Game Semantics and Subtyping

Juliusz Chroboczek*
École Normale Supérieure
Paris, France

Abstract

While Game Semantics has been remarkably successful at modelling, often in a fully abstract manner, a wide range of features of programming languages, there has to date been no attempt at applying it to subtyping. We show how the simple device of explicitly introducing error values in the syntax of the calculus leads to a notion of subtyping for game semantics. We construct an interpretation of a simple λ-calculus with subtyping and show how the range of the interpretation of types is a complete lattice thus yielding an interpretation of bounded quantification.

Introduction

Game Semantics is a framework for the modelling of programming languages which combines the elegant mathematical structure of Denotational Semantics [17] with explicit operational notions such as sequentiality. In the last few years, Game Semantics has provided a fully abstract model of Plotkin's functional calculus PCF [16, 15, 12, 4], and has been used to model a remarkably large range of programming language features, including recursive types [5], imperative variables [6], continuations [13], references [3] and nondeterminism [11]. It is commonly believed that this success is at least in part due to the fact that Game Semantics explicitly considers concepts of an operational nature.

Subtyping is a feature of type systems of programming languages which greatly facilitates the manipulation of heterogeneous data structures and improves the possibilities for reuse of code; as such, subtyping is an essential feature of object-oriented languages. Given types A and B, the statement that A is a subtype of B means that whenever a term M is of type A, it is *ipso facto* of type B.

Subtyping makes possible an interesting generalisation of quantification over types known as *bounded quantification*. While usual polymorphism forces quantifiers to range over all types, bounded quantification allows them to range over a well-defined set of types, typically a (principal) order ideal. Thus, bounded quantification allows the natural expression of limited polymorphism (in the case of universals) and partly abstract datatypes (in the case of existentials).

Notwithstanding its practical importance, subtyping has only rarely been modelled (of particular note are the inclusive subsets model [9] and the P.E.R. model [8]). In this paper, we show how subtyping can be modelled in Game Semantics, in a way that we believe respects the operational intuitions behind the formalism. In order to do so, we start by modelling an untyped calculus; this calculus explicitly incorporates the notion of error, which typing is meant to avoid. We then introduce the notion of *game* which will be used for interpreting types, and define the *liveness ordering* on games, an ordering that will be used for modelling subtyping. Finally, we show that the set of games equipped with the liveness ordering is a complete lattice, which will allow us to interpret bounded quantification.

1. A λ-calculus with errors

Consider a simple λ-calculus with ground values, defined by the following syntax

$$\begin{aligned} M, N, N' ::= {} & x \mid \lambda x.M \mid (M\ N) \\ \mid {} & (M, N) \mid \pi_l(M) \mid \pi_r(M) \\ \mid {} & \mathbf{tt} \mid \mathbf{ff} \mid \mathbf{if}\ M\ \mathbf{then}\ N\ \mathbf{else}\ N'\ \mathbf{fi} \\ \mid {} & \mathbf{top}. \end{aligned}$$

The syntax of the calculus consists of variables, λ-abstractions, function applications, as well as two ground values $\mathbf{tt}$ and $\mathbf{ff}$. We furthermore include a term $\mathbf{top}$ that will be used for representing the result of badly-typed terms.

The simplest way of defining the operational semantics is by using a *one step reduction* relation $\rightsquigarrow$ on terms; this style of presentation is known as the *small-step* style. For our calculus, a common choice — the *call-by-name* semantics — consists of the β-reduction rule

$$((\lambda x.M)\ N) \rightsquigarrow M[x \backslash N]$$

*The research leading to this work was partly supported by an an EU Marie Curie scholarship and an EPSRC student grant.

as well as two symmetric rules for products

$$\pi_l((M,N)) \rightsquigarrow M \qquad \pi_r((M,N)) \rightsquigarrow N$$

and two rules for reduction of the conditional

$$\mathbf{if\ tt\ then}\ N\ \mathbf{else}\ N'\ \mathbf{fi} \rightsquigarrow N,$$
$$\mathbf{if\ ff\ then}\ N\ \mathbf{else}\ N'\ \mathbf{fi} \rightsquigarrow N'.$$

Finally, there is a structural rule that allows reduction within a term under some conditions; these conditions are defined by a set of *evaluation contexts*

$$\begin{aligned} E[\cdot] ::= {} & ([\cdot]\ N) \mid \pi_l([\cdot]) \mid \pi_r([\cdot]) \\ \mid {} & \mathbf{if}\ [\cdot]\ \mathbf{then}\ N\ \mathbf{else}\ N'\ \mathbf{fi} \end{aligned}$$

and we allow reduction to happen in evaluation contexts:

$$\frac{M \rightsquigarrow M'}{E[M] \rightsquigarrow E[M']}$$

In general, we will be interested in computations that take more than one step. The *reduction relation* $\rightsquigarrow^*$ is the transitive reflexive closure of $\rightsquigarrow$.

The notion of *reduction to a value* is defined by specifying a set of *values*, which can be seen as the possible outcomes of satisfactory computations. In our case, the set of values is defined as:

$$V ::= \mathbf{top} \mid \mathbf{tt} \mid \mathbf{ff} \mid \lambda x.M \mid (M,N)$$

We say that a term M *reduces to value* V, written $M \downarrow V$, if $M \rightsquigarrow^* V$ where V is a value. We write $M \downarrow$ when there exists a value V such that $M \downarrow V$, and $M \uparrow$ otherwise.

1.1. Errors in the calculus

The relation $\downarrow$ is not total; a number of terms do not reduce to values. This is expected, as we have done nothing whatsoever to prevent the formation of meaningless terms.

Let δ be the term $\lambda x.(x\ x)$. The term $(\delta\ \delta)$ does not reduce to a value; $(\delta\ \delta)$ leads to an infinite sequence of one-step reductions:

$$(\delta\ \delta) \rightsquigarrow (\delta\ \delta) \rightsquigarrow (\delta\ \delta) \rightsquigarrow \cdots$$

Another example of a term that fails to reduce to a value is

$$M = \mathbf{if}\ \lambda x.x\ \mathbf{then\ tt\ else\ ff\ fi}$$

In this case, small-step semantics shows that the reduction remains "stuck" at a non-value term:

$$\text{there is no } M' \text{ such that } M \rightsquigarrow M'.$$

In our view, this situation corresponds to a runtime error — an exceptional situation detected during the reduction of a term. As our calculus contains no constructs that allow us to handle ("trap") such errors, we shall use the term *untrappable error*.

We will use the term $\mathbf{top}$ to represent untrappable errors. This term is easy to introduce in the small-step presentation: we only need to introduce a new rule

$$M \rightsquigarrow \mathbf{top} \qquad \begin{array}{l}\text{whenever } M \text{ is not a value and} \\ \text{there is no } M' \neq \mathbf{top} \\ \text{such that } M \rightsquigarrow M'.\end{array}$$

The addition of this rule, together with the fact that $\mathbf{top}$ is a value, makes $\downarrow$ into a total relation.

1.2. Errors and denotational semantics

It is not immediately obvious how to model errors in Denotational Semantics. Consider for example the domain of Booleans presented in Fig. 1(a). One choice would be to add an error value **error** "on the side" (Fig. 1(b)); another one would be to add a value **top** as a top element (Fig. 1(c)).

