Subtyping Recursive Games

Juliusz Chroboczek

Université de Paris VII
Paris, France

Abstract. Using methods drawn from Game Semantics, we build a sound and computationally adequate model of a simple calculus that includes both subtyping and recursive types. Our model solves recursive type equations up to equality, and is shown to validate a subtyping rule for recursive types proposed by Amadio and Cardelli.

Introduction

Subtyping is an ordering relation over types that is an essential feature of a wide range of programming languages. While at first order subtyping corresponds to inclusion of the carriers, there is no simple set-theoretic interpretation of subtyping at higher order.

Many programming languages also include recursive types — types that are defined implicitly, as fixpoints of maps over types. The interaction of recursive types with subtyping has been studied before [4], and shown to present a number of interesting challenges. While both are important features of many programming languages, there are only few interpretations that satisfactorily model both.

Game Semantics is a framework for modelling programming languages that combines the elegant mathematical structure of Denotational Semantics with explicitly operational notions. Due to the blend of the two, Game Semantics has been successful at modelling a wide range of programming language features. In a previous work [7], we have shown how the simple feature of adding explicit error elements to Game Semantics allows us to model subtyping; in this paper, we extend this model to include recursive types, and show that it validates the subtyping rule for recursive types proposed by Amadio and Cardelli [4].

There are two main new results in this paper. First, we show how a minor modification of the operational semantics of the untyped model presented in an earlier makes the model computationally adequate (Sections 1.1 and 2.2), thus solving the main open problem in our previous paper [7]. Second, we show how the space of games, used for modelling types, can be equipped with a metric that allows us to construct recursive types; the metric is shown to interact with the order structure related to subtyping so as to validate the desired subtyping rules (Sections 4 and 5).

1 A λ-calculus with errors

We consider an untyped λ-calculus with ground values, defined by the following syntax:

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

The only unusual feature of this calculus is the presence of a ground value **top** that will be used for representing the result of badly-typed terms.

Our calculus may be equipped with an operational semantics *e.g.* by defining a *one step reduction* relation $\rightsquigarrow$ on terms. For our calculus, a common choice — the *call-by-name* semantics — consists of the rules

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

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

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

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

where the set of *evaluation contexts* $E[\cdot]$ is defined by

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

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

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:

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

A very different 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: there is no M' 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 **top** to represent untrappable errors. Intuitively, a term should reduce to **top** whenever its reduction gets "stuck" with no applicable rule; unfortunately, such a simple extension does not quite work. Indeed, consider the "identity on the Booleans" $\mathbf{I_{Bool}} = \lambda x.\mathbf{if}\ x\ \mathbf{then\ tt\ else\ ff\ fi}$; this term behaves as the identity when applied to a Boolean, but returns an error when applied to a function or a pair. Let now $\mathbf{Y}$ be a fixpoint combinator, and consider the "looping Boolean" $(\mathbf{Y}\ \mathbf{I_{Bool}})$; intuitively, we would expect this term to loop when invoked in a Boolean context, but return an error when *e.g.* applied. The reduction relation $\downarrow$, augmented as suggested above, causes it to loop (this problem is expressed technically by the failure of computational adequacy of our model w.r.t. $\downarrow$ [7, Section 2.3]).

We therefore define a different reduction relation, written $\Downarrow$, which is explicitly decorated with the locus of the computation. To do so, we introduce a notion of *initial component*, a finite sequence over $\{1, l, r\}$ (the empty sequence is written ϵ). Walking the syntax tree of a type, computation happening on the right-hand-side of an arrow is marked by 1; computation happening on the left-hand-side (resp. right-hand-side) of a product is marked by l (resp. r). A family of reduction relations, indexed by initial components, is defined in Fig. 1.

As usual, we write $M \Uparrow_c$ when there is no V such that $M \Downarrow_c V$. We write ϵ for the empty component, and $|c|$ for the length of component c.

To clarify this definition, note that the form of a value resulting from reduction at initial component c is determined by c. More precisely, if M is a closed term and c an initial component such that $M \Downarrow_c V$, then one of the following is true:

- $V = \mathbf{top}$; or
- $c = \epsilon$ and $V = \mathbf{ff}$ or $V = \mathbf{tt}$; or
- c is of the form $1 \cdot c'$ and V is of the form $\lambda x.M'$; or
- c is of the form $l \cdot c'$ or $r \cdot c'$, and V is of the form (N, P).

