim^1 s'$ when $s = m \cdot n$ and $s' = n \cdot m$ for two moves $m, n \in M_A$. The *homotopy equivalence* \sim between paths is defined as the least equivalence relation containing \sim^1 , and closed under composition; that is, for every four paths $s_1 : x_1 \rightarrow x_2$ and $s, s' : x_2 \rightarrow x_3$ and $s_2 : x_3 \rightarrow x_4$:

$$s \sim s' \Rightarrow s_1 \cdot s \cdot s_2 \sim s_1 \cdot s' \cdot s_2.$$

We also use the notation \sim in our diagrams to indicate that two (necessarily different) moves m and n are permuted:

$$\begin{array}{ccc} & z & \\ n \nearrow & & \nwarrow m \\ y_1 & \sim & y_2 \\ m \nwarrow & & \nearrow n \\ & x & \end{array} \quad (13)$$

Note that our current definition of asynchronous game implies that two paths $s_1 : x_1 \rightarrow y_1$ and $s : x_2 \rightarrow y_2$ are homotopic iff $x_1 = x_2$ and $y_1 = y_2$. Thus, homotopy becomes informative only in the presence of an *independence* relation between moves, see Section 7.

Alternating paths. A path $m_1 \cdots m_k : x \rightarrow y$ is *alternating* when:

$$\forall i \in \{1, \dots, k-1\}, \quad \lambda_A(m_{i+1}) = -\lambda_A(m_i).$$

Alternating homotopy. Given two paths $s, s' : x \rightarrow y$ in $\mathcal{G}(A)$, we write $s \sim_{OP}^1 s'$ when $s = m_1 \cdot n_1 \cdot m_2 \cdot n_2$ and $s' = m_2 \cdot n_2 \cdot m_1 \cdot n_1$ where the moves $m_1, m_2 \in M_A$ are Opponent and the moves $n_1, n_2 \in M_A$ are Player. The situation is summarized in diagram (12). The relation \sim_{OP} is defined as the least equivalence relation containing \sim_{OP}^1 and closed under composition. Note that $s \sim_{PO} s'$ implies $s \sim s'$, but that the converse is not true, since in diagram (12) one has $m_1 \cdot n_1 \cdot m_2 \cdot n_2 \sim m_1 \cdot n_2 \cdot m_2 \cdot n_1$ without having $m_1 \cdot n_1 \cdot m_2 \cdot n_2 \sim_{OP} m_1 \cdot n_2 \cdot m_2 \cdot n_1$.

Plays. A play is a path starting from the empty position $*_A$:

$$*_A \xrightarrow{m_1} x_1 \xrightarrow{m_2} \cdots \xrightarrow{m_{k-1}} x_{k-1} \xrightarrow{m_k} x_k$$

in the positional graph $\mathcal{G}(A)$. The set of plays is noted P_A .

Equivalently, a play of A is a finite sequence $s = m_1 \cdots m_k$ of moves, without repetition, such that the set $\{m_1, \dots, m_j\}$ is downward closed in (M_A, \leq_A) for every $1 \leq j \leq k$.

Strategy. A strategy σ is a set of alternating plays of even length such that:

- the strategy $s \in \sigma$ contains the empty play,
- every nonempty play $s \in \sigma$ starts by an Opponent move,
- σ is closed by even-length prefix:

$$\forall s \in P_A, \forall m, n \in M_A, \quad s \cdot m \cdot n \in \sigma \Rightarrow s \in \sigma,$$

- σ is deterministic: $\forall s \in P_A, \forall m, n, n' \in M_A,$

$$s \cdot m \cdot n \in \sigma \text{ and } s \cdot m \cdot n' \in \sigma \Rightarrow n = n'.$$

We write $\sigma : A$ when σ is a strategy of A .

