Interaction Systems and Linear Logic, a different games semantics

Pierre Hyvernat

Interaction Systems and Linear Logic

¿A Different Games Semantics?

Pierre Hyvernat

Laboratoire de mathématiques
université de Savoie
campus scientifique
73376 Le Bourget-du-Lac Cedex
France

Abstract

We define a model for linear logic based on two well-known ingredients: games and simulations. This model is interesting in the following respect: while it is obvious that the objects interpreting formulas are *games* and that everything is developed with the intuition of *interaction* in mind, the notion of morphism is very different from traditional morphisms in games semantics. In particular, we make no use of the notion of strategy! The resulting structure is very different from what is usually found in categories of games.

We start by defining several constructions on those games and show, using elementary considerations, that they enjoy the appropriate algebraic properties making this category a denotational model for intuitionistic linear logic. An interesting point is that the tensor product corresponds to a strongly *synchronous* operation on games

This category can also, using traditional translations, serve as a model for the simply typed λ-calculus. We use some of the additional structure of the category to extend this to a model of the *simply typed differential λ-calculus* of [1]. Once this is done, we go a little further by constructing a reflexive object in this category, thus getting a concrete non-trivial model for the *untyped* differential λ-calculus.

We then show, using a highly non-constructive principle, that this category is in fact a model for full *classical* linear logic ; and we finally have a brief look at the related notions of predicate transformers ([2]) and containers ([3]).

Key words: Denotational Semantics, Linear Logic, Differential λ-calculus, Interaction Systems, Simulations, Games Semantics, Containers

---

Email address: `pierre.hyvernat@univ-savoie.fr` (Pierre Hyvernat).

Introduction

Transition systems and simulation relations are well known tools in computer science. More recent is the use of games to give models for different programming languages [4,5], or as an interesting tool for the study of other programming notions [6]. We devise a denotational model of linear logic based on those two ideas. Basically, a formula will be interpreted by an "alternating transition system" (called an *interaction system*) and a proof will be interpreted by a *safety property* for this interaction system. Those concepts which were primarily developed to model imperative programming and interfaces turned out to give a rather interesting games model: a formula is interpreted by a game (the interaction systems), and a proof by a witness that a non-loosing strategy exists (the safety property). The notion of morphism corresponds to the notion of simulation relation, a particular case of safety properties.

Part of the interest is that the notion of safety property is very simple. They are only subsets of states in which the first player can remain, whatever the second player does. In other words, from any state in the safety property, the first player has an infinite strategy *which never leaves the safety property.* This is to be contrasted with traditional notions where morphisms are *functions* (usually depending on some subset of the history) giving the strategy.

The structure of safety properties is much richer than the structure of proofs. In particular, safety properties are closed under arbitrary unions. Since there is no notion of "sum" of proofs, this doesn't reflect a logical property. However, this is a feature rather than a bug: the differential λ-calculus of Ehrhard and Regnier ([1]) is an extension of λ-calculus with a notion of differentiation and non-deterministic sum. As we will show, interaction systems can interpret this extra structure quite naturally.

Even better, this category enjoys such properties that we can, without much difficulty, construct a *reflexive object* allowing an interpretation of *untyped* differential λ-calculus. This reflexive object is constructed using a fixpoint construction which is already available in the category **Rel** of sets and relations between them.

The last thing we look at in this category is the object "$\perp$". As far as interaction is concerned, this is one of the most boring objects. However, it satisfies a very strong algebraic property: it is *dualizing* and we can thus interpret the whole of *classical* linear logic. This was rather unexpected and has a few surprising consequences (see corollary 34). The main reason for this is that the principle used to prove this fact (the contraposition of the axiom of choice) is relatively counter-intuitive. It is also the only part of this work where highly non-constructive principles are used, which explains why we separate this from the rest.

We conclude this paper by relating interaction systems with other notions, namely the notion of *predicate transformers* ([2]) and the notion of *containers* ([3]) and "dependent containers".

1 Interaction Systems

The definition of interaction system we are using was developed primarily by Peter Hancock and Anton Setzer. Their aim was to describe *programming interfaces* in dependent type theory ([7,8]). The ability to use dependent types makes it possible to add logical specifications to usual specification. The result is a notion of formal interface describing:

- the way the programmer is allowed to use commands;
- and the logical properties he can expect from those commands.

In practice, it is usually the case that the logical specification is ensured *a posteriori*: a command might be legally used in a situation, even though the logical specification prevents it

Because of their definition however, interaction systems can be interpreted in many different ways. In order to develop some intuitions, we prefer using a "games" interpretation: an interaction system describes the modalities of a two persons game.

1.1 The Category of Interaction Systems

Let's start with the raw definition:

Definition 1 *Let S be a set (of* states*); an* interaction system *on S is given by the following data:*

- *for each $s \in S$, a set $A(s)$ of* possible actions;
- *for each $a \in A(s)$, a set $D(s,a)$ of* possible reactions *to a;*
- *for each $d \in D(s,a)$, a* new state *$n(s,a,d) \in S$.*

We usually write $s[a/d]$ instead of $n(s,a,d)$.

We use the letter w for an arbitrary interaction system, and implicitly name its components A, D and n. To resolve ambiguity, we sometimes use the notation

$w.A$, $w.D$ and $w.n$ to denote the components of interaction system w. The set of states is usually implicit, and we write it $w.S$.

Following standard practice within computer science, we distinguish the two "characters" by calling them the Angel (choosing actions, hence the A) and the Demon (choosing reactions, hence the D). Depending on the authors' background, other names could be Player and Opponent, Eloise and Abelard, Alice and Bob, Master and Slave, Client and Server, System and Environment, alpha and beta, Arthur and Bertha, Left and Right etc.

As stated above, one of the original goals for interaction systems was to represent programming interfaces. Here is for example the interface of a stack of booleans:

- $S = \mathsf{List}(\mathbf{B})$;

  The set of states S represents the "virtual" internal states of an implementation of stacks: lists (stacks) of booleans.

- $\begin{cases} A([]) = \{\mathsf{Push}(b) \mid b \in \mathbf{B}\} \\ A(\_) = \{\mathsf{Push}(b) \mid b \in \mathbf{B}\} \cup \{\mathsf{Pop}\} \end{cases}$

  In a non-empty state, we can issue a "pop" or a "push(b)" command, but if the state is empty, we can only issue a push command.

- $D(\_,\_) = \{\mathsf{Akn}\}$

  The responses are (in this case) trivial: we can only get an "Aknowledgment".

- $\begin{cases} n(s, \mathsf{Push}(b)) = b :: s \\ n(b :: s, \mathsf{Pop}) = s \end{cases}$

  The next state is defined by either adding a boolean in front of the list (push) or removing the first element of the list (pop).

What is still missing from this description is the "side-effects" part: it doesn't say anywhere that a "pop" command will return the first element of the state. [1] This is however much more precise than traditional interfaces which are usually given by a collection of types. Compare with this poor description of stacks:

- $\mathsf{Pop} : \mathbf{B}$
- $\mathsf{Push} : \mathbf{B} \to ()$

For any given stack, there are two commands: "Pop" and "push": the first one *returns* a boolean and the second one takes a single boolean argument and "does something".

Since we are aiming at a games semantics, we will rather use the following interpretation:

- S is a set of states describing the possible states of a game;
- for a state $s \in S$, the set $A(s)$ is the set of legal moves in state s;
- for such a move $a \in A(s)$, the set $D(s, a)$ is the set of countermoves to a;
- finally, $s[a/d]$ is just the new state of the game after a/d has been played.

[1] It is possible to devise interaction systems with such side-effects, but the theory of those is still to be developed.

Departing from the well established tradition of “morphisms as strategies”, we use the following notion of “simulation relation”:

Definition 2 *If w_1 and w_2 are two interaction systems on S_1 and S_2 respectively; a relation $r \subseteq S_1 \times S_2$ is called a* simulation *if:*

$$\begin{array}{lll} (s_1, s_2) \in r & \Rightarrow & (\forall a_1 \in A_1(s_1)) \\ & & (\exists a_2 \in A_2(s_2)) \\ & & (\forall d_2 \in D_2(s_2, a_2)) \\ & & (\exists d_1 \in D_1(s_1, a_1)) \\ & & \quad (s_1[a_1/d_1], s_2[a_2/d_2]) \in r \ . \end{array}$$

This definition is very similar to the usual definition of simulation relations between labeled transition systems, but adds one layer of quantifiers to deal with reactions. That $(s_1, s_2) \in r$ means that “s_2 simulates s_1”. By extension, if a_2 is a witness to the first existential quantifier, we say that “a_2 simulates a_1”. Note that because the left hand side would be vacuous, the empty relation is always a simulation.

It is illuminating to look at the usual “copycat strategy” with this in mind: a simulation is just a generalization of what happens there. Intuitively, a simulation from w to w' means that if the Angel knows how to play in w, she can simulate, move after move, a play in w'. (And vice and versa for the Demon.)

To continue on the previous example, programming a stack interface amounts to implementing the stack commands using a lower level interface (arrays and pointer for examples). If we interpret the quantifiers constructively, this amounts to providing a (constructive) proof that a non-empty relation is a simulation from this lower level interaction system to stacks: for each of the stacks commands, we need to provide a witness command in the low level world in such a way as to guarantee simulation. (See [9] and [10] for a more detailed description of programming in terms of interaction systems.)

Recall that the composition of two relations is given by:

$$(s_1, s_3) \in r_2 \cdot r_1 \quad \Leftrightarrow \quad (\exists s_2)\ (s_1, s_2) \in r_1 \text{ and } (s_2, s_3) \in r_2$$

It should be obvious that the composition of two simulations is a simulation and that the equality relation is a simulation from any w to itself. Thus, we can put:

Definition 3 *We call* **Int** *the category of interaction systems with simula-*

tions.

Note that this category is locally small but that, for any given set of states S, the collection of interaction systems on S forms a proper class. It is possible to restrict to "finitary" interaction systems: those for which the sets of actions and the sets of reactions are always finite. For our purposes, it is impossible to restrict to finite sets of states (see the definition of $!w$) or to countable actions/reactions sets (see the definition of $w_1 \multimap w_2$). Subtler considerations show that it is however possible to restrict to sets of states of cardinalities $\aleph_0$ *and* sets of actions of cardinalities $2^{\aleph_0}$ and sets of reactions of cardinalities $\aleph_0$. (See the proof of proposition 38 for a hint.)

We have the following "forgetful" functor from **Int** to **Rel**, the category of sets and relations between them:

Lemma 4 *The operation* $w \mapsto |w| = w.S$ *is a faithful functor from* **Int** *to* **Rel**. *Its action on morphisms is just the identity.*

This functor has a right adjoint $S \mapsto \mathsf{magic}(S)$ *and a left adjoint* $S \mapsto \mathsf{abort}(S)$ *defined by* *(where* $\{*\}$ *denotes a singleton set)*

$$\begin{array}{lcllcl}
\mathsf{magic}(S).A(s) & = & \{*\} & \mathsf{abort}(S).A(s) & = & \emptyset \\
\mathsf{magic}(S).D(s,*) & = & \emptyset & \mathsf{abort}(S).D(s,\_) & = & \_ \\
\mathsf{magic}(S).n(s,*,\_) & = & \_ & \mathsf{abort}(S).n(s,\_,\_) & = & \_
\end{array}$$

In terms of games, magic means "the Demon resigns" (he cannot answer any move) while abort means "the Angel resigns" (she cannot play).