Conversely,

- if $(M, N) \Downarrow_c P$, then either c is of the form $l \cdot c'$ or $r \cdot c'$, or $P = \mathbf{top}$;
- if $\lambda x.M \Downarrow_c N$, then either c is of the form $1 \cdot c'$ or $N = \mathbf{top}$;
- $\mathbf{tt} \Downarrow_c N$ or $\mathbf{ff} \Downarrow_c N$, then either $c = \epsilon$ or $N = \mathbf{top}$.

There is also a simple relationship between $\Downarrow$ and the simple reduction relation $\downarrow$; it shows that the extension that we introduce only concerns erroneous reductions. Roughly speaking, the relations $\Downarrow$ and $\downarrow$ coincide, except in the case in which $\Downarrow$ yields an untrappable error and $\downarrow$ diverges. More precisely, $M \downarrow V$ implies that for some initial component c, $M \Downarrow_c V$. Conversely, $M \uparrow$ implies that for all c, either $M \Uparrow_c$ or $M \Downarrow_c \mathbf{top}$. Finally, if for some c, $M \Uparrow_c$, then $M \uparrow$.

$$\mathbf{tt} \Downarrow_\epsilon \mathbf{tt} \qquad \mathbf{ff} \Downarrow_\epsilon \mathbf{ff} \qquad \mathbf{tt} \Downarrow_c \mathbf{top} \quad (c \neq \epsilon) \qquad \mathbf{ff} \Downarrow_c \mathbf{top} \quad (c \neq \epsilon)$$

$$\lambda x.M \Downarrow_{1\cdot c} \lambda x.M \qquad \lambda x.M \Downarrow_c \mathbf{top} \quad (c \neq 1 \cdot c')$$

$$(M, N) \Downarrow_{l\cdot c} (M, N) \qquad (M, N) \Downarrow_{r\cdot c} (M, N) \qquad (M, N) \Downarrow_c \mathbf{top} \quad (c = \epsilon \text{ or } c = 1 \cdot c')$$

$$\frac{M \Downarrow_{1\cdot c} \lambda x.M' \qquad M'[x\backslash N] \Downarrow_c P}{(M\ N) \Downarrow_c P} \qquad \frac{M \Downarrow_{1\cdot c} \mathbf{top}}{(M\ N) \Downarrow_c \mathbf{top}}$$

$$\frac{M \Downarrow_{l\cdot c} (N, P) \qquad N \Downarrow_c N'}{\pi_l(M) \Downarrow_c N'} \qquad \frac{M \Downarrow_{r\cdot c} (N, P) \qquad P \Downarrow_c P'}{\pi_r(M) \Downarrow_c P'}$$

$$\frac{M \Downarrow_{l\cdot c} \mathbf{top}}{\pi_l(M) \Downarrow_c \mathbf{top}} \qquad \frac{M \Downarrow_{r\cdot c} \mathbf{top}}{\pi_r(M) \Downarrow_c \mathbf{top}}$$

$$\frac{M \Downarrow_\epsilon \mathbf{tt} \qquad N \Downarrow_\epsilon N'}{\mathbf{if}\ M\ \mathbf{then}\ N\ \mathbf{else}\ P\ \mathbf{fi} \Downarrow_\epsilon N'} \qquad \frac{M \Downarrow_\epsilon \mathbf{ff} \qquad P \Downarrow_\epsilon P'}{\mathbf{if}\ M\ \mathbf{then}\ N\ \mathbf{else}\ P\ \mathbf{fi} \Downarrow_\epsilon P'}$$

$$\frac{M \Downarrow_\epsilon \mathbf{top}}{\mathbf{if}\ M\ \mathbf{then}\ N\ \mathbf{else}\ P\ \mathbf{fi} \Downarrow_\epsilon \mathbf{top}} \qquad \mathbf{if}\ M\ \mathbf{then}\ N\ \mathbf{else}\ P\ \mathbf{fi} \Downarrow_c \mathbf{top} \quad (c \neq \epsilon)$$

Fig. 1. Big-step semantics with errors and initial components

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. One choice would be to add an error value **error** "on the side" (Fig. 2(a)); another one would be to add a value **top** as a top element (Fig. 2(b)).

