UNIVERSITÉ DE LA MÉDITERRANÉE, AIX-MARSEILLE 2
U.F.R. DE MATHÉMATIQUES

N° attribué par la bibliothèque

Année : 2005

THÈSE

pour l'obtention du diplôme de

DOCTEUR DE L'UNIVERSITÉ AIX-MARSEILLE 2

Spécialité : mathématiques discrètes et fondements de l'informatique

présentée et soutenue publiquement

par

Pierre HYVERNAT

le 12 décembre 2005

TITRE :

Une investigation logique des systèmes d'interaction

Directeur de thèse : M. Thomas EHRHARD
Codirecteur de thèse : M. Thierry COQUAND

JURY

MM.	Thierry	COQUAND	directeur
	René	DAVID	
	Thomas	EHRHARD	directeur
	Marcelo	FIORE	rapporteur
	Laurent	RÉGNIER	président
	Thomas	STREICHER	rapporteur

Remerciements :

Je tiens en premier lieu à remercier Thierry Coquand qui a commencé par encadrer mon stage de maîtrise pendant l'été 2001. C'est pendant ce stage que j'ai découvert les systèmes d'interaction, objet d'étude de ce travail. C'est également lui qui m'a proposé d'approfondir ce sujet pendant une partie de ma thèse.

Merci également à Thomas Ehrhard, mon directeur à Marseille qui m'a autorisé à passer la première année de ma thèse en Suède, me permettant ainsi de poursuivre des travaux en dehors des thèmes de recherche du laboratoire. C'est également lui qui m'a encouragé à poursuivre l'étude des systèmes d'interaction dans un contexte plus "marseillais", à savoir, la logique linéaire.

La première partie de ce travail a été effectuée au département d'informatique de l'université de Chalmers, à Göteborg en Suède. Je tiens à remercier l'ensemble des étudiants qui ont grandement contribué à l'ambiance à la fois studieuse et détendue pendant mon séjour là bas. Un merci particulier va à Markus Forsberg, mon collègue de bureau.

La seconde partie de ma thèse à été effectuée à l'Institut mathématique de Luminy, à Marseille. Merci à toute l'équipe pour l'ambiance de travail très appréciable. Un merci particulier va à Laurent Régnier, Pierre Boudes et Lionel Vaux pour toutes les discussions, parfois animées, qui ont parsemé cette période.

Je tiens également à remercier Peter Hancock sans qui ce travail n'aurait jamais vu le jour. Son expérience de l'informatique concrète ainsi que sa culture scientifique ont ont été à l'origine de nombreuses idées que l'on retrouve plus loin. Ce sont les innombrables discussions et emails échangés qui ont permis de développer de nombreux concepts présents dans ce travail. Je le remercie également infiniment pour avoir suivi de près la rédaction de ce document et de m'avoir fait part de ses nombreuses remarques.

Un autre merci va à Giovanni Sambin, à Padoue en Italie, pour m'avoir permis de présenter mes travaux au "Second workshop on formal topology" et pour m'avoir invité plusieurs jours à Padoue.

Finalement, un dernier merci à tous les autres, familles et amis, trop nombreux pour être cités. Ils ont su me rappeler qu'il y a d'autres aspects que la recherche et ont réussit à me faire garder les pieds sur terre.

Un remerciement de dernière minute va également à mes deux rapporteurs externes, Marcelo Fiore et Thomas Streicher pour avoir accepté la lourde tache de relire et valider ce travail...

Table of Contents

This document was typeset using Donald Ervin Knuth's TEX system. The fonts used for the text are mainly from the "Concrete" family of fonts designed by D. Knuth for is book "Concrete Mathematics". The main fonts used for mathematics are from the Euler family and were designed by Herman Zapf for the American Mathematical Society.

Introduction

The Curry-Howard isomorphism, whose existence can be traced back to the 50's in the work of Haskell Curry and William Howard, has proved a key notion in the development of modern proof theory. In essence, the Curry-Howard isomorphism is the following slogan:

"a proof is a program and a program is a proof".

Its conceptual importance cannot be ignored but, ever since the basis of this correspondence has been laid, the isomorphism has mostly worked in one direction: interpreting a proof as a program. People have developed stronger and stronger "programming languages"[1] to give computational meaning to bigger and bigger proofs, but only little attention has been given to providing mathematical content to "real-life" programs. This is particularly true if one looks at "interactive" programs, which do not directly correspond to λ-terms. Some work has been done in this direction in [42], where Peter Hancock takes the view that interactive programs are proofs of well-foundedness, thus linking interactive programs with traditional proof theory.

One of the key notions appearing in [42] is the notion of *interaction system*. Briefly, an interaction system is given by the following data:

- a set of states;
- for each state, a set of labels for outgoing "Angel" transitions;
- for each Angel transition, a set of labels for outgoing "Demon" transitions;
- each pair Angel-transition/Demon-transition leads to a new state.

The notion is very close to the usual notion of *labeled transition system* except that there are two kinds of label: angelic and demonic ones. What is important is that a Demon transition *follows* an Angel transition, and that there is no "intermediate state". We travel in an interaction system in the following manner:

1) we start in a state;
2) the Angel chooses one of its own transitions from that state;
3) the Demon chooses one of its own transitions following the Angel transition;
4) we reach a new state.

In particular, there is no state between *2* and *3*. By making the Demon transitions depend on a particular Angel transition, we obtain a notion which is very different from a transition system where labels are pairs (Angel label/Demon label). In par-

[1]: or, to be more precise, stronger and stronger type theories

ticular, it is possible to distinguish between an "Angel deadlock" (the Angel cannot move) and a "Demon deadlock" (the Angel can move, but the Demon cannot).

We extend the theory of interaction systems by adding a notion of morphism bearing similarities, both formal and intuitive, to the usual notion of simulation. One of the goals is to develop a viable denotational model for "component based programming": an interaction system can be seen as an interface, *i.e.* as the abstract description of the possible use of a program (a *specification*). Implementing an interface *depending* on other interfaces (*i.e.* writing a component) is captured by the notion of *refinement*, also called *generalized simulations:* we translate high-level commands (which we want to implement) into low-level commands (which we assume to be already implemented).

The next goal of this work is to extend the Curry-Howard correspondence by linking this model of programming to mathematical notions. We exhibit an almost perfect relationship between interaction systems and the concept of "inductively generated basic topologies".[2] The intuition is that states serve as representations for *basic* open sets of a topology. The label structure is then an abstract way to describe the different ways one can *cover* a basic open by other basic opens. General simulations (*i.e.* implementations) correspond exactly to the notion of *continuous function*. This extends the Curry-Howard isomorphism by linking very concrete notions to more abstract ones in the following manner:

- an interface corresponds to a (basic) topology;
- a client program corresponds to a (proof of a) covering;
- a server program corresponds to a (proof that a subset is a) closed set;
- a generalized simulation corresponds to a continuous function.

One drawback of interaction systems is that they represent, in general, "strictly basic" topologies:[3] starting from concrete programming interfaces, we usually obtain non-distributive complete sup-lattices (binary "intersection" of open sets doesn't distribute over arbitrary "unions" of open sets)! It is however possible to do a little more work and interpret the extra condition yielding distributivity: this is linked with the notion of *concurrent execution*. Unfortunately, the intuitions are not as clear as in the simple, non-distributive case.

A nice class of non-distributive interaction systems arises naturally: by weakening a particular class of formal spaces giving models for *geometric theories* ([15] and [24]) into a class of pretopologies giving models for *"linear" geometric theories*, we obtain a completeness result for linear geometric theories. Both geometric theories and linear geometric theories can be described by interaction systems, but while the former enjoy "localization" (a property making the lattice of open sets distributive), the latter do not.

Interaction systems have a very rich structure. They developed naturally in a second, unsuspected direction: linear logic. To put it simply, linear logic can be seen as a logic of *resources* whereas classical logic is a logic of *truth*. The second part of

[2]: This takes the form of a full and faithful functor from the category of interaction systems to the category of basic topologies (proposition 4.2.8).

[3]: Basic topologies are a weaker form of *formal spaces*, which are the usual predicative version of *locales* or *frames*.

this work shows how to use the notion of interaction system to give a denotational semantics to linear logic. A lot of work has already been done in this area and there are several denotational models, both "static" (coherence spaces and related) and "dynamic" (games semantics). The interest of interaction systems is that they yield a semantics with the two aspects:

- dynamic since the notion itself of interaction systems and their morphisms is defined using an *interaction* intuition;
- static since the interpretation of a proof is just a set (of states).

Both aspect are related in the sense that the interpretation of a proof is a set of states satisfying a *safety property:* it behaves well w.r.t. *all* possible interactions. More precisely, from any state in the interpretation, the Angel has (at least) one move s.t. all the following Demon moves go back into the interpretation. This means in particular that the Angel can avoid deadlock, were interaction to start in the interpretation of a proof.

Because of the objects at hands, part II can be seen as an "unorthodox" games semantics for linear logic. A comprehensive comparison between traditional games and interaction systems is yet to be done, but we can give some of the differences.

A mostly inaccurate slogan for interaction systems as a model for (linear) logic could be something like *"games without explicit strategies"*. It is quite clear that interaction systems can be seen as a notion of games, but the notion of morphism doesn't contain an explicit strategy: they are only relations! An implicit notion of strategies is present in the notion of *safety property*, and since morphisms are safety properties we can guarantee, as a side effect, the existence of certain strategies intrinsic in any morphism. Those strategies however have no real interest since they are not part of the data defining morphisms. (Different strategies may be used for the same safety property!)[4] The reason strategies are not important comes from the fact that moves are individually unimportant! Their only goal is to serve as links between states. One can even devise an equivalent category where the notion of moves has disappeared: this is the category of *predicate transformers* with forward data-refinements. All the structure of interaction systems can be translated faithfully in terms of predicate transformers. Going from an interaction system to its associated predicate transformer is very similar to going from a labeled transition system to its associated unlabeled graph (binary relation). The reason for using one category or the other is, in an impredicative setting mostly a matter of taste.[5]

To come back to the comparison between interaction systems and games, one can say that simulations are at the same time more concrete and more abstract than traditional morphisms between games (which are special cases of strategies):

- they are more concrete because they correspond to the usual notion of simulation between labeled transition systems;[6]
- they are more abstract because the computational part of the simulation is ab-

[4]: We thus avoid this "unfortunate" aspect of traditional games semantics where a proof (*i.e.* a tree) is interpreted by a strategy (*i.e.* another tree).

[5]: In a predicative setting, we only have a full and faithful functor from interaction systems to predicate transformers. It is then easier to work with interaction systems, as most of the operations cannot be defined predicatively on predicate transformers.

[6]: Interaction systems are themselves very adequate to model "concrete", non-logical situations.

stracted away: equality of simulations is equality of the underlying relations. The second point accounts for the relative simplicity of composition of morphisms. In several of the traditional games semantics, total morphisms (given by total strategies) are not closed under composition. The reason is that while defining the composition of strategies, there could be some "infinite chattering" in the middle game. Even when this problem is solved (by considering partial strategies for example), proving transitivity is, though not difficult, not entirely trivial. Here, composition of morphisms (plain relational composition) is trivially associative.

Finally, one of the nice features about this semantics is that it projects in the simplest denotational model of linear logic: the relational model. Projecting an interaction system just means forgetting the move structure and only keeping the set of states. (A simulation relation is simply sent to itself: a relation.)

There are several crucial differences between the notion of "point" (potential interpretation of proofs) in this model and in other models. In most cases, the collection of points forms a Scott domain: there is a notion of finite element and of directed limit. In our case, the collection of points (safety properties) forms a complete sup-lattice which is generally not algebraic. The two main facts are the following:

- a subset of a safety property need not be a safety property;
- a union of safety properties is still a safety property.

The situation is thus radically different from most usual denotational models. The first point means that we do not really consider partial objects: to be an interpretation of a proof requires the subset of states to be "big enough". (Even though there always is a trivial smallest safety property: the empty set.) If one compares this to the closest situation, coherent spaces, the difference is obvious: there, a subset of a clique (complete subgraph) is always a clique.

The second point means that it is "semantically" sound to add proofs: if $[\![\pi_1]\!]$ and $[\![\pi_2]\!]$ are interpretations of proofs, then $[\![\pi_1]\!] \cup [\![\pi_2]\!]$ is also a "potential" interpretation of a proof. A possible intuition is that this is the interpretation of $\pi_1 + \pi_2$, the *non-deterministic sum* of π_1 and π_2. This intuition is even more convincing when applied to interpretations of λ-terms, where $+$ represents non-deterministic sum of programs. The reason this is not possible in most other denotational models is that they are based on deterministic intuitions (even when, like in the case of Scott domains, functions such as the parallel boolean "or" are accepted). For example, strategies in mainstream game semantics are deterministic, which prevents the union from being well-defined; in more traditional models, we can only add coherent partial objects: they need to have a common extension.

In the denotational model we obtain, two points are worth noting:

- the additive connectives $\oplus$ and $\&$ are identified;
- when atomic formulas are restricted to the logical constants, the model is trivial.

The first point is a consequence of the previous remark that safety properties, and thus simulations, are closed under unions: any model with a notion of "sum" on morphisms does identify the additives.[7] Rather than trying to change the semantics, we try to

[7]: Technically: in any category enriched over commutative monoids, the product and coproduct coincide (if they exist).

find a logic corresponding to this. One such logic exists: the *differential* λ*-calculus* of Thomas Ehrhard and Laurent Régnier. It has, besides a notion of sum, a very rich additional structure. We show that we can interpret this differential structure in a non trivial way within interaction systems. This is interesting because it gives a model of differential λ-calculus having very different intuitions from the original models: Köthe spaces and finiteness spaces, both based on a notion of "finitary" sets.

The second problem is, from a categorical point of view, not very important. From a more concrete point of view however, it questions the relevance of interaction systems. To justify them as a good model, we extend the semantics to second order linear logic. In order to do so, we follow closely the model appearing in [38] and the work of Alexandra Bruasse from [18]. We obtain in this way a canonical interpretation of Π^1_1 formulas, *i.e.* propositional linear logic. Once this is done, getting full second order is, though a little technical, neither difficult nor very exciting. We check on very simple examples that this model is non trivial and does correspond to what we have in mind. It shows that as opposed to the simple relational model, interaction systems or predicate transformers have a real discriminative power. The situation seems to be very close to the case of coherent spaces, except that we have added unions and a differential structure.

Content

This thesis is divided in two parts: the first one is mostly carried out in a strongly constructive setting, namely predicative dependent type theory; the second one lives in a traditional classical setting. Those two parts correspond, roughly speaking, to the work done respectively in Chalmers (Göteborg, Sweden) and in Luminy (Marseille, France). They are representative of the local interests. A simple way to summarize the difference between the two parts, besides constructivity requirements, is the following:

- at the intersection of both worlds lies a category **Int**. Its objects are given by interaction systems and its morphisms by *linear* simulations;
- programming and topology are concerned with a Kleisli construction over a monad $\_^*$ of "reflexive and transitive closure";
- (linear) logic is concerned with a Kleisli construction over the (co)monad $!\_$ of "synchronous multithreading".

Unfortunately, as of this writing, the two constructions have almost no relation to each other, save for the core category **Int**.

Briefly, after some preliminaries about type theory (chapter 1), the first part introduces, together with their computational relevance, the notion called *interaction systems* and their basic structure (chapter 2 and 3). The aim is to show that interaction systems are adequate to represent both the notion of programming interfaces (section 2.6) and the notion of (inductively generated) basic topology (chapter 4). By its very nature, this part has a strong computational content. We thus avoid as much as possible the use of non-constructive principles, and go even further by working in a predicative framework.

The second part drops all constructivity requirements and uses the abstract structure of interaction systems to give a "synchronous" model for full propositional

linear logic (chapter 6). Not all the structure of interaction systems is used, but we can extend the model to take into account the operation of *differentiation* present in Ehrhard and Régnier's differential λ-calculus (section 6.4).

If one is not interested in the interactive intuition of interaction systems, it is possible to simplify the presentation and obtain the very concise model presented in chapter 7 based on the notion of *predicate transformers*. Using this version of the model as a starting point, we finally interpret full second order linear logic in chapter 8.

Notes

This work is, except when explicitly stated, original with the following proviso:

- chapters 1 and 5 are mostly standard introductions, except for the discussions about equality in sections 1.1.6 and 1.1.7 which recall some of the ideas present in Peter Hancock's thesis ([42]).
- Most of the definitions appearing at the beginning of chapter 2 were already developed by Peter Hancock and Anton Setzer.
- The categorical structure of interaction systems with simulations (chapter 3) and of the category of predicate transformers with forward data-refinement is original. (But the link between the two is mostly due to Peter Hancock.)
- The link with basic / formal topology is original but benefited from many discussion with Thierry Coquand and Giovanni Sambin.
- Finally, the extension of the model of section 7.1 to second order is inspired by the work of Alexandra Bruasse and Thomas Ehrhard on the relational model.

Parts of this work can be found in more concise form:

- sections 4.2 and 4.3 on the application of interaction systems to constructive topology (together with the relevant parts from chapter 2) appeared in [43];
- section 6.4 about the differential λ-calculus is contained in [55] and in [54];
- and section 7.1 about the denotational model based on predicate transformers is the subject of [53].

1 Preliminaries

Part I of this work will be developed with *constructivity* in mind. Motivating and introducing the general concepts of constructive mathematics would take us too far and we refer to the abundant literature on the subject ([61], [83] and [84], and [17]). To be more precise, most of part I is developed within a framework of "predicative constructive type theory". Since the second part will abandon the goal of constructivity (except in section 6.4), we will try to make the framework as transparent as possible. It must be noted that only the ambient logic is constructive and that everything from this first part also holds constructively.

1.1 Martin-Löf Type Theory

Martin-Löf dependent type theory ([62]) can be described as an expressive typed λ-calculus. The core consists of λ-terms with a strict typing discipline (dependent function types) ensuring strong normalization. In addition to the usual function types, we also have at our disposal a notion of dependent sum and a notion of inductive definitions. This theory is described in details in [62] and [68]. We then extend this with a notion of coinductive definitions and discuss the problems of general equality. Since they will be central in the sequel, we also show how to deal with the concepts of subsets and binary relations.

1.1.1 The Type Theory and its Associated Logic

We assume basic knowledge about the simply typed λ-calculus. For all this work, we only need two kinds for types:

- **Set** is the collection of datatypes, also called *sets;*
- **Type** is the collection of *proper types,* containing, among others, **Set**.

To simplify notation, we pretend that any set is also a proper type: **Set** $\subseteq$ **Type**.[1]

We make a typographic distinction between sets (capital roman letters like S) and proper types (calligraphic capital letters, like $\mathcal{S}$). We write membership in a set with the "$\in$" symbol: "$s \in S$" while membership in a proper type is written with a colon: "$X : \mathcal{A}$".

[1]: This is harmless in practice.

Intuitively speaking, **Set** consists of all the "datatypes". It is closed under most usual set-forming operations, with the notable exception of the powerset construction (see the discussion about *predicativity* in section 1.2.1). Elements of **Set** are called ... sets, and they are "small".

On the other hand, **Type** consists of the collections too big to be sets. The most trivial example is **Set** itself: the collection of all datatypes is certainly not a datatype. Similarly, the collection of functions from **Set** to **Set** is not a set, but is an element of **Type**. Elements of **Type** are called *proper types* and are intuitively "big". The difference between sets and proper types is in a way similar to the difference between sets and classes in von Neumann/Bernays/Gödel set theory or between different levels of Grotendieck universes.

↬ REMARK 1: this terminology can be very confusing at first, especially for computer scientists who are used to using the word "type" for usual datatypes (*i.e.* sets). We use the generic term "type" when we do not really care about the kind of objects, and we may specify using the adjectives "small" (for sets) or "big" (for proper types).

§ *Dependent Product.* We follow a rather informal presentation. To be precise, one needs to define types, contexts, typing judgments, etc. Some care is definitely needed, but this is is irrelevant for our purposes.

The most important set constructor is the dependent function type. It is called *dependent product* and is governed by the following rules:

- $\dfrac{A : \mathbf{Set} \qquad x \in A \vdash B(x) : \mathbf{Set}}{(\Pi x \epsilon A)\ B(x) : \mathbf{Set}}$ formation;
- $\dfrac{x \in A \vdash f \in B(x)}{(\lambda x \epsilon A).f \in (\Pi x \epsilon A)\ B(x)}$ introduction;
- and $\dfrac{t \in (\Pi x \epsilon A)\ B(x) \qquad a \in A}{t(a) \in B(a)}$ elimination.

The reduction rule for the dependent product is:[2]

$$\big((\lambda x \epsilon A).f\big)\ a \quad = \quad f[a/x]\ .$$

Thus, a term f of type $(\Pi x \epsilon A)\ B(x)$ is a function taking any $a \in A$ to an element of $B(a)$. This is exactly the definition of indexed cartesian product in classical mathematics. When the set $B(x)$ doesn't depend on $x \in A$, we recover the usual function space which we abbreviate by "$A \to B$". To make explicit the fact that the dependent product is a function space, we use the notation:

$$(x \epsilon A) \to B(x)$$

as a synonym for $(\Pi x \epsilon A)\ B(x)$.

We also have the same construction at the level of types, with the same rules and the same notation. We also allow mixed constructions of the form $A \to \mathcal{B}$ but then the kind of the dependent product will always be **Type**.

[2]: Just like in usual λ-calculus, "$f[a/x]$" is the term f where x has been substituted by a. As always, we need to make sure this doesn't capture free variables by first doing some α-conversion.

↬ REMARK 2: the "pure type system part" corresponds to having the two sorts $\mathbf{Set} : \mathbf{Type}$ and the rules $(\mathbf{Set}, \mathbf{Set})$, $(\mathbf{Set}, \mathbf{Type})$ and $(\mathbf{Type}, \mathbf{Type})$, *i.e.* it corresponds to $\lambda\mathbf{P}\underline{\omega}$. (See [11].)
In particular for any set S, we are allowed to form the types $S \to \mathbf{Set}$ (using the rule $(\mathbf{Set}, \mathbf{Type})$) and $(S \to \mathbf{Set}) \to \mathbf{Set}$ (using the rule $(\mathbf{Type}, \mathbf{Type})$).

§ *Dependent Sum.* There is a second set constructor, dual to the dependent product: the *dependent sum*. Just like the dependent product is an indexed cartesian product, the dependent sum is an indexed disjoint sum. It obeys the rules:

- $$\frac{A : \mathbf{Set} \qquad x \in A \vdash B(x) : \mathbf{Set}}{(\Sigma x \epsilon A)\ B(x) : \mathbf{Set}}$$ formation
- $$\frac{a \in A \qquad b \in B(a)}{(a,b) \in (\Sigma x \epsilon A)\ B(x)}$$ introduction
- $$\frac{p \in (\Sigma x \epsilon A)\ B(x) \qquad f \in (x \epsilon A) \to \big(y \epsilon B(x)\big) \to C\big((x,y)\big)}{\mathrm{split}(p,f) \in C(p)}$$ elimination

with the following reduction rule:

$$\mathrm{split}\big((a,b),f\big) \quad = \quad f\big((a,b)\big)\ .$$

The elimination rule may look unnecessarily complex, but for our purposes, it suffices to note that one can define the two projections:

- $$\frac{p \in (\Sigma x \epsilon A)\ B(x)}{\pi_1(p) \in A}$$ first projection
- $$\frac{p \in (\Sigma x \epsilon A)\ B(x)}{\pi_2(p) \in B\big(\pi_1(p)\big)}$$ second projection

as $\pi_1(p) \triangleq \mathrm{split}\big(p, (\lambda x y.x)\big)$ and $\pi_2(p) \triangleq \mathrm{split}\big(p, (\lambda x y.y)\big)$.

Note that when $B(x)$ doesn't depend on $x \in A$, then this is just a usual cartesian product.[3] We then write $A \times B$ rather than $(\Sigma x \epsilon A)\ B$.

§ *Other Constructions.* Since we need to start with something, we also have the singleton set $\{*\}$ and the empty set $\emptyset$ with the obvious rules. There is also a notion of disjoint sum $A + B$ with rules

- $$\frac{A : \mathbf{Set} \qquad B : \mathbf{Set}}{A + B : \mathbf{Set}}$$ formation
- $$\frac{a \in A}{\mathrm{inl}(a) \in A + B}$$ intro (left) and $$\frac{b \in B}{\mathrm{inr}(b) \in A + B}$$ intro (right)
- $$\frac{x \in A + B \qquad f \in (a \epsilon A) \to C\big(\mathrm{inl}(a)\big) \qquad g \in (b \epsilon B) \to C\big(\mathrm{inr}(b)\big)}{\mathbf{case}(x,f,g) \in C(i)}$$ elim

with reduction rule:

$$\mathbf{case}\big(\mathrm{inl}(a),f,g\big) \quad = \quad f(a) \qquad \text{and} \qquad \mathbf{case}\big(\mathrm{inr}(b),f,g\big) \quad = \quad g(b)\ .$$

Following standard programming practice, we use the notation

$$\begin{array}{l}\mathbf{case}\ x\ \text{of}\ \ \mathrm{inl}(a) \Rightarrow\ f(a)\\ \qquad\qquad\ \ \mathrm{inr}(b) \Rightarrow\ g(b)\ .\end{array}$$

[3]: Yes! Cartesian product is an instance of the dependent sum!