This category enjoys a very strong algebraic property:

Proposition 5 *The category* **Int** *is enriched over complete sup-lattices.*

Note that enrichment over sup-lattice is stronger than enrichment over commutative monoids.

PROOF. Proving that an arbitrary unions of simulations in $\mathbf{Int}(w_1, w_2)$ is still a simulation in $\mathbf{Int}(w_1, w_2)$ is trivial; as well as showing that the empty relation is always a simulation.

It thus only remains to show that composition commutes with unions, on the right and on the left. Since this is true in **Rel**, it is also true in **Int**!

□

1.2 Notation

Let's recall some traditional notation.

- An element of the indexed cartesian product $\prod_{a \in A} D(a)$ is given by a function f taking any $a \in A$ to an $f(a)$ in $D(a)$. When the set $D(a)$ doesn't depend on a, it amounts to a function $f : A \to D$.
- An element of the indexed disjoint sum $\sum_{a \in A} D(a)$ is given by a pair (a, d) where $a \in A$ and $d \in D(a)$. When the set $D(a)$ doesn't depend on a, this is simply the cartesian product $A \times D$.
- We write $\mathsf{List}(S)$ for the set of "lists" over set S. A list is a tuple $(s_1, s_2, \ldots s_n)$ of elements of S. The empty list is written $()$.
- The collection $\mathcal{M}_f(S)$ of finite multisets over S is the quotient of $\mathsf{List}(S)$ by permutations. We write $[s_1, \ldots s_n]$ for the equivalence class containing $(s_1, \ldots s_n)$. We use "+" for the sum of multisets; it simply corresponds to concatenation on lists.

1.3 Constructions

We now define, at the level of interaction systems, the connectives of linear logic. With those, making **Int** into a denotational model of intuitionistic linear logic more or less amounts to showing that it is symmetric monoidal closed, has finite products and coproducts and has a well behaved comonad.

1.3.1 Constants.

A very simple, yet important interaction system is "I", the interaction system without interaction.

Definition 6 *Define I to be the interaction system on the singleton set $\{*\}$:*

$$\begin{array}{lcl} A_I(*) & = & \{*\} \\ D_I(*,*) & = & \{*\} \\ n_I(*,*,*) & = & \{*\} \ . \end{array}$$

This interaction systems is also called "skip*".*

This is the perfect example of "stable" or "constant" game: no knowledge is ever gained by any of the players. Since the interaction traces are ultimately constant, this is sometimes considered a terminating system: both sides have reached an agreement.

One last interaction system of (theoretical) interest is given by the interaction system on the empty set of states:

Definition 7 *Define* $\mathbf{0}$ *to be the unique interaction system on* $\emptyset$.

This interaction system is even more boring than I: there are no states! From a practical point of view, this system doesn't even exist. The following is trivial:

Lemma 8 *In* **Int**, *the object* $\mathbf{0}$ *is a zero object: it is both initial and terminal.*

1.3.2 Product and Coproduct

Since the forgetful functor $w \mapsto |w|$ has a right adjoint, we know it commutes with colimits. As a result, we know that if the coproduct of w_1 and w_2 exists in **Int**, its set of states is isomorphic to the coproduct of S_1 and S_2 in **Rel**. We thus define:

Definition 9 *Suppose* w_1 *and* w_2 *are interaction systems on* S_1 *and* S_2. *Define the interaction system* $w_1 \oplus w_2$ *on* $S_1 + S_2$ *as follows:*[2]

$$\begin{array}{lcl} A_{w_1 \oplus w_2}((i, s_i)) & = & A_i(s_i) \\ D_{w_1 \oplus w_2}((i, s_i), a) & = & D_i(s_i, a) \\ n_{w_1 \oplus w_2}((i, s_i), a, d) & = & (i, s_i[a/d]) \end{array}$$

In other words, the game $w_1 \oplus w_2$ is simply a disjoint sum of w_1 and w_2, and interaction takes place in only one of the games. Because we have no initial state, there is no need to specify who, among the Angel or the Demon, is making the choice of the game to use. With this in mind, lemma 11 isn't very surprising.

We have:

Lemma 10 *The operation* $\_ \oplus \_$ *is the coproduct in* **Int**.

[2] Recall that $A + B = \{1\} \times A \cup \{2\} \times B$.

PROOF. We just "lift" the constructions from the category **Rel**:

- injections: we put

$$\begin{aligned} i_1 \quad &: \quad \mathbf{Int}(w_1, w_1 \oplus w_2) \\ &= \quad \{(s_1, (1, s_1)) \ | \ s_1 \in S_1\} \end{aligned}$$

  and similarly for $i_2 \in \mathbf{Int}(w_2, w_1 \oplus w_2)$.
- copairing: suppose $r_1 \in \mathbf{Int}(w_1, w)$ and $r_2 \in \mathbf{Int}(w_2, w)$, define:

$$\begin{aligned} [r_1, r_2] \quad &: \quad \mathbf{Int}(w_1 \oplus w_2, w) \\ &= \quad \{((1, s_1), s)) \ | \ (s_1, s) \in r_1\} \\ & \quad \cup \{((2, s_2), s)) \ | \ (s_2, s) \in r_2\} \end{aligned}$$

Checking that those constructions yield simulations is direct.

Commutativity of the appropriate diagrams as well as universality can be lifted from **Rel**...

□

The next result is only surprising at first sight: the situation is similar in **Rel**. It will however to be quite important in the sequel since we cannot interpret the differential λ-calculus without it.

Lemma 11 *In* **Int**, $\oplus$ *is also the product.*

PROOF. This is a direct consequence of commutative monoid enrichment (proposition 5).[3]

□

When dealing with linear logic, we use the usual symbol & when it denotes a conjunction...

[3] This is well known for abelian categories (see [11, chap. 8]), but the existence of inverses is irrelevant.

1.3.3 Synchronous Product.

There is an obvious tensor construction reminiscent of the synchronous product found in SCCS (synchronous calculus of communicating systems, [12]):

Definition 12 *Suppose w_1 and w_2 are interaction systems on S_1 and S_2. Define the interaction system $w_1 \otimes w_2$ on $S_1 \times S_2$ as follows:*

$$\begin{array}{lcl} A_{w_1 \otimes w_2}((s_1,s_2)) & = & A_1(s_1) \times A_2(s_2) \\ D_{w_1 \otimes w_2}((s_1,s_2),(a_1,a_2)) & = & D_1(s_1,a_1) \times D_2(s_2,a_2) \\ n_{w_1 \otimes w_2}((s_1,s_2),(a_1,a_2),(d_1,d_2)) & = & (s_1[a_1/d_1], s_2[a_2/d_2]) \ . \end{array}$$

This is a kind of *lock-step synchronous parallel composition* of w_1 and w_2: the Angel and the Demon exchange pairs of actions/reactions. In terms of games, the players simply play two games in parallel *at the same pace.*

For any sensible notion of morphism, I should be a neutral element for this product. It is indeed the case, for the following reason: the components of $w \otimes I$ and w are isomorphic by dropping the second (trivial) coordinate:

$$\begin{array}{lclcl} w \otimes I & & & & w \\ S \times \{*\} & & & \simeq & S \\ A((s,*)) & = & A(s) \times \{*\} & \simeq & A(s) \\ D((s,*),(a,*)) & = & D(s,a) \times \{*\} & \simeq & D(s,a) \\ n((s,*),(a,*),(d,*)) & = & (s[a/d],*) & \simeq & s[a/d] \end{array}$$

This implies trivially that $\{((s,*),s) \mid s \in S\}$ is an isomorphism. For similar reasons, this product is transitive and commutative.

Lemma 13 *$\_ \otimes \_$ is a commutative tensor product in the category* **Int**. *Its action on morphisms is given by:*

$$((s_1,s_1'),(s_2,s_2')) \in r \otimes r' \quad \Leftrightarrow \quad \left\{ \begin{array}{r} (s_1,s_2) \in r \\ \textit{and } (s_1',s_2') \in r' \end{array} \right.$$

Checking that $r \otimes r'$ is a simulation is easy.

The notion of "componentwise" isomorphism is too fine for most purposes and the notion of isomorphism inherited from simulations is coarser. In particular, it is easy to find examples of isomorphic interaction systems in **Int** where the actions/reactions sets are not isomorphic. For example, if w is an interaction system, let ε_s be a choice function from $\mathcal{P}^*(A(s))$, the collection of non-empty subsets of $A(s)$ to $A(s)$; and define $\widehat{w}$ with

- $\widehat{A}(s) = \mathcal{P}^*(A(s))$;
- $\widehat{D}(s,U) = D(s,\varepsilon_s(U))$;
- $\widehat{n}(s,U,d) = n(s,\varepsilon_s(U),d)$.

The systems w and $\widehat{w}$ are isomorphic, but the sets $A(s)$ and $\widehat{A}(s)$ have different cardinalities.

1.3.4 Linear Arrow.

Any category with a zero object cannot be cartesian closed. We thus cannot hope to model the simply typed λ-calculus inside **Int**. One of the points of linear logic is precisely to give logical status to a simpler kind of structure: linear implication. We do not require our denotational model to be cartesian closed but only symmetric monoidal closed w.r.t. to a tensor which is generally not the cartesian product. This is the case for **Int**, but the definition of the functor $\_ \multimap \_$ is a little more involved:

Definition 14 *If w_1 and w_2 are interaction systems on S_1 and S_2, define the interaction system $w_1 \multimap w_2$ on $S_1 \times S_2$ as follows:*

$$\begin{aligned}
A_{\multimap}((s_1,s_2)) &= \sum_{f \in A_1(s_1) \to A_2(S_2)} \ \prod_{a_1 \in A_1(s_1)} D_2(s_2,f(a_1)) \to D_1(s_1,a_1) \\
D_{\multimap}((s_1,s_2),(f,G)) &= \sum_{a_1 \in A_1(s_1)} D_2(s_2,f(a_1)) \\
n_{\multimap}((s_1,s_2),(f,G),(a_1,d_2)) &= (s_1[a_1/G_{a_1}(d_2)]\,,\ s_2[f(a_1)/d_2])\ .
\end{aligned}$$

It may seem difficult to get some intuition about this interaction system; but it is *a posteriori* quite natural. Let's unfold this definition with simulations in mind:

- An action in state (s_1,s_2) is given by a pair consisting of:
  (1) a function f (the index for the element of the disjoint sum) translating actions from s_1 into actions from s_2;
  (2) for any action a_1, a function G_{a_1} translating reactions to $f(a_1)$ into reactions to a_1.
- A reaction to such a "one step translating mechanism" is given by:
  (2) an action a_1 in $A_1(s_1)$ (which we want to simulate);

(1) and a reaction d_2 in $D_2(s_2, f(a_1))$ (which we want to translate back).