It is our view that errors on the side model (trappable) exceptions, while errors at top model untrappable errors. Consider, indeed, the addition to our calculus of a term **ignore-errors** that would satisfy

$$\mathbf{ignore\text{-}errors\ tt} \Downarrow_\epsilon \mathbf{tt}$$
$$\mathbf{ignore\text{-}errors\ ff} \Downarrow_\epsilon \mathbf{tt}$$
$$\mathbf{ignore\text{-}errors\ top} \Downarrow_\epsilon \mathbf{ff}$$

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

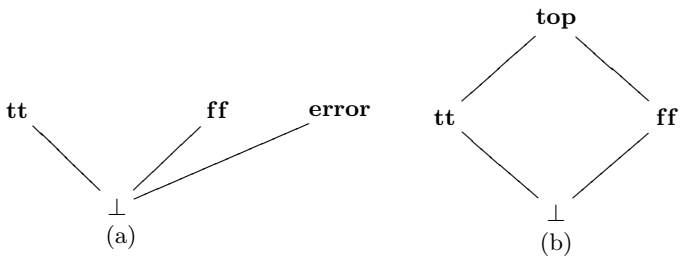

Fig. 2. Two domains of Booleans with errors

Errors "on the side," or exceptions, have been studied before [6]; in this paper, we adopt the domain in Fig. 2(b).

The addition of a top value to Scott domains was a common feature of early Denotational Semantics. 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*. We choose our set of observations to consist of the observations "reduction to **top** at ϵ," "reduction to **tt** at ϵ" and "reduction to **ff** at ϵ," ordered analogously to Fig. 2(b).

Definition 1. *(Observational preorder)*

$$M \lesssim N \text{ iff } \forall C[\cdot] \begin{cases} C[M] \Downarrow_\epsilon \mathbf{top} \Rightarrow C[N] \Downarrow_\epsilon \mathbf{top}; \\ C[M] \Downarrow_\epsilon \mathbf{tt} \Rightarrow C[N] \Downarrow_\epsilon \mathbf{tt} \text{ or } C[N] \Downarrow_\epsilon \mathbf{top}; \\ C[M] \Downarrow_\epsilon \mathbf{ff} \Rightarrow C[N] \Downarrow_\epsilon \mathbf{ff} \text{ or } C[N] \Downarrow_\epsilon \mathbf{top}. \end{cases}$$

We say that two terms M and N are observationally equivalent*, and write $M \cong N$, when $M \lesssim N$ and $N \lesssim M$.*

As usual in calculi with ground values, the observational preorder can be defined by just one well-chosen observation; one possible choice is reduction to **top** at ϵ.

Lemma 2. *$M \lesssim N$ if and only if*

$$\forall C[\cdot]\ C[M] \Downarrow_\epsilon \mathbf{top} \Rightarrow C[N] \Downarrow_\epsilon \mathbf{top}.$$

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

It is worthwhile to compare our calculus with Abramsky's *lazy λ-calculus* [1]. Writing Ω for the looping term (*e.g.* $\Omega = ((\lambda x.(x\ x))\ \lambda x.(x\ x))$), notice that Ω and $\lambda x.\Omega$ are observationally distinct. Indeed, taking

$$C[\cdot] = \mathbf{if}\ \cdot\ \mathbf{then\ tt\ else\ ff\ fi}$$

we have $C[\Omega] \Uparrow_\epsilon$, while $C[\lambda x.\Omega] \Downarrow_\epsilon \mathbf{top}$. On the other hand, as we shall see in Section 2, we introduce no explicit lifting in a sound and computationally adequate model. Thus, we believe that our calculus combines the most desirable characteristics of what Abramsky calls the *standard interpretation* of the λ-calculus with those of the lazy calculus. The fundamentally call-by-name nature of the construction is reflected in the syntax by the fact that the terms **top** and $\lambda x.\mathbf{top}$ are observationally equivalent (see the end of Section 2.2).

2 A game semantics for the untyped calculus

This section roughly outlines the semantic framework used for modelling untyped terms. As Game Semantics has been described before [2, 11], and so has our particular framework [7, 8], this section remains informal.

In Game Semantics, a term is represented by a *strategy*, 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 languages. By convention, Opponent plays first when modelling a call-by-name calculus.