↬ REMARK 3: the disjoint sum could be defined as an indexed sum over a two elements set, but this requires ... a two element set.

Following another programming practice, we allow the use of constructors. They are introduced with the "**data**" keyword: for example,

$$\begin{array}{ll}\textbf{data} & \mathrm{Cons}_1\left(a\epsilon A, b\epsilon B(a)\right)\\ & \mathrm{Cons}_2(a_1\epsilon A, a_2\epsilon A, a_3\epsilon A)\end{array}$$

is a verbose way to describe the set "$(\Sigma a\epsilon A)\,B(a) + A\times A\times A$". Note that this allows to define the empty set and the singleton sets as:

$$\emptyset \triangleq \textbf{data} \qquad \text{and} \qquad \{*\} \triangleq \textbf{data}\ *$$

i.e. respectively as the set with no constructor and the set with a unique constructor.

§ *Curry-Howard Isomorphism.* So far, **Set** contains $\emptyset$, $\{*\}$ and is closed under Π, $\to$, Σ, $\times$ and $+$. The Curry-Howard isomorphism shows how to translate formulas into sets and proofs into terms, and vice and versa:

type theory :	$\emptyset$	$\{*\}$	Π	Σ	$\times$	$+$	$\to$
logic :	False	True	$\forall$	$\exists$	$\wedge$	$\vee$	$\Rightarrow$

The equivalence between terms and proofs is more subtle and requires some knowledge about intuitionistic natural deduction: a term of type F where F is a logical formula can be seen as a proof of the formula F. (In the Brouwer-Heyting-Kolmogorov sense.)

Martin-Löf type theory identifies sets and propositions in the sense that proving a proposition F is identified with giving a term of type F. Depending on the context, we may switch from the type theoretical notation to the logical notation transparently. We even mix the symbols to make things more readable. No confusion arises from this because until the beginning of part II, the logical symbols are always interpreted by their intuitionistic predicative versions (except when explicitly stated).

1.1.2 Inductive Definitions

The main interest of "high-level" type theories like Martin-Löf type theory lies in the possibility to have user-friendly inductive definitions. As we'll briefly recall in the next section, inductive definitions are available "for free" in impredicative theories, but impredicativity is too big of a prize to pay and we try to avoid it. We thus introduce *ad hoc* principles to deal with them. We will not go into the details about the justification of inductive definitions: the literature on this subject is complete enough. (This is treated in [4].)

Rather than giving the formal definition, we'll only look at an example: the case of lists over an arbitrary set. The following is the definition as we could write it in a functional programming language:

```
List (A:Set) : Set
List A := data  Nil | Cons(a:A,t:List A)
```

Since "`List A`" appears on the right hand side of "$\triangleq$", it means we are actually solving an equation: in a less verbose way, $X = \{*\} + A \times X$. We write this definition as:

$$\begin{array}{lcl} \textsc{List}(A) & : & \textbf{Set} \\ \textsc{List}(A) & \triangleq & (\mu X : \textbf{Set})\ \textbf{data}\ \mathrm{Nil} \\ & & \qquad\qquad \mathrm{Cons}(a \in A\ ,\ t \in X) \end{array}$$

where the binder "μ" is here to mean "inductive definition".

We restrict inductive definition to *strictly positive* functors, *i.e.* in an inductive definition, the variable X may not appear on the left of an arrow type.[4]

The induction principle (or the fact that we are using the *least* solution) is used in the following way: define the length of a list to be a natural number with:

$$\begin{array}{lcl} \mathrm{length}(l) & \triangleq & \textbf{case}\ l\ \text{of}\ \mathrm{Nil} \Rightarrow 0 \\ & & \qquad\qquad\quad \mathrm{Cons}(a,t) \Rightarrow 1 + \mathrm{length}(t)\ . \end{array}$$

We also use "pattern matching" notation as in:

$$\begin{array}{lcl} \mathrm{length}(\mathrm{Nil}) & \triangleq & 0 \\ \mathrm{length}\big(\mathrm{Cons}(a,t)\big) & \triangleq & 1 + \mathrm{length}(t)\ . \end{array}$$

We will later extend the schema of definition to allow the definitions of objects of type $S \to \textbf{Set}$ by least fixpoint, in the spirit of [70]. We will detail that when necessary.

§ *Agda.* This part of the theory (with a more general schema for inductive definitions) has been implemented as a "programming" environment in the Agda system ([26]). Inside this system, the definition of lists would take the exact form:

```
List (A::Set) :: Set
  = data   Nil
         | Cons (a::A)(l::List A)
```

and (supposing `A` is a set, and that natural numbers are defined) the length function would be written as

```
length (l::List A) :: Nat
  = case l of  (Nil)      -> 0
               (Cons a t) -> 1+(length t)
```

Writing a term and checking that it is of the correct type in Agda is, by the Curry-Howard isomorphism, equivalent to proving a proposition in intuitionistic logic. Several lemmas and propositions from the first part of this work have been formalized in this way: of particular interest are proposition 3.3.1 and proposition 2.6.8.

1.1.3 Coinductive Definitions

By the Knaster-Tarski theorem, the collection of fixpoints of a monotonic operator on a complete lattice forms itself a complete lattice. In particular, there is a *least* fixpoint and a *greatest* fixpoint. The notion of inductive definition can be seen as a computational way to introduce least fixpoints, and there ought to be a dual computational principle to introduce greatest fixpoints: *coinductive definitions*. We present

[4]: This restriction is stronger than usual positivity where the variable may only occur positively, *i.e.* only at the left of an even number of arrows.

below a standard approach. It should be noted that in the presence of equality, coinductive definitions can be defined within Martin-Löf type theory: see [59], [64] or the definition of the positivity predicate in [23].

We first treat an example, and briefly give the general principle. Let's define *streams* (infinite lists) over an arbitrary set. Streams over A cannot be defined as $(\mu X).A \times X$, since this is easily seen to be empty. Streams will be defined as a greatest fixpoint:

$$\begin{array}{lcl} \textsc{Stream}(A) & : & \mathbf{Set} \\ \textsc{Stream}(A) & \triangleq & (\nu X \in \mathbf{Set})\ \mathbf{data}\ \ \mathrm{Cons}(a \in A\,,\ t \in X) \end{array}$$

i.e. a stream is something of the form "$\mathrm{Cons}(a, s)$" where a is an element of A and s is a new stream. Thus, we can get as many elements of A as we want by looking deeper and deeper inside s. The rules associated to this definition are:

- $\dfrac{A : \mathbf{Set}}{\textsc{Stream}(A) : \mathbf{Set}}$ formation
- $\dfrac{X : \mathbf{Set} \qquad C : X \to A \times X \qquad x \in X}{\mathrm{coiter}(X, C, x) \in \textsc{Stream}(A)}$ introduction
- $\dfrac{s \in \textsc{Stream}(A)}{\mathrm{elim}(s) \in A \times \textsc{Stream}(A)}$ elimination

with the reduction rule:

$$\mathrm{elim}\big(\mathrm{coiter}(X, C, x)\big) \quad = \quad (a, t) \ \textit{ where } \ a \triangleq \pi_1\big(C(x)\big) \qquad t \triangleq \mathrm{coiter}\big(X, C, \pi_2\big(C(x)\big)\big)\ .$$

For the general case, suppose F is a strictly positive operator from **Set** to **Set**, so that we can in particular define an action of F on functions:[5]

$$\begin{array}{lcl} F & : & \mathbf{Set} \to \mathbf{Set} \\ & & X \ \mapsto F(X) \qquad \text{and} \qquad (f \in X \to Y) \ \mapsto \ \big(F_f \in F(X) \to F(Y)\big)\ . \end{array}$$

We can then define νF with the rules:

- $\dfrac{}{\nu F : \mathbf{Set}}$ formation
- $\dfrac{X : \mathbf{Set} \qquad C : X \to F(X) \qquad x \in X}{\mathrm{coiter}(X, C, x) \in \nu F}$ introduction
- $\dfrac{s \in \nu F}{\mathrm{elim}(s) \in F(\nu F)}$ elimination

with the reduction rule:

$$\mathrm{elim}\big(\mathrm{coiter}(X, C, x)\big) \quad = \quad F_f\big(C(x)\big) \ \textit{ where } \ f \in X \to \nu F \qquad f(y) \triangleq \mathrm{coiter}(X, C, y)\ .$$

Let's just mention that the introduction rule can be understood as the specification of an appropriate morphism from a specific coalgebra (the pair (X, C) in our example)

[5]: *i.e.* F is a covariant functor

to the (weakly) final coalgebra defined by $(\nu F, \mathrm{elim})$. We have $\mathrm{coiter}(X, C) \in X \to \nu F$ and the appropriate diagram commute by the reduction rule:

$$\mathrm{elim} \cdot \mathrm{coiter}(X, C) \quad = \quad F_{\mathrm{coiter}(X,C)} \cdot C \ .$$

Equality of terms in a coinductive type is usually identified with a notion of *bisimulation*. Since we will not need this equality, we skip the actual definition.[6]

1.1.4 Predicates

The Curry-Howard isomorphisms shows that type theory can be used as a logical framework. One can add "simple" mathematical objects like natural numbers, functions, etc. What about notions which are not datatypes in the usual sense? One such example is the notion of subset. We now show how to represent "subsets" in Martin-Löf type theory. This is done by developing a toolbox allowing one to manipulate subsets almost transparently without leaving the framework already described. Refer to [80] for the details.

Set theory usually identifies a subset with its characteristic function. We do the same here, though the notion of "characteristic function" is different. Instead of taking its values in $\{\mathrm{True}, \mathrm{False}\}$, the characteristic function will take its values in the collection of proposition: if $U \subseteq S$ and c_U is its characteristic function,

- *classical:* "$c_U(s)$" is the truth value of "$s \in U$;
- *constructive:* "$c_U(s)$" is the *proposition* "$s \in U$".

Martin-Löf type theory identifies a proposition with the set of its proofs:

- *Martin-Löf:* "$c_U(s)$" is the set (of proofs that) "$s \in U$".

Thus, we define:

▷ **Definition 1.1.1:** for any set S, define the collection of *predicates* on S as:

$$\mathcal{P}(S) \quad \triangleq \quad S \to \mathbf{Set} \ .$$

Similarly, if $\mathcal{S}$ is a proper type, define $\mathcal{P}(\mathcal{S}) \triangleq \mathcal{S} \to \mathbf{Set}$.
We write the predicate $\varphi : \mathcal{P}(S)$ more seductively as $\{s \epsilon S \mid \varphi(s)\}$.

This is reminiscent of the comprehension axiom scheme of ZF set theory[7] which guarantees that such a $\{s \in S \mid \varphi(s)\}$ does indeed form a set.

↬ Remark 4: it is tempting to define subsets of S as $S \to \{\mathrm{True}, \mathrm{False}\}$, but this would restrict to *computable subsets:* we would then need to deal with "algorithms" rather than formulas. As a naive example, consider the definition of the subset of even natural numbers:

- $E(n) \triangleq (\exists k)\, n = 2k$;
- $E(n) \triangleq \mathsf{case}\ n\ \mathsf{of}\ 0 \Rightarrow \mathrm{True}\ \ 1 \Rightarrow \mathrm{False}\ \ m+2 \Rightarrow E(m)$.

The first definition is obviously better as far as mathematics is concerned.

[6]: Using the bisimulation intuition, it is possible to encode coinductive types within predicative theory in the presence of extensional equality (see [59]), or strong forms of intensional equality (see [64]). The problem of coinductive definitions is thus pertinent only when restricting the use of equality...

[7]: $\forall x \exists y\ (\forall z\, z \epsilon y \Leftrightarrow (z \in x \wedge \varphi(z))$, for any proposition φ with at most one free variable

The type $\mathcal{P}(S)$ is very well-behaved and all the usual (simple) operations can be defined in a systematic way:

▷ **Definition 1.1.2:** let S be a set, and let X, Y be predicates on S; define:
- $s \varepsilon X$ is a synonym for $X(s)$;
- $X \subseteq Y$ is an abbreviation for $(\Pi s \epsilon S)\, X(s) \to Y(s)$, or using the logical notation: $(\forall s \epsilon S)\, s \varepsilon X \Rightarrow s \varepsilon Y$;
- $X \between Y$ (read "X overlaps Y") is an abbreviation for $(\Sigma s \epsilon S)\, X(s) \times Y(s)$, or using the logical notation: $(\exists s \epsilon S)\, s \varepsilon X \wedge s \varepsilon Y$;
- $\emptyset_S : \mathcal{P}(S) \triangleq (\lambda s \epsilon S).\emptyset$ *i.e.* no element $s \epsilon S$ belongs to $\emptyset_S$;
- $\mathrm{Full}_S : \mathcal{P}(S) \triangleq (\lambda s \epsilon S).\{*\}$ *i.e.* any $s \epsilon S$ belongs to Full_S;
- $X \cup Y \triangleq \{s \epsilon S \mid s \varepsilon X \vee s \varepsilon Y\}$;
- $X \cap Y \triangleq \{s \epsilon S \mid s \varepsilon X \wedge s \varepsilon Y\}$.

We can also define indexed extrema: if I : **Set** and if $X_i : \mathcal{P}(S)$ for all $i \epsilon I$,
- $\bigcap_{i \epsilon I} X_i \triangleq \{s \epsilon S \mid (\forall i \epsilon I)\, s \varepsilon X_i\}$;
- $\bigcup_{i \epsilon I} X_i \triangleq \{s \epsilon S \mid (\exists i \epsilon I)\, s \varepsilon X_i\}$.

The only point deserving some comment is the new "$\between$" symbol. It acts as a positive dual to inclusion: just like "$\subseteq$" hides a universal quantifier, so does "$\between$" hide an existential quantifier. Despite its simplicity, it seems that Giovanni Sambin was the first to stress its importance in constructive frameworks ([80]?).

The expected result holds almost trivially:

○ **Lemma 1.1.3:** for any type S, the proper type $\mathcal{P}(S)$ with $\subseteq$, $\emptyset_S$, Full_S, $\bigcup$ and $\bigcap$ is a complete[8] Heyting algebra.

↬ Remark 5: traditionally, a complete Heyting algebra $\mathcal{H}$ is a complete lattice s.t. for any a, the operation $a \wedge \_$ has a right adjoint $a \Rightarrow \_$. Impredicatively, this is equivalent to saying that $\mathcal{H}$ is a complete lattice satisfying the "infinite distributivity law": $a \wedge \bigvee_i b_i = \bigvee_i (a \wedge b_i)$. In our context, we cannot prove the equivalence, but $\mathcal{P}(S)$ is a Heyting algebra according to both definitions: put $U \Rightarrow V \triangleq \lambda s.U(s) \to V(s)$.

Note that there is a typographic distinction between elements of a set ("$s \epsilon S$") and elements of a predicate ("$s \varepsilon X$"). Those two assertions have a completely different nature: the first one is a judgment while the second one is a set.

1.1.5 Relations

Relations are special cases of predicates: a binary relation R between S_1 and S_2 is a predicate on the cartesian product $S_1 \times S_2$. We write $\mathbf{Rel}(S_1, S_2)$ as a synonym for $\mathcal{P}(S_1 \times S_2)$. Equivalently, using "curryfication", a relation between S_1 and S_2 is a function from S_1 to predicates of S_2:

$$(S_1 \times S_2) \to \mathbf{Set} \quad \simeq \quad S_1 \to (S_2 \to \mathbf{Set}) \quad \simeq \quad S_1 \to \mathcal{P}(S_2)\ .$$

Consequently, if R is a relation between S_1 and S_2, there are several ways to state that $s_1 \epsilon S_1$ and $s_2 \epsilon S_2$ are related through R:

[8]: where by "complete", we means that all *set-indexed* suprema exist. See section 4.1 for more details.

- $(s_1, s_2) \varepsilon R$;
- $s_2 \varepsilon R(s_1)$;
- $R(s_1, s_2)$.

We will usually prefer the first notation.

The structure of predicates lifts to relations: relations are ordered by (pointwise) inclusion and they have a structure of complete Heyting algebra. We have the following additional operations:

- converse;
- composition;
- transitive closure.

Those do not differ from the traditional definitions:

1) converse: if $R : \mathbf{Rel}(S_1, S_2)$, define $R^\sim : \mathbf{Rel}(S_2, S_1)$ as:

$$(s_2, s_1) \varepsilon R^\sim \quad \triangleq \quad (s_1, s_2) \varepsilon R \ ;$$

2) composition: for $R : \mathbf{Rel}(S_1, S_2)$ and $R' : \mathbf{Rel}(S_2, S_3)$, define $R' \cdot R : \mathbf{Rel}(S_1, S_3)$ as:

$$\begin{array}{lcl} (s_1, s_3) \varepsilon R' \cdot R & \triangleq & (\exists s_2 \epsilon S_2)\ (s_1, s_2) \varepsilon R \wedge (s_2, s_3) \varepsilon R' \\ & = & R(s_1) \between R'^\sim(s_3) \ ; \end{array}$$

3) transitive closure: for $R : \mathbf{Rel}(S, S)$, define $R^+ : \mathbf{Rel}(S, S)$ as $R \cup R \cdot R \cup R \cdot R \cdot R \ldots$ More precisely, using an inductive definition:

$$\begin{array}{lcll} (s, s') \varepsilon R^+ & \triangleq & (\mu X : \mathbf{Set}) & \\ & & \mathbf{data}\ \mathrm{Leaf}(r) & \textit{where}\ r \epsilon R(s, s') \\ & & \quad \mathrm{Cons}(s_i, r, r') & \textit{where}\ s_i \epsilon S \\ & & & \quad r \epsilon R(s, s_i) \\ & & & \quad r' \epsilon R^+(s_i, s') \ . \end{array}$$

We have:

- **Lemma 1.1.4:**
  - composition is associative; its neutral element is the equality;
  - converse is involutive and $(R \cdot R')^\sim = R'^\sim \cdot R^\sim$.

  If ones defines the reflexive transitive closure R^* to be $\mathbf{Eq} \cup R^+$, we obtain a Kleene algebra $\left(\mathbf{Rel}(S, S), \cup, \cdot, \_^*\right)$.

1.1.6 Families and Equality

§ *The Problem of Equality.* The equality relation on S: $\mathbf{Eq}_S \triangleq \{(s, s') \epsilon S \times S \mid s = s'\}$ is of utmost importance in mathematics. However, the notion of equality in (predicative) type theory is not clear at all. Real mathematical equality is extensional: we inherit it from set theory and its "extensionality axiom".[9] However, type theory with extensional equality has undecidable type checking! One solution is to make the proof objects for equality explicit. For this reason, Martin-Löf's early theories

[9]: "$\forall x \forall y\ (\forall z\ z \epsilon x \Leftrightarrow z \epsilon y) \Rightarrow x = y$"

had a notion of "intensional equality". This equality allows to keep type checking decidable, but the notion is at the least awkward!
The idea to have the following formation rule:

- $$\frac{S : \mathbf{Set} \qquad s \in S \qquad s' \in S}{\mathbf{Id}(S, s, s') : \mathbf{Set}}$$
  where "$s =_S s'$" is a synonym for the set $\mathbf{Id}(S, s, s')$
- with introduction rule: $\dfrac{S : \mathbf{Set} \qquad s \in S}{\mathrm{refl}(s) \in \mathbf{Id}(S, s, s)}$.

We refer to [68], [50] or [48] for the elimination and computation rules.

One particular problem with intensional equality comes with the following statement, called "uniqueness of identity proofs":

$$\mathbf{UIP_A}: \quad (\forall a_1, a_2 \in A)\,\big(\forall p_1, p_2 \in \mathbf{Id}_A(a_1, a_2)\big)\ \mathbf{Id}_{\mathbf{Id}_A(a_1,a_2)}(p_1, p_2) \ .$$

This principle asserts that two proofs that $a_1 =_A a_2$ must be equal. This is derivable in a type theory enriched with "pattern matching" but independent of the core type theory, see [50].

In practice, many usual datatypes have an implicit notion of equality which is definable if needs be. This suggest that one could reject the equality type and define it when necessary. However, what properties should an equality relation satisfy? There are two main possibilities:

- it should be reflexive and substitutive: if $s = s'$ and $P(s)$ then $P(s')$, *i.e.* the following type is inhabited: $(\forall s, s')\ \mathbf{Id}(s, s') \to P(s) \to P(s')$ for any $P : S \to \mathbf{Set}$. This is the notion of *datoid:* set with a reflexive / substitutive relation.
- it should be an equivalence relation: sets equipped with an equivalence relation are called *setoids.*[10] This allows to define quotient setoids but the proofs become very verbose: all the operation defined on a setoid must be *extensional,* that is, respect the internal equivalence.

The extreme position is not using equality! While this objective is infeasible in the long run, it allows to notice details that are otherwise invisible. For example, one handicap when rejecting identity is that we cannot talk about singleton subsets any more: if $s \in S$, we cannot form the predicate $\{s\}$ as it is defined as $\{s' \in S \mid s' =_S s\}$!

The approach taken here is mixed: in Part I, we will try to avoid equality as much as possible and make its use explicit when necessary. When needed, we informally use an extensional equality but it seems that everything can be done in intentional type theory. Since the second part of this thesis lives in classical mathematics, we will forget about this after page 110.

↬ Remark 6: in impredicative type theory, the problem of equality is not so problematic. Using impredicative quantification one can define the so-called "Leibniz" equality

$$x =_X y \quad \triangleq \quad (\forall P : X \to \mathbf{Set})\ P(x) \Rightarrow P(y) \ .$$

[10]: Those also correspond to Bishop's notion of *set.*

§ *Families.* Even if we refrain from using equality, it is quite natural to define the *family* $\{(s,s) \mid s \in S\}$, obviously representing equality. This suggest an alternative way to define subsets of S:

$$\{f(i) \mid i \in I\} \quad \textit{where} \; I : \mathbf{Set}, \; f \in I \to S$$

which corresponds to the axiom of replacement from ZF set theory. We put:

▷ **Definition 1.1.5:** let S be a set, define the collection of *families* over S as:

$$\mathcal{F}(S) \quad \triangleq \quad (\Sigma I{:}\mathbf{Set})\; I \to S \; .$$

Similarly, if $\mathcal{S}$ is a proper type, define $\mathcal{F}(\mathcal{S}) \triangleq (\Sigma I{:}\mathbf{Set})\; I \to \mathcal{S}$. We write the family (I,f) either as $\{f(i) \mid i \in I\}$ or as $\big(f(i)\big)_{i \in I}$.
Just like predicates, $\mathcal{F}(S)$ is always a proper type.

Equality of families is, for our purposes, equality of the underlying subsets: we do not care about multiplicities of elements.

$$(I,f) \approx (J,g) \quad \Leftrightarrow \quad \begin{cases} & (\exists \sigma{:}I \to J)\; f = g \cdot \sigma \\ \wedge & (\exists \rho{:}J \to I)\; g = f \cdot \rho \; . \end{cases}$$

(This of course requires equality on the underlying set.)

A somewhat moral difference between families and predicates is that the former are "concrete": they give a way to generate their elements while the latter are "abstract": they only give a property to be satisfied. In standard mathematical practice, the two notions coincide:

- a predicate $X = \{s \in S \mid \varphi(s)\}$ is translated into the family $\{s \mid s \varepsilon X\}$;
- conversely, a family $F = \{f(i) \mid i \in I\}$ is turned into $\{s \in S \mid (\exists i \epsilon I)\; s = f(i)\}$.

Written in type theory:

$$\begin{array}{lcl} \{s \in S \mid \varphi(s)\} & \mapsto & \{\pi_1(p) \mid p \in (\Sigma s \epsilon S)\; \varphi(s)\} \\ \{f(i) \mid i \in I\} & \mapsto & \{s \in S \mid (\Sigma i \epsilon I)\; s =_S f(i)\} \; . \end{array}$$

Since the translation from families to subset requires a notion of equality on S, those two notions become different when the notion of equality is questioned. Moreover, this translation between predicates and families doesn't work in either direction when considering "subsets" of a proper type $\mathcal{A}$:

- to go from a predicate to a family, we need to index the family by $(\Sigma A{:}\mathcal{A})\; \varphi(A)$, which is not a set;
- to go from a family to a predicate, we need to have a notion of equality on $\mathcal{A}$, which is in general impossible.

The two notions are thus definitely different when dealing with "big" types.