- Given such a reaction, we can simulate a_1 by $a_2 = f(a_1) \in A_2(s_2)$; and translate back d_2 into $d_1 = G_{a_1}(d_2) \in D_1(s_1, a_1)$. The next state is just the pair of states $s_1[a_1/d_1]$ and $s_2[a_2/d_2]$.

In essence, the Angel translates what the Demon gives her.

This connective is indeed a "linear arrow":

Proposition 15 *In* **Int**, *$\_\otimes\_$ is left adjoint to $\_\multimap\_$: there is an isomorphism*

$$\mathbf{Int}(w_1 \otimes w_2\,,\, w_3) \quad \simeq \quad \mathbf{Int}(w_1\,,\, w_2 \multimap w_3)\ ,$$

natural in w_1, w_2 and w_3

PROOF. The proof is not really difficult. First notice that the axiom of choice can be written as

$$\begin{aligned}\mathsf{AC}: \quad & (\forall a \in A)(\exists d \in D(a))\ \varphi(a,d)\\ & \Leftrightarrow\\ & (\exists f \in \textstyle\prod_{a\in A} D(a))(\forall a \in A)\ \varphi(a, f(a))\ .\end{aligned}$$

When the domain $D(a)$ for the existential quantifier doesn't depend on $a \in A$, we can simplify it into:

$$\begin{aligned}\mathsf{AC}: \quad & (\forall a \in A)(\exists d \in D)\ \varphi(a,d)\\ & \Leftrightarrow\\ & (\exists f \in A \to D)(\forall a \in A)\ \varphi(a, f(a))\ .\end{aligned}$$

We will use AC to shuffle quantifiers and complexify the domains of quantification. This will transform the condition defining a simulation from $w_1 \otimes w_2$ to w_3 into the condition defining a simulation from w_1 to $w_2 \multimap w_3$.

In the sequel, the part of the formula being manipulated will be written in bold. That r is a simulation from $w_1 \otimes w_2$ to w_3 takes the form[4]

$$\begin{aligned}(s_1,s_2,s_3)\in r \Rightarrow\ & (\forall a_1 \in A_1(s_1))\boldsymbol{(\forall a_2 \in A_2(s_2))}\\ & \boldsymbol{(\exists a_3 \in A_3(s_3))}\\ & (\forall d_3 \in D_3(s_3,\boldsymbol{a_3}))\\ & (\exists d_1 \in D_1(s_1,a_1))(\exists d_2 \in D_2(s_2,a_2))\\ & \quad (s_1[a_1/d_1], s_2[a_2/d_2], s_3[\boldsymbol{a_3}/d_3]) \in r\ .\end{aligned}$$

Using AC on $\forall a_2 \exists a_3$, we obtain:

$$\begin{aligned}(s_1,s_2,s_3)\in r \Rightarrow\ & (\forall a_1 \in A_1(s_1))\\ & (\exists f \in A_2(s_2) \to A_3(s_3))\\ & (\forall a_2 \in A_2(s_2))\boldsymbol{(\forall d_3 \in D_3(s_3,f(a_2)))}\\ & (\exists d_1 \in D_1(s_1,a_1))\boldsymbol{(\exists d_2 \in D_2(s_2,a_2))}\\ & \quad (s_1[a_1/d_1], s_2[a_2/\boldsymbol{d_2}], s_3[f(a_2)/d_3]) \in r\ .\end{aligned}$$

We can now apply AC on $\forall d_3 \exists d_2$:

$$\begin{aligned}(s_1,s_2,s_3)\in r \Rightarrow\ & (\forall a_1 \in A_1(s_1))\\ & (\exists f \in A_2(s_2) \to A_3(s_3))\\ & \boldsymbol{(\forall a_2 \in A_2(s_2))}\\ & \boldsymbol{(\exists g \in D_3(s_3,f(a_2)) \to D_2(s_2,a_2))}\\ & (\forall d_3 \in D_3(s_3,f(a_2)))\\ & (\exists d_1 \in D_1(s_1,d_1))\\ & \quad (s_1[a_1/d_1], s_2[a_2/\boldsymbol{g}(d_3)], s_3[f(a_2)/d_3]) \in r\end{aligned}$$

[4] modulo associativity $(S_1 \times S_2) \times S_3 \simeq S_1 \times (S_2 \times S_3) \simeq S_1 \times S_2 \times S_3$.

and apply AC one more time on $\forall a_2 \exists g$ to obtain:

$$\begin{aligned}
(s_1, s_2, s_3) \in r \Rightarrow\ & (\forall a_1 \in A_1(s_1)) \\
& (\exists f \in A_2(s_2) \to A_3(s_3)) \\
& (\exists G \in \textstyle\prod_{a_2 \in A_2(s_2)} D_3(s_3, f(a_2)) \to D_2(s_2, a_2)) \\
& (\forall a_2 \in A_2(s_2))(\forall d_3 \in D_3(s_3, f(a_2))) \\
& (\exists d_1 \in D_1(s_1, d_1)) \\
& \quad (s_1[a_1/d_1], s_2[a_2/G_{a_2}(d_3)], s_3[f(a_2)/d_3]) \in r
\end{aligned}$$

which is equivalent to

$$\begin{aligned}
(s_1, s_2, s_3) \in r \Rightarrow\ & (\forall a_1 \in A_1(s_1)) \\
& \left(\exists (f, G) \in \begin{array}{l} \sum_{f \in A_2(s_2) \to A_3(s_3)} \\ \prod_{a_2 \in A_2(s_2)} D_3(s_3, f(a_2)) \to D_2(s_2, a_2) \end{array} \right) \\
& (\forall (a_2, d_3) \in \textstyle\sum_{A_2(s_2)} D_3(s_3, f(a_2))) \\
& (\exists d_1 \in D_1(s_1, d_1)) \\
& \quad (s_1[a_1/d_1], s_2[a_2/G_{a_2}(d_3)], s_3[f(a_2)/d_3]) \in r\ .
\end{aligned}$$

By definition, this means that r is a simulation from w_1 to $w_2 \multimap w_3$.

Naturality is trivial: it corresponds to naturality of associativity in **Rel**. $\Box$

In particular, proposition 15 implies that

$$\mathbf{Int}(w_1, w_2) \quad \simeq \quad \mathbf{Int}(I, w_1 \multimap w_2)\ .$$

We call a simulation from I to w a *safety property* for w.

Lemma 16 (with Def.) *A subset $x \subseteq S$ is a simulation from I to w iff*[5]

$$s \in x \quad \Rightarrow \quad (\exists a \in A(s))(\forall d \in D(s, a))\ s[a/d] \in x\ .$$

We write $\mathcal{S}(w)$ for the collection of such subsets, and we call such an x a safety property for w.

[5] This is well defined since $\mathcal{P}(\{*\} \times S) \simeq \mathcal{P}(S)$.

Finally, we have

$$\mathbf{Int}(w_1, w_2) \quad = \quad \mathcal{S}(w_1 \multimap w_2) \ .$$

The analogy with traditional notions of morphisms as strategies is rather subtle. A safety property x satisfies the property

> *if interaction is started from a state in x, then the Angel has a move that guarantees that the next state will also be in x (provided the Demon does answer).*

This is a safety property in the sense that it guarantees that "nothing bad happens". (As opposed to liveness properties, which ensure that "something good happens"...) In particular, this means that the Angel has an infinite strategy from any state in x: she can always find a move to play. The choice of those moves is irrelevant to the notion of safety property: we only know they exist. In particular, such a move needs not be unique.

As special case, let's look at the definition of linear negation. The orthogonal $w^\perp$ of w is defined as usual as the interaction system $w \multimap \perp$. For intuitionistic linear logic, any object can formally be used as $\perp$, but anticipating on proposition 33, we use $\perp = I$. We have:

$$\begin{aligned}
A^\perp((s,*)) \quad &= \quad \sum_{f \in A(s) \to \{*\}} \ \prod_{a \in A(s)} \{*\} \to D(s,a) \\
D^\perp((s,*),(f,G)) \quad &= \quad \sum_{a \in A(s)} \{*\} \\
n^\perp((s,*),(f,G),(a,*)) \quad &= \quad (s[a/G_{a_1}(*)],*)
\end{aligned}$$

which, after simplification, is equivalent to

$$\begin{aligned}
A^\perp(s) \quad &= \quad \prod_{a \in A(s)} D(s,a) \\
D^\perp(s,f) \quad &= \quad A(s) \\
n^\perp(s,f,a) \quad &= \quad s[a/f(a)] \ .
\end{aligned}$$

One important point to notice is that with this definition, the set of states of $w^\perp$ is the same as the set of states of w. In particular, the canonical morphism in $\mathbf{Int}(w, w^{\perp\perp})$ will be given by the equality relation.

The definition looks complex but can be interpreted in a very traditional way: *negation interchanges the two players.* In our context, it is not possible to simply interchange actions and reactions since reactions *depend* on a particular action. We have to do the following:

- an action in $A^{\perp}(s)$ is a *conditional* reaction; or a *one move strategy* to react to any action;
- a reaction in $D^{\perp}(s, f)$ is just an action in w;
- the new state after "action" f and "reaction" a is just the state obtained after playing a followed by $f(a)$.

The "polarities" of moves and coutermoves is interchanged, and it does swap the players in the sense that it transforms Angel strategies into Demon strategies and vice and versa. (See [10] for more details.)

The dual $w^{\perp}$ enjoys a surprising property: the set of possible reactions to a particular action *doesn't depend on the particular action!*

1.3.5 Multithreading.

We now come to the last connective needed to interpret intuitionistic linear logic. Its computational interpretation is related to the notion of *multithreading, i.e.* the possibility to run several instances of a program in parallel. In our case, it corresponds to the ability to play several synchronous instances of the same game in parallel. Let's start by defining synchronous multithreading in the most obvious way:

Definition 17 *If w is an interaction system on S, define $\text{Ł}(w)$, the multithreaded version of w to be the interaction system on $\mathsf{List}(S)$ with:*

$$\begin{aligned}
\text{Ł}.A((s_1, \dots, s_n)) &= A(s_1) \times \ldots \times A(s_n) \\
\text{Ł}.D((s_1, \dots, s_n), (a_1, \dots, a_n)) &= D(s_1, a_1) \times \ldots \times D(s_n, d_n) \\
\text{Ł}.n((s_1, \dots, s_n), (a_1, \dots, a_n), (d_1, \dots, d_n)) &= (s_1[a_1/d_1], \dots, s_n[a_n/d_n]) \ .
\end{aligned}$$

This interaction system is just the sum of all "n-ary" versions of the synchronous product.