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. Precisely, a move is of the form m_c, where m is one of q, the *question*, $a^{\mathtt{tt}}$, the *answer true*, or $a^{\mathtt{ff}}$, the *answer false*, and c, the *component* of the move, is a finite string over 0, 1, l, r. In addition, moves are decorated with justification pointers which, while absolutely necessary for the correction of the interpretation, are not essential for the ideas in this paper.

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.

The main novelty of the formalism used in this work and introduced in [8, 7] is that we allow strategies to refuse moves, which is used for modelling untrappable errors. Concretely, this is realised by allowing strategies to contain both even- and odd-length positions. In a spirit similar to that of Harmer [9, Chapter 4], even-length positions represent moves that are played by Player, while odd-length positions represent situations in which player loops.

Definition 3. *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 Pref A for the prefix completion of A.

The even-prefix-closedness condition in this definition says that a strategy cannot mandate that Opponent should 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. 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.

In [7], we define a certain number of strategies. The strategy **top** consists of the single empty position ϵ; this strategy never accepts an Opponent's move. The strategy Ω consists of all positions of length 1; thus, it always accepts an initial move from Opponent, but never plays a move. The strategy **tt** consists of all even-length positions composed of alternations of the moves q and $a^{\mathbf{tt}}$; thus, it always accepts an initial question, and replies with the answer true (**ff** is analogous).

The class of strategies that copy moves between components are known as the *copy-cat* strategies; this class includes the identity **I**, the projections π_r and π_l, and, to a certain extent, the "if-then-else" strategy **ite**. In addition, we use a number of operations on strategies, including (functional) pairing $\langle \cdot, \cdot \rangle$, currying $\Lambda(\cdot)$, as well as the *injection* $K(\cdot)$ which "shifts" a strategy into component 1.

Composition of strategies s and t is performed by ranging over all behaviours in s and t, selecting those that are agree on a common component, and composing them, similarly to Baillot *et al.* [5]. However, we cannot just use their formalism, as 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 4. *Given a natural integer n, we say that two positions 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 $q \upharpoonright 0$ (or $p \upharpoonright 1 = q \upharpoonright 0$ if both projections are of length smaller than n).*

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

2.1 The liveness ordering

We now introduce an ordering — the *liveness ordering* $\preccurlyeq$ — which will model the observational preorder, the typing relation, and the subtyping relation (Lemmata 10 and 21 and Theorem 22), and is inspired by Abramsky's "back-and-forth inclusion relation" [2]. The definition of the liveness ordering is analogous to that of the observational preorder. Just like for terms M and N we have $M \lesssim N$ when M produces errors in less contexts than N, and N produces results in more contexts than M (Definition 1), 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 5. *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. A clarification of the intuitions behind the liveness ordering may be found in [8, 7].

Theorem 6. *The relation $\preccurlyeq$ is a partial order on non-empty prefix-closed collections of positions.*

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 7. *If s and t are strategies, then $\mathrm{Pref}(s) = \mathrm{Pref}(t)$ implies $s = t$. The relation $\preccurlyeq$ is therefore a partial order on strategies.*

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

2.2 Interpretation of the calculus

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.

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

The notion of soundness that we use is somewhat complicated by the fact that we use a family of reduction relations. Given a component c, we say that two strategies s and t are equal at component c, and write $s =_c t$, when the sets of positions starting with q_c in s and t coincide.

Lemma 8. (Equational Soundness) *If $[\![\Gamma \vdash M]\!]$ is defined and $M \Downarrow_c N$, then $[\![\Gamma \vdash N]\!]$ is defined and $[\![\Gamma \vdash N]\!] =_{1 \cdot c} [\![\Gamma \vdash M]\!]$.*

The interpretation is also computationally adequate.

Lemma 9. (Computational Adequacy) *If $[\![\Gamma \vdash M]\!]$ is defined and there is no term N such that $M \Downarrow_\epsilon N$, then $[\![\Gamma \vdash M]\!] =_1 \bot$.*

In order to prove this property, we use a variant of Plotkin's method of formal approximation relations. We say that a family $\lhd_c$ of relations between strategies and terms, indexed by initial components, is a *family of formal approximation relations* when it satisfies a number of fairly natural properties that imply in particular that $s \lhd_\epsilon M$ implies $s =_1 \bot$ or $M \Downarrow_\epsilon$. We then show the existence of such a family, and that for any closed term M and initial component c, $[\![\Gamma \vdash M]\!] \lhd_c M$, which allows us to conclude by a standard argument.