When $\mathcal{S}$ is a proper type equipped with an equality, it is sometimes possible to turn a predicate $\{X \mid \varphi(X)\}$ of $\mathcal{S}$ into a family $(Y_i)_{i \in I}$:

$$\{X \mid \varphi(X)\} \simeq (Y_i)_{i \in I} \quad \triangleq \quad (\forall X : \mathcal{S}) \quad \big(\varphi(X) \leftrightarrow (\exists i \epsilon I)\; X =_{\mathcal{S}} Y_i\big) \; .$$

(see on page 27 for a discussion about this type of quantification)

We then say that $\{X \mid \varphi(X)\}$ is a *set-indexed predicate.*

A technical difference is that as functors, the operators $\mathcal{P}(\_)$ and $\mathcal{F}(\_)$ have opposite variance:

▷ **Definition 1.1.6:** extend $\mathcal{P}(\_)$ and $\mathcal{F}(\_)$ to functors **Set** $\to$ **Type** or **Type** $\to$ **Type** in the following way:

$$(f \in X \to Y) \quad \mapsto \quad \begin{array}{l}\big(\mathcal{P}f : \mathcal{P}(Y) \to \mathcal{P}(X)\big)\\ \big(\mathcal{F}f : \mathcal{F}(X) \to \mathcal{F}(Y)\big)\end{array}$$

with:

$$\begin{array}{lll}\mathcal{P}f: & \{y \epsilon Y \mid \varphi(y)\} & \mapsto \big\{x \epsilon X \mid \varphi\big(f(x)\big)\big\}\\ \mathcal{F}f: & \big(g(i)\big)_{i\epsilon I} & \mapsto \big(g \cdot f(i)\big)_{i\epsilon I}\ .\end{array}$$

"$\mathcal{P}(\_)$" is thus contravariant while "$\mathcal{F}(\_)$" is covariant.

↬ Remark 7: a second difference between subsets and families lies in the fact that families allow to talk about *multiplicities* of elements: since we do not ask the indexing function to be injective (this would require equality), elements may appear many times in a family.

With equality, the structure of $\mathcal{F}(S)$ is the same as that of $\mathcal{P}(S)$: we have a (complete) Heyting algebra whose atoms are given by singletons. If we remove equality, the situation is more colorful:

- the empty family is defined as the unique family indexed by the empty type;
- the full family can only be defined on sets: $\mathrm{Full}_S \triangleq \{s \mid s \in S\}$;
- the union is defined as $\{f(i) \mid i \in I\} \cup \{g(j) \mid j \in J\} \triangleq \{\mathsf{case}(k, f, g) \mid k \in I + J\}$.[11]

On the other hand, we cannot say that an element belongs to a family, as it requires an equality:

$$s \text{ “}\varepsilon\text{”} \ \big\{f(i) \mid i \epsilon I\big\} \quad \Leftrightarrow \quad (\exists i \epsilon I)\ s = f(i)\ .$$

For the same reason, neither intersection, inclusion nor overlapping can be defined. However, we can define inclusion of a family in a subset and overlapping between a family and a subset:

- $\{f(i) \mid i \epsilon I\} \subseteq \{s \mid \varphi(s)\}$ iff $(\forall i \epsilon i)\ \varphi\big(f(i)\big)$;
- $\{f(i) \mid i \epsilon I\} \between \{s \mid \varphi(s)\}$ iff $(\exists i \epsilon I)\ \varphi\big(f(i)\big)$.

Finally, the singleton family $\{s\}$ is trivially defined as $\{s \mid i \epsilon I\}$ for any non empty I.

1.1.7 Transition Systems

Equipped with this new notion of subset, we take a second look at relations. We get two different notions:

- if we take $R : \mathcal{F}(A \times B)$, we get the notion of *span:* a triple (I, f, g) with $I :$ **Set**, $f \in I \to A$ and $g \in I \to B$;
- if we take $R : A \to \mathcal{F}(B)$, we get a notion which we call a *transition system*: a function from A to $\mathcal{F}(B)$.

Those two notions are isomorphic only with equality. Each has some advantages and drawbacks: for example, spans are "reversible" using a converse operation (just swapping the two "legs") but are not composable while transition systems are composable

[11]: This operation is also called *concatenation* of families.

but not reversible. We keep the notion of transition systems, as they are a simple version of interaction system, defined at the beginning of chapter 2.

Let's unfold the definition: if S_1 and S_2 are sets and $\nu : S_1 \to \mathcal{F}(S_2)$, we have that ν maps any $s_1 \in S_1$ to:

- an indexing set which we call $\nu.A(s_1)$;
- together with an indexing function $\nu.n_{s_1} \in \nu.A(s_1) \to S_2$.

The intuition we have about such a structure is the following:

- S_1 and S_2 are sets of *states;*
- $\nu.A(s_1)$ is the set of outgoing transitions (labels) from state s_1. We also use the term "actions" to denote elements of $\nu.A(s_1)$;
- $\nu.n_{s_1}(a)$ is the state (in S_2) reached after transition a from s_1. When the transition system is clear from the context, we write it $s_1[a]$.

Thus, a transition system from S_1 to S_2 is some kind of labeled directed bipartite graph, with all the transitions going from S_1 to S_2. When we use the same set of states S as the domain and the codomain, we get something which is very close to the usual notion of labeled transition system. The transition from s to $s[a]$ is usually denoted by $s \xrightarrow{a} s[a]$.

▷ **Definition 1.1.7:** a *transition system* from set S_1 to set S_2 is given by:
- a function $A : S_1 \to \mathbf{Set}$;
- and a function $n \in (s \epsilon S_1) \to A(s_1) \to S_2$.

Equivalently, a transition system from S_1 to S_2 is a function $\nu : S_1 \to \mathcal{F}(S_2)$. A transition system is called homogeneous when its domain and codomain are the same set.

To any transition system ν, we can associate a relation with the meaning that s_2 and s_1 are related iff there is a transition from s_1 to s_2:

▷ **Definition 1.1.8:** let $\nu = (A, n)$ be a transition system from S_1 to S_2, define a relation ν° on $S_2 \times S_1$ as:

$$(s_2, s_1)\ \varepsilon\ \nu^\circ \quad \Leftrightarrow \quad \big(\exists a \epsilon A(s_1)\big)\ s_1[a] = s_2\ .$$

This obviously requires equality on S_2.

§ *Operations on Transition Systems.* Without equality, the collection of transition systems enjoys different properties than the collection of relations. The three main points are:

- there is an identity $\mathbf{skip}_S : S \to \mathcal{F}(S)$ corresponding to equality;
- we can define the composition of $\nu : S_1 \to \mathcal{F}(S_2)$ and $\nu' : S_2 \to \mathcal{F}(S_3)$;
- we can define the reflexive and transitive closure ν^* of $\nu : S \to \mathcal{F}(S)$.

Thus, even without equality, the collection of transition systems will form a category. This is not the case for real relations since we need equality to define the identities. Similarly, the collection of homogeneous transition systems on a set will form a Kleene algebra while relations do not (we need equality to define the reflexive and transitive closure).

The concrete definitions go as follow:

1) the transition system $\mathbf{skip} : S \to \mathcal{F}(S)$:

$$\begin{array}{lcl} \mathbf{skip}_S.A(s) & \triangleq & \{*\} \\ \mathbf{skip}_S.n(s,*) & \triangleq & s\ ; \end{array}$$

2) the composition $\nu\,;\nu'$ of $\nu = (A,n) : S_1 \to \mathcal{F}(S_2)$ and $\nu' = (A',n') : S_2 \to \mathcal{F}(S_3)$:

$$\begin{array}{lcl} (\nu\,;\nu').A(s_1) & \triangleq & \big(\Sigma a \epsilon A(s_1)\big)\ A'\big(s_1[a]\big) \\ (\nu\,;\nu').n\big(s_1,(a,a')\big) & \triangleq & (s_1[a])[a']\ . \end{array}$$

Thus, an action in $\nu\,;\nu'$ is a pair of two consecutive actions: the first one in ν and the second in ν'.

3) the reflexive and transitive closure of $\nu : S \to \mathcal{F}(S)$ is defined as $\nu^* = (A^*, n^*)$:

$$\begin{array}{lcl} A^* & \triangleq & (\mu X : S \to \mathbf{Set})\ (\lambda s \in S) \\ & & \mathbf{data}\ \mathrm{Nil} \\ & & \quad \mathrm{Cons}(a,a')\ \mathit{where}\ a \in A(s) \\ & & \qquad\qquad\qquad\quad a' \in X(s[a]) \end{array}$$

and

$$\begin{array}{lcl} n^*(s,\mathrm{Nil}) & \triangleq & s \\ n^*\big(s,(a,a')\big) & \triangleq & n^*(s[a],a')\ . \end{array}$$

The definition of A^* uses a schema which is slightly more general than traditional inductive definitions: we define a predicate on S (a function from S to **Set**) rather than a set. In the Agda language, this definition would be simply written as

```
Astar (s::S) :: Set
  = data   Nil
         | Cons (a::A(s)) (a'::Astar(n(s,a)))
```

and the definition of n^* would be:

```
nstar(s::S , a'::Astar(s)) :: S
  = case a' of
      (Nil)       -> s
      (Cons a a') -> nstar( n(s,a) , a' )
```

Unfortunately, the converse of a transition system is not definable without equality: this makes transition system an asymmetric structure, adequate to model non reversible phenomena.

1.2 Impredicativity

Martin-Löf type theory is "predicative". This terminology which originated from Russel was supposed to mean that there were no "vicious circularities". The notion is difficult to formalize, and encompasses several concepts. Below is a tentative explanation of the kind of predicativity we have in mind.[12]

[12]: There is a second notion of predicativity related to the proof theoretic strength of a system. A system whose strength is greater than the ordinal Γ_0 is called "impredicative" (see [34]).

1.2.1 A Tentative Explanation of Predicativity

§ *Constructive Set Theory.* The perfect example of theory which is predicative is *constructive set theory* (CZF, see [6]) which originated from the work of John Myhill. The basic idea is to take the axioms of ZF set theory and:

- work with intuitionistic logic;
- remove the powerset axiom: "$\forall x \exists y \forall z\ z \subseteq x \leftrightarrow z \in y$".

One needs to modify other axioms in order to get a sensible system. For example, having the full foundation axiom allows to get both the powerset axiom and the excluded middle!

This gives a first "definition" for impredicativity: is impredicative a definition which uses the powerset axiom. Examples of such definitions are any definition using quantification over the collection of subsets of a set. Here is for example an impredicative definition of the vector space generated by a set of vectors:

> *the vector space generated by a set* V *of vectors is the smallest vector space containing* V*. More precisely,*
>
> $$\langle V \rangle \quad \triangleq \quad \bigcap \{W \mid W \textit{ is a vector space}\,,\ V \subseteq W\}\ .$$

Compare with the following predicative definition of the same concept:

> *the vector space generated by a set* V *of vectors is the set of all linear combinations of elements of* V*.*

The question of "why" the powerset axiom should be avoided is more a philosophical question than a real technical problem. As Peter Hancock once told me,

> *to me, the main lesson of about 150 years of mathematical logic is that the idea of a powerset is unfathomably mysterious. We can't even say anything reasonable about its cardinal! (generalized continuum hypothesis). How on earth can people feel they are on solid ground here??*

One problem when using such a system as a foundation for mathematics is that its logical strength is very low: CZF is below second order arithmetics, *i.e.* below analysis! The same applies to any other predicative system and in particular to Martin-Löf type theory.

§ *PI-1 Quantification.* Predicativity thus amounts to removing quantification over subsets. However, there are cases where such a quantification does make sense, even in a predicative framework: this is the case of $\mathbf{\Pi}^1_1$ quantification. The intuition is

$$\vdash (\forall X{:}\mathbf{Set})\ \varphi(X) \qquad \text{iff} \qquad X : \mathbf{Set} \vdash \varphi(X)\ .$$

Thus, while the expression "$(\forall X{:}\mathbf{Set})\ \varphi(X)$" is predicatively not a proposition, the *judgment* "$X : \mathbf{Set} \vdash \varphi(X)$" still makes sense. We will freely use such $\mathbf{\Pi}^1_1$ quantification.

However, we cannot nest such quantification with the other constructions. In particular, such a universal quantification should *never* occur negatively in a formula. The technical details of why $\mathbf{\Pi}^1_1$ is predicatively acceptable can be found in [21]: it is shown that the strength of $\mathbf{\Pi}^1_1$ is equivalent to the strength of iterated inductive definitions. A result inspired by this, but in a simpler context, can be found in [7] (see also [25]).

§ *Type Theory.* The notion of predicativity can also get a "precise" definition in the framework of type theories. Pure type systems ([11]) are type systems based on the pure λ-calculus with the following formation rules for function type:

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, a : A \vdash B : s_2}{\Gamma \vdash (\Pi x{:}A)\ B : s_3}$$

where (s_1, s_2, s_3) is a triple of *kinds*. A system is predicative if for all such rules, we have $s_2 \leqslant s_3$ and $s_1 \leqslant s_3$.

The typical example of such impredicative system is Girard system-F ([37]) or Reynolds polymorphic λ-calculus ([73]). It has a single rule

$$\frac{\mathbf{Type} : * \qquad \alpha : \mathbf{Type} \vdash \tau : \mathbf{Type}}{(\Pi\alpha{:}\mathbf{Type})\ \tau : \mathbf{Type}}$$

where the order on kinds is only $\mathbf{Type} < *$.

An alternative view on this condition is to say that a system is predicative if it has a well-founded notion of "subformula". It is simple to define subformulas for the simply typed λ-calculus, but the notion is not so simple for system-F. This is what made the proof of strong normalization so difficult and required the introduction of the notion of "candidats de réducibilité" by Jean-Yves Girard.

The problem with such a definition is that it is highly syntactical: there is no guarantee *a priori* that we cannot find an equivalent presentation of the type system satisfying the above condition. However, as far as system-F is concerned, we have a semantical counterpart stating that system-F has no "naive" model: [74] shows that system-F doesn't have a set theoretic model.

> ↬ Remark 8: this problem is however "easily" solved by using subtler models having notions of continuity or stability. We will in fact develop such a model (for second order linear logic and thus for system-F in chapter 8) when the question of predicativity will not bother us anymore...

§ *The Case of Martin-Löf Type Theory.* The type theory presented in section 1.1 can be seen as a predicative theory, as strong as it can get. It is possible to show ([60]) that this type theory (with intensional equality) enriched with a powerset constructor allows to get classical logic. Since Martin-Löf type theory with the excluded middle is as strong as ZFC set theory, this is very bad from a constructive point of view... However, as this work will show, Martin-Löf type theory is still a decent mathematical framework.

It should be noted that Peter Aczel has shown that Martin-Löf type theory, enriched with a notion of "generalized inductive definition" is "equivalent" to CZF, in the sense that they have the same expressivity ([5]). Working in one system or the other is thus just a matter of taste.

§ *Inductive Definitions.* The principle of inductive definitions can be seen as a "predicative" counterpart to the Knaster-Tarski theorem:

let F *be a monotonic operator on a complete lattice, then the collection of fixpoints of* F *is a complete lattice with least element* μF *and greatest element* νF. *Moreover, we have:*

$$\mu F = \bigwedge \{V \mid F(V) \leqslant V\}$$
$$\nu F = \bigvee \{V \mid V \leqslant F(V)\} \ .$$

Let's look how it applies to the inductive definition of lists over A: the operator in question is $F : \mathbf{Set} \to \mathbf{Set}$ with $F(X) \triangleq \{*\} + A \times X$. The inductive definition introduces an element $\textsc{List}(A)$ of $\mathbf{Set}$ such that:

- we have a function $F\big(\textsc{List}(A)\big) \to \textsc{List}(A)$, namely:

$$\begin{array}{lll} \lambda x \ . \ \mathbf{case} \ x \ \mathrm{of} \ * & \Rightarrow & \mathrm{Nil} \\ \qquad\qquad\qquad\quad (a, l) & \Rightarrow & \mathrm{Cons}(a, l) \ . \end{array}$$

  That is, $\textsc{List}(A)$ is a pre-fixpoint;
- this pre-fixpoint is smaller than any other pre-fixpoint: if $g \in F(X) \to X$, then we have a function $f \in \textsc{List}(A) \to X$:

$$\begin{array}{lcll} f(l) & \triangleq & \mathbf{case} \ l \ \mathrm{of} \ \mathrm{Nil} & \Rightarrow \ g(*) \\ & & \mathrm{Cons}(a, t) & \Rightarrow \ g\big((a, f(t))\big) \ . \end{array}$$

Thus, $\textsc{List}(A)$ is indeed a least pre-fixpoint. A similar analysis of coinductive definition is possible...

> ↬ REMARK 9: the technology of categories allows to be a little more precise: an inductive definition is not just a least pre-fixpoint, but it should have some structure. This is achieved by requiring that an inductive definition is an initial (=least) algebra (=pre-fixpoint) of a covariant (=monotonic) functor (=operator). Dually, coinductive definition are terminal coalgebras for covariant functors.

1.2.2 Impredicative Systems, Encodings

One of the nice features about impredicative systems is their expressive power. It is well-known that second-order universal quantification and implication allow to define all the intuitionistic connectives via the so-called Prawitz encoding:

- $\bot \triangleq (\forall \alpha) \ \alpha$;
- $F \wedge G \triangleq (\forall \alpha) \ (F \to G \to \alpha) \to \alpha$;
- $F \vee G \triangleq (\forall \alpha) \ \big((F \to \alpha) \to (G \to \alpha)\big) \to \alpha$;
- $(\exists \beta) \ F(\beta) \triangleq (\forall \alpha) \ \big((\forall \beta) \, (F(\beta) \to \alpha)\big) \to \alpha$.

We can in the same way define the product "$\times$" and sum "$+$" in system-F, or in other impredicative type theories. What is even more surprising is the fact that we can encode inductive definitions! Let's first look at the type of natural numbers:

$$\begin{array}{lcl} N & \triangleq & (\mu X : \mathbf{Set}) \ \mathbf{data} \ \mathrm{zero} \\ & & \qquad\qquad\quad \mathrm{succ}(n \in X) \ . \end{array}$$

In system-F, the definition of natural number is ("Church numerals"):

$$N \quad \triangleq \quad (\forall \alpha) \ \alpha \to (\alpha \to \alpha) \to \alpha \ .$$

The second definition may not be as intuitive, but is surely very elegant!

The general recipe to translate an inductive definition $(\mu X)\,F$ inside an impredicative theory is the following: if F is a monotonic functor on X,

$$(\mu X)\ F \quad \triangleq \quad (\forall \alpha)\ \big(F(\alpha) \to \alpha\big) \to \alpha\ .$$

The problem however, is to see the computational meaning of other impredicative quantifications. For example, what is the meaning of the following type:

$$\kappa \quad \triangleq \quad (\forall \alpha)\ \big((\alpha \to \alpha) \to \alpha\big) \to \alpha\ ?$$

This certainly doesn't correspond to an inductive definition since the functor F would be $F(X) = X \to X$, which is not monotonic.

↬ REMARK 10: since the type κ is Π^1_1, we have a simple description of terms inhabiting it: they are known as "Kierstead" λ-terms and their normal forms are:

$$(\lambda F)\quad F\Big((\lambda x_1 \epsilon \alpha)\ F\big((\lambda x_2 \epsilon \alpha)\ F\big((\lambda x_3 \epsilon \alpha)\ \ldots\ F(x_i)\ldots\big)\big)\Big)\ .$$

Similarly, but not as widely known is the fact that we can encode coinductive definitions in a completely dual way. For example, the type of streams over a set A would give, in system-F:

$$\textsc{Stream}(A) \quad \triangleq \quad (\exists \alpha)\ \alpha \times (\alpha \to A \times \alpha)\ ,$$

which gets hopelessly unreadable if one expands the definition. Just for fun, have a look at the unfolded definition:

$$(\forall \beta)((\forall \alpha)(((\forall \gamma)(\alpha \to (\alpha \to ((\forall \delta)(A \to \alpha \to \delta) \to \delta)) \to \gamma) \to \gamma) \to \beta)) \to \beta\ .$$

For the general case, we put:

$$\nu F \quad \triangleq \quad (\exists \alpha)\ \alpha \times \big(\alpha \to F(\alpha)\big)\ .$$

1.3 Classical Logic

Even if predicative type theory will be the framework of choice for the first part of this work, impredicative systems like the calculus of construction are still perfectly "constructive". However, adding new principles easily brings the full power of classical logic, thus taking us beyond any obvious constructive interpretation. The only place where we will really make use of such a principle in Part I is in section 3.5. The three most usual ways of getting classical logic from intuitionistic logic are:

- add the law of excluded middle $A \vee \neg A$ for all formula A;
- add the double negation $\neg\neg A \to A$ for all formula A;
- add Pierce's law $\big((A \to B) \to A\big) \to A$ for all formulas A and B.

It is a traditional exercise to show that they are equivalent in intuitionistic logic...

Due to the structure of our objects, classical logic is most adequately introduced with another principle: the contraposition of the axiom of choice. It was noted from

the very beginning by Per Martin-Löf that the following formulation of the axion of choice is *provable* in his system:[13]

$$\textbf{AC}: \quad (\forall x \epsilon X)\big(\exists y \epsilon Y(x)\big)\ \varphi(x,y) \Leftrightarrow \big(\exists f \epsilon (x \epsilon X) \to Y(x)\big)(\forall x \epsilon X)\ \varphi\big(x, f(x)\big)\ .$$

The classical dual of the axiom of choice is thus the following principle, which defies (so it seems) intuition:

$$\textbf{CtrAC}: \quad \big(\forall f \epsilon (x \epsilon X) \to Y(x)\big)(\exists x \epsilon X)\ \varphi\big(x, f(x)\big) \Leftrightarrow (\exists x \epsilon X)\big(\forall y \epsilon Y(x)\big)\ \varphi(x,y)\ .$$

The contraposition of the axiom of choice implies Pierce's law in the following manner: take two sets X and Y, and let $\varphi(x,y)$ be the singleton set $\{*\}$. We can simplify as follows:

- $(\Sigma x \epsilon X)\, \varphi(x,y)$ simplifies into X;
- $(\Pi f \epsilon X \to Y)\, X$ simplifies into $(X \to Y) \to X$;
- and on the right-hand-side, $(\Sigma x \epsilon X)(\Pi y\ \epsilon\ Y)\ \varphi(x,y)$ simplifies into X.

In the end, **CtrAC** becomes:

$$\big((X \to Y) \to X\big) \leftrightarrow X$$

which is just Pierce's law! To derive Pierce's law in a more traditional logical context, if A and B are formulas, define $X \triangleq \{x \epsilon \{*\} \mid A\}$ and $Y \triangleq \{k \epsilon \{*\} \mid B\}$. Wim Veldman apparently studies some constructive restrictions of **CtrAC** in [85].

While we are on this matter, it should be noted that Pierce's law does have a constructive interpretation in the form of the "call with current continuation" operation present in the LISP programming language. Such interpretations were first studied by Griffin in [40]. However, this computational interpretation doesn't lift to strong frameworks like Martin-Löf type theory: it is incompatible with the axiom of choice ([47] or [58]). The best way to give a constructive analysis of "**AC** + classical logic" seems to use a game interpretation and a double negation translation of **AC** ([14]), or to use a cc operator, together with a new operator like a "clock" ([57]).

1.4 Notations and Conventions

To finish this introduction, let's try to give some notation. Since type theory tends to be very verbose, it is important to decide on implicit conventions to simplify expressions without loosing information...

- Elements of **Type** are denoted by calligraphic capital letter like $\mathcal{A}$. One notable exception is the proper type of all sets, written **Set**.
- Elements of a proper type $\mathcal{A}$ are written as capital, roman letters: for example, we have S : **Set**, U : $\mathcal{P}(S)$ and R : **Rel**(S,T).
- For actual sets, we try to keep letters from the beginning of the alphabet together with S (set of states). In order to get enough diversity, we put decorations on the names: A_4, S', etc. .
- For predicates, we usually use U, V and W.

[13]: The fact that **AC** is constructively valid had been noted before by Howard and Bishop.

- For relations, we almost exclusively use R, with decorations.
- Element of a set are themselves written in small, roman letters. We have $s \in S$, $u \in U(s)$ and $r \in R(s,t)$. When the sets have decoration, we try to keep them on the names of elements, like $s_2 \in S_2$.
- Variable objects (sets or their elements) are usually written with letters from the end of the alphabet: x, y, ... or X, Y, ...

We apply (un)currification ($A \to B \to C \simeq A \times B \to C$) transparently. In particular, if f is of type $A_1 \to A_2 \to A_3 \to B$, $f(a_1, a_2, a_3)$ is a notation for the repeated application $((f\ a_1)\ a_2)\ a_3$. This is to keep standard mathematical notation rather than type theoretic notation which is not easily parsed.

The symbol $\triangleq$ is used for definitions: "name $\triangleq$ definition".

For technical reason, it wasn't possible to keep a consistent notation across the whole thesis. Some of the notation becomes obsolete in a classical setting and we will change some of the conventions in the second part of this work. Those changes will be explained when appropriate (page 111).

Part I

General Theory and Applications

2 Interaction Systems

The object of study of this thesis is a structure called *interaction system* (Peter Hancock's terminology). Because of its "genericity", this structure has been introduced (with different degrees of generality) by many authors under different names and with many different purposes. Let's mention some of the interesting uses we have seen:

- [82]: Alfred Tarski seems to have used a finitary version as Post-systems;
- [41]: Kripke like semantics for intuitionistic logic;
- [4]: abstract description of generalized inductive definitions;
- [29]: complete model for intuitionistic logic;
- [70]: grammars with ideas of applications to linguistic;
- [68]: justification for families of mutually dependent inductive types;
- [23] and [27]: description of inductively generated formal topologies;
- [44], [45] and [66]: abstract description of a "programming language";
- [15] and [24] as a topological model for geometric theories;
- [46] and [64] as representations for "polynomial (set-based) functors".