3 Innocent strategies

The notion of *innocence* has been introduced by Martin Hyland, Luke Ong and Hanno Nickau in the framework of arena games [13,24]. It is designed to capture the interactive behaviour of the simply-typed λ -calculus with a constant Ω for non-termination, either formulated as η -long Böhm trees [7], as proofs of Polarized Linear Logic [17], or (after a continuation-passing style translation) as PCF programs augmented with local control [16,4,11]. Asynchronous games enable to reformulate the notion of *innocence* in a concurrency friendly way. The original definition of innocence is based on the notion of *Player view* of a justified play (s, φ) , defined using the pointer structure φ . In asynchronous games, the situation is slightly simpler than in arena games, because the play s is non repetitive. In particular, there is no need to distinguish a move m from its occurrences in the play. More: every play s comes with an implicit pointer structure φ derived from the causality relation \leq between moves, as follows.

Justification pointers. Suppose that m and n are two different moves of an asynchronous game A . We write $m \vdash_A n$, and say that m justifies n , when:

- $m \leq_A n$, and
- for every move $p \in M_A$ such that $m \leq_A p \leq_A n$, either $m = p$ or $p = n$.

View extraction. We define the binary relation $\overset{\text{OP}}{\rightsquigarrow}$ as the smallest relation between alternating plays such that:

$$s_1 \cdot m \cdot n \cdot s_2 \overset{\text{OP}}{\rightsquigarrow} s_1 \cdot s_2$$

for every alternating play s_1 and nonempty path s_2 such that m is an Opponent move which does not justify any move in s_2 , and n is a Player move which does not justify any move in s_2 .

Player view. The relation $\overset{\text{OP}}{\rightsquigarrow}$ defines a noetherian and locally confluent rewriting system on alternating plays. By Newman's lemma, the rewriting system is confluent. Thus, every alternating play $s \in P_A$ induces a unique normal form noted $\ulcorner s \urcorner \in P_A$ and called its *Player view*:

$$s \overset{\text{OP}}{\rightsquigarrow} s_1 \overset{\text{OP}}{\rightsquigarrow} \dots \overset{\text{OP}}{\rightsquigarrow} s_k \overset{\text{OP}}{\rightsquigarrow} \ulcorner s \urcorner.$$

Asynchronous innocence. A strategy σ is *innocent* in an asynchronous game A when for every plays $s, t \in \sigma$, for every Opponent move $m \in M_A$ and Player move $n \in M_A$:

$$s \cdot m \cdot n \in \sigma \text{ and } t \cdot m \in P_A \text{ and } \ulcorner s \cdot m \urcorner \sim_{OP} \ulcorner t \cdot m \urcorner \Rightarrow t \cdot m \cdot n \in \sigma.$$

Asynchronous innocence is equivalent to usual innocence in the intuitionistic fragment [13,24]. In that fragment, indeed, every move has at most one justifying move, and thus, the two Player views $\ulcorner s \cdot m \urcorner$ and $\ulcorner t \cdot m \urcorner$ are \sim_{OP} -equivalent iff they are equal. On the other hand, asynchronous innocence generalizes the usual notion of innocence to more "concurrent" arenas, in which several moves n_1, \dots, n_k may justify the same move m — a situation which does not occur in arena games associated to intuitionistic types.

4 Diagrammatic innocence

In this section, we reformulate diagrammatically the notion of *innocence* in asynchronous games.

Backward consistency. A strategy σ is called *backward consistent* (see Figure 1 in Appendix) when for every play $s_1 \in P_A$, for every path s_2 , for every moves $m_1, n_1, m_2, n_2 \in M_A$, it follows from

$$s_1 \cdot m_1 \cdot n_1 \cdot m_2 \cdot n_2 \cdot s_2 \in \sigma \text{ and } \neg(n_1 \vdash_A m_2) \text{ and } \neg(m_1 \vdash_A n_2)$$

that

$$\neg(n_1 \vdash_A n_2) \text{ and } \neg(m_1 \vdash_A m_2) \text{ and } s_1 \cdot m_2 \cdot n_2 \cdot m_1 \cdot n_1 \cdot s_2 \in \sigma.$$

Forward consistency. A strategy σ is called *forward consistent* (see Figure 2 in Appendix) when for every play $s_1 \in P_A$ and for every moves $m_1, n_1, m_2, n_2 \in M_A$, it follows from

$$s_1 \cdot m_1 \cdot n_1 \in \sigma \text{ and } s_1 \cdot m_2 \cdot n_2 \in \sigma \text{ and } m_1 \neq m_2$$

that

$$n_1 \neq n_2 \text{ and } s_1 \cdot m_1 \cdot n_1 \cdot m_2 \cdot n_2 \in \sigma.$$