Soundness, computational adequacy and Lemma 2 imply inequational soundness.

Lemma 10. (Inequational Soundness) *For any two terms M and N, if $[\![M]\!] \preccurlyeq [\![N]\!]$ then $M \lesssim N$.*

Inequational soundness can often be used for proving properties about the calculus itself. For example, as the terms **top** and $\lambda x.\mathbf{top}$ have the same interpretation, we may conclude that they are in fact observationally equivalent.

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,

$$A, B ::= \mathbf{Bool} \mid \top \mid X \mid A \times B \mid A \to B \mid \mu X.C$$

where the type C is *guarded* in the type variable X. Thus, types consist of the ground type **Bool** of Booleans, the type $\top$ of all terms, type variables, product types, arrow types, and recursive types.

The set of types guarded in a type variable X is defined by the grammar

$$C, D ::= \mathbf{Bool} \mid \top \mid Y \mid A \times B \mid A \to B \mid \mu Y.C$$

In order to speak about subtyping of recursive types, we need a notion of *covariant* type. The set of types covariant in a type variable X is defined by the grammar

$$E, F ::= \mathbf{Bool} \mid \top \mid X \mid Y \mid E \times F \mid G \to E \mid \mu Y.E$$

where G is *contravariant* in X; the set of types contravariant in a type variable X is defined by the grammar

$$G, H ::= \mathbf{Bool} \mid \top \mid Y \mid G \times H \mid E \to G \mid \mu Y.G$$

An *environment* is a set of type variables and a map from variables to types. We use the letter E to range over environments, and write

$$X, Y, x : C, y : D$$

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

We use two kinds 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.

The set of inference rules used for typing is given in Figures 3 and 4. Somewhat unusual is the fact that there are no explicit rules for the folding and unfolding of recursive types; these rules can in fact be derived from the penultimate rule in Fig. 4 and subsumption (the last rule in Fig. 3). The last rule in Fig. 4 is the subtyping rule proposed by Amadio and Cardelli [4].

3.1 Games and the liveness ordering

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.

As we use the liveness ordering to interpret typing, 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. Technically, this is expressed by the reachability condition in the definition of the liveness ordering.

Definition 11. *A* game *is a non-empty prefix-closed set of positions.*

$$E \vdash M : \top \qquad E, x : A \vdash x : A$$

$$\frac{E, x : A \vdash M : B}{E \vdash \lambda x.M : A \to B} \qquad \frac{E \vdash M : A \to B \quad E \vdash N : A}{E \vdash (M\ N) : B}$$

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

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

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

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

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

Fig. 3. Typing rules

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

The game $\top = \{\epsilon\}$ is the maximal element of the lattice of games. The game **Bool** of Booleans it is defined as the set of all interleavings of positions in Pref $\mathbf{tt}$ and Pref $\mathbf{ff}$. 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.

4 A metric on games

In order to solve recursive type equations, we use Banach's fixpoint theorem. We recall that a metric space is said to be *complete* when every Cauchy sequence has a limit. A map over a metric space (X, d) is said to be Lipschitz with constant $\lambda > 0$ when for all $x, y \in X$, $\mathrm{d}(f(x), f(y)) \leq \lambda \mathrm{d}(x, y)$. Such a map is said to be *nonexpanding* when $\lambda \leq 1$, and *contractive* when $\lambda < 1$.

Theorem 12. (Banach) *A contractive map f over a complete metric space has a unique fixpoint* $\mathrm{fix}(f)$.

In order to solve recursive type equations using Banach's theorem, we need to equip the set of games $\mathcal{G}$ with a metric that makes it into a complete space;

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

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

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

$$E \vdash \mu X.B[X] = B[\mu X.B[X]]$$

$$\frac{E, X \vdash A[X] \leq B[X]}{E \vdash \mu X.A[X] \leq \mu X.B[X]} \quad (A,\ B \text{ covariant in } X)$$

Fig. 4. Subtyping rules

furthermore, the metric should make all type constructors into contractive maps. Games, being prefix-closed collections of sequences, may be seen as trees, so it would seem natural to equip $\mathcal{G}$ with the tree metric. Unfortunately, this simple approach does not yield enough contractive maps, failing in particular to make the product contractive. For this reason, we apply the tree metric method twice, once to components and once to positions.