To a lesser extend, one can also see any specie of *games* (like in [22], [51] or [3]) as a variant of interaction systems. The converse is also true (*i.e.* interaction systems are a kind of two players game) but the developments differ in many ways.

Our first motivation ([52]) for looking at interaction systems came from Peter Hancock and Anton Setzer who used interaction systems to model interactive programs ([44], [45] and [66]). The notion of interaction systems as we use it was brought to its present form by Anton Setzer and Peter Hancock.

This first chapter presents, in a non-technical way, the basic structure of interaction systems, their morphisms and several properties they enjoy.

2.1 Basic Definitions and Examples

2.1.1 Interaction Systems

Interaction system bear many similarities with the simple notion of transition system defined in section 1.1.7, but there now are two kinds of transitions:

▷ **Definition 2.1.1:** let S_1 and S_2 be sets; an *interaction system from* S_1 *to* S_2 is given by the following data:
- a function $A : S_1 \to \mathbf{Set}$;
- a function $D : (s_1 \epsilon S_1) \to A(s_1) \to \mathbf{Set}$;
- a function $n \epsilon (s_1 \epsilon S_1) \to \big(a \epsilon A(s_1)\big) \to D(s_1, a) \to S_2$.

If w is an interaction system, we name its components $w.A$, $w.D$ and $w.n$. When no confusion arises, we drop the "$w.$" and simply write A, D and n, possibly with decorations. When the interaction system is clear from the context, we write $s[a/d]$ instead of $n(s, a, d)$.

An interaction system from S to S is called *homogeneous*. In this case, we say that w is an interaction system *on* S.

Since most of this work deals with homogeneous systems, we implicitly assume that the "domain" and "codomain" of the interaction system are identical.

A first intuition about interaction systems is that:
- the set S is a set of *states;*
- if s is a state, $A(s)$ is the set of possible *actions* available in state s;
- if a is an action in state s, the set $D(s, a)$ is the set of possible *reactions* to a;
- finally, if d is a reaction to action a, the state $s[a/d]$ is the *new state* after the action a had been "performed" and reaction d has been "received".

More specific interpretations will be given in section 2.1.2

In practice, just like for transition systems, we might like to have a notion of "initial states": states from which interaction can be initiated.

> ↬ Remark 11: the main reason we do not bother with initial states is simplicity. Having initial states naturally brings forward the problem of *reachability* of states: initialized interaction systems ought to be identified when their "connected component containing the initial state" coincide (whatever that really means), *i.e.* we do not really care about unreachable states . Dealing with simple interaction systems allows to evacuate this problem, at least for the time being...

Following standard (??) terminology in computer science, we call the entity choosing the actions the *Angel,* hence the A. For the sake of simplicity, the Angel will be a female and referred to as a "she". The entity responding to the actions, *i.e.* the entity choosing the reactions is called the *Demon,* hence the D. The Demon will be a male and referred to as a "he". Depending on the audience's background, they could have been named Player and Opponent, Eloise and Abelard, Alice and Bob, Master and Slave, Client and Server, System and Environment, alpha and beta, Arthur and Berta etc.

§ *Structural Isomorphism.* There is a natural notion of "structural isomorphism" for interaction systems: say that two interaction systems are *structurally isomorphic* if they are isomorphic component-wise:

▷ **Definition 2.1.2:** if w and w' are interaction systems respectively on S and on S', we say that w and w' are *structurally isomorphic* if we have the following:
- an isomorphism $\sigma \epsilon S \xrightarrow{\sim} S'$;
- for each $s \epsilon S$, an isomorphism $\alpha_s \epsilon A(s) \xrightarrow{\sim} A'\big(\sigma(s)\big)$;

- for each $a \in A(s)$, an isomorphism $\delta_{s,a} \in D(s,a) \xrightarrow{\sim} D'\big(\sigma(s), \alpha_s(a)\big)$

such that

$$\sigma\big(n(s,a,d)\big) \quad = \quad n'\big(\sigma(s), \alpha_s(a), \delta_{s,a}(d)\big) \ .$$

We write $w \approx w'$ to mean that w is structurally isomorphic to w'. This relation is obviously an equivalence relation.

Of course, this definition requires equality.

This notion of isomorphism is too fine for most purposes and section 2.4 is devoted to finding a "good" notion of (iso)morphism between interaction systems. Sections 2.6.2 and 3.3.1 will latter introduce other notions of morphism, adequate for some particular applications of interaction systems.

§ *An Alternative View.* Just like transition systems, interaction systems have a more concise, abstract definition using families. This allows to see interaction systems as a higher-order variation on transition systems.

- **Lemma 2.1.3:** an interaction system from S_1 to S_2 is equivalent to a function $w : S_1 \to \mathcal{F}^2(S_2)$.

If we recall that transition systems are concrete representations for relations, we can see interaction systems as concrete representations for functions $S_1 \to \mathcal{P}^2(S_2)$. Such functions are called *predicate transformers* and will be introduced in details in section 2.5.

2.1.2 Many Possible Interpretations

Definition 2.1.1 is very general, and many situations can be modeled, or at least approximated by interaction systems. Here is a (non exhaustive) list.

§ *Physical world.* In the most naive interpretation, S represents the set of physical states of a system. It could for example consist of physical quantities like temperature, pressure and the like. The Angel represents any entity which can try to influence the world described by S. The Demon is then given by the laws of physics. The fact that their might be many possible reactions comes from the fact that the state may not describe everything (flipping a coin is non-deterministic if the knowledge about the environment is not precise enough) or because we have a level of details such that quantum phenomena do occur.

§ *Games.* A natural interpretation is to see an interaction system as a *game*. For example, the game of chess is easily described by an interaction system: S will be the set of configurations of the board, $A(s)$ is the set of possible moves for White in state s, and $D(s,a)$ is the set of possible moves for Black after White's move a. The new state $s[a/d]$ is just the state of the board obtained from s after the pair of moves White-a/Black-d.

This kind of symmetric games might however be more adequately described using a symmetric variant of interaction systems called *Janus system:*

▷ **Definition 2.1.4:** a *Janus system* on states S_A and S_D is given by:
- a function $A : S_A \to \mathbf{Set}$;
- a function $n_A \in (s \epsilon S_A) \to A(s) \to S_D$;
- a function $D : S_D \to \mathbf{Set}$;
- a function $n_D \in (s \epsilon S_D) \to D(s) \to S_A$.

Equivalently, a Janus system on S_A and S_D is given by a pair of opposite transition systems:

$$\nu_A : S_A \to \mathcal{F}(S_D) \quad \text{and} \quad \nu_D : S_D \to \mathcal{F}(S_A) \ .$$

The idea is that the Angel and Demon have their own, disjoint sets of states and that they alternate moves. This notion is being studied by Markus Michelbrinks in Swansea; it is also at the heart of Giovanni Sambin's work on "basic pairs" ([78]).

> ↬ Remark 12: many logicians used to games semantics are taken aback by the asymmetric nature of interaction systems. Most would prefer working with the more symmetric Janus systems. Anticipating on the second part of this thesis, let's explain why the notion of Janus system is inadequate for our purposes. While it is simple enough to define connectives like $\oplus$ and $\otimes$ on Janus systems, the notion of morphism is not as obvious.
>
> One argument invoked is that negation is very easy in Janus system: just change the Angel and the Demon:
>
> $$\eta = (S_A, S_D, A, D, n_A, n_D) \mapsto \eta^\perp \triangleq (S_D, S_A, D, A, n_D, n_D) \ .$$
>
> However, it is difficult to see how the above operation could achieve the goal of changing an Angel strategy into a Demon strategy: if a strategy for the Angel in η is of the form $(\exists a_1)(\forall d_1)(\exists a_2)(\forall d_2)\ldots$, then a strategy for the Angel in $\eta^\perp$ will be of the form $(\exists d_1)(\forall a_1)(\exists d_2)(\forall a_1)\ldots$ This is a strategy for the Demon in η *if we allow the Demon to start.* This is in essence the reason of the presence of "dummy moves" in negation in many games semantics. This makes realizing $F^{\perp\perp} = F$ not trivial. One very nice feature of our negation operator will be that *it doesn't change the set of state,* while still interchanging the Angel and the Demon strategies.
>
> Finally, even if all those problems are set aside, one cannot ignore the fact that, with the synchronous definition of tensor, this negation would make the category compact closed,[1] *i.e.* the multiplicatives would collapse into a single connective.

One last reason why this structure is inadequate for our purposes is that is not at all obvious how to define the reflexive closure of a Janus system.

§ *Knowledge.* An interpretation which turns out to be interesting in the sequel is to see $s \in S$ as a state of *knowledge* the Angel has about the world. She can try to extend her knowledge by asking questions. Responses come of course from the Demon. A response will make her knowledge increase. The fact that the state "increases" with time will be quite important when we talk about *localized* interaction systems in section 4.3. (See also section 4.4.1.)

§ *Resources.* We can easily devise a "non-monotonic" variant of the previous interaction system: S doesn't represent knowledge about the world but *resources* at the disposal of the Angel. She can use those resources to conduct experiments which

[1]: It is possible to use them to construct a non-trivial $\star$-autonomous category see [65], but the intuitions are entirely different.

may have different outcomes. What is produced by those experiments is added to the available resources, but what was used ... is used. This is in essence the subject of section 4.4.2.

§ *Post Systems.* An interaction system can be seen as a generalized discharge-free deduction system (a Post system): given a proposition to prove, there can be many different inference rules one can apply (Angel's choice). For one such inference rule, there are several premises one needs to prove (Demon's choice).This is the notion used by Peter Aczel under the name *rule set* to describe generalized inductive definitions ([4]): the Angel chooses a particular constructor and the Demon responds by choosing one argument for this constructor. The notion of *strategy* for the Angel is quite important since it is linked with the notion of proof and term...

§ *Grammars.* One other idea is to use interaction systems to model grammars: a state is a non-terminal token, an action for the Angel is a rule with this token as the left hand side and a reaction is one of the non-terminal tokens appearing on the right hand side of the rule. This was the original motivation for introducing interaction systems by Kent Petersson and Dan Synek in [70]. They defined a scheme for special inductive definitions relative to the type signature:

$$\begin{array}{lll} A & : \mathbf{Set} & \\ B(a) & : \mathbf{Set} & \textit{where } a \in A \\ C(a,b) & : \mathbf{Set} & \textit{where } a \in A, b \in B(a) \\ d(a,b,c) \in A & & \textit{where } a \in A, b \in B(a), c \in C(a,b) \end{array}$$

i.e. relative to a pair (A, w) of a set A and an interaction system on A. They called the resulting inductive type "tree set".

§ *Interfaces.* This will somehow be the main "concrete" example: describing the services offered by an interface for programming. Since we will describe interfaces in details in section 2.6, we do not go into the details for the moment.

§ *Topological Space.* This interpretation is quite different in nature. Since this will be the subject of section 4.2, we omit the details and simply say that a state can be seen as an element of a *base* for a topological space and that the actions and reactions give the possible ways to *cover* a particular basic open set by other basic open sets.

Here is table summarizing all this:

$s \in S$	$a \in A(s)$	$d \in D(s,a)$	$n(s,a,d)$
physical state	action	reaction	next state
state of board	move	counter-move	next state
state of knowledge	question	answer	new knowledge
resources	experiment	outcome	new resources
proposition	inference rule	premise	new proposition
inductive type	constructor	argument	type of argument
non-terminal	production rule	RHS token	new token
state	command	response	new state
basic open	covering	index for...	new basic open

2.2 Combining Interaction Systems

Given two interaction systems, there are natural ways to combine then. We give the most obvious ones below.

§ *Disjoint Sum.* A very simple thing to do is to make the "disjoint union" of the interaction systems, reminiscent of the disjoint sum of two labeled transition systems:

▷ **Definition 2.2.1:** let w_1 and w_2 be interaction systems on S_1 and S_2. Define the interaction system $w_1 \oplus w_2$ on $S_1 + S_2$ as:

$$\begin{array}{lcl}
(w_1 \oplus w_2).A(s) & \triangleq & \textbf{case } s \text{ of } \begin{array}{l} \mathrm{inl}(s_1) \Rightarrow A_1(s_1) \\ \mathrm{inr}(s_2) \Rightarrow A_2(s_2) \end{array} \\
(w_1 \oplus w_2).D(s,a) & \triangleq & \textbf{case } s \text{ of } \begin{array}{l} \mathrm{inl}(s_1) \Rightarrow D_1(s_1,a) \\ \mathrm{inr}(s_2) \Rightarrow D_2(s_2,a) \end{array} \\
(w_1 \oplus w_2).n(s,a,d) & \triangleq & \textbf{case } s \text{ of } \begin{array}{l} \mathrm{inl}(s_1) \Rightarrow s_1[a/d] \\ \mathrm{inr}(s_2) \Rightarrow s_2[a/d] \ . \end{array}
\end{array}$$

We call $w_1 \oplus w_2$ the *disjoint sum* of w_1 and w_2.

The interaction system $w_1 \oplus w_2$ is quite boring: interaction takes place either in w_1 or in w_2, but always on the same side!

§ *Synchronous Tensor.* On the other side of the spectrum, we can impose the Angel and the Demon to play on both sides all the time. This is a kind of "lock-step" synchronous product, similar to the operation with the same name defined in [67] for labeled transition systems.

▷ **Definition 2.2.2:** suppose w_1 and w_2 are interaction systems on S_1 and S_2; define $w_1 \otimes w_2$ to be the following interaction system on $S_1 \times S_2$:

$$\begin{array}{lcl}
(w_1 \otimes w_2).A\big((s_1,s_2)\big) & \triangleq & A_1(s_1) \times A_2(s_2) \\
(w_1 \otimes w_2).D\big((s_1,s_2),(a_1,a_2)\big) & \triangleq & D_1(s_1,a_1) \times D_2(s_2,d_2) \\
(w_1 \otimes w_2).n\big((s_1,s_2),(a_1,a_2),(d_1,d_2)\big) & \triangleq & \big(s_1[a_1/d_1], s_2[a_2/d_2]\big) \ .
\end{array}$$

We call $w_1 \otimes w_2$ the "*synchronous tensor*" of w_1 and w_2.

This is very restrictive since a failure to play on one side yields a failure to play in the synchronous tensor. Because of its algebraic properties (see section 3.4), this operation will be central in the second part of this work where it will model the tensor of linear logic.

§ *Angelic and Demonic Tensors.* Somewhere between "interaction only on one side" and "interaction always on both sides" lie two other possibilities:

- interaction on one side at a time, the Angel decides which;
- interaction on one side at a time, the Demon decides which.

This means that we can interleave interaction in w_1 and w_2. Such an interaction is biased either toward the Angel, in which case we talk about the Angelic tensor, or toward the Demon, in which case we talk about the Demonic tensor.

▷ **Definition 2.2.3:** if w_1 and w_2 are interaction systems on S_1 and S_2, define $w_0 \boxplus w_2$ and $w_1 \boxtimes w_2$ on $S_1 \times S_2$ with components $(A_\boxplus, D_\boxplus, n_\boxplus)$ and $(A_\boxtimes, D_\boxtimes, n_\boxtimes)$:

$$\begin{array}{lcl}
A_\boxplus\big((s_1,s_2)\big) & \triangleq & A_1(s_1) + A_2(s_2) \\
D_\boxplus\big((s_1,s_2),a)\big) & \triangleq & \textsf{case } a \textsf{ of } \mathrm{inl}(a_1) \Rightarrow D_1(s_1,a_1) \\
 & & \qquad\qquad\quad \mathrm{inr}(a_2) \Rightarrow D_2(s_2,a_2) \\
n_\boxplus\big((s_1,s_2),a,d)\big) & \triangleq & \textsf{case } a \textsf{ of } \mathrm{inl}(a_1) \Rightarrow (s_1[a_1/d], s_2) \\
 & & \qquad\qquad\quad \mathrm{inr}(a_2) \Rightarrow (s_1, s_2[a_2/d])
\end{array}$$

and

$$\begin{array}{lcl}
A_\boxtimes\big((s_1,s_2)\big) & \triangleq & A_1(s_1) \times A_2(s_2) \\
D_\boxtimes\big((s_1,s_2),(a_1,a_2)\big) & \triangleq & D_1(s_1,a_1) + D_2(s_2,d_2) \\
n_\boxtimes\big((s_1,s_2),(a_1,a_2),d\big) & \triangleq & \textsf{case } d \textsf{ of } \mathrm{inl}(d_1) \Rightarrow (s_1[a_1/d_1], s_2) \\
 & & \qquad\qquad\quad \mathrm{inr}(d_2) \Rightarrow (s_1, s_2[a_2/d_2])\ .
\end{array}$$

The first one is called the *Angelic tensor* of w_1 and w_2 and the second one is called the *Demonic tensor* of w_1 and w_2.

Note that in a $\boxtimes$, the Angel needs not be consistent in her choice of moves: if she chooses (a_1, a_2) and the Demon responds with d_1, then for the next interaction, the Angels may choose (a_1', a_2') where $a_2' \neq a_2$.

All of $\oplus$, $\otimes$, $\boxplus$ and $\boxtimes$ can be defined as well for heterogeneous systems.

§ *Obvious Properties.* Those four operations are commutative and associative in a strong sense:

○ **Lemma 2.2.4:** for any interaction systems w_1, w_2 and w_3, we have:
- $w_1 \clubsuit (w_2 \clubsuit w_3)$ is structurally isomorphic to $(w_1 \clubsuit w_2) \clubsuit w_3$;
- $w_1 \clubsuit w_2$ is structurally isomorphic to $w_2 \clubsuit w_1$;

where $\clubsuit$ is one of $\oplus$, $\otimes$, $\boxplus$ or $\boxtimes$.
Moreover, $\otimes$ distribute over $\oplus$: $w \otimes (w_1 \oplus w_2) \approx (w \otimes w_1) \oplus (w \otimes w_2)$.

They all have a neutral element:

▷ **Definition 2.2.5:** define the following interaction systems:
- **null** is the unique interaction system on the empty set of state;
- **skip** is the following interaction system on $\{*\}$:

$$\begin{array}{lcll}
\textbf{skip}: & A(*) & \triangleq & \{*\} \\
 & D(*,*) & \triangleq & \{*\} \\
 & n(*,*,*) & \triangleq & * \quad ;
\end{array}$$

- **abort** and **magic** are the following interaction systems on $S \triangleq \{*\}$:

$$\begin{array}{lcllcllcl}
\textbf{abort}: & A(*) & \triangleq & \emptyset & \text{and} & \textbf{magic}: & A(*) & \triangleq & \{*\} \\
 & D(*,_) & \triangleq & _ & & & D(*,*) & \triangleq & \emptyset \\
 & n(*,_,_) & \triangleq & _ & & & n(*,*,_) & \triangleq & _\ .
\end{array}$$

Those constants have natural interpretations in terms of interaction:

- **null** is probably the most boring interaction system: there are no state!
- **abort** is strongly "winning" for the Demon: the Angel cannot move, interaction doesn't even start!
- **magic** is strongly "winning" for the Angel: the Demon cannot answer! The system stops ("hangs") after the first action.
- **skip** is the second most boring interaction system after **null**. Interaction doesn't bring any information because there is only one way to interact. It is the perfect example of a "stable" system. This system, as simple as it may seem enjoys a highly non-trivial property: see section 3.5.

Those interaction systems satisfy:

- **Lemma 2.2.6:** for any interaction system w, we have
  - $w \oplus \mathbf{null} \approx w$;
  - $w \otimes \mathbf{skip} \approx w$;
  - $w \boxplus \mathbf{abort} \approx w$;
  - $w \boxtimes \mathbf{magic} \approx w$.

2.3 Sequential Composition and Iteration

The reason we are mainly interested in *homogeneous* interaction systems is that interaction can be iterated: after interaction (a/d) from s, the Angel can chose a new action in $A(s[a/d])$ to which the Demon can respond, etc. We omit parenthesis and write $s[a_1/d_1][a_2/d_2]\dots[a_n/d_n]$ for the state reached after the sequence of interaction $(a_1/d_1, \dots, a_n/d_n)$. Such a sequence of interaction is usually called a *trace*.

This section deals with this idea of iteration by defining, for any interaction system w on S, new interaction systems w^* and w^∞ on S for which actions are "sequences" of actions and reactions are "sequences" of reactions.

2.3.1 Sequential Composition

The first step is to define a notion of *sequential composition* $w_1 ; w_2$ for interaction systems. The idea is simply that an interaction pair (action/reaction) in $w_1 ; w_2$ will be a pair of interactions $(a_1/d_1, a_2/d_2)$ where a_2/d_2 follows a_1/d_1:

- **Definition 2.3.1:** suppose w_1 and w_2 are interaction systems respectively from S_1 to S_2 and from S_2 to S_3; define $w_1 ; w_2$ to be the following interaction system, from S_1 to S_3:
  - $(w_1 ; w_2).A(s_1) \triangleq \big(\Sigma a_1 \epsilon A_1(s_1)\big)\big(\Pi d_1 \epsilon D_1(s_1, a_1)\big) A_2(s_1[a_1/d_1])$;
  - $(w_1 ; w_2).D\big(s_1, (a_1, k)\big) \triangleq \big(\Sigma d_1 \epsilon D_1(s_1, a_1)\big) D_2\big(s_1[a_1/d_1], k(d_1)\big)$;
  - $(w_1 ; w_2).n\big(s_1, (a_1, k), (d_1, d_2)\big) \triangleq s_1[a_1/d_1][k(d_1)/d_2]$.

  The interaction system $w_1 ; w_2$ is called the *sequential composition* of w_1 and w_2.

This definition certainly looks frightening but is in fact quite natural: recall that Σ and Π respectively denote pairs and functions,

- an action from state s_1 in $w_1 ; w_2$ is given by:
  - an action a_1 in $A_1(s_1)$,
  - together with a continuation k mapping any reaction d_1 to a_1 to a new action a_2 from state $s_1[a_1/d_1]$ (a "conditional" action in w_2);
- a reaction to such a pair is given by:
  - a reaction d_1 to the first action a_1,
  - and a reaction d_2 to the action obtained from the continuation $k(d_1)$,
- the resulting state is simply $s_1[a_1/d_1][k(d_1)/d_2]$.

Thus, a single move for the Angel in $w_1 ; w_2$ is a *strategy* to play one move in w_1 followed by one move in w_2.

This operation is naturally associative but definitely not commutative. For any set S, there is an interaction system $\mathbf{skip}_S$ which is neutral on both sides:

$$\begin{array}{lcll} \mathbf{skip}_S ; w & \approx & w & \textit{where } w : S \to \mathcal{F}^2(S') \\ w ; \mathbf{skip}_S & \approx & w & \textit{where } w : S' \to \mathcal{F}^2(S) \ , \end{array}$$

where $\mathbf{skip}_S$ is the following interaction system on S:

$$\begin{array}{lll} \mathbf{skip} : \ A(s) & \triangleq & \{*\} \\ \quad D(s,*) & \triangleq & \{*\} \\ \quad n(s,*,*) & \triangleq & s \ . \end{array}$$

This small section can be summarized by saying that we have a category where objects are sets and morphisms interaction systems.

2.3.2 Factorization of Interaction Systems

The very notion of interaction system is sequential: the Demon's reactions *follow* the Angel's actions. As we will show, it is possible to see any interaction system as the sequential composition of two simple interaction systems: one for the Angel and one for the Demon.

§ *Angelic and Demonic Updates.* First, let's see how we can lift a transition system to an interaction system:

▷ **Definition 2.3.2:** suppose $\nu = (A, n)$ is a transition system from S_1 to S_2; define the *Angelic update* $\langle\nu\rangle$ of ν to be the following interaction system from S_1 to S_2:

$$\begin{array}{lll} \langle\nu\rangle.A(s_1) & \triangleq & A(s_1) \\ \langle\nu\rangle.D(s_1, a_1) & \triangleq & \{*\} \\ \langle\nu\rangle.n(s_1, a_1, *) & \triangleq & n(s_1, a_1) \ . \end{array}$$

Dually, define the *Demonic update* $[\nu]$ of ν to be the following interaction system from S_1 to S_2:

$$\begin{array}{lll} [\nu].A(s_1) & \triangleq & \{*\} \\ [\nu].D(s_1, *) & \triangleq & A(s_1) \\ [\nu].n(s_1, *, a_1) & \triangleq & n(s_1, a_1) \ . \end{array}$$

An "update" of a transition system amounts to giving a name (Angel or Demon) to the player choosing the transitions.