We prove by a diagrammatic reasoning inspired by Rewriting Theory that, for every strategy σ of an asynchronous game A :

Proposition 1 (diagrammatic characterization). *The strategy σ is innocent iff it is backward and forward consistent.*

5 Positional innocence

We establish below the main result of the paper: innocent strategies are positional strategies (Theorem 2). We then characterize innocent strategies as positional strategies (Proposition 3) and identify them as concurrent strategies in the sense of [5] (Proposition 4).

Positional strategy. A strategy $\sigma : A$ is called *positional* when for every two plays $s_1, s_2 : *A \rightarrow x$ in the strategy σ , and every path $t : x \rightarrow y$ of $\mathcal{G}(A)$, one has:

$$s_1 \sim s_2 \text{ and } s_1 \cdot t \in \sigma \Rightarrow s_2 \cdot t \in \sigma.$$

Every positional strategy is characterized by the set of positions of $\mathcal{D}(A)$ it reaches, defined as:

$$\sigma^\bullet = \{x \in \mathcal{D}(A), \exists s \in \sigma, s : *A \rightarrow x\}.$$

Theorem 2 (positionality). *Every innocent strategy σ is positional.*

The positional characterization of innocence (Proposition 3) works in any asynchronous game in which justification is alternated, that is, where $m \vdash n$ implies $\lambda(n) = -\lambda(m)$ for every move m and n . In particular, it works in any interpretation of a formula of intuitionistic linear logic.

Proposition 3 (positional characterization). *A positional strategy σ is innocent iff the set σ^\bullet of positions satisfies:*

- σ^\bullet is closed under intersection: $x, y \in \sigma^\bullet \Rightarrow x \cap y \in \sigma^\bullet$,
- σ^\bullet is closed under union: $x, y \in \sigma^\bullet \Rightarrow x \cup y \in \sigma^\bullet$,
- forward confluence: if $\sigma^\bullet \ni x \xrightarrow{m} w \rightarrow z \in \sigma^\bullet$ and m is an Opponent move, then there exists a unique Player move $w \xrightarrow{n} y$ such that $\sigma^\bullet \ni y \rightarrow z \in \sigma^\bullet$,
- backward confluence: if $\sigma^\bullet \ni x \rightarrow w \xrightarrow{n} z \in \sigma^\bullet$ and n is a Player move, then there exists a unique Opponent move $y \xrightarrow{m} w$ such that $\sigma^\bullet \ni x \rightarrow y \in \sigma^\bullet$,
- initial condition: $*_A$ is element of σ^\bullet .

Proposition 4. *Every innocent strategy $\sigma : A$ defines a closure operator σ^\bullet on the complete lattice $\mathcal{D}(A)^\top$ of positions.*

This series of properties explicates the true concurrency nature of innocence. Proposition 4 bridges sequential arena games with concurrent games as formulated in [5]. In particular, positionality implies that strategies may be composed just as relations, or as cliques in the hypercoherence space model [8].

If the reader finds the idea of *positionality* difficult to grasp, we hope that the Proposition below will clarify the situation. It is quite straightforward to define a notion of *innocent* counter-strategy τ interacting against the strategy σ . The counter-strategy τ may *withdraw* at any stage of the interaction. Every such withdraw of τ induces an even-length play $s : *A \rightarrow x$ in the strategy τ , whose target position $x \in \tau^\bullet$ is of even cardinality. Our next result states that the static evaluation (by intersection) of σ^\bullet against τ^\bullet coincides with the dynamic evaluation (by interaction) of σ against τ .