Definition 13. *Let $p = m_0 \cdots m_{n-1}$ be a position, and M_p the set of moves m such that $p \cdot m$ is a position. For any move $m \in M_p$, the* weight *of m w.r.t. p is defined by $\mathrm{w}_p(m) = 2^{-|c|}$, where c is the component of m.*

The ultrametric d_p on M_p is defined, for distinct moves m, m', as $\mathrm{d}_p(m, m') = 2^{-|c|}$, where c is the longest common prefix of the components of m, m'.

We are now ready to define the metric on positions that will serve our needs. Given two distinct positions p and p', either one is the prefix of the other, in which case we will use the weight of the first differing move, or neither is a prefix of the other, in which case we use the distance between the first differing moves.

Definition 14. *Given a position $p = q \cdot m_n = m_0 \cdot m_1 \cdots m_{n-1} \cdot m_n$, the* weight *of p is defined as $\mathrm{w}(p) = 2^{-n}\mathrm{w}_q(m_n)$.*

The metric d on the set of positions is defined as follows. Given two distinct positions p, p', let $q = m_0 \cdots m_{n-1}$ be their longest common prefix. If

$$p = q \cdot m_n \cdot r, \qquad p' = q \cdot m'_n \cdot r',$$

then $\mathrm{d}(p, p') = 2^{-n}\mathrm{d}_q(m_n, m'_n)$. On the other hand, if

$$p = q \cdot m_n \cdot r, \qquad p' = q,$$

then $\mathrm{d}(p, p') = 2^{-n}\mathrm{w}_q(m_n)$.

Note that this metric does not induce the discrete topology on the set of positions; games, however, are closed with respect to it, and therefore we may still apply the Hausdorff formula to games.

Definition 15. *The metric* d *on the set of games is defined by the Hausdorff formula*

$$\mathrm{d}(A, B) = \max(\sup_{p \in A} \inf_{q \in B} \mathrm{d}(p, q), \sup_{p \in B} \inf_{q \in A} \mathrm{d}(p, q)).$$

As the space of positions is not complete, and games are not necessarily compact[1], we cannot take any of the properties of Hausdorff's metric for granted. However, a fairly standard proof shows that in fact d does have all the desired properties.

Theorem 16. *The space of games* $(\mathcal{G}, \mathrm{d})$ *is a complete ultrametric space.*

There is another property that we will need in order to prove soundness of typing: the fact that least upper bounds preserve the ordering in some cases. The following property is simple enough to prove directly and is sufficiently strong for our needs:

Lemma 17. *If* A *is a game, then the order ideal* $\{B \mid B \preccurlyeq A\}$ *is closed with respect to* d.

This is proved by considering a game $C \not\preccurlyeq A$. If A contains a position p such that $p \notin C$ but all strict prefixes of p are in C, we define the real number δ as the minimum of the weights of all prefixes of p, and show that for any $B \preccurlyeq A$, $\mathrm{d}(B, C) \geq \delta$. A similar argument applies in the case when $p \in C$, $p \notin A$ and all strict prefixes of p are in A.

This property implies the following one, which we will need in order to prove soundness:

Lemma 18. *Let* $f, g : \mathcal{G} \to \mathcal{G}$ *be monotone, contractive maps over games such that for any game* A, $f(A) \preccurlyeq g(A)$. *Then* $\mathrm{fix}(f) \preccurlyeq \mathrm{fix}(g)$.

Finally, as the metric was constructed *ad hoc*, it is a simple matter to show that all type constructors are contractive.

Lemma 19. *The maps over games* $\cdot \times \cdot$ *and* $\cdot \to \cdot$ *are contractive in all of their arguments.*

5 Interpreting 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

[1] Actually, they are in this case, but would no longer necessarily be so if we chose to use an infinite set of ground values.

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:

$$\begin{array}{cc} [\![\mathbf{Bool}]\!]\eta = \mathbf{Bool} & [\![A \times B]\!]\eta = [\![A]\!]\eta \times [\![B]\!]\eta \\ [\![\top]\!]\eta = \top & [\![A \to B]\!]\eta = [\![A]\!]\eta \to [\![B]\!]\eta \\ [\![X]\!]\eta = \eta(X) & [\![\mu X.A[X]]\!]\eta = \mathrm{fix}(\lambda\mathcal{X}.[\![A]\!](\eta[X\backslash\mathcal{X}])) \end{array}$$

The well-foundedness of this definition is a consequence of the following lemma:

Lemma 20. *If A is a type, then*

(i) $[\![A]\!]$ *is well-defined;*
(ii) $[\![A]\!]$ *is a pointwise nonexpanding map;*
(iii) if A is guarded in X, then $[\![A]\!](\eta[X\backslash\mathcal{X}])$ is contractive in $\mathcal{X}$;
(iv) if A is covariant (resp. contravariant) in X, then $[\![A]\!](\eta[X\backslash\mathcal{X}])$ is monotone (resp. antimonotone) in $\mathcal{X}$.