§ *Factorization.* We just saw how to lift a transition system to an interaction system, and we saw that there were two ways to do so. We now do the converse and show how to dismantle an interaction system into two transition systems. If $w = (A, D, n)$ is an interaction system from S_1 to S_2, define:

- a transition system w_A from S_1 to $(\Sigma s_1 \epsilon S_1)\, A(s_1)$:

$$\begin{array}{lcl} w_A.A(s_1) & \triangleq & A(s_1) \\ w_A.n(s_1, a_1) & \triangleq & (s_1, a_1)\ ; \end{array}$$

- and a transition system w_D from $(\Sigma s_1 \epsilon S_1)\, A(s_1)$ to S_2:

$$\begin{array}{lcl} w_D.A\big((s_1, a_1)\big) & \triangleq & D(s_1, a_1) \\ w_D.n\big((s_1, a_1), d_1\big) & \triangleq & n(s_1, a_1, d_1)\ . \end{array}$$

This operation of "surgery" is somewhat right inverse to the previous lifting operations:

◇ **Proposition 2.3.3:** *for any homogeneous interaction system w, we have $w \approx \langle w_A \rangle\ ;\ [w_D]$.*

proof: since the set of states of w and $\langle w_A \rangle\ ;\ [w_D]$ are the same, it is enough to show that the sets of actions and reactions are isomorphic, in a way that is compatible with the next state functions: for the actions,

$$\begin{array}{lcl} (\langle w_A \rangle\ ;\ [w_D]).A(s) & = & \big(\Sigma a \epsilon \langle w_A \rangle.A(s)\big) \\ & & \quad \big(d \epsilon \langle w_A \rangle.D(a)\big) \to [w_D].A\big(w_A.n(s, a, d)\big) \\ & = & \big(\Sigma a \epsilon A(s)\big)\ \{*\} \to \{*\} \\ & = & A(s) \times \big(\{*\} \to \{*\}\big) \\ & \simeq & A(s) \end{array}$$

for the reaction,

$$\begin{array}{lcl} (\langle w_A \rangle\ ;\ [w_D]).D(s, (a, k)) & = & \big(\Sigma d \epsilon \langle w_A \rangle.D(s)\big) \\ & & \quad [w_D].D\big(w_A.n(s, a, d), k(d)\big) \\ & = & \big(\Sigma\_ \epsilon \{*\}\big)\ [w_D].D\big(w_A.n(s, a, *), k(*)\big) \\ & = & \{*\} \times [w_D].D\big((s, a), *\big) \\ & = & \{*\} \times D(s, a) \\ & \simeq & D(s, a) \end{array}$$

and the next state functions

$$\begin{array}{lcl} (\langle w_a \rangle\ ;\ [w_D]).n\big(s, (a, k), (*, d)\big) & = & [w_D].n\big(\langle w_A \rangle.n(s, a, *), k(*), d\big) \\ & = & [w_D].n\big((s, a), *, d\big) \\ & = & w_D.n\big((s, a), d\big) \\ & = & n(s, a, d)\ . \end{array}$$

This concludes the proof. ☑

2.3.3 Reflexive and Transitive Closure: Angelic Iteration

If we can compose interaction systems (when the codomain of the first one coincide with the domain of the second one), it is possible to compose a homogeneous system with itself, many times in a row if needed: this corresponds to doing a sequence of interactions. However, the iterated composition $w;\ldots;w$ suffers from a big drawback: all traces of interaction have the same length. The next definition is an answer to this problem:

▷ **Definition 2.3.4:** let $w = (A, D, n)$ be an interaction system on S; define the *reflexive transitive closure* of w, written w^*, on S as:

$$\begin{array}{lcl} A^* & \triangleq & (\mu X : S \to \mathbf{Set})\ (\lambda s \in S) \\ & & \quad \mathbf{data}\ \mathrm{Exit} \\ & & \qquad \mathrm{Call}(a,k)\ \ \textit{where}\ a \in A(s) \\ & & \qquad\qquad\qquad\qquad k \in \big(\Pi d \epsilon D(s,a)\big)\, X(s[a/d]) \end{array}$$

$$\begin{array}{lcl} D^*(s, \mathrm{Exit}) & \triangleq & \mathbf{data}\ \mathrm{Nil} \\ D^*\big(s, \mathrm{Call}(a,k)\big) & \triangleq & \mathbf{data}\ \mathrm{Cons}(d,d')\ \ \textit{where}\ d \in D(s,a) \\ & & \qquad\qquad\qquad\qquad\qquad\qquad d' \in D^*\big(s[a/d], k(d)\big) \end{array}$$

and

$$\begin{array}{lcl} n^*(s, \mathrm{Exit}, \mathrm{Nil}) & \triangleq & s \\ n^*\big(s, \mathrm{Call}(a,k), \mathrm{Cons}(d,d')\big) & \triangleq & n^*\big(s[a/d], k(d), d'\big)\ . \end{array}$$

This interaction system is also called the *Angelic iteration* of w.

A single move in w^* is thus a *strategy* to play in w, until the Angel decides she doesn't want to continue. A response to such a strategy is simply a sequence of reactions to the consecutive moves given by the strategy.

Here is how these definitions would be written in the Agda system:

```
RTCA (s::S) :: Set
   = data Exit  |  Call (a::A s) (k::(d::D s a) -> RTCA (n s a d))
RTCD (s::S) (a':: RTCA s) :: Set
  = case a' of
     (Exit)     ->  data Nil
     (Call a k) ->  data Cons (d::D s a)  (d'::RTCD (n s a d) (k d))
RTCn (s::S) (a':: RTCA s) (d':: RTCD s a) :: S
  = case a' of
     (Exit)     -> s
     (Call a k) -> RTCn (n s a d'.fst) (k d'.fst) d'.snd
```

where "`d'.fst`" denotes the projection of `d'` on the first coordinate.

This interaction system will play an important rôle in sections 2.5.5, 2.6.2 and 4.2.

2.3.4 Demonic Iteration

Angelic iteration is concerned with well-founded interaction where the Angel decides when to stop. There is a dual notion of potentially infinite plays for the Angel, where termination is decided by the Demon. This notion of Demonic iteration uses a definition by greatest fixpoint over $S \to \mathbf{Set}$ similar to the previous generalized inductive

definition. We start by recalling the full rules for such coinductive definitions as they are described in [46].

§ *"State Dependent" Coinductive Definitions.* Usual coinductive definitions allow to define greatest fixpoints for functors **Set** → **Set**; "state dependent" coinductive definitions will allow to define greatest fixpoints for a restricted class of functors from $(S \to \mathbf{Set})$ to $(S \to \mathbf{Set})$ (*i.e.* from $\mathcal{P}(S)$ to $\mathcal{P}(S)$). For predicativity reasons, it is not possible to justify the introduction of such greatest fixpoints for arbitrary functors. Instead, we use the notion of interaction system to define so called "set-based predicate transformers".[2] The interest of interaction systems as representations for endofunctors on $\mathcal{P}(S)$ will be discussed in section 2.5.

For any interaction system $w = (A, D, n)$ on S, define the following operator on subsets:

$$\begin{array}{lcl} w^\circ & : & \mathcal{P}(S) \to \mathcal{P}(S) \\ & & U \quad \mapsto \{s \epsilon S \mid (\exists a \epsilon A(s))(\forall d \epsilon D(s,a))\ s[a/d] \ \varepsilon\ U\}\ ; \end{array}$$

or, to use type theoretic notation:

$$\begin{array}{lcl} w^\circ & : & (S \to \mathbf{Set}) \to (S \to \mathbf{Set}) \\ w^\circ & \triangleq & (\lambda U{:}S \to \mathbf{Set})(\lambda s \epsilon S)\ (\Sigma a \epsilon A(s))(\Pi d \epsilon D(s,a))\ U(s[a/d])\ . \end{array}$$

It is quite trivial to check that w° is a functor. Formally, this means that w° is a monotonic operator from $\mathcal{P}(S)$ to $\mathcal{P}(S)$.

In words, $s\ \varepsilon\ w^\circ(U)$ means that the Angel has a foolproof way to reach U in exactly one interaction (provided the Demon does react to her action). An element of the set $s\ \varepsilon\ w^\circ(U)$ is simply a pair (a, k) where a is an action and $k(d)$ provides a proof that $s[a/d]\ \varepsilon\ U$ for any reaction d.

We now define $\nu_X.w^\circ(X)$, the greatest fixpoint of the operator just given. The intuition is that $s\ \varepsilon\ \nu_X.w^\circ(X)$ if, from state s, there is an infinite strategy choosing actions for the Angel.

- $$\frac{w = (A, D, n) \text{ interaction system on } S}{\nu_X.w^\circ(X) : S \to \mathbf{Set}}\ ;$$
- $$\frac{X : S \to \mathbf{Set} \qquad C\ \epsilon\ (s \epsilon S) \to X(s) \to w^\circ(X)(s) \qquad s\ \epsilon\ S \qquad x\ \epsilon\ X(s)}{\mathrm{coiter}(X, C, s, x)\ \epsilon\ \nu_X.w^\circ(X)(s)}\ ;$$
- $$\frac{s\ \epsilon\ S \qquad p\ \epsilon\ \nu_X.w^\circ(X)(s)}{\mathrm{elim}(p, s)\ \epsilon\ w^\circ(\nu_X.w^\circ(X))(s)}\ .$$

The reduction rule is:

$$\begin{array}{rcl} \mathrm{elim}\big(\mathrm{coiter}(X, C, s, x)\big) & = & \Big(a\ ,\ (\lambda d \epsilon D(s,a))\ .\ \mathrm{coiter}\big(X, C, s[a/d], g(d)\big)\Big) \\ & & \textit{where}\ \ C(s, x) = (a, g)\ . \end{array}$$

(With "$C(s,x) = (a,g)$" denoting a pattern matching: $C(s,x)$ is a pair, because it is in a sigma type.)
Thus, if $p\ \epsilon\ \nu_X.w^\circ(X)$, $\mathrm{elim}(p)$ is of the form (a, k), where a is an action in $A(s)$ and k is a continuation sending any $d\ \epsilon\ D(s, a)$ to a new infinite strategy from state $s[a/d]$.

[2]: The same restriction also applies to least fixpoints.

§ *Demonic Iteration.* We now have all the tools to define Demonic iteration:

▷ **Definition 2.3.5:** let $w = (A, D, n)$ be an interaction on S; define a new interaction system $w^\infty = (A^\infty, D^\infty, n^\infty)$ on S with:

$$A^\infty \triangleq \nu_X.w^\circ(X) \qquad \textit{(see above)}$$

$$\begin{array}{lll} D^\infty & \triangleq & \big(\mu X : (s \epsilon S) \to A^\infty(s) \to \textbf{Set}\big)\ \big(\lambda s\ \epsilon\ S\big)\ \big(\lambda p\ \epsilon\ A^\infty(s)\big) \\ & & \textbf{data}\ \mathrm{Nil} \\ & & \quad \mathrm{Cons}(d, d')\ \textit{where}\ (a, k) = \mathrm{elim}(p) \\ & & \qquad\qquad\qquad\quad d\ \epsilon\ D(s, a) \\ & & \qquad\qquad\qquad\quad d'\ \epsilon\ X\big(s[a/d], k(d)\big) \end{array}$$

and

$$\begin{array}{lll} n^\infty(s, p, \mathrm{Nil}) & \triangleq & s \\ n^\infty\big(s, p, \mathrm{Cons}(d, d')\big) & \triangleq & n^\infty\big(s[a/d], k(d), d'\big)\ \textit{where}\ (a, k) = \mathrm{elim}(p) \end{array}$$

This interaction system is called the *Demonic iteration* of w.

So, when choosing an action in w^∞, the Angel needs to decide on a potentially infinite strategy to play in w and the Demon reacts by a finite sequence of counter moves. Plays are still finite, but the Angel doesn't know when interaction will stop. This kind of situation is very common in computer science when dealing with server programs. We will come back to this example in section 2.6.3. Also note that a an element of $A^\infty(s)$ is a *deadlock avoiding strategy:* no matter what happens, provided the Demons reacts, the Angel always has a move to play.

2.4 Simulations

We now, at last, come to the notion of morphisms between interaction systems. This notion generalizes the natural notion of *simulation* between transition systems and was first formalized in the context of interaction systems by Anton Setzer and Peter Hancock. It will of course be compatible with the notion of structural isomorphism defined on page 37.

2.4.1 The Case of Transition Systems

The usual notion of *simulation relation* between labeled transition systems can be written as:

> *if* $\longrightarrow_1$ *and* $\longrightarrow_2$ *are LTS on sets* S_1 *and* S_2*, with labels in* L*, a relation* R *on* $S_1 \times S_2$ *is a simulation if the following holds:*
>
> $$\begin{cases} a\ \epsilon\ L \\ s_1 \xrightarrow{a}_1 s_1' \\ (s_1, s_2)\ \varepsilon\ R \end{cases} \quad \Rightarrow \quad s_2 \xrightarrow{a}_2 s_2' \quad \textit{for some } s_2' \textit{ s.t. } (s_1', s_2')\ \varepsilon\ R\ .$$

In our case, where the set of labels is local to each interaction system (and is even local to each state), we modify this definition into:

let $\nu_1 = (A_1, n_1)$ *and* $\nu_2 = (A_2, n_2)$ *be transition systems on* S_1 *and* S_2*; a relation* $R : \mathbf{Rel}(S_1, S_2)$ *is called a simulation if the following holds for all* $s_1 \in S_1$ *and* $s_2 \in S_2$*:*

$$\begin{array}{lll} (s_1, s_2)\ \varepsilon\ R & \Rightarrow & (\forall a_1 \in A_1(s_1)) \\ & & (\exists a_2 \in A_2(s_2)) \\ & & \quad (s_1[a_1], s_2[a_2])\ \varepsilon\ R\ . \end{array}$$

This definition shows that we are mainly concerned about how the states are linked and not so much about the actual transition names between them.

2.4.2 The General Case

It is quite straightforward to extend the above definition to take into account the presence of reactions:

▷ **Definition 2.4.1:** let $w_1 = (A_1, d_1, n_1)$ and $w_2 = (A_2, D_2, n_2)$ be interaction systems on S_1 and S_2; a relation $R : \mathbf{Rel}(S_1, S_2)$ is a *linear simulation relation* (or simply a *simulation*) from w_1 to w_2 if the following holds:
for all $s_1 \in S_1$ and $s_2 \in S_2$,

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

To be really pedantic, the actual definition of the collection of simulations from w_1 to w_2 is of the form:

$$\begin{array}{lll} (\Sigma R : \mathbf{Rel}(S_1 \times S_2)) & & \\ (\forall s_1 \epsilon S_1)(\forall s_2 \epsilon S_2)\ (s_1, s_2)\ \varepsilon\ R & \Rightarrow & (\forall a_1 \epsilon A_1(s_1)) \\ & & (\exists a_2 \epsilon A_2(s_2)) \\ & & \ldots \end{array}$$

i.e. a simulation is a pair (R, p) where p is a proof that R is a simulation.

The intended meaning is that if R is a simulation from w_1 to w_2 and $(s_1, s_2)\ \varepsilon\ R$, then we can simulate s_1 (in w_1) from s_2 (in w_2). There is one subtlety in the order of quantifiers which allows to get a flow of interaction coherent with the intuition of simulations: a "black-box" allows to simulate a state $s_1 \in S_1$ by a state $s_2 \in S_2$ if:

- when given an action $a_1 \in A_1(s_1)$ (action to simulate),
- it can send a command $a_2 \in A_2(s_2)$ to the environment (simulating command);
- and once the environment responds to a_2 with some $d_2 \in D_2(s_2, a_2)$,
- it can translate this reaction to a reaction d_1 in $D_1(s_1, a_1)$.

2.4.3 The Category of Interfaces

We are now ready to define the "category of interaction systems".

▷ **Definition 2.4.2:** an *interface* is given by a set S together with an interaction system w on S. We call the collection of interfaces **Int**. We sometimes omit the set of states S and refer to the interface (S, w) as w.

This proper type, with the notion of simulation just defined forms a category:

- the (relational) composition of simulations is a simulations;
- the identity is a simulation from any (S, w) to itself.

We will omit the proof that the identity (if available) is always a simulation from an interface to itself: this is just the usual "copycat" strategy which copies actions from left to right, and reactions from right to left.

Let's quickly check that the relational composition of two simulations is a simulation. Let R be a simulation from w_1 to w_2 and R' be a simulation from w_2 to w_3; suppose that $(s_1, s_3) \varepsilon R' \cdot R$:

1) we know that $(s_1, s_2) \varepsilon R$ and $(s_2, s_3) \varepsilon R'$ for some $s_2 \in S_2$;

2) suppose we are given an action $a_1 \in A_1(s_1)$ to simulate:

  a) since $(s_1, s_2) \varepsilon R$, we can simulate a_1 by some $a_2 \in A_2(s_2)$,

  b) since $(s_2, s_3) \varepsilon R'$, we can now simulate a_2 by some $a_3 \in A_3(s_3)$,

3) we produce the action a_3 to simulate a_1;

4) suppose we are given a reaction $d_3 \in D_3(s_3, a_3)$ to translate back:

  a) because a_2 is simulated by a_3 (via R'), we can translate d_3 back into a reaction d_2 in $D_2(s_2, a_2)$,

  b) similarly, since a_1 is simulated by a_2, we can translate d_2 back into a reaction d_1 in $D_1(s_1, a_1)$,

5) we produce reaction d_1;

6) we have indeed that $(s_1[a_1/d_1], s_3[a_3/d_3]) \varepsilon R' \cdot R$ because there is a mediating element: $(s_1[a_1/d_1], s_2[a_2/d_2]) \varepsilon R$ and $(s_2[a_2/d_2], s_3[a_3/d_3]) \varepsilon R'$.

Thus:

○ **Lemma 2.4.3:** the proper type **Int** with simulations forms a category.

This category inherits some of the structure of the simpler category of sets and relations: in particular, it is order enriched. This simply means that each collection $\mathbf{Int}(w_1, w_2)$ is equipped with a partial order (inclusion) and that composition is monotonic in both its arguments. This is trivial.

There is another property which deserves some comments: simulations are closed under arbitrary unions: the verification is direct. Since composition of relation commutes with unions on the left and on the right, we can conclude:

◇ **Proposition 2.4.4:** **Int** *is a category enriched over complete suplattices.*

In particular, the empty relation (the empty union) is *always* a simulation. (In the condition for simulation, we have a vacuous left hand side in the implication...) In practice, one uses initialized interaction systems and requires the initial states to be related: this prevents this bug, but most of part II will not work in this context.

2.5 Interaction Systems and Predicate Transformers

We now look at the predicate (rather than family) version of interaction systems: just like transition systems are concrete representations for relations, interaction systems are concrete representations for *predicate transformers*. We recall some of the traditional theory of predicate transformers (see [8]) and link that to the previous sections.

A predicate transformer is simply an operator on subsets:

▷ **Definition 2.5.1:** if S_1 and S_2 are sets, a *predicate transformer* from S_1 to S_2 is a function from $\mathcal{P}(S_1)$ to $\mathcal{P}(S_2)$. A predicate transformer is *monotonic* if it is monotonic w.r.t. inclusion.
Inclusion and equality of predicate transformers is defined pointwise. (It is an instance of $\mathbf{\Pi}^1_1$ quantification.)
Predicate transformers are implicitly assumed to be monotonic with respect to inclusion.

Predicate transformers were introduced by E. W. Dijkstra ([28]) in order to develop a compositional semantics for sequential programs. Each program was interpreted by a predicate transformer taking final states to initial states with the following interpretations:

- **wp**-*calculus:* the meaning of "$s \varepsilon P(U)$" is: "if the program is started in state s, then execution will terminate and the final state will be in U". Thus, $P(U)$ is the set of initial states from which we can guarantee termination in U. "**wp**" stands for Weakest Precondition;
- **wlp**-*calculus:* we weaken the meaning of $s \varepsilon P(U)$ to "if the program is started in state s, and if execution terminates, then the final state will be in U". Thus, we do not guarantee termination. "**wlp**" stands for Weakest Liberal Precondition.

This idea of using predicate transformers to model programs was later extended in order to deal with *specifications* as well: a specification usually takes the form:

> *if execution is started from a state satisfying* ψ*, then execution should terminate, and the final state should satisfy* φ.

Just like above, we may weaken such a specification and prefer conditional termination. Such a specification can be identified with the predicate transformer:

$$\begin{array}{lcl} P & : & \mathcal{P}(S_f) \to \mathcal{P}(S_i) \\ & & \varphi \quad \mapsto \text{"biggest such } \psi\text{"} \ . \end{array}$$

One interesting point about this semantics is that programs and specifications belong to the same semantical domain: predicate transformers. The field of the *refinement calculus* ([8]) is a systematic exploration of the relation between programs and specifications in this framework. One of its interesting features is the ability to start with a specification, *i.e.* a monotonic predicate transformer, and mechanically transform it into a well-behaved predicate transformer[3] representing the semantics of an actual program. This program can then be extracted from the predicate transformer!

[3]: typically a predicate transformer commuting with arbitrary unions and directed intersections

2.5.1 Representing Predicate Transformers by Interaction Systems

An equivalent way to see predicate transformers is through the isomorphism:[4]

$$\begin{array}{lcl} \mathcal{P}(S_1) \to \mathcal{P}(S_2) & = & \mathcal{P}(S_1) \to (S_2 \to \mathbf{Set}) \\ & \simeq & \big(\mathcal{P}(S_1) \times S_2\big) \to \mathbf{Set} \\ & \simeq & \big(S_2 \times \mathcal{P}(S_1)\big) \to \mathbf{Set} \\ & \simeq & S_2 \to \big(\mathcal{P}(S_1) \to \mathbf{Set}\big) \\ & = & S_2 \to \mathcal{P}^2(S_1) \end{array}$$

which make predicate transformers look pretty much like interaction systems (modulo the difference between $\mathcal{P}(\_)$ and $\mathcal{F}(\_)$). Our intuition is that an interaction system from S_1 to S_2 is a concrete representation for a *monotonic* predicate transformer from S_2 to S_1. However, the translation from an interaction system to a predicate transformer is subtler than the translation from a transition system to a relation since we cannot apply the operation $\_^\circ$ from page 25 on proper types. Instead, we use the following:

▷ **Definition 2.5.2:** If $w = (A, D, n)$ is an interaction system from S_1 to S_2, define the monotonic predicate transformer w° from S_2 to S_1 (note the swap) as:

$$s \;\varepsilon\; w^\circ(U) \quad \Leftrightarrow \quad \big(\exists a \in A(s)\big)\big(\forall d \in D(s,a)\big)\; s[a/d] \;\varepsilon\; U \;.$$

Dually, define the monotonic predicate transformer $w^\bullet$ as:

$$s \;\varepsilon\; w^\bullet(U) \quad \Leftrightarrow \quad \big(\forall a \in A(s)\big)\big(\exists d \in D(s,a)\big)\; s[a/d] \;\varepsilon\; U \;.$$

It is easy to show that we only get in this way *monotonic* predicate transformers. It should also be noted that as opposed to the translation from transition systems to relations, this translation doesn't use equality.

The predicate transformer w° is concerned with the *reachability* of a subset of states by the Angel, in a single interaction; the predicate transformer $w^\bullet$ is concerned with reachability for the Demon. Surprisingly, the predicate transformer $w^\bullet$ is definable in terms of $\_^\circ$:

▷ **Definition 2.5.3:** if $w = (A, D, n)$ is an interaction system from S_1 to S_2, define $w^\perp = (A^\perp, D^\perp, n^\perp)$, an interaction system from S_1 to S_2 as:

$$\begin{array}{lcl} A^\perp(s_1) & \triangleq & \big(a \in A(s_1)\big) \to D(s_1, a) \\ D^\perp(s_1, f) & \triangleq & A(s_1) \\ n^\perp(s_1, f, a) & \triangleq & s_1[a/f(a)] \;. \end{array}$$

Thus, an action for the Angel in $w^\perp$ is a conditional reaction for the Demon in w, and a reaction for the Demon in $w^\perp$ is a an action for the Angel in w. An interesting point is that the set of reactions $D^\perp(s_1, f)$ doesn't depend on the action f. This operation will play a crucial role in the interpretation of linear logic developed in Part II. For the moment, we only note the following:

○ **Lemma 2.5.4:** for any interaction system, $w^\bullet = (w^\perp)^\circ$.

[4]: This works also with the traditional notion of subset: replace **Set** by $\mathbf{B} \triangleq \{\mathrm{True}, \mathrm{False}\}$.

proof: suppose $U : \mathcal{P}(S_2)$ and $s \in S_1$, we need to show that $s \varepsilon w^\bullet(U)$ iff $s \varepsilon (w^\perp)^\circ(U)$:

$$\begin{array}{lcl} s \varepsilon w^\bullet(U) & \Leftrightarrow & \big(\forall a \in A(s)\big)\big(\exists d \in D(s,a)\big)\ s[a/d] \varepsilon U \\ & & \Leftrightarrow \{\ \textbf{AC}, \text{page } 30\ \} \\ s \varepsilon w^{\perp\circ}(U) & \Leftrightarrow & \big(\exists f \in (a \epsilon A(s)) \to D(s,a)\big)\big(\forall a \in A(s)\big)\ s[a/f(a)] \varepsilon U\ . \end{array}$$

This proof has a strong "Dialectica" feeling: it somehow shows that formulas of the form $(\exists F)(\forall f)\, \varphi(F,f)$ are closed under negation.