It is our view that errors on the side model errors that may be trapped by a construct in the language, while errors at top model untrappable errors. Consider, indeed, the addition to our calculus of a term **ignore-errors** that would satisfy

$$\textbf{ignore-errors}\ \mathbf{tt} \rightsquigarrow \mathbf{tt}$$
$$\textbf{ignore-errors}\ \mathbf{ff} \rightsquigarrow \mathbf{tt}$$
$$\textbf{ignore-errors}\ \mathbf{top} \rightsquigarrow \mathbf{ff}$$

Denotationally, such a term would have to map $\mathbf{tt}$ to $\mathbf{tt}$ while mapping $\mathbf{top}$ to $\mathbf{ff}$, which would be a non-monotone semantics. On the other hand, modelling an analogous term using $\mathbf{error}$ instead of $\mathbf{top}$ would cause no problem at all.

In a calculus that would include both trappable and untrappable errors, the domain of Booleans would have to have two distinct error values, as in Fig. 1(d). As trappable errors have been modelled before [10], we shall restrict ourselves to a single error value, and adopt the domain of Fig. 1(c).

The addition of a top value to Scott domains was a common feature of early Denotational Semantics [17]. However, this value does not seem to be used for modelling anything, but is just added to domains in order to turn them into complete lattices.

1.3. Observational preorder

In order to complete the definition of the semantics of our calculus, we need to introduce a notion of equivalence of terms. This is usually done by defining a set of *observations*, which is then used to define a congruent preorder on terms known as the *observational preorder*.

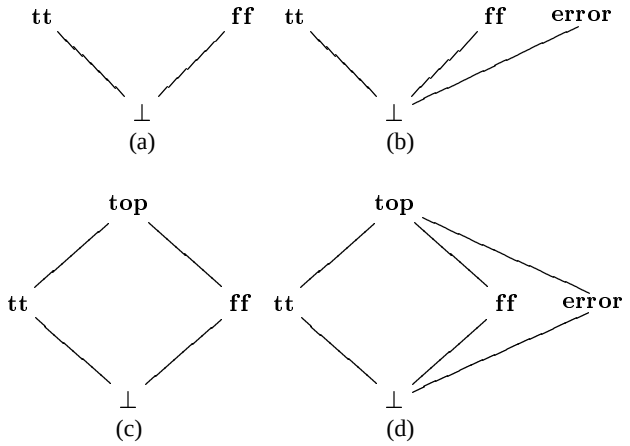

Figure 1. Four domains of Booleans

Of course, we will want the preorder to take into account the difference between looping and reduction to $\mathbf{top}$; in a view to doing so, we need to define the corresponding observation. The definition is somewhat complicated by the fact that we are aiming for a variant of *call-by-name* semantics (a variant of what Abramsky [1] calls the *standard theory* of the λ-calculus), in which $\mathbf{top}$ is undistinguishable from $\lambda x.\mathbf{top}$. In the following definition, we write Ω for the term $(\delta\ \delta) = ((\lambda x.xx)\ (\lambda x.xx))$.

Definition 1 *The set* $\mathbf{te}$ *of* top-equivalent *terms is inductively defined by:*

- $\mathbf{top} \in \mathbf{te}$*;*
- $(M, N) \in \mathbf{te}$ *if* $M \in \mathbf{te}$ *and* $N \in \mathbf{te}$*;*
- $\lambda x.M \in \mathbf{te}$ *if* $M[x\backslash\Omega] \in \mathbf{te}$*;*
- *if* $M \downarrow M'$, $M' \in \mathbf{te}$, *then* $M \in \mathbf{te}$*;*
- *if* $M[x\backslash\Omega] \in \mathbf{te}$ *then* $M \in \mathbf{te}$.

The first four rules in this definition define the set of closed top-equivalent terms; the last one extends the definition to non-closed terms.

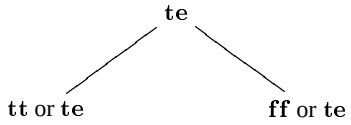

Figure 2. Observations

The set of observations consists of the observations $\mathbf{te}$, $\{\mathbf{tt}\} \cup \mathbf{te}$ and $\{\mathbf{ff}\} \cup \mathbf{te}$, ordered as in Fig. 2 (note the analogy with Fig. 1(c)). The observational preorder is then defined, as usual, to be the least discrete (largest) congruent preorder that makes contexts monotone.

Definition 2 *(Observational preorder)*

$$M \sqsubseteq N \text{ iff}$$
$$\forall C[\cdot] \begin{cases} C[M] \in \mathbf{te} \Rightarrow C[N] \in \mathbf{te} \\ C[M] \downarrow \mathbf{tt} \Rightarrow C[N] \downarrow \mathbf{tt} \text{ or } C[N] \in \mathbf{te} \\ C[M] \downarrow \mathbf{ff} \Rightarrow C[N] \downarrow \mathbf{ff} \text{ or } C[N] \in \mathbf{te} \end{cases}$$

This observational preorder can be defined by just one well-chosen observation. In our case, this is the top observation.

Lemma 3 $M \sqsubseteq N$ *if and only if*

$$\forall C[\cdot]\ C[M] \in \mathbf{te} \Rightarrow C[N] \in \mathbf{te}.$$

Informally, this lemma says that terms are equivalent if and only if they generate errors in the same set of contexts.

2. A game semantics for the untyped calculus

In Game Semantics, a term is represented by the set of its behaviours in all possible contexts. A behaviour is modelled as a play between two players, Player, who represents the term under consideration, and Opponent, who represents its environment (the context it is in). The two players exchange tokens of information known as *moves* — one may think of these as (visible) actions in process calculi, or messages in message-passing object oriented calculi. By convention, Opponent plays first when modelling a call-by-name calculus.

A *position* is an alternating sequence of moves — odd-ordered moves played by Opponent, even-ordered ones by Player. A *strategy* is a set of positions that specify the moves played by Player in response to a given sequence of moves from Opponent.

Moves are structured into *components* which correspond to paths in the syntax tree of a type. For example, a strategy corresponding to a term of type $\mathbf{Bool} \to \mathbf{Bool}$ exchanges moves in components 0 (the left-hand-side of the arrow) and 1. Similarly, a strategy playing at type $\mathbf{Bool} \times \mathbf{Bool}$ will play in components l (for left-hand-side) and r. Strategies are composed using a variant of parallel composition with hiding, with the addition of renaming between components.

The main novelty of the formalism used in this work is that we allow strategies to refuse moves. This is used for modelling untrappable errors.

2.1. Strategies

The set M of *unjustified moves* is inductively defined as follows:

- q, $a^{\mathbf{tt}}$, $a^{\mathbf{ff}}$ are unjustified moves;
- if m is an unjustified move, then so are m_0, m_1, m_l, m_r.

As a shorthand, we will write $m_{e \cdot f \cdot g}$ for $((m_g)_f)_e$ (note the inversion). An unjustified move is therefore of the form t_c, where t is one of q (the *question*), $a^{\mathbf{tt}}$ (the *answer true*) or $a^{\mathbf{ff}}$ (the *answer false*), and c, the *component* of t_c, a finite sequence over $\{0, 1, l, r\}$. An unjustified move m is said to be *in* component c if it is of the form m'_c for some unjustified move m'; in other words, if c is a prefix of the component of m. We write M for the set of unjustified moves.