Proposition 5. *For every position $x \in \mathcal{D}(A)$:*

$$\sigma^\bullet \cap \tau^\bullet = \{x\} \iff \sigma \cap \tau = \{s\} \text{ and } s : *A \rightarrow x.$$

It is nearly routine to construct a category \mathcal{G} with asynchronous games as objects, and innocent strategies as morphisms. The only difficulty is to interpret the exponentials, which is done by equipping every game with a left and right group action, in the spirit of [21]. The resulting category \mathcal{G} defines a model of intuitionistic linear logic without additives. The usual category of arena games and innocent strategies [13,24] embeds fully and faithfully (as a cartesian closed category) in the kleisli category associated to the category \mathcal{G} and to its comonad !.

6 The non uniform λ -calculus

We introduce a non-uniform variant of the λ -calculus. It is called *non-uniform* because the argument of a function $\lambda x.P$ is not a λ -term Q , but a vector \vec{Q} of λ -terms Q_i where $i \in \mathbb{N}$ is an index for each occurrence $x(i)$ (or function call) of the variable x in P . The calculus is affine in nature (never two occurrences of $x(i)$ occur in the same term), but the simply-typed λ -calculus may be encoded in it, thanks to group-theoretic ideas developed in our first article on asynchronous games [21].

Definition of the calculus. The non-uniform λ -terms P and vectors of arguments \vec{Q} are defined by mutual induction:

$$\begin{array}{l} P ::= x(i) \text{ located variable} \\ \quad | P \vec{Q} \text{ application} \\ \quad | \lambda x.P \text{ abstraction} \end{array}$$

$$\vec{Q} ::= (Q_i)_{i \in \mathbb{N}} \text{ vector of non-uniform } \lambda\text{-terms indexed by an integer } i \in \mathbb{N}$$

where a located variable $x(i)$ consists of a variable x in the usual sense, and an integer $i \in \mathbb{N}$. We require that every located variable $x(i)$ appears at most once in a term. Note that a non-uniform λ -term is generally infinite. The β -reduction is defined as

$$(\lambda x.P) \vec{Q} \rightarrow_\beta P[x(i) := Q_i]$$

where $P[x(i) := Q_i]$ denotes the non-uniform λ -term obtained by replacing each located variable $x(i)$ in P by the non-uniform λ -term Q_i . The non-uniform λ -terms are typed by the simple types of the λ -calculus, built on the base type α :

$$x(i) : A \vdash x(i) : A \quad \frac{\Gamma \vdash P : A \Rightarrow B \quad (\Delta_i \vdash Q_i : A)_{i \in \mathbb{N}}}{\Gamma, \Delta_0, \Delta_1, \Delta_2, \dots \vdash P \vec{Q} : B}$$

$$\frac{\Gamma, x(i_0) : A, x(i_1) : A, x(i_2) : A, \dots \vdash P : B}{\Gamma \vdash \lambda x. P : A \Rightarrow B}$$

Here, a context Γ, Δ, \dots may contain an infinite number of located variables, since the \Rightarrow -elimination rule involves a family of derivation trees $(\Delta_i \vdash Q_i : A)_{i \in \mathbb{N}}$. The point is that the \Rightarrow -introduction rule may migrate an infinite number of located variables $x(i)$ from the context to the λ -term.

Non-uniform η -long Böhm trees. The non-uniform η -long Böhm trees of simple type $A = A_1 \Rightarrow \dots \Rightarrow A_m \Rightarrow \alpha$ are of three kinds:

1. $\lambda x_1 \dots \lambda x_m. (y(i) \vec{Q}_1 \dots \vec{Q}_n)$ where
 - every variable x_j is of type A_j for $1 \leq j \leq m$,
 - the located variable $y(i)$ is of type $B = B_1 \Rightarrow \dots \Rightarrow B_n \Rightarrow \alpha$ for some type B ,
 - every non uniform η -long Böhm tree $(Q_k)_i$ is of type B_k , for $1 \leq k \leq n$ and $i \in \mathbb{N}$.
2. or Ω_B where Ω_B is a fixed constant of type B ,
3. or $\lambda x_1 \dots \lambda x_m. \mathcal{U}$ where \mathcal{U} is a fixed constant of type α , and every variable x_j is of type A_j , for $1 \leq j \leq m$.