☑

↬ REMARK 13: **classically,** we also have the converse, *i.e.* $w^\circ = (w^\perp)^\bullet$, but this requires the use of the contraposition of the axiom of choice (**CtrAC**, page 30) which doesn't hold constructively.

The property of being of the form w° appears in Peter Aczel's work under the name *set-based* predicate transformer: from [6]

> *Call a monotone operation* $f : \mathcal{P}(A) \to \mathcal{P}(A)$ *set-based if there is a subset* $\mathcal{B}$ *of* $\mathcal{P}(A)$ *such that whenever* $a \varepsilon f(X)$, *with* $X : \mathcal{P}(A)$, *then there is* $Y \varepsilon \mathcal{B}$ *such that* $Y \subseteq X$ *and* $a \varepsilon f(Y)$. *We call* $\mathcal{B}$ *a baseset for* f.

The important point in this definition is that $\mathcal{B}$ needs to be a sub*set*, *i.e.* it needs to be indexed by a set: we cannot take $\mathcal{B} \triangleq \mathcal{P}(A)$. It is easy to show that for an operator from $\mathcal{P}(S)$ to itself, being set-indexed and being of the form w° are equivalent:

- if f is set indexed, let $\{U_b \mid b \in B\}$ be the baseset, define
  - $A(s) \triangleq \{b \in B \mid s \varepsilon f(U_b)\}$
  - $D(s,b) = \{s \in S \mid s \varepsilon U_b\}$
  - $n(s,b,s') \triangleq s'$;
- for w°, define the baseset to be $\{U(s,a) \mid s \in S, a \in A(s)\}$ where the $U(s,a)$'s are defined as $U(s,a) \triangleq \{s[a/d] \mid d \in D(s,a)\}$.

All the structure of $\mathcal{P}(S)$ lifts pointwise to predicate transformers, so that the collection of predicate transformers from S_1 to S_2 forms a complete Heyting algebra. Moreover, predicate transformers are obviously closed under composition, and this corresponds exactly to the sequential composition of interaction systems:

> ○ **Lemma 2.5.5:** for all interaction systems w_1 from S_1 to S_2 and w_2 from S_2 to S_3, we have
>
> $$(w_1 ; w_2)^\circ \quad = \quad w_1^\circ \cdot w_2^\circ\ .$$

proof: suppose $s_1 \in S_1$ and $U : \mathcal{P}(S_3)$:

$$\begin{array}{l} s_1 \varepsilon (w_1 ; w_2)^\circ(U) \\ \quad \Leftrightarrow \quad \{\ \text{definition of } \_^\circ\ \} \\ \big(\exists a \in (w_1 ; w_2).A(s_1)\big)\big(\forall d \in (w_1 ; w_2).D(s_1,a)\big)\ (w_1 ; w_2).n(s_1,a,d) \varepsilon U \\ \quad \Leftrightarrow \quad \{\ \text{definition of } \_ ; \_\ \} \\ \big(\exists a_1 \epsilon A_1(s_1)\big) \\ \big(\exists k \epsilon \big(d_1 \epsilon D_1(s_1,a_1)\big) \to A_2(s_1[a_1/d_1])\big)\big(\forall d_1 \epsilon D_1(s_1,a_1)\big) \\ \qquad\qquad\qquad\qquad \big(\forall d_2 \epsilon A_2\big(s_2, k(d_1)\big)\big) \\ \qquad\qquad\qquad\qquad\quad s_1[a_1/d_1][k(d_1)/d_2] \varepsilon U \end{array}$$

$\Leftrightarrow$ { axiom of choice on "$\exists k \forall d_1$" }

$(\exists a_1 \epsilon A_1(s_1))$
$(\forall d_1 \epsilon D_1(s_1, a_1))(\exists a_2 \epsilon A_2(s_1[a_1/d_1]))$
$(\forall d_2 \epsilon A_2(s_2, a_2)))\ s_1[a_1/d_1][a_2/d_2] \ \varepsilon \ U$

$\Leftrightarrow$ { definition of w_2° }

$(\exists a_1 \epsilon A_1(s_1))(\forall d_1 \epsilon D_1(s_1, a_1))\ s_1[a_1/d_1] \ \varepsilon \ w_2^\circ(U)$

$\Leftrightarrow$ { definition of w_1° }

$s_1 \ \varepsilon \ w_1^\circ(w_2^\circ(U))$

☑

2.5.2 Angelic and Demonic Updates

There is an increase of complexity between the following notions of morphisms between sets: $S_1 \to S_2$, $S_1 \to \mathcal{P}(S_2)$ and $\mathcal{P}(S_1) \to \mathcal{P}(S_2)$. The link between those is explained in [35]. For us, the important remark is that we can lift operations from one level to the next. For functions, we can define (in the presence of equality) its *graph* relation $\mathbf{gr}(f) : \mathbf{Rel}(S_1, S_2)$ as $\{(s_1, s_2) \mid f(s_1) = s_2\}$. Lifting a relation to a predicate transformer can be done is two dual ways:

▷ **Definition 2.5.6:** let $R : \mathbf{Rel}(S_1, S_2)$ be a relation from S_1 to S_2; define the *Angelic update* $\langle R \rangle$ to be the following predicate transformer from S_2 to S_1 (note the swap):

$$s_1 \ \varepsilon \ \langle R \rangle(U) \quad \Leftrightarrow \quad (\exists s_2 \epsilon S_2)\ (s_1, s_2) \ \varepsilon \ R \wedge s_2 \ \varepsilon \ U \ .$$

We define the *direct image along* R to be the predicate transformer $\langle R^\sim \rangle$ and we usually write it simply R. Since $R(s_1)$ is equal to $\langle R^\sim \rangle(\{s_1\})$, there is no danger of confusion.

Dually, define the *Demonic update* $[R]$ to be the predicate transformer from S_2 to S_1:

$$s_1 \ \varepsilon \ [R](U) \quad \Leftrightarrow \quad (\forall s_2 \epsilon S_2)\ (s_1, s_2) \ \varepsilon \ R \Rightarrow s_2 \ \varepsilon \ U \ .$$

The choice of notation is not innocent (refer to definitions 2.3.2 and 1.1.8 for the actions of $\langle \_ \rangle$, $[\_]$ and $\_^\circ$ on transition systems): we have

- **Lemma 2.5.7:** for any *transition* system ν from S_1 to S_2,
  - $\langle \nu^\circ \rangle = \langle \nu \rangle^\circ$;
  - and $[\nu^\circ] = [\nu]^\circ$.

proof: let's only show the first one, let $U : \mathcal{P}(S_2)$ and $s_1 \ \epsilon \ S_1$,

$s_1 \ \varepsilon \ \langle \nu^\circ \rangle(U)$

$\Leftrightarrow$ { definition of $\langle \_ \rangle$ }

$(\exists s_2 \epsilon S_2)\ (s_1, s_2) \ \varepsilon \ \nu^\circ \wedge s_2 \ \varepsilon \ U$

$\Leftrightarrow$ { definition of ν° }

$(\exists s_2 \epsilon S_2)(\exists a \epsilon A(s_1))\ s_1[a] = s_2 \wedge s_2 \ \varepsilon \ U$

$\Leftrightarrow$ { logic }

$(\exists a \epsilon A(s_1))(\forall \_ \epsilon \{*\})\ s_1[a] \ \varepsilon \ U$

$\Leftrightarrow$ { definition of $\langle \nu \rangle$ }

$$\begin{array}{l}\big(\exists a \epsilon \langle \nu \rangle . A(s_1)\big)\big(\forall d \epsilon \langle \nu \rangle . D(s,a)\big)\ s_1[a/d] \ \varepsilon\ U \\ \quad \Leftrightarrow \quad \{\ \text{definition of } \_^\circ\ \} \\ s_1 \ \varepsilon\ \langle \nu \rangle^\circ(U)\end{array}$$

☑

Note that while the definitions of $\langle \nu^\circ \rangle$ and $[\nu^\circ]$ use equality, the definitions or $\langle \nu \rangle^\circ$ and $[\nu]^\circ$ don't, which makes them preferable.

Those two predicate transformers satisfy the following well-known facts:

◇ **Proposition 2.5.8:** *for any relation* R*, we have:*
- $\langle R \rangle$ *commutes with arbitrary unions;*
- $[R]$ *commutes with arbitrary intersections.*

Moreover, suppose F *is a predicate transformer:*
- **with equality,** *if* F *commutes with arbitrary unions, then it is of the form* $\langle R \rangle$ *for some relation* R*;*
- **impredicatively,** *if* F *commutes with arbitrary intersections, then it is of the form* $[R]$ *for some relation* R*.*

proof: the proofs that $\langle R \rangle$ and $[R]$ respectively commute with unions and intersections are trivial.

For the second part, suppose $F : \mathcal{P}(S_1) \to \mathcal{P}(S_2)$ commutes with arbitrary unions. Define $R : \mathbf{Rel}(S_2, S_1)$ as

$$(s_2, s_1) \ \varepsilon\ R \quad \triangleq \quad s_2 \ \varepsilon\ F\big(\{s_1\}\big)\ .$$

(We need equality to use singleton subsets...)
That $F = \langle R \rangle$ follows directly from the fact that $U = \bigcup \big\{\{s_1\} \mid s_1 \ \varepsilon\ U\big\}$.

Suppose that F commutes with arbitrary intersections. Define $R : \mathbf{Rel}(S_2, S_1)$ as

$$(s_2, s_1) \ \varepsilon\ R \quad \triangleq \quad \big(\forall U : \mathcal{P}(S_1)\big)\ s_2 \ \varepsilon\ F(U) \Rightarrow s_1 \ \varepsilon\ U\ .$$

(This is impredicative because of the quantification over $\mathcal{P}(S_1)$*.)*
Let $U : \mathcal{P}(S_1)$ and $s_2 \in S_2$,
- suppose $s_2 \ \varepsilon\ F(U)$, let's show that $s_2 \ \varepsilon\ [R](U)$. Suppose that $(s_2, s_1) \ \varepsilon\ R$, *i.e.* that $s_2 \ \varepsilon\ F(V) \Rightarrow s_1 \ \varepsilon\ V$ for all V. We need to show that $s_1 \ \varepsilon\ U$. We can take $V \triangleq U$, and since $s_2 \ \varepsilon\ F(U)$ by hypothesis, we can conclude that $s_1 \ \varepsilon\ U$.
- for the other direction, if $s_2 \ \varepsilon\ [R](U)$, let's show that $s_2 \ \varepsilon\ F(U)$. We obviously have that $s_2 \ \varepsilon\ \bigcap \{F(V) \mid s_2 \ \varepsilon\ F(V)\}$, which implies, since F commutes with intersections, that $s_2 \ \varepsilon\ F\big(\bigcap \{V \mid s_2 \ \varepsilon\ F(V)\}\big)$.
  Now, it is easy to show that $\bigcap \{V \mid s_2 \ \varepsilon\ F(V)\} \subseteq U$: this is exactly the meaning of $s_2 \ \varepsilon\ [R](U)$. Thus, by monotonicity of F, we obtain that $s_2 \ \varepsilon\ F(U)$.

☑

2.5.3 Factorization of Monotonic Predicate Transformers

We now come to the predicate transformer version of proposition 2.3.3: any predicate transformer can be seen as the sequential composition of a $[R]$ followed by a $\langle R' \rangle$. In this case however, the result is impredicative.

◇ **Proposition 2.5.9: (impredicative)**
for any monotonic predicate transformer F *from* S_1 *to* S_2, *there is a type* $\mathcal{Z}$ *and two relations* $R : \mathbf{Rel}(\mathcal{Z}, S_1)$ *and* $R' : \mathbf{Rel}(S_2, \mathcal{Z})$ *such that* $F = \langle R' \rangle \cdot [R]$.

proof: define the type $\mathcal{Z} \triangleq \mathcal{P}(S_1)$ and the two relations

- $(V, s_1) \varepsilon R$ iff $s_1 \varepsilon V$;
- and $(s_2, V) \varepsilon R'$ iff $s_2 \varepsilon F(V)$.

Let $U : \mathcal{P}(S_1)$ and $s_2 \epsilon S_2$; we need to show that $s_2 \varepsilon F(U)$ iff $s_2 \varepsilon \langle R' \rangle \cdot [R](U)$:

$s_2 \varepsilon \langle R' \rangle \cdot [R](U)$

$\Leftrightarrow$

$(\exists V)\ (s_2, V) \varepsilon R' \wedge V \varepsilon [R](U)$

$\Leftrightarrow$

$(\exists V)\ s_2 \varepsilon F(V) \wedge (\forall s_1)\ (V, s_1) \varepsilon R \Rightarrow s_1 \varepsilon U$

$\Leftrightarrow$

$(\exists V)\ s_2 \varepsilon F(V) \wedge (\forall s_1)\ s_1 \varepsilon V \Rightarrow s_1 \varepsilon U$

$\Leftrightarrow$

$(\exists V)\ s_2 \varepsilon F(V) \wedge V \subseteq U$

$\Leftrightarrow$ { by monotonicity }

$s_2 \varepsilon F(U)$.

☑

Note that this proof is impredicative because $\mathcal{Z}$ is not a set but a proper type. This proof is constructive but its computational content is next to empty.

2.5.4 Interior and Closure Operators

Recall that:

▷ **Definition 2.5.10:** an *interior operator* on S is a monotonic predicate transformer F on S such that:

- F is contractive: $F(U) \subseteq U$ for any $U : \mathcal{P}(S)$;
- $F \subseteq F \cdot F$.

Dually, a *closure operator* is a monotonic predicate transformer F such that:

- F is expansive: $U \subseteq F(U)$ for any $U : \mathcal{P}(S)$;
- $F \cdot F \subseteq F$.

We have the following:

○ **Lemma 2.5.11:** for any relation $R \subseteq S_1 \times S_2$:

- $\langle R \rangle$ is left Galois connected to $[R^\sim]$: $\langle R \rangle(U) \subseteq V \Leftrightarrow U \subseteq [R^\sim](V)$;
- $\langle R \rangle \cdot [R^\sim]$ is an interior operator on S_2;
- $[R^\sim] \cdot \langle R \rangle$ is a closure operator on S_1.

The second and third points are implied by the first one, which is immediate.

We have a representation theorem in the spirit of proposition 2.5.9. However, while proposition 2.5.9 is well-known, the next lemma doesn't appear anywhere in the reference [8].

○ **Lemma 2.5.12: (impredicative)** for any interior operator F on S, there is a type $\mathcal{Z}$ and a relation $R : \mathbf{Rel}(\mathcal{Z}, S)$ s.t. $F = \langle R \rangle \cdot [R^\sim]$.

proof: define $\mathcal{Z}$ to be the collection of fixpoints of F. This is predicatively not a set, but a proper type: $\mathcal{Z} \triangleq \mathbf{Fix}_F \triangleq \big(\Sigma V : \mathcal{P}(S)\big)\ V = F(V)$. Put $(V, s)\ \varepsilon\ R$ iff $s\ \varepsilon\ V$.

The proof relies on the following (impredicative) fact: if F is an interior operator, then

$$F(U) = \bigcup \{V\ \varepsilon\ \mathbf{Fix}_F \mid V \subseteq U\} \quad \text{for any } U : \mathcal{P}(S)\ . \tag{2-1}$$

The proof is simple:

- $F(U) \subseteq \bigcup \{V\ \varepsilon\ \mathbf{Fix}_F \mid V \subseteq U\}$: we know that $F(U)$ is itself a fixpoint of F because F is an interior operator. This implies that $F(U)$ appears in the RHS, which yields the inclusion.
- $F(U) \supseteq \bigcup \{V\ \varepsilon\ \mathbf{Fix}_F \mid V \subseteq U\}$: suppose V is fixpoint of F such that $V \subseteq U$. By monotonicity, we have that $F(V) \subseteq F(U)$, *i.e.* that $V \subseteq F(U)$.

Now, for the main part, suppose $U : \mathcal{P}(S)$ and $s \in S$:

$s\ \varepsilon\ \langle R \rangle \cdot [R^\sim](U)$
$\quad\Leftrightarrow\quad$ { definition of $\langle \_ \rangle$ }
$(\exists V \in \mathbf{Fix}_F)\ (V, s)\ \varepsilon\ R \wedge V\ \varepsilon\ [R^\sim](U)$
$\quad\Leftrightarrow\quad$ { definition of $[\_]$ }
$(\exists V \in \mathbf{Fix}_F)\ (V, s)\ \varepsilon\ R \wedge (\forall s')\ (V, s')\ \varepsilon\ R \Rightarrow s'\ \varepsilon\ U$
$\quad\Leftrightarrow\quad$ { definition of R }
$(\exists V \in \mathbf{Fix}_F)\ s\ \varepsilon\ V \wedge (\forall s')\ s'\ \varepsilon\ V \Rightarrow s'\ \varepsilon\ U$
$\quad\Leftrightarrow$
$(\exists V \in \mathbf{Fix}_F)\ s\ \varepsilon\ V \wedge V \subseteq U$
$\quad\Leftrightarrow$
$s\ \varepsilon\ \bigcup \{V \in \mathbf{Fix}_F \mid V \subseteq U\}$
$\quad\Leftrightarrow\quad$ { fact (2-1) }
$s\ \varepsilon\ F(U)$.

☑

↬ REMARK 14: apparently however, there is no constructive version of this theorem for closure operators! What we can do is factorize (impredicatively) any closure operator as $\lfloor R \rceil \cdot \lfloor R^\sim \rceil$, where $\lfloor R \rceil$ is the *antitonic* predicate transformer

$$s_2\ \varepsilon\ \lfloor R \rceil(U) \quad\Leftrightarrow\quad (\forall s_1 \in S_1)\ s_1\ \varepsilon\ U \Rightarrow (s_1, s_2)\ \varepsilon\ R\ .$$

The proof is very similar to that of lemma 2.5.12.

This is an example of non trivial *resolution* for an interior/closure operator. In a categorical setting, a resolution is a factorization of a (co)monad as the composition of two adjoint functors. There are always two trivial resolutions given by the Kleisli and monad algebra constructions: they correspond to factorizing F as "$F \cdot \mathbf{Id}$" or as "$\mathbf{Id} \cdot F$".

2.5.5 Angelic and Demonic Iterations

We now come to the less trivial operations of iteration. We know by impredicative reasoning (Knaster-Tarski theorem) that any monotonic operator F on $\mathcal{P}(S)$ has a least fixpoint and a greatest fixpoint, respectively called μF and νF. We are interested only in two forms of fixpoints which, anticipating on proposition 2.5.18, we call F^* and F^∞:

$$F^*(U) \quad \triangleq \quad \big(\mu X : \mathcal{P}(S)\big)\ U \cup F(X)\ ;$$

$$F^\infty(U) \quad \triangleq \quad \big(\nu X : \mathcal{P}(S)\big)\ U \cap F(X)\ .$$

They obey the rules:

- $\dfrac{F : \mathcal{P}(S) \to \mathcal{P}(S) \text{ monotonic}}{F^*, F^\infty : \mathcal{P}(S) \to \mathcal{P}(S)}$ formation;
- $\dfrac{}{U \cup F \cdot F^*(U) \subseteq F^*(U)}$ pre-fixpoint, and $\dfrac{U \cup F(X) \subseteq X}{F^*(U) \subseteq X}$ least;
- $\dfrac{}{F^\infty(U) \subseteq U \cap F \cdot F^\infty(U)}$ post-fixpoint, and $\dfrac{X \subseteq U \cap F(X)}{X \subseteq F^\infty(U)}$ greatest.

Such fixpoints cannot be predicatively justified. As we'll see in proposition 2.5.18, it is however possible to define them inductively if we restrict to set-based predicate transformers. Those operators enjoy another fixpoint property:

- **Lemma 2.5.13:** for any predicate transformer F, we have:

$$\begin{aligned} F^* &= (\mu P)\ .\ \mathbf{Id} \cup F \cdot P \\ F^\infty &= (\nu P)\ .\ \mathbf{Id} \cap F \cdot P\ . \end{aligned}$$

proof: easy. ☑

Let's look at some properties of F^* and F^∞:

- **Lemma 2.5.14:** for any predicate transformer F,
  - F^* is a closure operator;
  - F^∞ is an interior operator.

proof: let's check that F^* is a closure operator:

- F^* is contractive: $U \subseteq F^*(U)$. This follows directly from the "pre-fixpoint" rule: $\mathbf{Id} \cup F \cdot F^* \subseteq F^*$.
- $F^* \cdot F^* \subseteq F^*$: by the "pre-fixpoint" rule, we have $F^*(U) \cup F \cdot F^*(U) \subseteq F^*(U)$. By applying the "least" rule for $X \triangleq F^*(U)$, we get $F^*\big(F^*(U)\big) \subseteq F^*(U)$.

The proof that F^∞ is an interior operator is completely dual... ☑

Those predicate transformers are also linked with the notions of invariant and saturated predicates:

- **Definition 2.5.15:** if F is a predicate transformer on S,
  - an F-*invariant predicate* is a post-fixpoint of F, *i.e.* a predicate U such that $U \subseteq F(U)$;
  - an F-*saturated predicate* is a pre-fixpoint of F, *i.e.* a predicate U such that $F(U) \subseteq U$.

The notion of invariant predicate will be particularly important in the sequel, where invariant predicates (or, as we also call them, "safety properties") will be interpretations for proofs and λ-terms. (See section 7.1, proposition 7.1.17.) We have:

- **Lemma 2.5.16:** if F is a predicate transformer on S, for any $U : \mathcal{P}(S)$,
  - $F^*(U)$ is the least F-saturated subset containing U;
  - $F^\infty(U)$ is the greatest F-invariant contained in U.

proof: easy.

☑

As all the previous lemmas show, the predicate transformers F^* and F^∞ enjoy dual properties. This duality can be made very precise through the following statement, which only holds classically:

◦ **Lemma 2.5.17: (classically)** for any predicate transformer F on S:

$$\begin{aligned}(\complement \cdot F \cdot \complement)^* &= \complement \cdot F^\infty \cdot \complement \\ (\complement \cdot F \cdot \complement)^\infty &= \complement \cdot F^* \cdot \complement\ .\end{aligned}$$

(where $\complement$ represents complementation w.r.t. S)

proof: easy if one looks at the characterizations of F^*/F^∞ in terms of least/greatest pre-fixpoint/post-fixpoint.

☑

This (classical) notion of duality will be of great importance in the second part of this work (sections 7 and 8).

We now state the main result of this section:

◇ **Proposition 2.5.18:** *for any interaction system w on S, we have:*

$$\begin{aligned}w^{*\circ} &= w^{\circ *}\ ; \\ w^{\infty\circ} &= w^{\circ\infty}\ .\end{aligned}$$

A visual way to see this proposition is through the following

$$\begin{aligned}\exists a^* \forall d^* &\Leftrightarrow \exists a_1 \forall d_1\ \exists a_2 \forall d_2\ \ldots\ \exists a_n \\ \exists a^\infty \forall d^\infty &\Leftrightarrow \exists a_1 \forall d_1\ \exists a_2 \forall d_2\ \ldots\ \exists a_n \forall d_n\end{aligned}$$

with the additional remark that length of interaction n may depend on the trace of interaction $(a_1/d_1, a_2/d_2, \ldots)$. The left hand sides correspond respectively to $w^{*\circ}$ and $w^{\infty\circ}$ while the right hand sides correspond to $w^{\circ *}$ and $w^{\circ\infty}$.

proof:

→ "$w^{*\circ}(U) \subseteq w^{\circ *}(U)$": suppose we have $s\ \varepsilon\ w^{*\circ}(U)$, *i.e.* that

$$\big(\exists a' \epsilon A^*(s)\big)\big(\forall d' \epsilon D^*(s,a')\big)\ s[a'/d']\ \varepsilon\ U\ . \tag{2-2}$$

We proceed by induction on a':

- if $a' = \mathrm{Exit}$, (2-2) gives $\big(\forall d' \in \{\mathrm{Nil}\}\big)\, s[\mathrm{Exit}/d']\ \varepsilon\ U$, *i.e.* $s\ \varepsilon\ U$. This implies that $s\ \varepsilon\ w^{\circ *}(U)$ by the "pre-fixpoint" rule;
- if $a' = \mathrm{Call}(a,k)$, (2-2) gives

$$\big(\forall d \epsilon D(s,a)\big)\Big(\forall d' \epsilon D^*\big(s[a/d], k(d)\big)\Big)\ n^*\big(s[a/d][k(d)/d']\big)\ \varepsilon\ U\ .$$

This implies that, for all $d \in D(s,a)$, $s[a/d]\ \varepsilon\ w^{*\circ}(U)$. By induction hypothesis, this implies that whenever $d \in D(s,a)$, we have $s[a/d]\ \varepsilon\ w^{\circ *}(U)$, *i.e.* that $s\ \varepsilon\ w^\circ \cdot w^{\circ *}(U)$. By the "pre-fixpoint" rule, this yields $s\ \varepsilon\ w^{\circ *}(U)$.