In order to assign precise rôles to moves, we define a *labelling map*

$$\lambda : M \to \{O, P\} \times \{Q, A\}$$

which determines, for every move, whether it is played by Opponent or Player, and whether it is a question or an answer. The map λ is inductively defined by:

$$\begin{aligned} \lambda(q) &= OQ; \\ \lambda(a^{\mathbf{tt}}) = \lambda(a^{\mathbf{ff}}) &= PA; \\ \lambda(m_1) = \lambda(m_l) = \lambda(m_r) &= \lambda(m); \\ \lambda(m_0) &= \overline{\lambda}(m); \end{aligned}$$

where $\overline{\lambda}$ is derived from λ by inverting the Opponent/Player nature of moves; precisely

$$\begin{aligned} \overline{\lambda}(m) &= PQ \qquad \text{if } \lambda(m) = OQ \\ \overline{\lambda}(m) &= PA \qquad \text{if } \lambda(m) = OA \\ \overline{\lambda}(m) &= OQ \qquad \text{if } \lambda(m) = PQ \\ \overline{\lambda}(m) &= OA \qquad \text{if } \lambda(m) = PA \end{aligned}$$

When we introduce positions later in this section, some moves will be justified by others. In order to determine which moves may be played without justification, and which moves can be justified by which other moves, we define a relation $\vdash$ in $M + M \times M$, the *enabling relation*. Writing $\vdash m$ for $m \in \vdash$, and $m \vdash n$ for $(m, n) \in \vdash$, the relation $\vdash$ is inductively defined by

- $\vdash q$, $q \vdash a^{\mathbf{tt}}$ and $q \vdash a^{\mathbf{ff}}$;
- if $\vdash m$ then $\vdash m_1$, $\vdash m_l$ and $\vdash m_r$;
- if $\vdash m$ and $\vdash n$ then $m_1 \vdash n_0$;
- if $m \vdash n$ then $m_1 \vdash n_1$, $m_0 \vdash n_0$, $m_l \vdash n_l$ and $m_r \vdash n_r$.

We say that a move m is *initial* when $\vdash m$, and that a move m *enables* a move n when $m \vdash n$.

The following properties of $\vdash$ are immediate consequences of its definition:

(e1) if $\vdash m$ then $\lambda(m) = OQ$ and there is no n such that $n \vdash m$;

(e2) if $m \vdash n$ where n is an answer, then m is a question;

(e3) if $m \vdash n$, then either m is Opponent's and n Player's, or m is Player's and n Opponent's.

These properties correspond to McCusker's axioms [14, Section 3.1]. The triple $(M, \lambda, \vdash)$ is therefore a *Computational Arena* in the sense of McCusker; more precisely, it is isomorphic to the Arena that McCusker associates with the type

$$\mu X \,.\, \mathbf{Bool} \,\&\, (X \otimes X) \,\&\, (X \multimap X),$$

or, in an intuitionistic setting (the distinction being irrelevant at the level of Arenas),

$$\mu X \,.\, \mathbf{Bool} \times (X \times X) \times (X \to X).$$

A *justified move*, or simply *move*, is either an unjustified move or a couple (m, i) where m is an unjustified move and i a natural integer, known as the move's *justification*. A *justified sequence*, or *position*, is a sequence $m_1 \cdots m_i$, where every m_k is a move which either has no justification and $\vdash m_k$, or has a justification j such that $j < k$ and $m_j \vdash m_k$. If the move m_k has a justification j, then m_j is said to *justify* m_k. We write ϵ the position of length 0.

There is a convenient pictorial representation for positions, which we use in Fig. 3. In this representation, positions are written in a table, with columns representing components; justification pointers are represented by curves connecting moves. For example, Fig. 3(a) represents the position $q_1 \cdot (a^{\mathbf{tt}}_1, 1)$, *i.e.* the position consisting of an unjustified question in component 1, followed by the answer true in component 1 justified by the initial question.

Terms will be interpreted by *strategies*. A strategy specifies Player's behaviour, and is represented as a set of possible positions which, for any input for Opponent, uniquely specify the output from Player. Thus, an even-length position in a strategy (a position ending in a Player's move) specifies the output from Player to a given input from Opponent. Strategies may also contain odd-length positions, specifying that Player does not play any move in reply to a certain input.

Definition 4 *A set s of positions is*

- prefix-closed *if $p \cdot q \in s$ implies $p \in s$ for any positions p and q;*
- even-prefix-closed *if $p \cdot q \in s$ and $|p|$ even imply $p \in s$ for any positions p and q;*
- deterministic *if for any position $p \in s$, if $|p|$ is odd then, for any moves m and n,*

– $p \cdot m \in s$ *and* $p \cdot n \in s$ *imply* $m = n$*; and*

– $p \cdot m \in s$ *implies* $p \notin s$.

A strategy *is a non-empty even-prefix-closed deterministic set of positions.*

For any collection of positions A, we write $\mathrm{Pref}\, A$ for the prefix completion of A.

The even-prefix-closedness condition in this definition says that a strategy cannot mandate that Opponent play at a given position: a strategy must allow for the situation in which Opponent never plays a move. As to the determinacy condition, it states that a strategy cannot mandate either playing two distinct moves or both playing and not playing a move at a given position.

Of particular note is that we distinguish between a position containing an odd-length position, *e.g.* $\{\epsilon, q\}$ and a strategy not containing it, *e.g.* $\{\epsilon\}$. The former strategy accepts the initial move q from Opponent and then never plays a move, thus modelling a looping term. On the other hand, the latter strategy never accepts an initial move from Opponent: it represents a term that can never be correctly invoked, and thus models the untrappable error $\mathbf{top}$. This distinction is not present in previous frameworks for Game Semantics, where Player can never refuse an Opponent's move, and thus strategies contain all possible odd-length positions [12], (or, equivalently up to isomorphism, only contain even-length positions [14]).

The use of even-prefix-closedness (as opposed to prefix-closedness) is essential for the proper propagation of errors during composition. Intuitively, it says that moves are played in rigid units of two: Player accepts a move conditionally on the next move being accepted by Opponent; in terms of process calculi, strategies implement unbuffered communication.

Taken together, even-prefix-closedness and determinacy imply that an odd-length position in a strategy cannot be extended (*i.e.* if $p \in s$ and $|p|$ is odd, then no $p \cdot q$ is in s): once a strategy has refused to play a move, the play will not proceed further.

A few interesting strategies are sufficiently simple to be defined directly. The strategy Ω (which models looping terms) is defined as the set consisting of ϵ (the empty position) and all positions of length 1 (*i.e.* all positions consisting of a single initial move): this strategy specifies that Player should accept any initial Opponent's move, but never play a move. The strategy $\mathbf{top}$, used for modelling errors, consists of the single position ϵ: no Opponent's move is ever accepted by Player. The strategy $\mathbf{tt}$, which models the true Boolean value, is the set of all even-length positions consisting of the initial question q and the answer $a^{\mathbf{tt}}$; omitting for the sake of clarity the justification pointers,

$$\mathbf{tt} = \{\epsilon, q \cdot a^{\mathbf{tt}}, q \cdot a^{\mathbf{tt}} \cdot q \cdot a^{\mathbf{tt}}, \dots\},$$

with every $a^{\mathbf{tt}}$ move justified by the preceding q. Similarly,

$$\mathbf{ff} = \{\epsilon, q \cdot a^{\mathbf{ff}}, q \cdot a^{\mathbf{ff}} \cdot q \cdot a^{\mathbf{ff}}, \dots\}.$$

An important class of strategies are the so-called *copycat* strategies, which interpret the purely logical combinators. All that a copycat strategy ever does is copy moves between components. The simplest of those is the identity strategy $\mathbf{I}$ plays copycat between the components 1 and 0. Formally, using notions to be introduced in Section 2.2, this strategy can be defined as the set of all even-length positions p entirely composed of moves in components 1 and 0 such that for even-length prefix q of p, $q \upharpoonright 1 = q \upharpoonright 0$. Informally, the behaviour of a Player obeying $\mathbf{I}$ can be described as follows. When Opponent plays an unjustified move m in component 1, Player copies this move to component 0 and justifies it with m. When Opponent plays a move m justified by a previous Player's move n, Player copies this move to the other component justifying it with the move preceding n (*i.e.* the move of which n is a copy).