Trace semantics. We describe a trace semantics for non-uniform η -long Böhm trees, which coincides with the game semantics delivered by our asynchronous game model. The Opponent moves are generated by the rule

$$R^- : \Omega_A \longrightarrow \lambda x_1 \dots \lambda x_m. \mathcal{U}$$

where $A = A_1 \Rightarrow \dots \Rightarrow A_m \Rightarrow \alpha$ and the variable x_j is of type A_j for every index $1 \leq j \leq m$. The Player moves are generated by the rule

$$R_{x(i)}^+ : \mathcal{U} \longrightarrow x(i) \vec{\Omega}_{A_1} \dots \vec{\Omega}_{A_m}$$

where $x(i)$ is a located variable of type $A = A_1 \Rightarrow \dots \Rightarrow A_m \Rightarrow \alpha$, and $\vec{\Omega}_{A_j}$ is the vector which associates to every index $i \in \mathbb{N}$ the constant Ω_{A_j} , for every $1 \leq j \leq m$. Last point, every move from an η -long Böhm tree is labelled by a subtree of the type A , once translated in linear logic as an infinite formula, using the equation $A \Rightarrow B = !A \multimap B$, and the definition of the exponential modality as infinite tensor: $!A = \otimes_{i \in \mathbb{N}} A$.

Uniformity and bi-invariance. The usual (uniform) η -long Böhm trees of the λ -calculus are extracted from their non-uniform counterpart by applying a *bi-invariance* principle introduced in [21]. As recalled in the introduction, see (5), every game there is equipped with a left and right group action on moves. A strategy σ is called *bi-invariant* when, for every play $s \in \sigma$ and every right action $h \in H$, there exists a left action $g \in G$ such that $g \cdot s \cdot h \in \sigma$. This characterizes the strategies which are “blind to thread indexing”, and thus the strategies which behave as if they were defined directly in an arena game. The concept of bi-invariance remains formal and enigmatic in [21]. Here, quite fortunately, the non-uniform λ -calculus provides a simple syntactical explanation to this concept of bi-invariance, what we explain now.

Every intuitionistic type A defines a left and right group action (5) on the asynchronous game $[A]$ interpreting it in the asynchronous game model. These two group actions may be understood syntactically as acting on the non-uniform η -long Böhm trees P of type A , as follows: the effect of a right group action $h \in H$ is to permute the indices inside the vectors of arguments \vec{Q} in P , while the effect of a left group action $g \in G$ is to permute the indices of the located variables $x(i)$ in P . By analogy with [21], a non-uniform η -long Böhm tree P is called *bi-invariant* when for every permutation $h \in H$, there is a permutation $g \in G$ such that $g \cdot P \cdot h = P$. It is not difficult to see that an η -long Böhm tree in the usual λ -calculus is just a *bi-invariant* η -long Böhm tree in the non-uniform λ -calculus, modulo left group action (that is, permutation of the indices of the located variables.) For instance, let P_j denote the non-uniform η -long Böhm tree $P_j = \lambda x.\lambda y.(x(j) \vec{y})$ of type $A = (\alpha \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha)$, where \vec{y} associates to every index $i \in \mathbb{N}$ the located variable $y(i)$. Obviously, P_j is bi-invariant, and represents the uniform η -long Böhm tree $\lambda x.\lambda y.x y$ of same type A . Note that P_j is equivalent to any P_k modulo left group action. The trace (or game) semantics of P_j is given by:

$$\Omega_A \xrightarrow{m} \lambda x.\lambda y.\mathcal{U} \xrightarrow{n} \lambda x.\lambda y.(x(j) \vec{\Omega}_\alpha) \xrightarrow{m_k} \lambda x.\lambda y.(x(j) \vec{Q}_k) \xrightarrow{n_k} \dots$$

Here, the move m by Opponent (labelled by the type A) asks for the value of the head variable of P_j , and the move n by Player (labelled by the type $(\alpha \Rightarrow \alpha)_j$) answers $x(j)$; then, the move m_k by Opponent (labelled by α_k in $(\alpha \Rightarrow \alpha)_j$) asks for the value of the head variable of the k -th argument of $x(j)$, inducing the vector of arguments $(Q_k)_i = \Omega_\alpha$ for $i \neq k$ and $(Q_k)_k = \mathcal{U}$; finally the move n_k by Player (labelled by α_k) answers $y(k)$, etc... This example illustrates the fact that the trace (or game) semantics of a non-uniform η -long Böhm tree is simply the exploration (or parsing) of that tree by the Opponent.