Lemma 18 *The operator $w \mapsto \text{Ł}(w)$ can be extended to a functor from* **Int** *to* **Int***.*

To get the abstract properties we want, we need to quotient multithreading by permutations. Just like multisets are lists modulo permutations, so is $\mathbf{!}w$ the multithreaded $\text{Ł}(w)$ modulo permutations. This definition is possible because Ł(w) is "compatible" with permutations: if σ is a permutation, we have

$$\begin{aligned}
&\sigma \cdot ((s_1, \dots s_n)[(a_1, \dots a_n)/(d_1, \dots d_n)]) \\
&\qquad = \\
&(\sigma \cdot (s_1, \dots s_n))[\sigma \cdot (a_1, \dots a_n)/\sigma \cdot (d_1, \dots d_n)] \ .
\end{aligned}$$

The final definition is:

Definition 19 *If w is an interaction system on S, define $\mathbf{!}w$ to be the following interaction system on $\mathcal{M}_f(S)$:*

$$\begin{aligned}
\mathbf{!}A(\mu) &= \textstyle\sum_{\overline{s}\in\mu} Ł.A(\overline{s}) \\
\mathbf{!}D(\mu, (\overline{s}, \overline{a})) &= Ł.D(\overline{s}, \overline{a}) \\
\mathbf{!}n(\mu, (\overline{s}, \overline{a}), \overline{d}) &= [Ł.n(\overline{s}, \overline{a}, \overline{d})] \ .
\end{aligned}$$

(Note that as an element of μ, $\overline{s}$ is just a specific order for the element of μ.)

Unfolded, it gives:

- an action in state μ is given by an element $\overline{s}$ (a list) of μ (a multiset, *i.e.* an equivalence class) together with an element $\overline{a}$ in Ł$.A(\overline{s})$ (a list of actions);
- a reaction is given by a list of reactions $\overline{d}$ in Ł$.D(\overline{s}, \overline{a})$;
- the next state is the equivalence class containing the list $\overline{s}[\overline{a}/\overline{d}]$ (the orbit of $\overline{s}[\overline{a}/\overline{d}]$ under the action of the group of permutations).

Lemma 20 *This operation $w \mapsto \mathbf{!}w$ can be extended to a functor from* **Int** *to* **Int**. *Moreover, we have the following bisimulation (which is not an isomorphism)*

$$Ł(w) \quad \underset{p}{\overset{\sigma}{\rightleftarrows}} \quad \mathbf{!}w \tag{1}$$

where σ is just membership of a list in a multiset (equivalence class of lists) and p its converse.

This operation enjoys a very strong algebraic property:

Proposition 21 *$\mathbf{!}w$ is the free $\otimes$-comonoid generated by w.*

Note that because $\mathbf{!}w$ "is" $\bigoplus_{n\geq 0} w^{\otimes n}/\mathfrak{S}_n$, this is not very surprising.

PROOF. Quite a lot can be deduced from the same property of **Rel**, but we will look at some details.

Let's start by looking at the $\otimes$-comonoid structure of $\mathbf{!}w$: the counit and comultiplication are given by

$$\begin{array}{rcl} e & \in & \mathbf{Int}(\mathbf{!}w, I) \\ & = & \{([\,], *)\} \end{array} \quad \text{and} \quad \begin{array}{rcl} m & \in & \mathbf{Int}(\mathbf{!}w, \mathbf{!}w \otimes \mathbf{!}w) \\ & = & \{(\mu + \nu, (\mu, \nu)) \mid \mu, \nu \in \mathcal{M}_f(S)\} \end{array}$$

We need to show that for any interaction system w and $\otimes$-comonoid w_c, there is a natural isomorphism

$$\mathbf{CoMon}(\mathbf{Int}, \otimes)(w_c, \mathbf{!}w) \quad \simeq \quad \mathbf{Int}(w_c, w)\ .$$

Going from left to right is easy:

$$\begin{array}{lcl} \mathbf{CoMon}(\mathbf{Int}, \otimes)(w_c, \mathbf{!}w) & \to & \mathbf{Int}(w_c, w) \\ r & \mapsto & \{(s_c, s) \mid (s_c, [s]) \in r\}\ . \end{array}$$

Checking that this operation is well-defined (it sends a comonoid morphism to a simulation) is direct.

The other direction is more interesting. Let w_c be a commutative comonoid. This means we are given $e_c \in \mathbf{Int}(w_c, I)$ and $m_c \in \mathbf{Int}(w_c, w_c \otimes w_c)$, satisfying additional commutativity and associativity conditions.

Suppose r is a simulation from w_c to w. This is a relation with no condition about the comonoid structure of w_c. We construct a relation from w_c to $\mathbf{!}w$ in the following way:

- we start by extending comultiplication m_c to $\overline{m}_c : \mathbf{Int}(w_c, Ł(w_c))$;
- we then compose that with $Ł(r) : \mathbf{Int}(Ł(w_c), Ł(w))$;
- and finally compose that with $\sigma : \mathbf{Int}(Ł(w), \mathbf{!}w)$, see (1) in lemma 20.

We then check that this simulation respects the comonoid structures of w_c and $\mathbf{!}w$.

Define $\overline{m}_c \subseteq S_c \times \mathsf{List}(S_c)$ by the following clauses: (inductive definition)

$$\begin{array}{lll}
(s,()) \in \overline{m}_c & \text{iff} & s \in e_c \\
(s,s') \in \overline{m}_c & \text{iff} & s = s' \\
(s,(s_1,\dots,s_n)) \in \overline{m}_c & \text{iff} & (s,(s_1,s')) \in m_c \wedge (s',(s_2,\dots,s_n)) \in \overline{m}_c \\
 & & \text{for some } s' \in S_c\ .
\end{array}$$

Using the fact that e_c and m_c are simulations, we can easily show (by induction) that $\overline{m}_c$ is a simulation from w_c to $\text{Ł}(w_c)$.

Moreover, we have:

$$\begin{array}{l}
(s_c,(s_{c,1},\dots,s_{c,n+m})) \in \overline{m_c} \\
\quad\Leftrightarrow \\
(\exists s_c^1, s_c^2 \in S_c)\ (s_c,(s_c^1,s_c^2)) \in m_c \wedge (s_c^1,(s_{c,1},\dots,s_{c,n})) \in \overline{m}_c \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge\, (s_c^2,(s_{c,n+1},\dots,s_{c,n+m})) \in \overline{m}_c
\end{array} \tag{2}$$

by transitivity and

$$\begin{array}{l}
(s_c,(s_{c,1},\dots,s_{c,i},s_{c,i+1},\dots,s_{c,n})) \in \overline{m_c} \\
\quad\Leftrightarrow \\
(s_c,(s_{c,1},\dots,s_{c,i+1},s_{c,i},\dots,s_{c,n})) \in \overline{m_c}
\end{array} \tag{3}$$

by commutativity.

We know that $\widetilde{r} = \sigma \cdot \text{Ł}(r) \cdot \overline{m}_c$ is a simulation from w_c to $\mathbf{!}w$. We need to check that this simulation respects the comonoid structures of w_c and $\mathbf{!}w$, *i.e.* that both

$$\begin{tikzcd}
w_c \arrow[r, "\widetilde{r}"] \arrow[dr, "e_c"'] & \mathbf{!}w \arrow[d, "e"] \\
 & \mathbf{1}
\end{tikzcd} \quad \text{and} \quad \begin{tikzcd}
w_c \arrow[r, "c"] \arrow[d, "\widetilde{r}"'] & w_c \otimes w_c \arrow[d, "\widetilde{r} \otimes \widetilde{r}"] \\
\mathbf{!}w \arrow[r, "m_c"'] & \mathbf{!}w \otimes \mathbf{!}w
\end{tikzcd}$$

are commutative. The first diagram is easily shown to be commutative. For the second one: suppose $(s_c,[s_1,\dots,s_n],[s_{n+1},\dots,s_{n+m}]) \in m{\cdot}\widetilde{r}$. This is equivalent to saying that there are $s_{c,1},\dots,s_{c,n+m}$ in S_c s.t.

- $(s_{c,i},s_i) \in r$ for all $i = 1,\dots,n+m$

- and $(s_c, (s_{c,1}, \dots, s_{c,n+m})) \in \overline{m}_c$.

That $(s_c, [s_1, \dots, s_n], [s_{n+1}, \dots, s_{n+m}])$ is in $\widetilde{r} \otimes \widetilde{r} \cdot c$ means that there are s_c^1 and s_c^2 in S_c s.t.

- $(s_c, (s_c^1, s_c^2)) \in m_c$
- and $(s_c^1, [s_1, \dots, s_n]) \in \widetilde{r}$ and $(s_c^2, [s_{n+1}, \dots, s_{n+m}]) \in \widetilde{r}$,

i.e. there are s_c^1 and s_c^2 in S_c, and $s_{c,1}, \dots, s_{c,n}, s_{c,n+1}, \dots, s_{c,n+m}$ in S_c s.t.

- $(s_c, (s_c^1, s_c^2)) \in m_c$
- $(s_c^1, (s_{c,1}, \dots, s_{c,n})) \in \overline{m}_c$
- $(s_c^2, (s_{c,n+1}, \dots, s_{c,n+m})) \in \overline{m}_c$
- and $(s_i, s_{c,i}) \in r$ for all $i = 1, \dots, n+m$.

By using (2) and (3), it is trivial to show that the two conditions are equivalent. This proves that the second diagram is commutative.

It only remains to show that the two operations defined are inverse of each other. This is not difficult. $\square$

2 Interpreting Linear Logic

We now have all the necessary ingredients to construct a denotational model for intuitionistic linear logic. The details of categorical models for linear logic are quite intricate, and there are several notions, not all of which are equivalent. We refer to the survey [13] and the references given there.

In the case of **Int**, the situation is however quite simple: proposition 21 makes **Int** into a “Lafont category” (see [14]).

Corollary 22 *With the construction defined in the previous sections,* **Int** *is a Lafont category. In particular, “$!\_$” is a comonad; and we have for any w_1 and w_2, we have the following natural isomorphism:*

$$!(w_1 \mathbin{\&} w_2) \quad \simeq \quad !w_1 \otimes !w_2 \ .$$

Recall that since the product and coproduct coincide, & is the same as $\oplus$.

A direct proof of the fundamental isomorphism is easy: there is a “componentwise” isomorphism between the interaction systems $!(w_1 \mathbin{\&} w_2)$ and $!w_1 \otimes !w_2$.

Lafont categories were used to give a semantics to linear logic and were latter subsumed by Seely categories and linear categories.

It is thus possible to give a semantics to formulas and proofs in the usual way. We write F for the interpretation of a formula F (no confusion arises) and $[\![\pi]\!]$ for the interpretation of a proof π.