Similarly, π_l and π_r play copycat between 1 and $0 \cdot l$ and between 1 and $0 \cdot r$ respectively. The Δ strategy simultaneously plays copycat between components $1 \cdot l$ and $1 \cdot r$ and component 0. Finally, of particular interest is the strategy $\mathbf{eval}$, which simultaneously plays copycat between components 1 and $0 \cdot l \cdot 1$ and between components $0 \cdot l \cdot 0$ and $0 \cdot r$ (think of the syntax tree of a type $((A \to B) \times A) \to B$).

Another important strategy is the $\mathbf{ite}$ (for "if-then-else") strategy; while not purely logical, $\mathbf{ite}$ is almost a copycat strategy. The only initial move that $\mathbf{ite}$ accepts is q_1; after receiving this move, it plays an initial move in component $0 \cdot l$; when it receives the answer $a^{\mathbf{tt}}$ in component $0 \cdot l$ (resp. the answer $a^{\mathbf{ff}}$), it starts playing copycat between components $0 \cdot r \cdot l$ and 1 (resp. between components $0 \cdot r \cdot r$ and 1).

2.2. Operations on strategies

More complex strategies may be built from simpler strategies. The essential operations here are *injection*, *currying* and *projection*.

The injection $I_c(s)$ of a strategy s into component c, where c is one of 1, l or r, simply consists in the renaming of moves of all positions in s by prepending c to their components. Of particular interest is injection into component 1, written $K(s)$, which is used for interpreting constant maps.

Currying is a slightly more complex renaming. Given a strategy s, the currying $\Lambda(s)$ is built by considering all the positions in s that are entirely composed of moves in components $0 \cdot l$, $0 \cdot r$ or 1, and renaming them into components, respectively, 0, $1 \cdot 0$ and $1 \cdot 1$ (think of the syntax trees of the types $A \times B \to C$ and $A \to B \to C$).

Projection is a left inverse of injection. Given a position $p = m_1 \cdots m_n$, let c be one of 1, 0, l or r. Consider the sequence of moves $p' = m_{n_1} \cdots m_{n_k}$ of p that are in component c, and rename them by removing the prefix c from all moves. In the process, justification pointers that point at a move in p' are adjusted, while justification pointers that don't are simply removed (thus leading to an unjustified move); the result of this operation is written $p \upharpoonright c$. The projection $s \upharpoonright c$ of a strategy s is the set of all $p \upharpoonright c$ for $p \in s$; in the particular case of c being one of 1, l or r, this yields a strategy.

Given a position $p = m_1 \cdots m_n$, let $N = \{n_1, \ldots n_k\}$ be a subset of $\{1, \ldots n\}$. If for any $n \in N$, m_n either carries no justification or is justified by a move $m_{n'}$ with $n' \in N$, then there exists a unique position

$$q = m_{n_1} \cdots m_{n_k}$$

consisting of exactly the moves in p indexed by N with the justification pointers adjusted. We will call q the *subposition* of p associated to N.

Given positions q and q', a position p is said to be an *interleaving* of q and q' when there exists an exhaustive partition N, N' of $\{1, \ldots |q|\}$ such that q, resp. q', is the subposition of p associated to N, resp. to N'.

Given strategies s and t, the pairing (s, t) consists of all positions p such that p is an interleaving of a position $q \in I_l(s)$ and a position $q' \in I_r(t)$. In order to define the *tensor product* $s \otimes t$, consider all positions in (s, t) that are entirely within components $l \cdot 1$, $l \cdot 0$, $r \cdot 0$, $r \cdot 1$, (*i.e.* all positions that are at a type $(A \to B) \times (C \to D)$), and rename them as follows:

$$\begin{aligned} l \cdot 1 &\mapsto 1 \cdot l, \\ l \cdot 0 &\mapsto 0 \cdot l, \\ r \cdot 1 &\mapsto 1 \cdot r, \text{ and} \\ r \cdot 0 &\mapsto 0 \cdot r. \end{aligned}$$

Finally, composing this with the strategy Δ, we obtain the functional pairing $\langle s, t \rangle = \Delta; (s \otimes t)$ (composition is defined below).

Strategies are composed using a variant of parallel composition with hiding. Given two strategies s and t, in order to compute $s; t$ we compose them by letting component 1 of s interact with component 0 of t; the common component is then hidden. Consider for example the constant strategy $K(\mathbf{tt})$, a position of which is depicted in Fig. 3(a), and the identity strategy, a position of which is in Fig. 3(b). The two positions *agree* in that the projection of the first into component 1 is equal to the projection of the second into component 0. After hiding the common component, we obtain the position in Fig. 3(c), which is a position in $K(\mathbf{tt}); \mathbf{I} = K(\mathbf{tt})$.

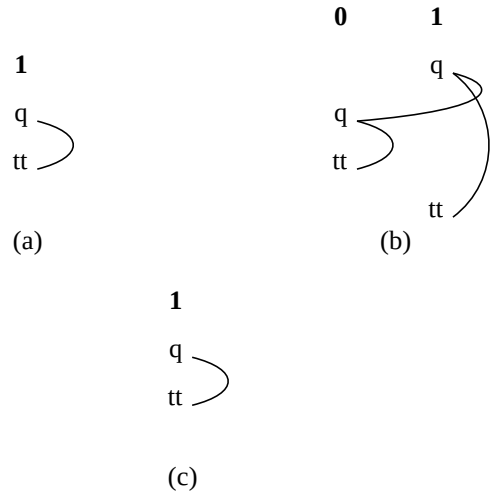

Figure 3. Composition of two positions

Roughly speaking, composition of strategies s and t is performed by ranging over all behaviours in s and t, selecting those that are compatible, and composing them. Thus, we obtain a total operation of composition (as needed for modelling an untyped calculus), but one that maps incompatible strategies to $\top$. For example, if s is $\mathbf{tt}$ (*not* $K(\mathbf{tt})$), and t is $\mathbf{ff}$, the only compatible positions in s and t is ϵ; thus, $t; s = \epsilon$. This example is somewhat artificial, in that it does not occur in the actual interpretation (see below); however, the case of

$$[\![\vdash (\mathbf{tt}\ \mathbf{ff})]\!] = \langle K(\mathbf{tt}), K(\mathbf{ff}) \rangle; \mathbf{eval}$$

is quite analogous.

To chose a more interesting example, let $s = \mathbf{top}$ and $t = \mathbf{I}$. As $\mathbf{I}$ only contains even-length positions, and every position in $\mathbf{I}$ has moves in component 0, the only compatible positions in s and t are ϵ. Thus, $\mathbf{top}; \mathbf{I} = \mathbf{top}$, and not Ω as would be the case if we chose a notion of composition that implements buffered communication.

The formalisation is somewhat complicated, however, by the fact that we need to take into account *livelock*, or *infinite chattering*, the situation in which two strategies never disagree but never have positions that coincide. Indeed, suppose that when composing s with t, after the initial move is played in component 1 of t, both t and s keep playing in the common component. In this case, the two strategies would never ultimately reach agreement, and yet neither would ever play a move that is not accepted by the other.