- "$w^{\circ *}(U) \subseteq w^{* \circ}(U)$": by using the "least" rule for $X \triangleq w^{* \circ}(U)$, we only need to show $U \cup w^{\circ} \cdot w^{* \circ}(U) \subseteq w^{* \circ}(U)$.
  - We have trivially that $U \subseteq w^{* \circ}(U)$ by taking the Exit action: if $s \varepsilon U$, then $\big(\forall d' \epsilon D^*(s, \mathrm{Exit})\big)\ s[\mathrm{Exit}/d'] \varepsilon U$.
  - Suppose now that $s \varepsilon w^{\circ} \cdot w^{* \circ}(U)$, *i.e.* that there is an action $a \epsilon A(s)$ such that $\big(\forall d \epsilon D(s,a)\big)\ s \varepsilon w^{* \circ}(U)$. By definition this means:

    $$\big(\forall d \epsilon D(s,a)\big)\ \big(\exists a' \epsilon A^*(s[a/d])\big) \big(\forall d' \epsilon D^*(s[a/d], a')\big)\ r^*(s[a/d], a', d') \varepsilon U\ .$$

    Using the axiom of choice on $\forall d \exists a'$, we get

    $$\Big(\exists k\ \epsilon\ \big(d \epsilon D(s,a)\big) \to A^*\big(s[a/d]\big)\Big) \big(\forall d \epsilon D(s,a)\big)\Big(\forall d' \epsilon D^*\big(s[a/d], k(d)\big)\Big)\ r^*(s[a/d], k(d), d') \varepsilon U\ ;$$

    we can thus take $\mathrm{Call}(a,k) \epsilon A^*(s)$ and we have

    $$\Big(\forall d'\ \epsilon\ D^*\big(s, \mathrm{Call}(a,k)\big)\Big)\ r^*(s, \mathrm{Call}(a,k), d') \varepsilon U$$

    i.e. $s \varepsilon w^{* \circ}(U)$.
    This finishes the proof that $w^{\circ} \cdot w^{* \circ} \subseteq w^{* \circ}$, and thus that $w^{\circ *} \subseteq w^{* \circ}$.

- "$w^{\infty \circ}(U) \subseteq w^{\circ \infty}(U)$": by using the "greatest" rule defining F^{∞} for $X \triangleq w^{\infty \circ}(U)$, it is enough to show that $w^{\infty \circ}(U)$ is a post-fixpoint for $U \cap w^{\circ}(\_)$: suppose that $s \varepsilon w^{\infty \circ}(U)$, *i.e.*

  $$\big(\exists a'\ \epsilon\ A^{\infty}(s)\big)\big(\forall d'\ \epsilon\ D^{\infty}(s,a')\big)\ r^{\infty}(s,a',d') \varepsilon U\ . \tag{2-3}$$

  We need to show that $s \varepsilon U \cap w^{\circ} \cdot w^{\infty \circ}(U)$:
  - for $d' \triangleq \mathrm{Nil}$, we have that $r^{\infty}(s,a',\mathrm{Nil}) \varepsilon U$, *i.e.* that $s \varepsilon U$;
  - we have that $\mathrm{elim}(a')$ is an element of $w^{\circ}(A^{\infty})(s)$, *i.e.* is of the form (a,k) where $a \epsilon A(s)$ and $k \epsilon \big(d \epsilon D(s,a)\big) \to A^{\infty}(s[a/d])$.
    We claim that $(\forall d \epsilon D(s,a))\ s[a/d] \varepsilon w^{\infty \circ}(U)$. For any $d \epsilon D(s,a)$, take the action $k(d) \epsilon A^{\infty}(s[a/d])$. We need to show that

    $$\Big(\forall d'' \epsilon D^{\infty}\big(s[a/d], k(d)\big)\Big)\ r^{\infty}(s[a/d], k(d), d'') \varepsilon U\ .$$

    Let $d'' \epsilon D^{\infty}\big(s[a/d], k(d)\big)$, we can construct $\mathrm{Cons}(d,d'') \epsilon D^{\infty}(s,a')$ and by formula (2-3), we know that $r^{\infty}\big(s,a',\mathrm{Cons}(d,d')\big) \varepsilon U$.
    Since $r^{\infty}\big(s,a',\mathrm{Cons}(d,d'')\big) = r^{\infty}(s[a/d], k(d), d'')$, we get the result.

  This finishes the proof that $w^{\infty \circ}(U)$ is a post-fixpoint for $U \cap w^{\circ}(\_)$, and thus the proof that $w^{\infty \circ}(U) \subseteq w^{\circ \infty}(U)$.

- "$w^{\circ \infty}(U) \subseteq w^{\infty \circ}(U)$": suppose $s \varepsilon w^{\circ \infty}(U)$, we need to find an action $a' \epsilon A^{\infty}(s)$ such that

  $$\big(\forall d'\ \epsilon\ D^{\infty}(s,a')\big)\ r^{\infty}(s,a',d') \varepsilon U\ . \tag{2-4}$$

  By the introduction rule for A^{∞}, we need a coalgebra (X,C) where $X : S \to \mathbf{Set}$ and $C \epsilon (s \epsilon S) \to X(s) \to w^{\circ}(X,s)$.

Take $X \triangleq w^{\circ\infty}(U)$; by the "post-fixpoint" rule for $w^{\circ\infty}$, we know that $X \subseteq w^{\circ}(X)$: this defines C. This allows to construct $a' \triangleq \mathrm{coiter}(X, C, s, x)$ where x is the proof that $s \varepsilon w^{\circ\infty}(U)$.
Instead of proving directly (2-4), we will prove something slightly more general: define $a'(s,x) \triangleq \mathrm{coiter}(X, C, s, x) \in A^{\infty}(s)$. We claim

$$(\forall s \epsilon S)\big(\forall x \epsilon X(s)\big)\Big(\forall d' \epsilon D^{\infty}\big(s, a'(s,x)\big)\Big)\ n^{\infty}(s, a'(s,x), d') \varepsilon U\ .$$

This implies (2-4) by specializing s and x as above...
Let $s \in S$, $x \in X(s)$ (*i.e.* x is an element of "$s \varepsilon w^{\circ\infty}(U)$") and $d' \in D^{\infty}(s, a')$. We proceed by induction on d':

- if $d' = \mathrm{Nil}$, then $n^{\infty}(s, a'(s,x), d') \varepsilon U$ becomes "$s \varepsilon U$". This holds because $s \varepsilon w^{\circ\infty}(U)$.
- if $d' = \mathrm{Cons}(d, d'')$ then, by definition, $n^{\infty}(s, a'(s,x), d') \varepsilon U$ is equivalent to $n^{\infty}(s[a/d], k(d), d'') \varepsilon U$ where $\mathrm{elim}\big(a'(s,x)\big) = (a, k)$.
  But since $a'(s,x) = \mathrm{coiter}(X, C, s, x)$, this implies, by the computation rule, that $\mathrm{elim}\big(a'(s,x)\big)$ is of the form

  $$\Big(a\ ,\ (\lambda d)\,.\,\mathrm{coiter}\big(X, C, s[a/d], g(d)\big)\Big)$$

  where $C(s,x) = (a, g)$. This means that a is a witness for $s \varepsilon w^{\circ}(U)$ (by definition of C).
  So, we have that $k(d)$ is in fact $\mathrm{coiter}\big(X, C, s[a/d], g(d)\big)$, *i.e.* $a'\big(s[a/d], g(d)\big)$.
  We can thus apply the induction hypothesis to conclude that

  $$n^{\infty}\Big(s[a/d], a'\big(s[a/d], g(d)\big), d''\Big) \varepsilon U\ .$$

This finishes the proof that $w^{\circ\infty}(U) \subseteq w^{\infty\circ}(U)$ and concludes the proof of proposition 2.5.18.

☑

As a corollary to proposition 2.5.18 and lemma 2.5.14, we get:

- **Lemma 2.5.19:** for any interaction system w, $w^{*\circ}$ is a closure operator and $w^{\infty\circ}$ is an interior operator.

This proof uses impredicative reasoning in the definition of $w^{\circ *}$ and $w^{\circ\infty}$. It is however not difficult to show *directly*, using only predicative reasoning, that w^{*} and w^{∞} are respectively closure and interior operators. The proof can in fact be extracted from the directions "$w^{\circ *} \subseteq w^{*\circ}$" and "$w^{\infty\circ} \subseteq w^{\circ\infty}$" in the proof of proposition 2.5.18.

2.5.6 An Equivalence of Categories

Homogeneous predicate transformers have their own notion of morphism, called *data-refinements*: (see [9])

- **Definition 2.5.20:** if F_1 and F_2 are predicate transformers on S_1 and S_2, a predicate transformer $P : \mathcal{P}(S_1) \to \mathcal{P}(S_2)$ is said to be a *data-refinement* from F_1 to F_2 if: $P \cdot F_1 \subseteq F_2 \cdot P$.

A data-refinement is said to be a *forward data-refinement* if it commutes with arbitrary unions; it is said to be a *backward data-refinement* if it commutes with arbitrary intersections.

It is trivial to show that homogeneous predicate transformers with data-refinements form a category and that predicate transformers with forward/backward refinements form two "subcategories".[5] By proposition 2.5.8, we know that forward and backward data-refinements are in fact given by relations. In particular, a relation $R : \mathbf{Rel}(S_1 \times S_2)$ is a forward data-refinement from F_1 to F_2 iff $R \cdot F_1 \subseteq F_2 \cdot R$.[6]

What is rather surprising is that this notion of forward data-refinement corresponds exactly to the notion of simulation:

- **Lemma 2.5.21:** for all interfaces w_1 and w_2, a relation $R : \mathbf{Rel}(S_1, S_2)$ is a simulation from w_1 to w_2 iff it is a forward data-refinement from w_1° to w_2°. In other words, R is a simulation iff $R \cdot w_1^\circ \subseteq w_2^\circ \cdot R$.

The "$\Leftarrow$" direction uses equality.

proof:

- suppose first that R is a simulation; let's show that $R \cdot w_1^\circ \subseteq w_2^\circ \cdot R$.

  $s_2 \;\varepsilon\; R \cdot w_1^\circ(U)$

  $\Rightarrow$ { definition of $\langle R^\sim \rangle$: for some s_1 }

  $(s_1, s_2) \;\varepsilon\; R \wedge s_1 \;\varepsilon\; w_1^\circ(U)$

  $\Rightarrow$ { definition of w_1°: there is an $a_1 \in A_1(s_1)$ }

  $(s_1, s_2) \;\varepsilon\; R \wedge \big(\forall d_1 \epsilon D_1(s_1, a_1)\big)\; s_1[a_1/d_1] \;\varepsilon\; U$

  $\Rightarrow$ { since R is a simulation, there is an $a_2 \in A_2(s_2)$ simulating a_1 }
  { moreover, for any d_2, there is a d_1 s.t. $(s_1[a_1/d_1], s_2[a_2/d_2]) \;\varepsilon\; R$ }

  $\big(\exists a_2 \epsilon A_2(s_2)\big)\big(\forall d_2 \epsilon D_2(s_2, a_2)\big)$
  $\big(\exists d_1 \epsilon D_1(s_1, a_1)\big)\; (s_1[a_1/d_1], s_2[a_2, d_2]) \;\varepsilon\; R \wedge s_1[a_1/d_1] \;\varepsilon\; U$

  $\Rightarrow$ { take s_1' to be $s_1[a_1/d_1]$ }

  $\big(\exists a_2 \epsilon A_2(s_2)\big)\big(\forall d_2 \epsilon D_2(s_2, a_2)\big)(\exists s_1' \epsilon S_1)\; (s_1', s_2[a_2, d_2]) \;\varepsilon\; R \wedge s_1' \;\varepsilon\; U$

  $\Leftrightarrow$ { definition }

  $s_2 \;\varepsilon\; w_2^\circ \cdot R(U)$

- For the other direction, let $R \cdot w_1^\circ \subseteq w_2^\circ \cdot R$, $(s_1, s_2) \;\varepsilon\; R$ and $a_1 \in A_1(s_1)$, we want to show that

  $$\big(\exists a_2 \epsilon A_2(s_2)\big)\big(\forall d_2 \epsilon D_2(s_2, a_2)\big)$$
  $$\big(\exists d_1 \epsilon D_1(s_1, a_1)\big)\; (s_1[a_1/d_1], s_2[a_2/d_2]) \;\varepsilon\; R\,.$$

  By definition, this is equivalent to

  $$s_2 \;\varepsilon\; w_2^\circ\left(\bigcup_{d_1 \epsilon D_1(s_1, a_1)} R(s_1[a_1/d_1])\right).$$

[5]: Precisely, the identity is a faithful functor from the category of predicate transformers with forward (backward) data-refinements to the category of predicate transformers with data-refinements.

[6]: Recall that we also write R for the predicate transformer $\langle R^\sim \rangle \ldots$

Since the predicate transformer R (which is in fact $\langle R^{\sim}\rangle$) commutes with unions, this is equivalent to

$$s_2 \;\varepsilon\; w_2^{\circ} \cdot R\left(\bigcup_{d_1 \in D_1(s_1,a_1)} \{s_1[a_1/d_1]\} \right) .$$

Since by hypothesis $R \cdot w_1 \subseteq w_2 \cdot R$, it is sufficient to show

$$s_2 \;\varepsilon\; R \cdot w_1^{\circ}\left(\bigcup_{d_1 \in D_1(s_1,a_1)} \{s_1[a_1/d_1]\} \right) .$$

We trivially have that $s_1 \;\varepsilon\; w_1^{\circ}\big(\bigcup_{d_1}\{s_1[a_1/d_1]\}\big)$, and since $(s_1, s_2) \;\varepsilon\; R$, we can conclude.
Notice that we need equality to be able to form the singleton predicates.
☑

An important corollary to this is that

$$w_1^{\circ} \subseteq w_2^{\circ} \quad \Leftrightarrow \quad \textbf{Id} \text{ is a simulation from } w_1 \text{ to } w_2 \; .$$

In other words, when dealing with the lattice of set-based predicate transformers, we are fully predicative: the order is not given by a $\mathbf{\Pi}^1_1$ formula, but by a proposition.

The moral of this section is the following:

> ◇ **Proposition 2.5.22:** *the operation $w \mapsto w^{\circ}$ is a full and faithful functor from* **Int** *to the category of predicate transformers with forward-data refinements.*

The category **Int** can be seen as the "predicative core" of predicate transformers, and as shown in the previous sections, this category is closed under all the relevant operations on predicate transformers.

Predicatively speaking, not all predicate transformers are set-based: define the predicate transformer $P : \mathcal{P}(\mathbf{B}) \to \mathcal{P}(\mathbf{B})$ with: *(where* $\mathbf{B} = \{\mathtt{False}, \mathtt{True}\}$*)*

$$U \quad \mapsto \quad U^{\diamond\diamond}$$

where $b \;\varepsilon\; U^{\diamond}$ iff $(\forall b' \epsilon U)\; b' \leqslant b$ and $\leqslant$ is the standard boolean order: False $\leqslant$ True. It is shown in [27](section 4.6) that this operator cannot be represented by an interaction system.

Impredicatively however, it is quite easy to find an interaction system to represent *any* predicate transformer: if F is a (monotonic) predicate transformer, define w_F as follows:

$$\begin{array}{lcl} w_F.A(s) & \triangleq & (\Sigma U{:}\mathcal{P}(S))\; s \;\varepsilon\; F(U) \\ w_F.D\big(s,(U,\_)\big) & \triangleq & (\Sigma s' \epsilon S)\; s' \;\varepsilon\; U \\ w_F.n\big(s,(U,\_),(s',\_)\big) & \triangleq & s' \; ; \end{array}$$

or, if we want to use traditional mathematic notations:

$$w_F.A(s) \qquad \triangleq \quad \{U : \mathcal{P}(S) \mid s \;\varepsilon\; F(U)\}$$

$$\begin{array}{lcl} w_F.D(s,U) & \triangleq & U \\ w_F.n(s,U,s') & \triangleq & s' \ . \end{array}$$

It is easy to check that $F = (w_F)^\circ$, which essentially means that the full and faithful functor $w \mapsto w^\circ$ is surjective. Moreover, we leave it as an exercise to check that we have $\mathbf{Eq}_S : w \simeq w_{w^\circ}$ which proves that:

◇ **Proposition 2.5.23: (impredicative)**
the categories **Int** *of interaction systems with simulations and the category of homogeneous predicate transformers with forward data-refinements are equivalent. Moreover, this equivalence is a retract from interaction systems to predicate transformers.*[7]

2.6 A Model for Component based Programming

We gave several interpretations for interaction systems in section 2.1.2. The main idea is that an interaction system is a *contract* between two entities (the Angel and the Demon) describing possible interactions. This is just what programming is about...

2.6.1 Interfaces

In object oriented programming, an *interface* is given by a collection of types for different methods an object is supposed to provide. As a basic example, the object STACK consisting of stacks of booleans is specified by the following interface:

$$\begin{array}{lcl} \mathrm{pop} & \epsilon & \mathbf{B} \\ \mathrm{push} & \epsilon & \mathbf{B} \to () \end{array}$$

with the following meaning:

- you can apply the method "pop" (on a stack) to obtain a boolean (type **B**);
- you can apply the method "push", which needs one argument of type boolean, which will "do something".

What those commands actually *do* is only specified in the documentation and is not available from the interface.

Interaction systems can serve as much more expressive interfaces which contains also the specification of the commands. For the example of STACK, we could have an interface like:

$$\begin{array}{lcl} S & \triangleq & \textsc{List}(\mathbf{B}) \\ A(l) & \triangleq & \textbf{case } l \text{ of } \mathrm{Nil} \Rightarrow \{\mathrm{push}(b) \mid b \epsilon \mathbf{B}\} \\ & & \qquad\qquad\quad \_ \ \Rightarrow \{\mathrm{push}(b) \mid b \epsilon \mathbf{B}\} \cup \{\mathrm{pop}\} \\ D\big(l, \mathrm{push}(b)\big) & \triangleq & \{\mathrm{akn}\} \\ D(l, \mathrm{pop}) & \triangleq & \{\mathrm{akn}\} \end{array}$$

[7]: This equivalence is a retract in the sense that the functors satisfy $F \mapsto (w_F)^\circ = \mathbf{Id}$ while we only have $w \mapsto w_{w^\circ} \simeq \mathbf{Id}$.

$$\begin{array}{lcl} n(l, \mathrm{push}(b)) & \triangleq & \mathrm{Cons}(b, l) \\ n(\mathrm{Cons}(b, l), \mathrm{pop}) & \triangleq & l \end{array}$$

with the meaning:

- an object of type STACK has the possible states described by lists of booleans;
- if the state is not empty, we can either do a pop or a push; if the state is empty, we can only do a push;
- in both cases, the environment can only acknowledge the command;
- performing a pop removes the first element of the state; performing a push puts an element at the beginning of the state.

The only thing that is still missing from this specification is the fact that a pop command will actually produce a boolean which is given by the first element of the state. Dealing with that is possible in concrete setting, but requires the introduction of interaction systems with "side-effects", which lies outside the scope of this thesis. In a way, this interaction system only specifies *how* a stack can be legally used.

In order to get less trivial specifications, it is possible to add the possibility of errors from the environment (the Demon): the set $D(s, a)$ would then be something like $\{\mathrm{akn}, \mathrm{error}\}$. Any textbook on process calculus would contain many examples of vending machines and other automata which can be adequately described using interaction systems. In most cases, the environment is not "deterministic" in the sense that the set of reactions to a command is not a singleton.

A specification also comes with a subset of states, from which it is supposed to work. This justifies the following:

▷ **Definition 2.6.1:** an *initialized interface* is given by a set of states S, an interaction w on S and a predicate $\textsc{Init} : \mathcal{P}(S)$ of initial states.

The intuition is that the initial state predicate represents states from which the Angel may be asked to start interaction. For initialized interface $(w, \textsc{Init})$, interaction goes as follows:

0) the Demon starts by choosing a state $s_0 \in \textsc{Init}$;
1) the Angel chooses an action $a_0 \in A_0(s_0)$;
2) the Demon chooses a reaction $d_0 \in D_0(s_0, a_0)$;
3) the Angel chooses an action $a_1 \in A(s_0[a_0/d_0])$;
-) ...

2.6.2 Components: Refinements

A programmer is hired to program: he is given a description of the program he is supposed to write, together with a description of the libraries he is allowed to use. In our context, descriptions are simply interfaces; we call the former "high-level" and the latter "low-level". Implementing a particular high-level command amounts to producing a sequence of low-level commands and show that this sequence behaves according to the high-level specification. This program is called a *component* between the high-level and the low-level. This justifies the following definition:

▷ **Definition 2.6.2:** let $(w_h, \textsc{Init}_h)$ and $(w_l, \textsc{Init}_l)$ be two initialized interfaces; a *refinement*, or a *component* from $(w_h, \textsc{Init}_h)$ to $(w_l, \textsc{Init}_l)$ is a simulation R from w_h to w_l^* s.t. $\textsc{Init}_h \subseteq R(\textsc{Init}_l)$.

With this definition, one can identify the activity of programming with the act of proving that a carefully crafted relation is a refinement from a high-level specification (which we want implemented) to a low-level specification (which is already implemented).

It is trivial to see that the identity on S (if available) is a refinement from any specification to itself. As we'll see in section 3.3.1, we can also prove that the relational composition of two refinements is a refinement: this is due to the fact that $\_^*$ is a monad in the category **Int**. For now, we can simply rely on the intuition we have about refinement and assert:

- **Lemma 2.6.3:** initialized interfaces with refinements form a category.

We will slightly refine this category in section 2.6.5 by giving an appropriate definition of equality between refinements.

2.6.3 Clients and Servers

We saw that interaction systems can be seen as "contracts", or "protocols" describing interaction between two entities. The two mains kinds of programs obeying such contracts are given by the notions of *server programs* and *client programs*.

Before looking at the details, let's introduce the following notation: for any interaction systems w,

- $s \lhd_w U$ for $s \varepsilon w^{*\circ}(U)$;
- $U \lhd_w V$ for $U \subseteq w^{*\circ}(V)$;
- $s \ltimes_w U$ for $s \varepsilon w^{\perp\infty\circ}(U)$;
- $U \ltimes_w V$ for $U \between w^{\perp\infty\circ}(V)$.

The symbol $\lhd$ is read "covered by" and the symbol $\ltimes$ is read "restricted by". The intuition behind covering is that $s \lhd U$ means "provided the Demon reacts, the Angel has a way to reach U from s". For restriction, the intuition is slightly subtler: we have

$s \ltimes V$

$\Leftrightarrow$ { by definition of $\ltimes$, proposition 2.5.18 and lemma 2.5.4 }

$s \varepsilon w^{\bullet\infty}(V)$

$\Rightarrow$ { by the rule "post-fixpoint }

$s \varepsilon V \cap w^{\bullet}(w^{\bullet\infty}(V))$

$\Leftrightarrow$

$s \varepsilon V$ and $(\forall a \epsilon A(s))(\exists d \epsilon D(s,a))\ s[a/d] \ltimes V$

The meaning of $s \ltimes V$ is thus "no matter what the Angel does, the Demon has a way to remain in V".

§ *Server Programs.* A server program is simply a program which "runs forever". The perfect example of server is given by the "environment" in which we run other programs: the environment simply waits for a request, and sends back some response. In the Unix terminology, such programs are called "daemon programs". The goal of a server program is to make sure something always holds: it is strongly related to the notion of invariant predicates.

A server specification for the initialized interface $(S, A, D, n, \textsc{Init})$ is given by an invariant $\textsc{Inv} : \mathcal{P}(S)$ which can be maintained by the Demon. To make sure the server can be started, we also require the invariant to intersect the initial predicate.

- That INV is a Demon invariant means that from INV, the Demon can always respond in such a way as to remain in INV:

$$s \in \text{INV} \quad \Rightarrow \quad \big(\forall a \in A(s)\big)\big(\exists d \in D(s,a)\big)\ s[a/d] \in \text{INV} \tag{2-5}$$

  i.e. that $\text{INV} \subseteq w^\bullet(\text{INV})$;
- that INV intersects INIT is simply a way to ensure that the Demon will be able to launch the server program from some initial state:

$$\text{INIT} \between \text{INV} \ . \tag{2-6}$$

By lemma 2.5.16 and proposition 2.5.18, we know that (2-5) is equivalent to saying that $\text{INV} = w^{\bullet\infty}(V)$ for some V. Using the notation defined earlier, we thus have that (2-5) and (2-6) are equivalent to

$$\text{INIT} \ltimes_w V \tag{2-7}$$

where V is a predicate on S. A server program satisfying specification (2-7) is nothing more than a constructive proof of (2-7).

§ *Client Programs.* The notion of client server is dual to that of server program: a client interacts with a server by sending requests, and waiting for the server's response. A client has something in mind, a goal she wants to achieve. The simplest example takes the form of a predicate GOAL on states which the client wants to reach. This means that, whatever the initial state of the interface is, she will have a way to choose actions in such a way as to reach GOAL after a finite amount of interaction. Such a server program in entirely described by a constructive proof that

$$\text{INIT} \lhd_w \text{GOAL} \ . \tag{2-8}$$

The duality with the notion of server program is obvious: in (2-7), we are dealing with a Demon infinite strategy while in (2-8) we are dealing with a well-founded Angel strategy.