Proposition 23 *For all proof π_1 and π_2 of the same sequent, if π_1 and π_2 have the same cut-free normal form, then $[\![\pi_1]\!] = [\![\pi_2]\!]$.*

Moreover, since the interpretation is done using canonical morphisms, which are just lifting of the same morphisms in **Rel**, the interpretation of a proof is the same as its relational interpretation:[6]

Proposition 24 *For any proof π of a sequent $\Gamma \vdash F$, the relational interpretation $[\![\pi]\!]$ of π is a simulation from $\bigotimes \Gamma$ to F. (This holds for any valuation of the propositional variables.)*

The presence of propositional variables is crucial because without them, the model becomes trivial:

Proposition 25 *Suppose F is a formula without propositional variables; then its interpretation is trivial : any subset of its set of states is a safety property.*

More precisely, we have

- $A_F(s) = \{*\}$ *(singleton set) ;*
- $D_F(s, *) = \{*\}$ *(singleton set) ;*
- $n_F(s, *, *) = s$.

The proof is a trivial induction on the formula.

This model is thus only appropriate when interpreting Π^1_1 logic, *i.e.* propositional linear logic. There, it has a real discriminating power. The model can even be extended to deal with full second order, with the usual proviso: the interpretation may decrease (in some very special cases) during elimination of a second order cut. For more details, see [10, chap. 8].[7]

[6] the relational interpretation is folklore, at least in Marseille and Paris, but it is surprisingly difficult to find an early reference to it. For those who want to see the concrete definition of the interpretation, we refer to of [15, app. 4].

[7] There, the equivalent notion of predicate transformers rather than interaction systems is used, but as we will show in section 6.1, the two categories are equivalent.

3 Interpreting the Differential λ-calculus

So far, proposition 5 hasn't been used, except to deduce the existence of a product. The problem is that this proposition doesn't reflect a property of proofs. The reason is that

- not every formula has a proof;
- we do not see *a priori* how to sum proofs of a single formula.

Ehrhard and Regnier's *differential λ-calculus* ([1]) extends the λ-calculus by adding a notion of differentiation of λ-terms. One consequence is that we need a notion of sum of terms, interpreted as a non-deterministic choice. It is also possible to only add sums (and coefficients) to the usual λ-calculus as in [16].

It is not the right place to go into the details of the differential λ-calculus and we refer to [1] for motivations and a complete description. A complete definition can also be found in the Appendix on page 36.

In the typed case, we have the following typing rules:

$$(1)\quad \frac{}{\Gamma \vdash 0 : \tau} \quad \text{and} \quad \frac{\Gamma \vdash t : \tau \qquad \Gamma \vdash u : \tau}{\Gamma \vdash t + u : \tau}\ ;$$

$$(2)\quad \frac{\Gamma \vdash t : \tau \to \sigma \qquad \Gamma \vdash u : \tau}{\Gamma \vdash \mathrm{D}\, t \cdot u : \tau \to \sigma}\ .$$

The intuitive meaning is that "$\mathrm{D}\, t \cdot u$" is the result of (non-deterministically) replacing *exactly one occurrence* of the first variable of t by u. We thus obtain a sum of terms, depending on which occurrence was replaced. This gives a notion of differential substitution (or linear substitution) which yields a *differential-reduction.* The rules governing this reduction are more complex than usual β-reduction rules; we refer to [1] or the Appendix.

Besides the natural commutativity and associativity of addition, differential λ-terms are also quotiented modulo the following equivalence relations:

- $0 = (0)u = \lambda x.0 = \mathrm{D}\,0 \cdot t = \mathrm{D}\, t \cdot 0$;
- $(t_1 + t_2)\, u = (t_1)u \ + \ (t_2)u$;
- $\lambda x.(t_1 + t_2) = \lambda x.t_1 \ + \ \lambda x.t_2$;
- $\mathrm{D}(t_1 + t_2) \cdot u = \mathrm{D}\, t_1 \cdot u \ + \ \mathrm{D}\, t_2 \cdot u$;
- $\mathrm{D}\, t \cdot (u_1 + u_2) = \mathrm{D}\, t \cdot u_1 \ + \ \mathrm{D}\, t \cdot u_2$;
- $\mathrm{D}(\mathrm{D}\, t \cdot u) \cdot v = \mathrm{D}(\mathrm{D}\, t \cdot v) \cdot u$.

The last one is probably the most important one as it allows to link the notion of differentiation to the traditional, analytic notion of differentiation. Note that even if the first five rules can be oriented from left to right, a quotient is inevitable because of the sixth rule and the commutativity of addition. This quotient is natural because none of those rules carries any computational content.

Interpreting usual (without "D", "+" nor "0") λ-terms can be done using the well-known translation of the simply typed λ-calculus into intuitionistic linear logic with propositional variables. Just replace an atomic type by a propositional variable and the type $\tau \to \sigma$ by $!\tau \multimap \sigma$ inductively. That the resulting interpretation is sound follows directly from the fact that **Int** is a Lafont category. (In any model for linear logic, the co-Kleisli category of $!\_$ is cartesian closed.)

The general notion of categorical model for the differential λ-calculus (or differential proof nets, or "differential linear logic") is only beginning to emerge. The main paper on the subject is [17], where the categorical notion of "differentiation combinator" is studied in details. No real soundness theorem is however proved there because the authors work in a more general setting: the base category is not necessarily monoidal closed, *i.e.* the co-Kleisli category is not necessarily cartesian closed.

The notion of differential category is well-suited for our purposes, and we will show that the category **Int** is indeed a differential category. Together with the fact that **Int** is a Lafont category, it allows to deduce that we do get a categorical model for the differential λ-calculus.

Definition 26 *If $\mathcal{C}$ is a symmetric monoidal category with a coalgebra modality $!\_$, we call a natural transformation $d_X : X \otimes !X \to !X$ a* deriving transformation *in case the following hold:*

$$\begin{aligned}
d_X\, e_X &= 0\\
d_X\, \Delta &= (1 \otimes \Delta)\, (d_X \otimes 1) \;+\; (1 \otimes \Delta)\, (c \otimes 1)\, (1 \otimes d_X)\\
d_X\, \epsilon_X &= (1 \otimes e)\, u_X\\
d_X\, \delta &= (1 \otimes \Delta)\, (d_X \otimes \delta) d_{!X}
\end{aligned}$$

where

- *$e_X : !X \to I$ and $\Delta_X : !X \to !X \otimes !X$ are the operations of the coalgebra $!X$;*

- *$c : A \otimes B \to B \otimes A$ is the symmetry and $u_X : X \otimes I \to X$ is the unit;*

- $\delta_X : !X \to !!X$ *and* $\epsilon_X : !X \to X$ *come from the usual comonad laws.* [8]

In our case, we can lift d_X from the relational model:

Lemma 27 (and definition) *For any interaction system w on S, the relation d_w defined by*

$$d_w \quad = \quad \Big\{\Big((s_0,[s_1,\dots,s_n]),[s_0,s_1,\dots,s_n]\Big) \mid s_0,\dots,s_n \in S\Big\}$$

is a deriving transformation.

PROOF. Because it is already shown in [17] that this relation is indeed a deriving transformation in **Rel**, is suffices to show that d_w is a simulation from $w \otimes !w$ to $!w$. This is immediate.

□

We can now extend the interpretation to differential λ-terms:

- the additive structure (0 and +) is directly interpreted by the monoid structure ($\emptyset$ and $\cup$);
- the differential structure is interpreted in the only sensible way: suppose we have $\Gamma \vdash t : \tau \to \sigma$ and $\Gamma \vdash u : \tau$; by induction, we have $[\![t]\!] \in \mathbf{Int}(!\Gamma, !\tau \multimap \sigma)$ and $[\![u]\!] \in \mathbf{Int}(!\Gamma, \tau)$. We can define a morphism in $\mathbf{Int}(!\tau \multimap \sigma, (\tau \otimes !\tau) \multimap \sigma)$:

$$\begin{array}{lll} & 1 & \in \ \mathbf{Int}(!\tau \multimap \sigma \ , \ !\tau \multimap \sigma) \\ \Leftrightarrow & \widehat{1} & \in \ \mathbf{Int}((!\tau \multimap \sigma) \otimes !\tau \ , \ \sigma) \\ \Rightarrow & \widehat{1}\,(1 \otimes d_\tau) & \in \ \mathbf{Int}((!\tau \multimap \sigma) \otimes \tau \otimes !\tau \ , \ \sigma) \\ \Leftrightarrow & \widehat{1}\,\widehat{(1 \otimes d_\tau)} & \in \ \mathbf{Int}(!\tau \multimap \sigma \ , \ (\tau \otimes !\tau) \multimap \sigma) \end{array}$$

  We write D for this morphism. This is an internal version of the differential combinator from [17, def. 2.3]. From this, we get

$$\begin{array}{lll} & \mathrm{D}\,[\![t]\!] & \in \ \mathbf{Int}(!\Gamma \ , \ (\tau \otimes !\tau) \multimap \sigma) \\ \Leftrightarrow & \widetilde{\mathrm{D}[\![t]\!]} & \in \ \mathbf{Int}(!\Gamma \otimes \tau \ , \ !\tau \multimap \sigma) \end{array}$$

[8] The applications of the isomorphism $\sigma : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$ are omitted for readability.

which we'll call $[\![\mathrm{D}\,t]\!]$. We can now compose interpretations as in:

$$\mathbf{!}\Gamma \xrightarrow{\Delta} \mathbf{!}\Gamma \otimes \mathbf{!}\Gamma \xrightarrow{1\otimes[\![u]\!]} \mathbf{!}\Gamma \otimes \tau \xrightarrow{[\![\mathrm{D}\,t]\!]} \mathbf{!}\tau \multimap \sigma$$

This morphism in $\mathbf{Int}(\mathbf{!}\Gamma\ ,\ \mathbf{!}\tau \multimap \sigma)$ is the interpretation of $\mathrm{D}\,t \cdot u$.

Spelled out concretely in the case of **Rel** or **Int**, the inductive definition looks like:

$$(\gamma, \mu, s') \in [\![\mathrm{D}\,t \cdot u]\!]$$
$$\Leftrightarrow$$
$$(\gamma_1, \mu + [s], s') \in [\![t]\!] \text{ for some } (\gamma_2, s) \in [\![u]\!] \text{ s.t. } \gamma = \gamma_1 + \gamma_2$$

That the interpretation is sound follows rather directly from the properties of a deriving transformation, see [17] for some of the missing details:

Lemma 28 *For all differential λ-terms and valuations γ, we have*

- $[\![0]\!] = [\![(0)u]\!] = [\![\lambda x.0]\!] = [\![\mathrm{D}\,0 \cdot t]\!] = [\![\mathrm{D}\,t \cdot 0]\!]$;
- $[\![(t_1 + t_2)\,u]\!] = [\![(t_1)u\ +\ (t_2)u]\!]$;
- $[\![\lambda x.(t_1 + t_2)]\!] = [\![\lambda x.t_1\ +\ \lambda x.t_2]\!]$;
- $[\![\mathrm{D}(t_1 + t_2) \cdot u]\!] = [\![\mathrm{D}\,t_1 \cdot u\ +\ \mathrm{D}\,t_2 \cdot u]\!]$;
- $[\![\mathrm{D}\,t \cdot (u_1 + u_2)]\!] = [\![\mathrm{D}\,t \cdot u_1\ +\ \mathrm{D}\,t \cdot u_2]\!]$;
- $[\![\mathrm{D}(\mathrm{D}\,t \cdot u) \cdot v]\!] = [\![\mathrm{D}(\mathrm{D}\,t \cdot v) \cdot u]\!]$.