Definition 5 *Given a natural integer n, we say that two position p and q* agree at depth n *if p and q only contain moves within components 1 and 0, and the prefix of length n of $p \upharpoonright 1$ is equal to the prefix of same length of $p \upharpoonright 0$ (or*

$p \restriction 1 = q \restriction 0$ *if both projections are of length smaller than* n*).*

Given two strategies s *and* t*, the strategy* $s;t$ *is the set of all postions* p *such that for any natural integer* n*, there exist postions* $q \in s$ *and* $q' \in t$ *such that* q *and* q' *agree at depth* n*,* $q \restriction 0 = p$ *and* $q' \restriction 1 = t$*.*

Note how this operation eliminates all disagreeing positions, thus yielding errors for strategies that disagree on all positions.

Composition yields strategies and is an associative operation. As seen above, composition does not have a neutral element (left- or right-composing $\mathbf{tt}$ with any other strategy yields $\mathbf{top}$). However, if we restrict ourselves to the set of strategies that only contain moves in components 1 and 0 (and these are the only strategies that we will use in our interpretation), we do obtain a unary monoid with the strategy $\mathbf{I}$ as the neutral element.

2.3. Interpretation of the calculus

Untyped terms are not interpreted in isolation. We interpret a couple $\Gamma \vdash M$, where Γ is an (ordered) list of variables, and M a term such that $\mathrm{FV}(M) \subseteq \Gamma$. The interpretation is defined as follows.

$$\begin{aligned}
[\![x \vdash x]\!] &= \mathbf{I} \\
[\![\Gamma, x \vdash x]\!] &= \pi_r \\
[\![\Gamma, y \vdash x]\!] &= \pi_l; [\![\Gamma \vdash x]\!] \\
[\![\Gamma \vdash \lambda x.M]\!] &= \Lambda([\![\Gamma, x \vdash M]\!]) \\
[\![\Gamma \vdash (M\ N)]\!] &= \langle [\![\Gamma \vdash M]\!], [\![\Gamma \vdash N]\!] \rangle; \mathbf{eval} \\
[\![\Gamma \vdash (M, N)]\!] &= \langle [\![\Gamma \vdash M]\!], [\![\Gamma \vdash N]\!] \rangle \\
[\![\Gamma \vdash \pi_l(M)]\!] &= [\![\Gamma \vdash M]\!]; \pi_l \\
[\![\Gamma \vdash \pi_r(M)]\!] &= [\![\Gamma \vdash M]\!]; \pi_r \\
[\![\Gamma \vdash \mathbf{tt}]\!] &= K(\mathbf{tt}) \\
[\![\Gamma \vdash \mathbf{ff}]\!] &= K(\mathbf{ff}) \\
[\![\Gamma \vdash \mathbf{if}\ M\ \mathbf{then}\ N\ \mathbf{else}\ N'\ \mathbf{fi}]\!] &= \\
&= \langle [\![\Gamma \vdash M]\!], \langle [\![\Gamma \vdash N]\!], [\![\Gamma \vdash N']\!] \rangle \rangle; \mathbf{ite} \\
[\![\Gamma \vdash \mathbf{top}]\!] &= \mathbf{top}
\end{aligned}$$

Clearly, $[\![\Gamma \vdash M]\!]$ is defined as soon as Γ contains all the free variables of M.

Theorem 6 *The interpretation* $[\![\cdot]\!]$ *is sound,* i.e. *if* $[\![\Gamma \vdash M]\!]$ *is defined and* $M \rightsquigarrow N$ *then* $[\![\Gamma \vdash N]\!]$ *is defined and*

$$[\![\Gamma \vdash N]\!] = [\![\Gamma \vdash M]\!].$$

The proof of this soundness theorem consists of two parts. First, we prove (by cases on the structure of M) that whenever M reduces to $\mathbf{top}$ by using the "stuck computation" rule, $[\![\Gamma \vdash M]\!] = \mathbf{top}$. We then prove similarly soundness of all the single-step rules that don't involve errors.

While the interpretation of the untyped calculus is sound, it is not computationally adequate, and in fact not even inequationally sound. Indeed, let the identity on the booleans $\mathbf{I_{Bool}}$ be the term

$$\mathbf{I_{Bool}} = \lambda x.\mathbf{if}\ x\ \mathbf{then}\ x\ \mathbf{else}\ x\ \mathbf{fi}.$$

The associated strategy replies to a move with component 1 or 0 by copying it over to the other component, but refuses all moves with components other than 1 and 0. Let $\mathbf{Y}$ be the usual fixpoint combinator (the term $\lambda x.(\lambda y.x(yy))(\lambda y.x(yy))$; then

$$(\mathbf{Y}\ \mathbf{I_{Bool}}) \uparrow$$

while

$$[\![\vdash (\mathbf{Y}\ \mathbf{I_{Bool}})]\!] = K(\{\epsilon, q\}).$$

We will come back to this issue in Section 6.

3. Type assignment and subtyping

In order to define a type assignment on our calculus, we assume the existence of a countable set of type variables $X, Y, \ldots$ and define the syntax of types as follows.

$$\begin{aligned}
A, B ::= {} & \mathbf{Bool} \mid \top \mid X \mid A \times B \mid A \to B \\
& \mid \quad \forall X \leq A.B[X] \mid \exists X \leq A.B[X]
\end{aligned}$$

Environments specify bounds on type variables, and types of variables; an environment is a map from type variables and variables to types. We use the letter E to range over environments, and write

$$X \leq A, Y \leq B, x : C, y : D$$

for the environment that maps X to A, Y to B, x to C, y to D and all other type variables and variables to $\top$.

We use two types of *judgements*. A *subtyping judgement* is of the form

$$E \vdash A \leq B$$

and specifies that in the environment E, the type A is a subtype of the type B; we write $E \vdash A = B$ for $E \vdash A \leq B$ and $E \vdash B \leq A$. A *typing judgement* is of the form

$$E \vdash M : A$$

and states that in the environment E, the term M has type A. Typing is not functional: a term may in general have multiple distinct types.

The set of inference rules used for typing is given in Figures 4 through 6. Somewhat unusual is the absence of rules for the introduction and elimination of quantifiers, with the exception of $\forall$-introduction; in fact, these rules are not useful in our framework as they may be derived from the subtyping rules for quantifiers and subsumption (the last rule in Fig. 4). Thus, the $\forall$-elimination rule

$$\frac{E \vdash M : \forall X \leq A.B[X] \qquad E \vdash C \leq A}{E \vdash M : B[C]}$$

may be obtained from an application of the seventh rule in Fig. 5 and subsumption. Similarly, $\exists$-introduction

$$\frac{E \vdash M : B[C] \qquad E \vdash C \leq A}{E \vdash M : \exists X \leq A.B[X]}$$

may be handled by an application of the subtyping rule for existentials and subsumption. The case of $\exists$-elimination,

$$\frac{E, X \leq A, x : B[X] \vdash M : D \quad E \vdash N : \exists X \leq A.B[X]}{E \vdash (\lambda x.M)N : D}$$
(X not free in D)

is slightly more complex, as it requires successive application of $\forall$-introduction and $\forall$-dualisation (the first rule of Fig. 6):

$$\dfrac{\dfrac{\dfrac{\dfrac{E, X \leq A, x : B[X] \vdash M : D}{E, X \leq A \vdash \lambda x.M : B[X] \to D}}{E \vdash \lambda x.M : \forall X \leq A.B[X] \to D}}{E \vdash \lambda x.M : (\exists X \leq A.B[X]) \to D} \qquad E \vdash N : \exists X \leq A.B[X]}{E \vdash (\lambda x.M)N : D}$$

4. The liveness ordering and games

In this section, we introduce a preorder $\preccurlyeq$ — the *liveness ordering* — on arbitrary collections of positions; the liveness ordering will turn out to be a partial order on the particular classes of collections of positions that we are interested in. Up to the interpretation, $\preccurlyeq$ will be a refinement of the relations induced by typing and subtyping: whenever $\vdash M : A$, and $\vdash A \leq B$, we will have $[\![M]\!] \preccurlyeq [\![A]\!] \preccurlyeq [\![B]\!]$.