7 Additional structures

For clarity’s sake, we deliver the simplest possible definition of asynchronous game in Section 2. We review below possible extensions of this definition.

Compatibility. One may add an *incompatibility* relation $\#$ between moves, in order to obtain a model of intuitionistic linear logic *with additives*. The relation $\#$ indicates when two moves cannot appear in the same position, and thus cannot appear in the same

play. The coherence axiom ($m_1 \# m_2 \leq m_3 \Rightarrow m_1 \# m_3$) is required for every moves m_1, m_2, m_3 , just as in event structures [27].

Independence. There is a well-established tradition in trace semantics to describe *interference* mechanisms using an independence relation I between events [19]. Similarly, an independence relation between moves may be added to asynchronous games, in order to study interference in imperative programming languages. Take the game model of Idealized Algol presented in [1]. Suppose that an independence relation indicates that the moves `read` and `write(n)` are interfering in the interpretation of the variable type `var`, for every natural number n . In that case, the interference between `read` and `write(n)` induces obstructions (“holes”) to the homotopy relation \sim on the game `var`. Quite interestingly, the asynchronous definition of innocence adapts smoothly, and remains compositional in the presence of interfering moves (that is, it defines a category).

8 Conclusion

The theory of asynchronous games is designed to bridge the gap between mainstream game semantics and concurrency theory. Our preliminary results are extremely encouraging. We establish indeed that the cardinal notion of sequential game semantics: *innocence*, follows from elementary principles of concurrency theory, formulated in asynchronous transition systems. We deduce from this a non-uniform λ -calculus, whose game semantics is expressed as a trace semantics. This provides a concurrency-friendly picture of the λ -calculus, and new diagrammatic foundations for the understanding of its syntax and semantics.

References

1. S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions, 1997.
2. Samson Abramsky. Sequentiality vs. concurrency in games and logic. Report Research Report RR-01-08, Oxford University, Programming Research Group, April 2001.
3. Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Information and Computation*, 163(2):409–470, 2000.
4. Samson Abramsky and Guy McCusker. *Game Semantics*. Springer Verlag, 1999.
5. Samson Abramsky and Paul-André Melliès. Concurrent games and full completeness. In *Logic in Computer Science 99*, pages 431–442, Trento, July 1999. IEEE Computer Society Press.
6. Patrick Baillot, Vincent Danos, Thomas Ehrhard, and Laurent Regnier. Timeless games. In Morgen Nielsen and Wolfgang Thomas, editors, *Proceedings of CSL'97*, number 1414 in Lecture Notes in Computer Science, pages 56–77, Aarhus, 1997. Springer Verlag.
7. Vincent Danos, Hugo Herbelin, and Laurent Regnier. Games semantics and abstract machines. In *Proceedings of the 11th Symposium on Logic in Computer Science*, pages 394–405, New Brunswick, 1996. IEEE Computer Society Press.
8. Thomas Ehrhard. Hypercoherences: a strongly stable model of linear logic. *Mathematical Structures in Computer Science*, 3(4):365–385, 1993.