We finally obtain the desired result:

Proposition 29 *Suppose that $\Gamma \vdash t : \sigma$ where Γ is a context and t a differential λ-term. The relation $[\![t]\!]$ is a simulation relation from $\mathbf{!}\Gamma$ to σ. Moreover, for all t and u we have:*

$$[\![(\lambda x.t)u]\!] = [\![t[u/x]]\!]$$
$$[\![\mathrm{D}(\lambda x.t) \cdot u]\!] = [\![\lambda x\ .\ (\partial t/\partial x) \cdot u]\!]$$

PROOF. That we obtain a simulation is true by construction.

Invariance under β-reduction follows from the correctness of the interpretation of λ-calculus in a cartesian-closed category.

Invariance under linear substitution seems to follow from general considerations about deriving transformation in linear categories, even if this is not treated in [17]. (In our case, a direct verification is possible, but long and tedious...)

$\square$

4 Untyped Calculus

Interpreting *untyped* λ-calculus remained an open question for quite a long time: since the cardinality of a function space is strictly bigger than the cardinality of the original set, it seemed difficult to get a model where any λ-term can be either an argument or a function. Dana Scott finally found a model by constructing a special object in the category of domains.

The solution is quite elegant: to interpret untyped λ-terms in a cartesian closed category, one "just" needs to find a reflexive object in a cartesian closed category, *i.e.* an object X with a retraction / projection pair $[X \to X] \lhd X$.

We have at our disposal a cartesian closed category: the Kleisli category over the comonad $\mathbf{!}\_$. Were we to find a reflexive object W in this category, we could model the *untyped differential* λ-calculus in the λ-model $\mathcal{S}(W)$. We now show how to construct such a reflexive object, in a fairly straightforward way.

We start with a non-trivial interaction system w on a set of states S (natural numbers for example) and then define an interaction system W, satisfying the equation $W \simeq w \oplus (\mathbf{!}W \multimap W)$ as follows:

(1) the set of states S_W is defined as the least fixpoint of $X \mapsto S + \mathcal{M}_f(X) \times X$; in a more "programming" fashion"

$$\begin{array}{lcll} S_W & = & \mathbf{data} & \mathsf{Leaf}(s \in S) \\ & & | & \mathsf{Node}\Big(\mu \in \mathcal{M}_f(S_W)\,,\, u \in S_W\Big) \end{array}$$

(2) the possible actions in a given state are defined by induction on the state: using "pattern matching", we have

$$\begin{array}{lcl} A_W(\mathsf{Leaf}(s)) & = & A(s) \\ A_W(\mathsf{Node}(\mu, u)) & = & (\mathbf{!}A_W \multimap A_W)((\mu, u)) \end{array}$$

(3) reactions are defined similarly as

$$\begin{array}{lcl} D_W(\mathsf{Leaf}(s), a) & = & D(s, a) \\ D_W(\mathsf{Node}(\mu, u), b) & = & (\mathbf{!}D_W \multimap D_W)((\mu, u), b) \end{array}$$

(4) and finally, the next state function is defined as

$$\begin{array}{lcl} n_W(\mathsf{Leaf}(s), a, d) & = & \mathsf{Leaf}(s[a/d]) \\ n_W(\mathsf{Node}(\mu, u), b, p) & = & (!n_W \multimap n_W)((\mu, u), b, p) \end{array}$$

Due to the presence of multisets, an actual implementation of W in a dependently typed functional programming language would be a little more complex: we would need to work with lists rather than multisets, and reason modulo shuffling concretely.

Lemma 30 *The relation r between $S + (\mathcal{M}_f(S_W) \times S_W)$ and S_W defined by*

$$\begin{array}{lcl} ((1, s)\,,\, \mathsf{Leaf}(s)) & \in & r \\ ((2, (\mu, u))\,,\, \mathsf{Node}(\mu, u)) & \in & r \end{array}$$

is an isomorphism (in **Int***) from $w \oplus (!W \multimap W)$ to W.*

The proof is direct. (This is an instance of a strong, "componentwise" isomorphism.)

Corollary 31 *In the $!\_$-Kleisli category of* **Int***, which is cartesian closed, there is a retract $W^W \lhd W$.*

PROOF. First, notice that it is sufficient to find a retract in the category **Int**: any morphism in a category can also be seen as a morphism in a Kleisli category (in a way which is compatible with composition in the Kleisli category).

There is the canonical injection i_2 from $!W \multimap W$ to $w \oplus (!W \multimap W)$. Now, in the category **Int**, we have that product and coproduct coincide; in particular, we have the projection π_2 from $w \,\&\, (!W \multimap W) = w \oplus (!W \multimap W)$ to $!W \multimap W$. Moreover, by definition, we have $\pi_2 \cdot i_2 = \mathsf{Id}_{!W \multimap W}$.

We can now prove that $r \cdot i_2 \;/\; \pi_2 \cdot r^{\sim}$ is a retraction / projection from $!W \multimap W$ to W:[9] it follows from the previous remark that $\pi_2 \cdot i_2 = \mathsf{Id}$ and that $r^{\sim}$ is the inverse of r. (Recall that r is an isomorphism.) $\square$

From there, constructing a model for the untyped λ-calculus is standard. We refer to [18]. We obtain in this way a model where each term is interpreted by a safety property for W.

[9] The converse $r^{\sim}$ of a relation is defined as $(s_2, s_1) \in R^{\sim}$ iff $(s_1, s_2) \in r$.

Since $\mathcal{S}(W)$ is a complete sup-lattice, we can also model sums, and by the very same construction defined in section 3, model differentiation. (Remark that the interpretation of a term is not really by induction on the type inference, but directly by induction on the term: we can thus apply it to untyped terms as well.)

The interpretation becomes: if t is a differential λ-term with its free variables among $x_1, \ldots, x_n$, we interpret t by a subset of $\mathcal{M}_f(S_W) \times \cdots \times \mathcal{M}_f(S_W) \times S_W$. In the sequel, γ is a tuple in $\mathcal{M}_f(S_W) \times \cdots \times \mathcal{M}_f(S_W)$ and we use $\gamma(x)$ for the projection on the appropriate coordinate.

- $[\![x]\!] = \{(\gamma, s)\}$ where $\gamma(x) = [s]$ and $\gamma(y) = []$ otherwise
- $[\![\lambda x.t]\!] = \{(\gamma, (\mu, s)) \mid (\gamma_{x:=\mu}, s) \in [\![t]\!]\}$
- $(\gamma, s) \in [\![(t)u]\!]$ iff $\left(\begin{array}{l} (\gamma_0, \mathsf{Node}(\mu, s)) \in [\![t]\!] \text{ for some } \mu = [s_1, \ldots, s_n] \\ \text{s.t. } (\gamma_i, s_i) \in [\![u]\!] \text{ for } i = 1, \ldots, n \\ \text{and } \gamma = \gamma_0 + \gamma_1 + \ldots + \gamma_n; \end{array} \right.$
- $[\![0]\!] = \emptyset$;
- $[\![t_1 + t_2]\!] = [\![t_1]\!] \cup [\![t_2]\!]$;
- $(\gamma, \mu, s') \in [\![\mathrm{D}\, t \cdot u]\!]$ iff $\left(\begin{array}{l} (\gamma_1, \mathsf{Node}(\mu + [s], s')) \in [\![t]\!] \\ \text{for some } (\gamma_2, s) \in [\![u]\!] \text{ s.t. } \gamma = \gamma_1 + \gamma_2. \end{array} \right.$

Proposition 32 *For any closed differential λ-term t, we have that $[\![t]\!]$ is a safety property for W.*

We have thus, in effect, constructed a non-trivial (in the sense that not *all* subsets of S_W are safety properties) denotational model for the untyped differential λ-calculus. This is particularly interesting because the original model for differential λ-calculus (finiteness spaces) did not have a reflexive object: they could not interpret fixpoint combinators (see [15]).

5 Classical Linear Logic

If one has in mind the definition of negation (see page 15), the next result can look quite surprising: interaction systems can interpret *classical* linear logic. In other words, for any interaction system w, we have $w \simeq w^{\perp\perp}$. The reason behind that is that our notion of morphism is *not* the notion of "component-wise" morphism. Even though the actions/reactions in $w^{\perp\perp}$ are very complex sets, the way they interact with states remains relatively simple.

The reason we haven't shown this result in section 2 is that the principle at work is highly non-constructive and that natural generalizations of interaction systems are unlikely to satisfy it.

Recall that any object can be used to represent $\perp$ in the intuitionistic case. However, in or case, the object I plays a very special role. For $\perp = I$, we have:

Proposition 33 *In* **Int**, *for any interaction system w, the identity relation is an isomorphism between w and $w^{\perp\perp}$.*

Equivalently, the object $\perp$ is dualizing in **Int**.

PROOF. The principle at stake in the proof is the contrapositive of the axiom of choice:

$$\begin{array}{ll} \mathsf{CtrAC}: & (\exists a \in A)(\forall d \in D(a))\ \varphi(a,d) \\ & \Leftrightarrow \\ & (\forall f \in \prod_{a \in A} D(a))(\exists a \in A)\ \varphi(a, f(a)) \end{array}$$

When the domain $D(a)$ for the universal quantifier doesn't depend on $a \in A$, we can simplify it into:

$$\begin{array}{ll} \mathsf{CtrAC}: & (\exists a \in A)(\forall d \in D)\ \varphi(a,d) \\ & \Leftrightarrow \\ & (\forall f \in A \rightarrow D)(\exists a \in A)\ \varphi(a, f(a)) \end{array}$$

Here are the components of $w^{\perp\perp}$:

$$\begin{array}{lcl} A^{\perp\perp}(s) & = & \left(\prod_{a \in A(s)} D(s,a) \right) \rightarrow A(s) \\ D^{\perp\perp}(s,F) & = & \prod_{a \in A(s)} D(s,a) \\ n^{\perp\perp}(s,F,g) & = & s[F(g)/g(F(g))]\ . \end{array}$$

That equality is a simulation from $w^{\perp\perp}$ to w takes the form:

$$\begin{aligned}
(\forall s \in S) \quad & (\forall F \in A^{\perp\perp}(s))\boldsymbol{(\exists a \in A(s))} \\
& \boldsymbol{(\forall d \in D(s,a))}(\exists g \in D^{\perp\perp}(s,F)) \\
& \qquad s[a/\boldsymbol{d}] =_S s[F(g)/g(F(g))] \ .
\end{aligned}$$

By applying the contraposition of the axiom of choice on $\exists a \forall d$, this is equivalent to

$$\begin{aligned}
(\forall s \in S) \quad & (\forall F \in A^{\perp\perp}(s)) \left(\forall f \in \textstyle\prod_{a \in A(s)} D(s,a)\right) \\
& (\exists a \in A(s))(\exists g \in D^{\perp\perp}(s,F)) \\
& \qquad s[a/d] =_S s[F(g)/g(F(g))] \ .
\end{aligned}$$

We can swap quantifiers and obtain, by the definitions of $A^{\perp}$, $D^{\perp}$ and $A^{\perp\perp}$,

$$\begin{aligned}
(\forall s \in S) \quad & (\forall f \in A^{\perp}(s))\boldsymbol{(\forall F \in A^{\perp}(s) \rightarrow D^{\perp}(s,\_))} \\
& \boldsymbol{(\exists g \in D^{\perp\perp}(s,F))}(\exists a \in D^{\perp}(s,f)) \\
& \qquad s[a/f(a)] =_S s[\boldsymbol{F(g)}/g(\boldsymbol{F(g)})] \ .
\end{aligned}$$

We can now apply the contraposition of the axiom of choice on $\forall F \exists g$ to get the equivalent formulation

$$\begin{aligned}
(\forall s \in S) \quad & (\forall f \in A^{\perp}(s))(\exists g \in D^{\perp\perp}(s,F)) \\
& (\forall b \in D^{\perp}(s,g))(\exists a \in D^{\perp}(s,f)) \\
& \qquad s[a/f(a)] =_S s[b/g(b)] \ .
\end{aligned}$$

Since $D^{\perp\perp}$ is equal to $A^{\perp}$, this is obviously true.