Types will be interpreted as games. A game is a set of positions that provide a specification that a strategy may or may not satisfy. The specifications expressed are strictly safety specifications, *i.e.* if a given strategy belongs to a game, then so does any strategy below it.

A game A provides not only a specification for Player but also a specification for Opponent. A strategy s belongs to the game A if its behaviour satisfies the constraints expressed by A, but only as long as Opponent behaves according to A; Player's behaviour is otherwise unrestricted

$$E, x : A \vdash x : A$$

$$\frac{E, x : A \vdash M : B}{E \vdash \lambda x.M : A \to B}$$

$$\frac{E \vdash M : A \to B \quad E \vdash N : A}{E \vdash (M\ N) : B}$$

$$E \vdash M : \top$$

$$E \vdash \mathbf{tt} : \mathbf{Bool}$$

$$E \vdash \mathbf{ff} : \mathbf{Bool}$$

$$\frac{E \vdash M : \mathbf{Bool} \quad E \vdash N : A \quad E \vdash P : A}{E \vdash \mathbf{if}\ M\ \mathbf{then}\ N\ \mathbf{else}\ P\ \mathbf{fi} : A}$$

$$\frac{E \vdash M : A \qquad E \vdash N : B}{E \vdash (M, N) : A \times B}$$

$$\frac{E \vdash M : A \times B}{E \vdash \pi_l(M) : A} \qquad \frac{E \vdash M : A \times B}{E \vdash \pi_r(M) : B}$$

$$\frac{E, X \leq A \vdash M : B[X]}{E \vdash M : \forall X \leq A.B[X]}\ X \text{ does not appear in } E$$

$$\frac{E \vdash M : A \quad E \vdash A \leq B}{E \vdash M : B}$$

Figure 4. Typing rules

$$E \vdash A \leq A$$

$$\frac{E \vdash A \leq B \quad E \vdash B \leq C}{E \vdash A \leq C}$$

$$E \vdash A \leq \top$$

$$E \vdash \top \leq A \to \top$$

$$\frac{E \vdash A' \leq A \qquad E \vdash B \leq B'}{E \vdash A \to B \leq A' \to B'}$$

$$\frac{E \vdash A \leq A' \qquad E \vdash B \leq B'}{E \vdash A \times B \leq A' \times B'}$$

$$\frac{E \vdash C \leq A}{E \vdash \forall X \leq A.B[X] \leq B[C]}$$

$$\frac{E \vdash C \leq A}{E \vdash B[C] \leq \exists X \leq A.B[X]}$$

$$\frac{E, X \leq A \vdash B[X] \leq B'[X]}{E \vdash \forall X \leq A.B[X] \leq \forall X \leq A.B'[X]}$$

$$\frac{E, X \leq A \vdash B[X] \leq B'[X]}{E \vdash \exists X \leq A.B[X] \leq \exists X \leq A.B'[X]}$$

$$\frac{E \vdash A \leq A'}{E \vdash \forall X \leq A'.B[X] \leq \forall X \leq A.B[X]}$$

$$\frac{E \vdash A \leq A'}{E \vdash \exists X \leq A.B[X] \leq \exists X \leq A'.B[X]}$$

Figure 5. Subtyping rules

$$E \vdash \forall X \leq A.(B[X] \to C) = (\exists X \leq A.B[X]) \to C \quad (X \text{ not free in } C)$$

$$E \vdash \exists X \leq A.(B[X] \to C) = (\forall X \leq A.B[X]) \to C \quad (X \text{ not free in } C)$$

Figure 6. Subtyping rules: dualisation of quantifiers

(this is sometimes called the *assume-guarantee paradigm*). Technically, this will be expressed by the reachability condition in the definition of the liveness ordering (Section 4.1), which is inspired by what Abramsky calls the "back-and-forth inclusion relation" [2].

Definition 7 *A* game *is a non-empty prefix-closed set of positions.*

We write $\mathcal{G}$ for the set of games.

4.1. The liveness ordering

The definition of the *liveness ordering* $\preccurlyeq$ follows pretty much the definition of the observational preorder. Just like for terms M and N we have $M \sqsubseteq N$ when M produces errors in less contexts than N, and therefore N produces results in more contexts than M (Definition 2), we will want strategies s and t to satisfy $s \preccurlyeq t$ if and only if s accepts more positions and produces less positions than t when playing against any given opponent. We define $\preccurlyeq$ on prefix-closed sets of positions, and deduce a suitable definition for strategies from that.

For any non-empty position p, we write p_{-1} for the prefix of p of length $|p| - 1$ (*i.e.* p without its last move).

Definition 8 *Given non-empty prefix-closed sets of positions A and B, we say that B is* more live *than A, or A is* safer *than B, and write $A \preccurlyeq B$, if*

- *for every position of odd length $q \in B$, if $q_{-1} \in A$ then $q \in A$; and*
- *for every position of even length $p \in A$ ($p \neq \epsilon$), if $p_{-1} \in B$, then $p \in B$.*

The definition of $\preccurlyeq$ may be paraphrased as follows. Given a prefix-closed collection of positions A, a position p is said to be *reachable* at A if $p_{-1} \in A$ (or $p = \epsilon$). In order to have $A \preccurlyeq B$, the set of odd-length positions (positions ending in an Opponent's move) in A that are reachable at A needs to be a superset of the set of odd-length positions in B; and, dually, the set of even-length positions in B that are reachable at B should be a superset of the even-length positions in A.

The intution here is that A is a specification that is stricter that B. Unrolling the induction implicit in the definition, we first require that all positions of length 1 present in B be present in A; in other words, any initial Opponent's move that is specified to be accepted by B is also specified to be accepted by A. Moving on to positions of length 2, whenever Opponent played a move m, any Player's move n that A allows must also be allowed by B — but only if a Player obeying B might actually have an opportunity to play n, *i.e.*

if B allows playing the initial move m. Similarly, for positions of length 3, we require that any move required to be accepted by B is also required to be accepted by A, restricting this condition to Opponent's moves that might occur when playing against a Player obeying A.

To clarify this, consider the strategy $\mathbf{tt}$. Its prefix closure contains all sequences of the form

$$\begin{array}{l}\epsilon\\ q\\ q \cdot a^{\mathbf{tt}}\\ q \cdot a^{\mathbf{tt}} \cdot q\\ q \cdot a^{\mathbf{tt}} \cdot q \cdot a^{\mathbf{tt}}\\ \cdots\end{array}$$

As to the game of Booleans $\mathbf{Bool}$, it consists of all positions composed of the initial question q and the answers $\mathbf{tt}$ and $\mathbf{ff}$; $\mathbf{Bool}$ therefore consists of all positions of the form

$$\begin{array}{ll} q \cdot \mathbf{tt} \cdot q \cdot \mathbf{tt} \cdot q \cdots & q \cdot \mathbf{ff} \cdot q \cdot \mathbf{ff} \cdot q \cdots \\ q \cdot \mathbf{tt} \cdot q \cdot \mathbf{tt} \cdot q \cdot \mathbf{tt} \cdots & q \cdot \mathbf{ff} \cdot q \cdot \mathbf{ff} \cdot q \cdot \mathbf{ff} \cdots \end{array}$$

but also of arbitrary interleavings of such positions

$$q \cdot \mathbf{tt} \cdot q \cdot \mathbf{ff} \cdot q \cdots$$

The latter class of positions do not appear in the interpretation of any term; however, they would appear in an analogous interpretation of a calculus with imperative features (in which some strategies would not be *innocent* [6]).