9. Thomas Ehrhard. A relative definability result for strongly stable functions and some corollaries. *Information and Computation*, 1997.
10. Eric Goubault. Geometry and concurrency: A user's guide. *Mathematical Structures in Computer Science*, 10(4), August 2000.
11. R. Harmer. Games and full abstraction for nondeterministic languages. Phd thesis, University of London, 2000.
12. Martin Hyland. *Game Semantics*. Publications of the Newton Institute. Cambridge University Press, 1997.
13. Martin Hyland and Luke Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 163(2):285–408, December 2000.
14. Martin Hyland and Andrea Schalk. Games on graphs and sequentially realizable functionals. In *Logic in Computer Science 02*, pages 257–264, Copenhagen, July 2002. IEEE Computer Society Press.
15. Dietrich Kuske. Non deterministic automata with concurrency relations and domains. In *Proceedings of the Colloquium on Trees in Algebra and Programming, CAAP'94*, volume 787 of *Lecture Notes in Computer Science*. Springer Verlag, 1994.
16. James Laird. Full abstraction for functional languages with control. In *Logic in Computer Science*, pages 58–67, 1997.
17. Olivier Laurent. Polarized games (extended abstract). In *Proceedings of the seventeenth annual symposium on Logic In Computer Science*, pages 265–274, Copenhagen, July 2002. IEEE Computer Society Press.
18. Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. Technical Report DAIMI PB 78, Aarhus University, 1977.
19. Antoni Mazurkiewicz. *The book of traces*, chapter Introduction to trace theory. World Scientific Publishing, 1995.
20. Paul-André Melliès. Axiomatic rewriting 4: a stability theorem in rewriting theory. In *Logic in Computer Science '98*. IEEE Computer Society Press, July 1998.
21. Paul-André Melliès. Asynchronous games 1: a group-theoretic formulation of uniformity. Prépublication électronique PPS//04/06//n°31 (pp), Equipe Preuves, Programmes et Systèmes, April 2003.
22. Paul-André Melliès. Sequential algorithms and strongly stable functions. Prépublication électronique PPS//03/09//n°23 (pp), Equipe Preuves, Programmes et Systèmes, April 2003. To appear in the special issue "Game Theory Meets Theoretical Computer Science" of *Theoretical Computer Science*.
23. Robin Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4:1–22, 1977.
24. Hanno Nickau. Hereditarily sequential functionals. In A. Nerode and Yu. V. Matiyasevich, editors, *Proceedings of the Symposium on Logical Foundations of Computer Science: Logic at St. Petersburg*, volume 813 of *Lecture Notes in Computer Science*, pages 253–264. Springer Verlag, 1994.
25. E. W. Stark P. Panangaden, V. Shanbhogue. Stability and sequentiality in data flow networks. In A. Nerode and Yu. V. Matiyasevich, editors, *International Conference on Automates, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 253–264. Springer Verlag, 1990.
26. Vaughn Pratt. Modeling concurrency with geometry. In *Proceedings of the eighteenth annual symposium on Principles Of Programming Languages*, pages 311–322. ACM, IEEE Computer Society Press, January 1991.
27. G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, volume 4*. Oxford University Press, 1995.

Appendix: table of figures.

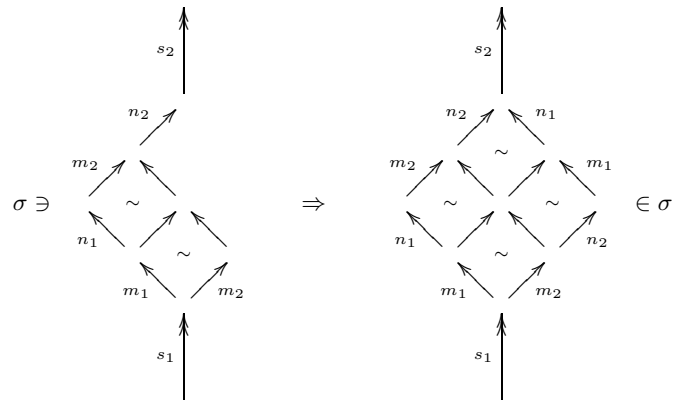


Fig. 1. Side consistency

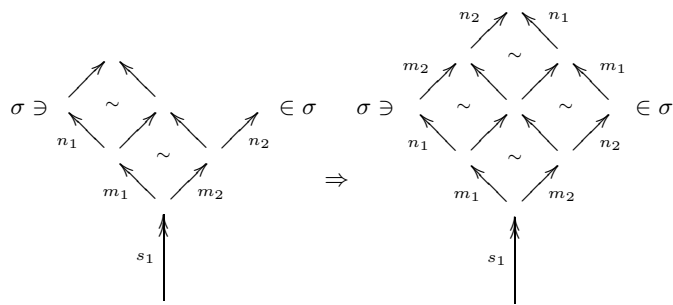


Fig. 2. Forward consistency