Thus, we can conclude that equality is a simulation from $w^{\perp\perp}$ to w. □

We obtain a surprising corollary:

Corollary 34 *Any interaction system is isomorphic to an interaction where the sets of reactions do not depend on a particular action. (More precisely, for any state s, the function $a \mapsto D(s,a)$ is constant.)*

PROOF. Just notice that $w^{\perp\perp}$ satisfies this property. □

Finally, we have

Corollary 35 *The category* **Int** *is $\star$-autonomous (see [19]), we can thus interpret classical linear logic.*

PROOF. Once we know that $\perp$ is dualizing, the remaining condition are fairly easy to check: the following diagram should be commutative

$$\begin{tikzcd}
w_1 \multimap w_2 \arrow[r, "\_^{\perp}"] \arrow[dr, "d_{w_1}^{-1} \cdot - \cdot d_{w_2}"'] & w_2^{\perp} \multimap w_1^{\perp} \arrow[d, "\_^{\perp}"] \\
 & w_1^{\perp\perp} \multimap w_2^{\perp\perp}
\end{tikzcd}$$

where d_w is the natural isomorphism from w to $w^{\perp\perp}$. This is immediate since d_w is the identity on S and $\_^{\perp}$ is the "converse" operation of a relation. (See footnote 9 on page 27.)

We can then unfold all the usual technology to give a denotational model for *classical* linear logic.

□

It is interesting to highlight some aspects of this model

- **Int** is a "games" model for full *classical* linear logic. The isomorphism between w and $w^{\perp\perp}$ is given by the identity relation.
- The constructions are quite different from usual games constructions; in particular, they have a strong *synchronous* feeling.
- The notion of strategy is not used to define morphisms; rather, we use the notion of *simulation.*
- The fact that $w \simeq w^{\perp\perp}$ seems rather accidental as it is not expected to hold in any generalized version of interaction systems. (See the discussion about containers in section 6.2). This fact is also highly non-constructive and almost counter-intuitive.

Putting the model of the differential λ-calculus with the dualizing object $\perp$, it is expected that we get a model for Lionel Vaux's "differential $\lambda\mu$-calculus" (see [20]), either in Vaux's setting (typed) or in an untyped setting.

6 Related Notions

6.1 Predicate Transformers

The category **Int** has a very concrete feeling. In [2], we have developed a model for full linear logic with a different intuition: the category of *predicate transformers* with *forward data-refinements:*

Definition 36 *If S is a set, a* predicate transformer *on S is a monotonic (w.r.t. inclusion of subsets) function from $\mathcal{P}(S)$ to $\mathcal{P}(S)$.*

If F_1 and F_2 are predicate transformers respectively on S_1 and S_2, a forward data-refinement *from P_1 to P_2 is a relation $r \subseteq S_1 \times S_2$ s.t. $\langle r \rangle \cdot F_1 \subseteq F_2 \cdot \langle r \rangle$.* [10] *(Extensional ordering.)*

Such predicate transformers with forward data-refinements form a category called **PT**.

An interaction system can be seen as a concrete representation for a predicate transformer. More precisely:

Proposition 37 *The operation $w \mapsto w^\circ$ from* **Int** *to* **PT** *defined as*

$$s \in w^\circ(x) \quad \Leftrightarrow \quad (\exists a \in w.a(s))(\forall d \in w.D(s,a))\ s[a/d] \in x$$

can be extended to a full and faithful functor from **Int** *to* **PT**.

The intuition in $s \in w^\circ(x)$ is that the Angel has a foolproof way to reach x in exactly one interaction.

PROOF. The action on morphisms is just the identity: we thus need to show that r is a simulation from w_1 to w_2 iff r is a forward data-refinement from w_1° to w_2°. The proof is not very difficult and can be found in [10]. $\square$

The interesting point is that all the constructions presented above are the concrete versions of the constructions presented in [2]. For example, we have

$$(w_1 \otimes w_2)^\circ \quad = \quad w_1^\circ \otimes w_2^\circ$$

[10] Where $\langle r \rangle$ is the *direct image* of r: $s_2 \in \langle r \rangle(x)$ iff $(\exists s_1)\,(s_1, s_2) \in r \wedge s_1 \in x$.

where $\otimes$ on the left is the synchronous tensor on interaction systems and the tensor on the right is the tensor on predicate transformers.[11]

Another interesting example is the fact that $(w^\perp)^\circ = (w^\circ)^\perp$. This is interesting because the definition of duality in the case of predicate transformers is very simple, and involutivity is trivial:

$$F^\perp(x) \quad = \quad \overline{F(\overline{x})}$$

where $\overline{x}$ represents the S-complement of x (*i.e.* $\overline{x} = S \setminus x$).

Surprisingly, proposition 37 can be strengthened in an ad-hoc way to read:

Proposition 38 *The categories* **Int** *and* **PT** *are equivalent. Moreover, this equivalence is a "retract".*

PROOF. By "retract", we mean the following: there is a functor $\mathcal{F}$ from **PT** to **Int** which satisfies $\mathcal{F}(F)^\circ = F$ and $\mathcal{F}(w^\circ) \simeq w$. In other words, we obtain equal object in one direction but only isomorphic objects in the other direction.

This functor $\mathcal{F}$ is defined as follows: let F be a predicate transformer on S, define $\mathcal{F}$ to be the interaction system on S with components

$$\begin{aligned} \mathcal{F}(F).A(s) \quad &= \quad \{x \subseteq S \mid s \in F(x)\} \\ \mathcal{F}(F).D(s,x) \quad &= \quad x \\ \mathcal{F}(F).n(s,x,s') \quad &= \quad s' \end{aligned}$$

Checking that this operation does define a functor is left as an exercise. (See [10].) $\square$

Once more, we separate propositions 37 and 38 because while we expect proposition 37 to hold for different generalizations of interaction systems / predicate transformers (see theorem 3.4 in [3]), proposition 38 seems very specific to this particular case.

[11] $(s_1, s_2) \in F_1 \otimes F_2(r)$ iff $(\exists x \subseteq S_1)(\exists y \subseteq S_2)\, x \times y \subseteq r \wedge s_1 \in F_1(x) \wedge s_2 \in F_2(y)$

6.2 Containers

In [3], the authors study the notion of *container,* a structure bearing several similarities with the notion of interaction system. They work in a variant of Martin-Löf type theory ([21,22]), a dependent predicative type theory.

Mild knowledge about this type theory is assumed in this section.

Simply, a container is given by the following:

- a set A of *shapes*;
- and for any $a \in A$, a set $D(a)$ of *positions.*

A morphisms from (A_1, D_1) to (A_2, D_2) is given by a pair (f, u) where f is a function $f : A_1 \to A_2$ and u is a family of functions indexed by A_1 and we have $u_{a_1} : D_2(f(a_1)) \to D_1(a_1)$.

This is reminiscent of interaction systems in the following way: any interaction system on the set of states $\{*\}$ (singleton set) can be seen as a container, and any container can be seen as an interaction system on $\{*\}$.

The links between container morphisms and simulations is subtler: a simulation from w_1 to w_2 (two interaction systems on $\{*\}$) is given by a relation $r \subseteq \{*\} \times \{*\} \simeq \{*\}$. In Martin-Löf type theory, a subset is seen as a propositional function $r : \{*\} \to \mathbf{Set}$, *i.e.* a set. The condition required to make this “relation” a simulation is the following:

$$\begin{aligned} r \qquad \Rightarrow \qquad & (\forall a_1 \in A_1)(\exists a_2 \in A_2) \\ & (\forall d_2 \in D_2(a_2))(\exists d_1 \in D_1(a_1))\ r\ . \end{aligned}$$

We can apply the (constructive) axiom of choice to skolemize this and we obtain

$$\begin{aligned} r \qquad \Rightarrow \qquad & (\exists f : A_1 \to A_2)\Big(\exists u : \textstyle\prod_{a_1 \in A_1} D_2(f(a_1)) \to D_1(a_1)\Big) \\ & (\forall a_1 \in A_1)(\forall d_2 \in D_2(f(a_1)))\ r \end{aligned}$$

which is *logically* equivalent to

$$r \qquad \Rightarrow \qquad (\exists f : A_1 \to A_2)\Big(\exists u : \textstyle\prod_{a_1 \in A_1} D_2(f(a_1)) \to D_1(a_1)\Big)\ \top$$

where $\top$ denotes the true proposition (or the singleton set in Martin Löf type theory).

Thus, a constructive (in the sense of Martin-Löf type theory) simulations between two interaction systems on $\{*\}$ is given by:

- a set r;
- and a function $r \to (\Sigma f : A_1 \to A_2) \prod_{a_1 \in A_1} D_2(f(a_1)) \to D_1(a_1)$.

Equivalently, a simulation between two interaction systems on $\{*\}$ is nothing but a *family of container morphisms* between the corresponding containers!

However, the difference is that while container morphisms are identified when the *functions* acting on actions / reactions are extensionally equal, simulations are identified when the *relations* between states are extensionally equal. In other words, two simulations $(r_1, (f_1, u_1))$ and $(r_2, (f_2, u_2))$ between interaction systems on $\{*\}$ are equal when there are "translating functions" $r_1 \rightleftarrows r_2$.[12]

If we adopt a classical point of view, then everything is rather boring: there is at most one non-empty simulation between two interaction systems on $\{*\}$: the relation $\{(*, *)\}$. The links with container morphisms is then the following:

> *if there is at least one container morphisms from w_1 to w_2, then there will be exactly one non-empty simulation from w_2 to w_2;*
> *if there are no container morphism from w_1 to w_2, then the only simulation from w_1 to w_2 will be the empty simulation.*

This whole theory of containers can be extended to work in a large class of locally cartesian closed categories. (See [3].) In such a setting, one needs to take care of additional coherence diagrams, but the idea is similar.