The four properties are shown simultaneously by induction on the syntax of types. The only issue with part (i) is that of the existence and unicity of fixpoints, which is a consequence of part (iii) of the induction hypothesis, the fact that we restrict the fixpoint operator to guarded types, and Banach's fixpoint theorem. Parts (ii) and (iii) follow from Lemma 19 and the fact that d is an ultrametric, and part (iv) only depends on itself.

5.1 Soundness of typing

The following lemma expresses the soundness of subtyping and is proved by induction on the derivation of $E \vdash A \leq A'$.

Lemma 21. (Soundness of subtyping) *Let E be an environment, and A and A' types such that $E \vdash A \leq A'$. Let η be a type environment; then $[\![A]\!]\eta \preccurlyeq [\![A']\!]\eta$.*

The main novelty in this lemma is the soundness of the last subtyping rule for the fixpoint operator; this is a consequence of Lemma 18.

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

Theorem 22. *Let E be a typing environment, M a term and A a type such that $E \vdash M : A$. Suppose that $E = X_1, \ldots X_n, x_1 : C_1, \ldots x_m : C_m$, and let $\Gamma = x_1, \ldots x_m$; let η be a typing environment, and C be the type $C = (\cdots(C_1 \times C_2) \times \cdots C_m)$. Then $[\![\Gamma \vdash M]\!] \preccurlyeq [\![C]\!]\eta \to [\![A]\!]\eta$.*

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 23. (Safety of Typing) *If $\vdash M : A$, where M is a closed term and A a closed type such that $[\![A]\!]\varnothing \neq_\epsilon \top$, then it is not the case that $M \Downarrow_\epsilon \mathbf{top}$.*

6 Conclusions and further work

In a previous work [7], we have shown how the addition of explicit untrappable errors to a simple λ-calculus with ground values induces a notion of subtyping, and have shown a sound Game Semantics model of the calculus with explicit errors and subtyping. In this paper, we have shown how an minor modification of the operational semantics makes our model computationally adequate. We believe that this calculus combines the best features of the standard and lazy semantics of the λ-calculus.

In addition, we have shown how the model supports recursive types by using fairly standard machinery, mainly a variant of the tree topology, and Banach's fixpoint theorem. By proving a property relating various order-theoretic and metric topologies, we have shown how our model validates the subtyping rule proposed by Amadio and Cardelli.

There is, however, an issue remaining. In [7], we have shown how the model supports bounded quantification. As we note in [8], we have been unable to make recursive types and quantifiers coexist in the same model. Indeed, while there is no problem with quantifying over fixpoints, there is no apparent reason why a least upper bound of contractive maps should itself be contractive; the issue is analogous to the well-known lack of properties of intersection with respect to Hausdorff's metric.

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.
3. 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.
4. R. M. Amadio and L. Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, 1993.
5. P. Baillot, V. Danos, T. Ehrhard, and L. Regnier. Believe it or not, AJM's games model is a model of classical linear logic. In *Proceedings of the Twelfth International Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1997.
6. R. Cartwright, P.-L. Curien, and M. Felleisen. Fully abstract models of observably sequential languages. *Information and Computation*, 111(2):297–401, 1994.
7. J. Chroboczek. Game Semantics and Subtyping In *Proceedings of the Fifteenth Annual IEEE Symposium on Logic in Computer Science*, Santa Barbara, California, June 2000.
8. J. Chroboczek. *Game Semantics and Subtyping.* Forthcoming PhD thesis, University of Edinburgh, 2000.
9. R. Harmer. *Games and Full Abstraction for Nondeterministic Languages.* PhD thesis, Imperial College, University of London, 1999.
10. J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. 1994.
11. G. McCusker. *Games and Full Abstraction for a Functional Metalanguage with Recursive Types*. PhD thesis, Imperial College, University of London, 1996.