Clearly, $\mathrm{Pref}\,\mathbf{tt} \subseteq \mathbf{Bool}$; therefore,

- all even-length positions in $\mathrm{Pref}\,\mathbf{tt}$ are in $\mathbf{Bool}$ (*i.e.* $\mathbf{tt}$ never plays a move that is not allowed in $\mathbf{Bool}$);
- the only odd-length positions in $\mathbf{Bool}$ that are reachable at $\mathrm{Pref}\,\mathbf{tt}$ are of the form

  $$q \cdot a^{\mathbf{tt}} \cdot q \cdots q$$

  and are therefore in $\mathrm{Pref}\,\mathbf{tt}$ (*i.e.* $\mathbf{tt}$ accepts all moves that are allowed by $\mathbf{Bool}$).

Thus, $\mathrm{Pref}\,\mathbf{tt} \preccurlyeq \mathbf{Bool}$.

Consider now the game $\top = \{\epsilon\}$ (we use $\mathbf{top}$ for the term and the for strategy, and $\top$ for the type and for the game). Clearly, all odd-length positions in $\top$ are in $\mathrm{Pref}\,\mathbf{tt}$; what is more, no non-empty even-length position is reachable at $\top$. Thus, it is vacuously true that $\mathrm{Pref}\,\mathbf{tt} \preccurlyeq \top$.

On the contrary, consider the game $\mathbf{Bool} \to \mathbf{Bool}$, consisting of all the interleavings of injections of positions in $\mathbf{Bool}$ into components 1 and 0. The prefix closure of the strategy $\mathbf{tt}$ contains the position q, which is (vacuously) reachable at $\mathbf{Bool} \to \mathbf{Bool}$, but is not in $\mathbf{Bool} \to \mathbf{Bool}$. Thus, $\mathrm{Pref}\,\mathbf{tt} \not\preccurlyeq \mathbf{Bool} \to \mathbf{Bool}$.

Theorem 9 *The relation $\preccurlyeq$ is a partial ordering on non-empty prefix-closed collections of positions.*

Indeed, $\preccurlyeq$ is clearly reflexive; as to transitivity and antireflexivity, they are both consequences of the following lemma.

Lemma 10 *Let A, B and C be prefix-closed collections of positions such that $A \preccurlyeq B \preccurlyeq C$, and let p be a position that is in both A and C; then $p \in B$.*

This lemma is not in general true for collections of positions that are not prefix-closed, and indeed the definition of $\preccurlyeq$ above does not yield a transitive or antireflexive relation on arbitrary sets of positions. We may, however, extend $\preccurlyeq$ to all non-empty sets of positions by writing $A \preccurlyeq B$ whenever $\mathrm{Pref}(A) \preccurlyeq \mathrm{Pref}(B)$; while this only makes $\preccurlyeq$ into a preorder on arbitrary sets of positions, it does actually make it into a partial order on strategies.

Lemma 11 *If s and t are strategies, then $\mathrm{Pref}(s) = \mathrm{Pref}(t)$ implies $s = t$. Therefore, $\preccurlyeq$ is a partial order on strategies.*

This property depends on the fact that we have restricted ourselves to deterministic strategies.

4.2. The complete lattice of games

We now show that $(\mathcal{G}, \preccurlyeq)$ is a complete lattice by explicitly constructing suprema and infima of arbitrary families of games. Doing this will lead to a sound interpretation of bounded quantification.

Definition 12 *Given a set of games $\mathcal{A} \subseteq \mathcal{G}$, define $\bigwedge \mathcal{A}$ to be the unique game such that:*

- *a position q of odd length is in $\bigwedge \mathcal{A}$ if and only if q_{-1} is in $\bigwedge \mathcal{A}$ and for some $A \in \mathcal{A}$, $q \in A$;*
- *a position p of even length is in $\bigwedge \mathcal{A}$ if and only if p_{-1} is in $\bigwedge \mathcal{A}$ (or $p = \epsilon$) and for all $A \in \mathcal{A}$ such that $p_{-1} \in A$, $p \in A$.*

Define $\bigvee \mathcal{A}$ dually:

- *a position q of odd length is in $\bigvee \mathcal{A}$ if and only if q_{-1} is in $\bigvee \mathcal{A}$ and for all $A \in \mathcal{A}$ such that $q_{-1} \in A$, $q \in A$;*
- *a position p of even length is in $\bigvee \mathcal{A}$ if and only if p_{-1} is in $\bigvee \mathcal{A}$ (or $p = \epsilon$) and for some $A \in \mathcal{A}$, $p \in A$.*

Again, these definitions are inductive, as they define the set of positions of length n as a function of the set of positions of length $n-1$. If games are "relations extended in time," then $\bigwedge$ and $\bigvee$ are intersection and union extended in time. More precisely, $\bigwedge$ takes the union of odd-numbered

(Opponent's) moves and the intersection of even-numbered (Player's) ones ($\bigvee$ does the opposite), but in doing this only considers moves that are reachable.

The proof that these definitions do in fact yield suprema and infima of families of games is a fairly straightforward verification of the following two facts (and their duals):

- if $A \in \mathcal{A}$, then $\bigwedge \mathcal{A} \preccurlyeq A$; and
- if $\bigwedge \mathcal{A} \preccurlyeq B \preccurlyeq A$ for all $A \in \mathcal{A}$, then $B = \bigwedge \mathcal{A}$.

The use of $\bigwedge$ and $\bigvee$ for interpreting quantifiers will not quite lead to a uniform interpretation. Indeed, let $A = \bigwedge_{X \in \mathcal{G}} X \to X$. If we consider the set of strategies dominated by A, we find all strategies that are dominated by the identity; for example, the strategy that behaves like the identity when applied to a ground value, but loops when applied to a couple or function, *i.e.* the strategy that copies q_1 over to component 0, but doesn't reply to moves $q_{1\cdot 1}$, $q_{1\cdot l}$ or $q_{1\cdot r}$; this strategy is clearly not the interpretation of any term in the calculus. This phenomenon is unavoidable in an approach that only considers safety properties of strategies.

5. Interpreting types

The games that will be used for modelling the ground types $\top$ and $\mathbf{Bool}$ have already been introduced in the examples of the previous section; *pro memoria*, $\top = \{\epsilon\}$ is the maximal element of the lattice of games, while $\mathbf{Bool}$ consists of all interleavings of positions in $\mathrm{Pref}\,\mathbf{tt}$ and $\mathrm{Pref}\,\mathbf{ff}$.

Given games A and B, the game $A \times B$ consists of the set of the injections of all positions in A in the component l, the injections of all positions in B in the component r, and all interleavings of such positions.

Finally, the game $A \to B$ consists of all positions p entirely within components 0 and 1 such that $p \upharpoonright 0$ is an interleaving of positions in A and $p \upharpoonright 1$ is an interleaving of positions in B.