There is currently some work being done on generalizing containers to a notion of dependent containers, *i.e.* interaction systems. The idea, following the original intuition of [23] and [24] is to define a dependent container in a locally cartesian closed category $\mathcal{C}$ as:

- an object S in $\mathcal{C}$;
- an object A in $\mathcal{C}/S$;
- an object D in $\mathcal{C}/(\Sigma_S A)$;[13]
- a morphism n in $\mathcal{C}(\Sigma_{\Sigma_S A} D\, ,\, S)$.

The appropriate notion(s) of morphism is not entirely clear and still under heavy discussion...

[12] Note that since r_1 and r_2 are "propositions", we do not require that those translating functions are inverse of each other.
[13] Recall that in a locally cartesian closed category, if B is a slice in $\mathcal{C}/A$, we write $\Sigma_A B$ for the codomain of B.

Conclusion

We have developed a new category where objects are games to model linear logic or λ-calculus. What is rather new is the use of a notion of *simulation* for morphisms: the notion of strategy is not used in this model! Strategies do however appear implicitly in the notion of safety property which are the "points" of our model: a safety property is set of states for which there is an infinite strategy which restrict interaction to stay in the safety property. This strategy is only guaranteed to exists, but there is in general no way to obtain it.

Some of the interesting points about this model are that is allows to model full linear logic (it can even be extended to second order). Moreover, and this is relatively new, it can interpret the untyped differential λ-calculus.

An interesting project is to see whether one can apply the technology developed here in order to give denotational models for more interesting programming languages. PCF-like languages ought to be rather easy, but interaction systems (or predicate transformers) are rooted in "real" programming[14] so that we might expect to have access to many programming features...

A The Simply Typed Differential λ-calculus

The syntax of the simply typed differential λ-calculus is given by the following grammar:

$$\begin{array}{lcl} t, u, t_1, t_2 & ::= & x \;\mid\; (t_1)\, t_2 \;\mid\; \lambda x.t \;\mid \\ & & 0 \;\mid\; (t_1 + t_2) \;\mid\; \mathrm{D}\, t \cdot u \end{array}$$

We define β-reduction in the usual way:

$$(\lambda x.t)\, u \quad \rightsquigarrow_\beta \quad t[u/x]$$

[14] Predicate transformers were used to give a semantics to sequential programs by Dijkstra (**wp** or **wlp** calculus) and have been extended to deal with *specifications* as well (whole field of refinement calculus); interaction systems, as hinted in the first section seem appropriate to describe interfaces.

where substitution is extended in the following "obvious" way:

$$\begin{array}{lcl}
x[u/x] & = & u \\
y[u/x] & = & y \quad \text{if } y \neq x \\
(t)v\ [u/x] & = & (t[u/x])v[u/x] \\
\lambda x.t\ [u/x] & = & \lambda x.t \\
\lambda y.t\ [u/x] & = & \lambda y\ .\ t[u/x] \quad \text{if } y \neq x \\
0[u/x] & = & 0 \\
t_1 + t_2\ [u/x] & = & t_1[u/x]\ +\ t_2[u/x] \\
\mathrm{D}\, t \cdot v\ [u/x] & = & \mathrm{D}\, t[u/x]\ \cdot\ v[u/x]
\end{array}$$

Differential reduction is defined as:

$$\mathrm{D}(\lambda x.t) \cdot u \quad \rightsquigarrow_{\mathrm{D}} \quad \lambda x\ .\ \frac{\partial t}{\partial x} \cdot u$$

where $\partial t/\partial x \cdot u$, the *linear substitution of x by u in t* is defined as:

$$\begin{array}{lcl}
\partial x/\partial x\ \cdot\ u & = & u \\
\partial y/\partial x\ \cdot\ u & = & 0 \quad \text{if } y \neq x \\
\partial (t)v/\partial x\ \cdot\ u & = & (\partial t/\partial x \cdot u)v + (\,\mathrm{D}\, t \cdot (\partial v/\partial x \cdot u))v \\
\partial \lambda x.t/\partial x\ \cdot\ u & = & \lambda x.t \\
\partial \lambda y.t/\partial x\ \cdot\ u & = & \lambda y.(\partial t/\partial x \cdot u) \quad \text{if } y \neq x \\
\partial 0/\partial x\ \cdot\ u & = & 0 \\
\partial (t_1 + t_2)/\partial x\ \cdot\ u & = & \partial t_1/\partial x \cdot u\ +\ \partial t_2/\partial x \cdot u \\
\partial (\mathrm{D}\, t \cdot v)/\partial x\ \cdot\ u & = & \mathrm{D}(\partial t/\partial x \cdot u) \cdot v\ +\ \mathrm{D}\, t \cdot (\partial v/\partial x \cdot u)
\end{array}$$

Terms are quotiented by the (contextual closure of the) following equations:

- $0 = (0)u = \lambda x.0 = \mathrm{D}\, 0 \cdot t = \mathrm{D}\, t \cdot 0$;
- $(t_1 + t_2)\, u = (t_1)u\ +\ (t_2)u$;
- $\lambda x.(t_1 + t_2) = \lambda x.t_1\ +\ \lambda x.t_2$;
- $\mathrm{D}(t_1 + t_2) \cdot u = \mathrm{D}\, t_1 \cdot u\ +\ \mathrm{D}\, t_2 \cdot u$;
- $\mathrm{D}\, t \cdot (u_1 + u_2) = \mathrm{D}\, t \cdot u_1\ +\ \mathrm{D}\, t \cdot u_2$;
- $\mathrm{D}(\mathrm{D}\, t \cdot u) \cdot v = \mathrm{D}(\mathrm{D}\, t \cdot v) \cdot u$.

The typing rules are

$$(1)\quad \frac{}{\Gamma \vdash x : \tau} \quad \text{if “}x : \tau\text{” appears in } \Gamma;$$

$$(2)\quad \frac{\Gamma, x : \tau \vdash t : \sigma}{\Gamma \vdash \lambda x.t : \sigma \to \tau}\ ;$$

$$(3)\quad \frac{\Gamma \vdash t : \tau \to \sigma \qquad \Gamma \vdash u : \sigma}{\Gamma \vdash (t)\, u : \tau}\ ;$$

$$(4)\quad \frac{}{\Gamma \vdash 0 : \tau} \quad \text{and} \quad \frac{\Gamma \vdash t : \tau \qquad \Gamma \vdash u : \tau}{\Gamma \vdash t + u : \tau}\ ;$$

$$(5)\quad \frac{\Gamma \vdash t : \tau \to \sigma \qquad \Gamma \vdash u : \tau}{\Gamma \vdash \mathrm{D}\, t \cdot u : \tau \to \sigma}\ .$$

Typed terms enjoy the Church-Rosser property and strong normalization (w.r.t. β/differential reduction).

References

[1] T. Ehrhard, L. Regnier, The differential lambda calculus, Theoret. Comput. Sci. 309 (1) (2003) 1–41.

[2] P. Hyvernat, Predicate transformers and linear logic: yet another denotational model, in: J. Marcinkowski, A. Tarlecki (Eds.), 18th International Workshop CSL 2004, Vol. 3210 of LNCS, Springer-Verlag, 2004, pp. 115–129.

[3] M. Abott, T. Altenkirch, N. Ghani, Containers - constructing strictly positive types, Theoretical Computer Science 342 (2005) 3–27, applied Semantics: Selected Topics.

[4] S. Abramsky, R. Jagadeesan, P. Malacaria, Full abstraction for PCF, Information and Computation 163 (2) (2000) 409–470.

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

[6] S. Abramsky, D. Ghica, L. Ong, A. Murawski, Applying game semantics to compositional software modelling and verification, in: Tools and Algorithms for the Construction and Analysis of Systems, Vol. 2988 of LNCS, Springer-Verlag, 2004, pp. 421–435.

[7] P. Hancock, A. Setzer, Interactive programs in dependent type theory, in: Computer science logic (Fischbachau, 2000), Vol. 1862 of Lecture Notes in Computer Science, Springer, Berlin, 2000, pp. 317–331.

[8] P. Hancock, A. Setzer, Specifying interactions with dependent types, in: Workshop on subtyping and dependent types in programming, Portugal, July 7th 2000, 2000.

[9] P. Hancock, P. Hyvernat, Programming interfaces and basic topology, Annals of Pure and Applied Logic 137 (1-3) (2006) 189–239.

[10] P. Hyvernat, A logical investigation of interaction systems, Thèse de doctorat, Institut mathématique de Luminy, Université Aix-Marseille II (2005).

[11] S. M. Lane, Categories for the working mathematician, 2nd Edition, Vol. 5 of Graduate Texts in Mathematics, Springer-Verlag, New York, 1998.

[12] R. Milner, Calculi for synchrony and asynchrony, Theoret. Comput. Sci. 25 (3) (1983) 267–310.

[13] P. A. Melliès, Categorical models of linear logic revisited, to appear in Theoretical Computer Science, (2002).

[14] Y. Lafont, Logiques, catégories et machines, Thèse de doctorat, Université Paris 7 (1988).

[15] T. Ehrhard, Finiteness spaces, Mathematical Structures in Computer Science 15 (4) (2005) 615–646.

[16] L. Vaux, λ-calculus in an algebraic setting, unpublished note (2006).

[17] R. F. Blute, J. R. B. Cockett, R. A. G. Seely, Differential categories, to appear in Mathematical Structures in Computer Science (2005).

[18] R. M. Amadio, P.-L. Curien, Domains and lambda-calculi, Vol. 46 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge, 1998.

[19] M. Barr, $*$-autonomous categories and linear logic, Mathematical Structures in Computer Science. A Journal in the Applications of Categorical, Algebraic and Geometric Methods in Computer Science 1 (2) (1991) 159–178.

[20] L. Vaux, Differential $\lambda\mu$-calculus, technical report (2005).

[21] P. Martin-Löf, Intuitionistic type theory, Bibliopolis, Naples, 1984, notes by Giovanni Sambin.

[22] B. Nordström, K. Petersson, J. M. Smith, Programming in Martin-Löf's type theory. An introduction, The Clarendon Press Oxford University Press, New York, 1990.

[23] R. A. G. Seely, Locally Cartesian closed categories and type theory, Mathematical Proceedings of the Cambridge Philosophical Society 95 (1) (1984) 33–48.

[24] M. Hofmann, On the interpretation of type theory in locally cartesian closed categories, in: CSL '94: Selected Papers from the 8th International Workshop on Computer Science Logic, Springer-Verlag, London, UK, 1995, pp. 427–441.