5.1. Interpretation of types

In order to interpret types, we need to give values to free type variables. A *type environment* is a map from type variables to games; we range over type environments with the Greek letter η. We write $\eta[X \backslash A]$ for the type environment that is equal to η except at X, which it maps to A, and interpret types as maps from type environments to types as follows.

$$\begin{aligned}
[\![\mathbf{Bool}]\!]\eta &= \mathbf{Bool} \\
[\![\top]\!]\eta &= \top \\
[\![X]\!]\eta &= \eta(X) \\
[\![A \times B]\!]\eta &= [\![A]\!]\eta \times [\![B]\!]\eta \\
[\![A \to B]\!]\eta &= [\![A]\!]\eta \to [\![B]\!]\eta \\
[\![\forall X \leq A.B[X]]\!]\eta &= \bigwedge_{\mathcal{X} \leq [\![A]\!]\eta} [\![B[X]]\!](\eta[X \backslash \mathcal{X}]) \\
[\![\exists X \leq A.B[X]]\!]\eta &= \bigvee_{\mathcal{X} \leq [\![A]\!]\eta} [\![B[X]]\!](\eta[X \backslash \mathcal{X}])
\end{aligned}$$

5.2. Soundness of typing

In order to establish the soundness of typing we need to relate environments to type environments. A type environment η is said to *satisfy* the environment E if whenever a bound $X \leq A$ appears in E, $\eta(X) \preccurlyeq [\![A]\!]\eta$. The following lemma expresses the soundness of subtyping and is proved by induction on the derivation of $E \vdash A \leq A'$.

Lemma 13 *Let E be an environment, and A and A' types such that $E \vdash A \leq A'$. Let η be a typing environment that satisfies E; then*

$$[\![A]\!]\eta \preccurlyeq [\![A']\!]\eta.$$

Expressing the soundness of typing is slightly more involved, as we need to consider not only free type variables but also free variables.

Theorem 14 *Let E be a typing environment, M a term and A a type such that $E \vdash M : A$. Suppose that $E = X_1 \leq B_1, \ldots X_n \leq B_n, x_1 : C_1, \ldots x_m : C_m$, and let $\Gamma = x_1, \ldots x_m$, let η be a typing environment that satisfies E, and C be the type $C = (\cdots (C_1 \times C_2) \times \cdots C_m)$. Then*

$$[\![\Gamma \vdash M]\!] \preccurlyeq [\![C]\!]\eta \to [\![A]\!]\eta.$$

This theorem is proved by induction on the derivation of $\Gamma \vdash M$; most cases in the proof use the following lemma.

Lemma 15 *Let s, t be strategies and A, B, C games such that $s \preccurlyeq A \to B$ and $t \preccurlyeq B \to C$; then $s; t \preccurlyeq A \to C$.*

The difficult case in its proof is that of an odd-length position in $p \in A \to C$; the existence of a position $p \cdot q$ in $s; t$ is proved by induction on the integer n in Definition 5.

The usual statement of the *safety* of typing — that "well-typed terms cannot go wrong" — translates in our setting into the statement that terms that have a non-trivial type do not generate untrappable errors.

Corollary 16 *If $\vdash M : A$, where M is a closed term and A a closed type such that $[\![A]\!]\varnothing \neq \top$, then $M \not\downarrow \top$.*

6. Conclusions and further work

In this work, we have shown how it is possible to construct a Game-semantic interpretation of subtyping with fairly modest technical means. The main technical devices are the introduction of explicit errors in an untyped calculus, and a suitably chosen ordering on prefix-closed collections of positions. This construction automatically yields an interpretation of bounded quantification.

Our construction is generic, in that it does not depend on the details of the calculus being used, and we believe that it could be adapted without trouble to more complex deterministic calculi for which Game models have been developed [6, 13]. Whether or not such extensions will yield any new types of interest, either from a logical or static analysis standpoint, remains to be seen.

The most obvious shortcoming of this work is the failure of computational adequacy and of inequational soundness. The author believes that this is not an intrinsic flaw of our semantic constructions, but rather reflects a shortcoming of the operational semantics chosen. It is notable that our semantics has the simplicity of a call-by-name (as opposed to lazy [1]) construction, and identifies the terms $\mathbf{top}$ and $\lambda x.\mathbf{top}$, but distinguishes Ω from $\lambda x.\Omega$, in a way that would appear to faithfully reflect the semantics of a number of programming languages; this fact comforts us in our trust in the semantics. Clearly, the mismatch is due to the fact that the notion of strategy distinguishes between evaluation in a ground context from evaluation at an arrow or product type (this information is contained in the component of the initial move), while the operational semantics does not. The author is currently investigating a different operational semantics which, by decorating reduction relations with component information, explicitly distinguishes reduction in ground contexts from reduction at more complex types, and appears to be adequately modelled by our construction.

In this work, we do not present a construction for modelling recursive types, which are useful (although not strictly necessary) for interpreting object-oriented languages and calculi. While it is possible to construct "recursive" games by applying Banach's fixpoint theorem to a suitable metric over games, and these games can be shown to validate the subtyping rule proposed by Amadio and Cardelli [7], attempts to interpret both recursive types and quantification in a single construction lead to certain difficulties caused by the subtle interaction between the order-theoretic and metric structures.

Acknowledgements

I would like to thank Samson Abramsky for his support and advice.

References

[1] S. Abramsky. The lazy Lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*. Addison Wesley, 1990.

[2] S. Abramsky. Semantics of interaction. In P. Dybjer and A. M. Pitts, editors, *Semantics and Logics of Computation*. Cambridge University Press, 1997. Based on lectures given at the CLICS-II Summer School on Semantics and Logics of Computation, Isaac Newton Institute for Mathematical Sciences, Cambridge, U.K., September 1995.

[3] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *Proceedings of the thirteenth annual IEEE symposium on Logic in Computer Science*, pages 334–344, Indianapolis, Indiana, June 1998.

[4] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF (extended abstract). In *Proc. TACS'94*, volume 789 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 1994.

[5] S. Abramsky and G. McCusker. Games for recursive types. In *Proceedings of the Second Workshop of the Theory and Formal Methods Section*. Department of Computing, Imperial College, 1994.

[6] S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions (extended abstract). In *Proceedings of the 1996 Workshop on Linear Logic*, volume 3 of *Electronic notes in Theoretical Computer Science*. Elsevier, 1996.

[7] R. M. Amadio and L. Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, 1993. Also in Proc. POPL'91.

[8] K. B. Bruce and G. Longo. A modest model of records, inheritance and bounded quantification. *Information and Computation*, 87:196–240, 1990.

[9] L. Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76(2/3):138–164, 1988.

[10] R. Cartwright, P.-L. Curien, and M. Felleisen. Fully abstract models of observably sequential languages. *Information and Computation*, 111(2):297–401, 1994.

[11] R. Harmer. *Games and Full Abstraction for Nondeterministic Languages*. PhD thesis, Imperial College, University of London, 1999.

[12] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. 1994.

[13] J. Laird. Full abstraction for functional languages with control. In *Proc. LICS'97*, Warsaw, Poland, 1997.

[14] G. McCusker. *Games and Full Abstraction for a Functional Metalanguage with Recursive Types*. PhD thesis, Imperial College, University of London, 1996.

[15] R. Milner. Fully abstract models of typed lambda calculi. *Theoretical Computer Science*, 4(1), 1977.

[16] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, December 1977.

[17] J. E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. The MIT Press, 1977.