2.6.4 The Execution Formula

Suppose we are given both a client program *and* a server program on the same specification (w, INIT). It is natural to look at the result of "connecting" the client to the server. If the client is given by a proof that $s_i \lhd \text{GOAL}$ and the server by a proof that $s_i \ltimes V$, then we can conduct interaction and obtain a final state $s_f \in S$ s.t.

- the client has reached her goal, *i.e.* $s_f \in \text{GOAL}$;
- the server can accept new connections and continue to maintain V, *i.e.* $s_f \ltimes V$.

Suppose that we have a proof that $s_i \lhd_w \text{GOAL}$; in (weak head) normal form, this proof has the shape (p, g) where

$$p \in A^*(s_i)$$
$$g \in \big(d' \in D^*(s_i, p)\big) \to \text{GOAL}(s_i[p/d']) \ .$$

In other words, this proof is either of the form (Exit, g) where $g(\text{Nil}) \in \text{GOAL}(s_i)$, or $\big(\text{Call}(a, f), g\big)$ where:

$$a \in A(s_i)$$

$$f \in \big(d \in D(s_i, a)\big) \to A^*(s_i[a/d])$$
$$g \in \Big((d, d') \in D^*\big(s_i, \mathrm{Call}(a, k)\big)\Big) \to \textsc{Goal}\big(s_i[a/d][f(d)/d']\big) \ .$$

For the server program, a proof of $s_i \ltimes_w V$ in (weak head) normal form looks like (q, l) where $q \in A^{\perp\infty}(s_i)$ and $l \in \big(a' \in D^{\perp\infty}(s_i, q)\big) \to V(s_i[q/a'])$. Applying "elim" on q yields a tuple (r, k) where:

$$r \in A^{\perp}(s_i) \quad = \quad \big(a \in A(s_i)\big) \to D(s_i, a)$$
$$k \in \big(a \in D^{\perp}(s_i, r)\big) \to A^{\perp\infty}(s_i[r/a]) \quad = \quad \big(a \in A(s_i)\big) \to A^{\perp\infty}\big(s_i[a/r(a)]\big) \ .$$

We can now define the function "exec". In the environment where $\textsc{Goal}, V \subseteq S$, we have: *(recall that $U \between U'$ is defined as $(\Sigma s \in S)\, U(s) \times U'(s)$ so that an element of $U \between U'$ is a triple)*

$$\mathrm{exec}\big((s, P, Q) \in w^*(\textsc{Goal}) \between w^{\perp\infty}(V)\big) \quad \in \quad \textsc{Goal} \between w^{\perp\infty}(V)$$

i.e. P is a proof that $s \lhd \textsc{Goal}$ and Q is a proof that $s \ltimes V$.

$$\begin{aligned}
\mathrm{exec}\big(s, (\mathrm{Exit}, g), (q, l)\big) &\triangleq \big(s, g(\mathrm{Nil}), (q, l)\big) \\
\mathrm{exec}\Big(s, \big(\mathrm{Call}(a, f), g\big), (q, l)\Big) &\triangleq \mathrm{exec}\big(s[a/d], (p', g'), (q', l')\big) \\
&\qquad \textit{where } (r, k) = \mathrm{elim}(q) \\
&\qquad\qquad d \triangleq r(a) \\
&\qquad\qquad p' \triangleq f(d) \\
&\qquad\qquad g' \triangleq \lambda d' \,.\, g\big(\mathrm{Cons}(d, d')\big) \\
&\qquad\qquad q' \triangleq k(a) \\
&\qquad\qquad l' \triangleq \lambda a' \,.\, l\big(\mathrm{Cons}(a, a')\big)
\end{aligned}$$

In the case of servers and clients described by interfaces as above, we can summarize this execution function in the following rule:

$$\frac{\textsc{Init} \lhd_w \textsc{Goal} \qquad \textsc{Init} \ltimes_w V}{\textsc{Goal} \ltimes_w V}\ \text{execution} \ .$$

As will be explained in section 4.2, this rule is exactly what is known as the "compatibility formula" required to hold in Giovanni Sambin's basic topologies.

An important feature is missing from the present model of client/server interaction: it is often the case that there may be several clients connecting to a single server. The server then needs to deal with several requests simultaneously. Being able to simulate such *concurrent* interaction in a sequential manner is important. This will be the subject of section 4.3.2.

2.6.5 Saturation of Refinements

The intuition we have about refinements is that a (proof of a) refinement is a process simulating high-level commands by sequences of low-level commands. Extensional equality of the underlying relation predicates is certainly too crude a notion for identifying refinements. We develop a notion of "saturation", which will take a refinement into a "biggest" refinements having the same simulating potential. The idea is to identify two refinements when they are extensionally equal, *up to internal interaction*.

▷ **Definition 2.6.4:** if R is a refinement from w_h to w_l, we call *saturation* of R the following relation:

$$(s_h, s_l) \in \overline{R} \quad \triangleq \quad s_l \lhd_{w_l} R(s_h)$$

i.e. $\overline{R} \triangleq w_l^* \cdot R$.

Thus, s_h and s_l are related via the saturation of R if there is a low-level Angel strategy going from s_l to states which are related to s_h via R.

○ **Lemma 2.6.5:** if R is a refinement from w_h to w_l, then $\overline{R}$ is also a refinement from w_h to w_l.

proof (checked in Agda): by lemma 3.3.4, we need to show, for any $a_h \in A_h(s_h)$, that

$$\overline{R}(s_h) \quad \lhd_{w_l} \quad \bigcup_{d_h \in D_h(s_h, a_h)} \overline{R}(s_h[a_h/d_h]) \ . \tag{2-9}$$

By this same lemma 3.3.4, because R is a refinement, we know that

$$R(s_h) \quad \lhd_{w_l} \quad \bigcup_{d_h \in D_h(s_h, a_h)} R(s_h[a_h/d_h]) \ .$$

Since w_l^* is a closure operator (lemma 2.5.19), we can conclude that

$$w_l^* \cdot R(s_h) = \overline{R}(s_h) \quad \lhd_{w_l} \quad \bigcup_{d_h \in D_h(s_h, a_h)} R(s_h[a_h/d_h]) \ . \tag{2-10}$$

Since we always have $R(s_h[a_h/d_h]) \subseteq w_l^* \cdot R(s_h[a_h/d_h])$, we also know that

$$R(s_h[a_h/d_h]) \quad \lhd_{w_l} \quad w_l^* \cdot R(s_h[a_h/d_h]) = \overline{R}(s_h[a_h/d_h])$$

and thus

$$\bigcup_{d_h \in D_h(s_h, a_h)} R(s_h[a_h/d_h]) \quad \lhd_{w_l} \quad \bigcup_{d_h \in D_h(s_h, a_h)} \overline{R}(s_h[a_h/d_h]) \tag{2-11}$$

By transitivity applied to (2-10) and (2-11), we obtain (2-9). This concludes the proof.

☑

This provides a better way to compare two refinements: compare their saturation.

▷ **Definition 2.6.6:** if R_1 and R_2 are two refinements from w_h to w_l, we say:
- that R_2 is *stronger* than R_1 if $\overline{R_1} \subseteq \overline{R_2}$, we write $R_1 \sqsubseteq R_2$;
- that R_1 and R_2 are equivalent if $R_1 \sqsubseteq R_2$ and $R_2 \sqsubseteq R_1$, we write $R_1 \approx R_2$.

The following is quite easy:

○ **Lemma 2.6.7:**
- $\sqsubseteq$ is a preorder on $\mathbf{Ref}(w_h, w_l)$;
- $\approx$ is an equivalence relation on $\mathbf{Ref}(w_h, w_l)$;
- $R \mapsto \overline{R}$ is a closure operation;
- $\overline{R}$ is the largest relation in the equivalence class of R.

Finally:

◇ **Proposition 2.6.8:** $(\mathbf{Ref}, \sqsubseteq)$ *is an order enriched category.*

proof (checked in Agda): the only remaining thing to check is that composition is monotonic w.r.t. $\sqsubseteq$ on the right and on the left.

Let R_1, R_2 be two simulations from w_h to w_m and Q_1, Q_2 two simulations from w_m to w_l such that $R_1 \sqsubseteq R_2$ and $Q_1 \sqsubseteq Q_2$. Suppose moreover that $s_h \in S_h$; we need to show that $\overline{Q_1 \cdot R_1}(s_h) \subseteq \overline{Q_2 \cdot R_2}(s_h)$:

Because $R_1 \sqsubseteq R_2$, we have

$$\overline{R_1}(s_h) \quad \subseteq \quad \overline{R_2}(s_h) \ .$$

We also claim that

$$Q_1 \cdot \overline{R_1}(s_h) \quad \lhd_l \quad Q_2 \cdot \overline{R_2}(s_h) \ . \tag{2-12}$$

Let $s_l \varepsilon Q_1 \cdot \overline{R_1}(s_h)$, *i.e.* $(s_m, s_l) \varepsilon Q_1$ for some s_m s.t. $(s_h, s_m) \varepsilon \overline{R_1}$. We will show that $s_l \lhd_l Q_2 \cdot \overline{R_2}(s_h)$:

- $Q_2(s_m) \subseteq Q_2 \cdot \overline{R_2}(s_h)$ since $s_m \varepsilon \overline{R_1}(s_h) \subseteq \overline{R_2}(s_h)$;
- $s_l \varepsilon w_l^* \cdot Q_2(s_m)$ because $Q_1 \sqsubseteq Q_2$ and $s_l \varepsilon \overline{Q_1}(s_m)$; *(since $s_l \varepsilon Q_1(s_m)$)*
- so by monotonicity, $s_l \varepsilon w_l^* \cdot Q_2 \cdot \overline{R_2}(s_h)$.

From (2-12), we get

$$w_l^* \cdot Q_1 \cdot \overline{R_1}(s_h) \quad \subseteq \quad w_l^* \cdot Q_2 \cdot \overline{R_2}(s_h) \ . \tag{2-13}$$

Now, for any simulation $R : \mathbf{Int}(w, w'^*)$, we have $w'^* \cdot R \cdot w^*(U) = w'^* \cdot R(U)$:

- "$\subseteq$": because $R \cdot w^*(U) \subseteq w'^* \cdot R(U)$ and w'^* is a closure operator;
- "$\supseteq$": $\mathbf{skip} \subseteq w^* \Rightarrow w'^* \cdot R \subseteq w'^* \cdot R \cdot w^*$.

Applying this remark to (2-13), we can conclude:

$$\overline{Q_1 \cdot R_1}(s_h) = w_l^* \cdot Q_1 \cdot R_1(s_h) \quad \subseteq \quad w_l^* \cdot Q_2 \cdot R_2(s_h) = \overline{Q_2 \cdot R_2}(s_h) \ .$$

☑

This allows to make the following definition:

▷ **Definition 2.6.9:** refinements modulo $\approx$ form a sound notion of morphisms between interaction systems. We call the resulting category $\mathbf{Ref}_{\approx}$.

3 Categorical Structure

3.1 A Few Words about Categories

§ *Universal Constructions in a Predicative Setting.* As we'll see in this chapter, and later in section 6.1, this category enjoys many algebraic properties. However, we haven't said anything about the way to formalize categories in predicative type theory: a category $\mathcal{C}$ is obviously given by a collection of objects and for each pair of objects A and B, a collection $\mathcal{C}(A, B)$ *with a notion of equality.* In our case, equality of simulations is simply extensional equality of the underlying relations. We also need a notion of composition and identities with the obvious requirements.

In order to make sense of universal constructions, which use heavily quantification on morphisms, we need to introduce specific constructions, and prove that they satisfy the universal properties (which involves only $\mathbf{\Pi}^1_1$ quantification). Let's look at the example of the cartesian product in $\mathcal{C}$:

- we should construct the object $A \times B$ with the projections $\pi^A_{A,B}$ and $\pi^B_{A,B}$;
- we should construct the pairing $\langle f, g\rangle$ for any pair of morphisms;
- those should satisfy:

$$\begin{aligned} &(\forall A, B, C)\big(\forall f : \mathcal{C}(C, A)\big)\big(\forall g : \mathcal{C}(C, B)\big) \\ &\qquad \pi^A_{A,B} \cdot \langle f, g\rangle = f \quad \wedge \quad \pi^B_{A,B} \cdot \langle f, g\rangle = g \\ \wedge \quad &\big(\forall h : \mathcal{C}(C, A \times B)\big)\ \pi^A_{A,B} \cdot h = f \wedge \pi^B_{A,B} \cdot h = g \Rightarrow h = \langle f, g\rangle\ . \end{aligned}$$

This is an instance of $\mathbf{\Pi}^1_1$ quantification and it does make sense in a predicative setting.

§ *The Category of Relations.* At the core of the category of interaction systems is the category of sets and relations: it is defined in the obvious way:

▷ **Definition 3.1.1:** the objects of the category **Rel** are sets and its morphisms are relations. The identity is the equality relation and composition is defined as the usual composition of relations.

We do not prove all the results about **Rel**: they belong to the folklore of categories.

There is an obvious faithful "forgetful" functor $|\_|$ from **Int** to **Rel** defined by

$$w = \big(S, (A, D, n)\big) \quad \mapsto \quad |w| = S \ .$$

As a result, we can lift some universality results from **Rel** to **Int**: if we can show that a universal construction from **Rel** is admissible in **Int** (*i.e.* the relations are in fact simulations for the corresponding interaction systems), then the construction is universal in **Int** as well.

Since we do not really accept the idea of an identity for all sets, the structure of **Rel** might be more adequately described by a weaker structure:

▷ **Definition 3.1.2:** a *precategory* is given by a collection (type) of objects $\mathcal{C}$, together with, for each pair A, B of objects, a collection $\mathcal{C}(A, B)$ of morphisms equipped with a notion of equality.[1] The structure also needs an associative notion of composition:

$$\mathrm{comp}_{A,B,C} \quad : \quad \mathcal{C}(A, B) \to \mathcal{C}(B, C) \to \mathcal{C}(A, C) \qquad \textit{for all objects } A\ B\ C.$$

Depending on the framework, we may downgrade the traditional category **Rel** to a precategory in order to make sense of constructions without relying on a general equality type.

3.2 Some Easy Properties

3.2.1 Constants

§ *The Initial/Terminal Object.* **null** is the only interaction system available on the empty set of states. Since $\mathbf{Rel}(\emptyset, S) = \mathcal{P}(\emptyset) \simeq \{*\}$, there is at most one simulation from **null** to any other interaction system: the empty relation. Since, as we have seen on page 49, the empty relation is *always* a simulation; we can conclude that for any interface (S, w), there is exactly one simulation from **null** to (S, w). The same argument applies to simulations from (S, w) to **null**.

○ **Lemma 3.2.1:** in **Int**, the object **null** is a *zero object:* it is both initial and terminal.

As a corollary, we can directly conclude that **Int** is not cartesian closed:

• **Corollary 3.2.2:** **Int** is not cartesian closed.

proof: folklore.

☑

[1]: *i.e.* there is an equivalence relation on each $\mathcal{C}(A, B)$.

§ *"**abort**" and "**magic**".* Computationally speaking, the **null** interaction system doesn't make sense: its set of states is empty! That it is terminal and initial is thus irrelevant. However, the objects **magic** and **abort** can almost play the rôle of terminal and initial:

- **Lemma 3.2.3:** for any interaction system w, all relations from $\{*\}$ to S are simulation from **abort** to w. Dually, all relations from S to $\{*\}$ are simulations from w to **magic**.
  More generally, the functor $|\_| : \mathbf{Int} \to \mathbf{Rel}$ has a right-adjoint and a left-adjoint $\mathbf{abort}(\_) \vdash |\_| \vdash \mathbf{magic}(\_)$.

proof: let (S, w) be an interface, and let R be a relation from $\{*\}$ to S. It is trivial to show that R is a simulation from **abort** to (S, w): suppose $(*, s)$ ε R, we need to show that

$$\big(\forall a \in \mathbf{abort}.A(*)\big)(\exists \ldots)(\forall \ldots)(\exists \ldots) \ \ldots$$

Since $\mathbf{abort}.A(*)$ is the empty set, this is always true!
The dual statement for **magic** is similar.

For the adjunction, define $\mathbf{magic}(S)$ and $\mathbf{abort}(S)$, two interaction systems on S as:

$$\begin{array}{lcl@{\qquad}lcl} \mathbf{abort}(S).A(x) & \triangleq & \emptyset & \mathbf{magic}(S).A(x) & \triangleq & \{*\} \\ \ldots & & & \mathbf{magic}(S).D(x,*) & \triangleq & \emptyset \\ & & & \ldots & & \end{array}$$

It is easy to check that both are functors and that we have the natural isomorphism

$$\mathbf{Int}\big(\mathbf{abort}(S), w\big) \simeq \mathbf{Rel}(S, |w|) \quad \text{and} \quad \mathbf{Int}\big(w, \mathbf{magic}(S)\big) \simeq \mathbf{Rel}(|w|, S) \ .$$

☑

§ *"**skip**" and Invariant Predicates.* There is one more constant which will be of great importance in Part II: **skip**. It enjoys a very strong categorical property (see section 3.5) which unfortunately only holds classically. For the moment, we will only note the connection between **skip** and invariant predicates. Remark first that any relation from S to $\{*\}$ can be identified with a predicate on S, so that the following lemma is well-formed:

- **Lemma 3.2.4:** let (S, w) be an interface, and $U : \mathcal{P}(S)$, we have
  - U is a simulation from **skip** to w iff $U \subseteq w^{\circ}(U)$;
  - U is a simulation from w to **skip** iff $U \subseteq w^{\bullet}(U)$.

3.2.2 Product and Coproduct

Let's now come to the first operation defined on page 40: the sum operation "$\_ \oplus \_$" on interaction systems. The intuition of "disjoint sum" turns out to be exact:

- **Lemma 3.2.5:** "$\_ \oplus \_$" is a bifunctor on **Int**, it is the coproduct.

proof: since disjoint union is the coproduct in the category **Rel**, it is enough to check that the constructions on **Rel** yield simulations when applied to simulations.
For the "injections" from any w_1 and w_2 to $w_1 \oplus w_2$, define:

$$\begin{array}{lcl} R_l & \triangleq & \{(s_1, \mathrm{inl}(s_1')) \mid s_1 =_{S_1} s_1'\} \\ R_r & \triangleq & \{(s_2, \mathrm{inr}(s_2')) \mid s_2 =_{S_2} s_2'\} \end{array}$$

i.e. those simulations require equality... It is trivial to check that they are indeed simulations from w_1 to $w_1 \oplus w_2$ and from w_2 to $w_1 \oplus w_2$ respectively.
For "copairing", if R_1 is a simulation from w_1 to w and R_2 is a simulation from w_2 to w, the copairing $[R_1, R_2] : \mathbf{Rel}(S_1 + S_2, S)$ is defined as:

$$\begin{array}{lcl} [R_1, R_2] & \triangleq & \{(\mathrm{inl}(s_1), s) \mid (s_1, s) \varepsilon R_1\} \\ & & \cup \{(\mathrm{inr}(s_2), s) \mid (s_2, s) \varepsilon R_2\} \end{array}$$

which is trivially seen to be a simulation. That $w_1 \oplus w_2$ with copairing satisfies the appropriate universal property follows from the fact that they do so in **Rel**...
☑

Just like **null** is both initial and terminal, so is $\_ \oplus \_$ both a coproduct and a product! This is not surprising since the situation is the same in the category **Rel**.

○ **Lemma 3.2.6:** "$\_ \oplus \_$" is the cartesian product on **Int**.

proof: this is *exactly* the converse of the previous lemma:
- projections are the converse of injections;
- pairing is the converse of copairing.

☑

The fact that the product and the coproduct are the same could have been deduced in section 2.4.3, since for any category enriched over commutative monoids,[2] finite products and finite coproducts coincide (if they exist).

3.3 Iteration

3.3.1 Angelic Iteration: a Monad

The operation of Angelic iteration (page 45) enjoys a strong algebraic property in the category of interfaces:

◇ **Proposition 3.3.1:** *in the category* **Int**, *"$\_^*$" is a monad.*

Using the well-known Kleisli construction, this will justify the notion of refinement defined in section 2.6.

proof (checked in Agda): we postpone the proof of this fact after the next paragraph, when an appropriate definition of monad has been given.
☑

[2]: Sup-lattice enrichment entails monoid enrichment in an obvious way: addition is given by binary sups...

§ *An Appropriate Definition of Monad.* A monad on a category $\mathcal{C}$ is an endofunctor M together with two natural transformations $\eta : \_ \to M(\_)$ and $\mu : MM(\_) \to M(\_)$ satisfying, for all objects A:

1) $\mu_A \cdot \mu_{M(A)} = \mu_A \cdot M\mu_A$;

2) $\mu_A \cdot \eta_{M(A)} = \mu_A \cdot M\eta_A = \mathbf{Id}_{M(A)}$.

The aim of this section is to show that the following makes $\_^*$ into a monad on **Int**:

$$\eta_w \triangleq \mathbf{Eq}_S \quad \text{and} \quad \mu_w \triangleq \mathbf{Eq}_S .$$

The above equations trivially hold, so that the only difficulty is proving that they are indeed natural transformations.

However, since we are trying to avoid the use of equality, this definition is not entirely appropriate. In particular, equality is not needed for the application of section 2.6. We will thus use an alternate definition: recall that a monad in "triple form" is given by an operation M on objects, with a morphism $\eta_A : \mathcal{C}\big(A, M(A)\big)$ for any object A and for any morphism $f : \mathcal{C}\big(A, M(B)\big)$, a morphism $f^\natural : \mathcal{C}\big(M(A), M(B)\big)$ such that:

1) $f^\natural \cdot \eta_A = f$ for all $f : \mathcal{C}\big(A, M(B)\big)$;

2) $\eta_A^\natural = \mathbf{Id}_{M(A)}$ for all object A;

3) $(g^\natural \cdot f)^\natural = g^\natural \cdot f^\natural$ for all $f : \mathcal{C}\big(A, M(B)\big)$ and $g : \mathcal{C}\big(B, M(C)\big)$.

With this definition, we can remove one occurrence of the identity and use:

$$\eta_w \triangleq \mathbf{Eq}_S \quad \text{and} \quad R^\natural \triangleq R .$$

We can do a little better and patch the definition of a monad in triple form in the following *ad-hoc* way:

▷ **Definition 3.3.2:** a *premonad* on a precategory $\mathcal{C}$ is given by an operation M on objects of $\mathcal{C}$, together with:

- for any morphism $f : \mathcal{C}\big(A, M(B)\big)$, a morphism $f^\natural : \mathcal{C}\big(M(A), M(B)\big)$;
- and for any morphism $g : \mathcal{C}\big(M(A), M(B)\big)$, a morphism $g_\natural : \mathcal{C}\big(A, M(B)\big)$

satisfying:

1) $(f^\natural)_\natural = f$ for all $f : \mathcal{C}\big(A, M(B)\big)$;

2) $(g^\natural \cdot f)^\natural = g^\natural \cdot f^\natural$ for all $f : \mathcal{C}\big(A, M(B)\big)$ and $g : \mathcal{C}\big(B, M(C)\big)$;

3) $(g^\natural \cdot f)_\natural = g^\natural \cdot f_\natural$. for all $f : \mathcal{C}\big(M(A), M(B)\big)$ and $g : \mathcal{C}\big(B, M(C)\big)$.

This definition is justified by:

○ **Lemma 3.3.3:** for a real category, a premonad M is a monad iff it satisfies $\big(\mathbf{Id}_{M(A)_\natural}\big)^\natural = \mathbf{Id}_{M(A)}$ for all objects A.

proof: just notice that one can go from a monad in triple form to a premonad by defining $f_\natural \triangleq \eta \cdot f$ and vice and versa by defining $\eta_A \triangleq (\mathbf{Id}_{M(A)})_\natural$. The rest is obvious.

☑

The interest of this notion, as far as we are concerned is that we can now define all the *data* for the premonad: the next paragraph will show that we can take:

$$R^\natural \triangleq R \quad \text{and} \quad R_\natural \triangleq R$$

to make $\_^*$ into a premonad. Since we do have $\big(R_\natural\big)^\natural = R$ for all relations R, we will have in particular $(\mathbf{Id}_\natural)^\natural = \mathbf{Id}$ when we allow identities.

The concept of premonad seems appropriate if one works in a precategory (definition 3.1.2). One problem with this concept is that when there are no identities, we *cannot define the action of* M *on morphisms!* In other words, a premonad M is not necessarily functorial.

§ *Proof of Proposition 3.3.1.* We need to show two things:
- if $R : \mathbf{Int}(w_1, w_2^*)$, then $R^\natural \triangleq R$ it is also a simulation from w_1^* to w_2^*;
- if $R : \mathbf{Int}(w_1^*, w_2^*)$, then $R_\natural \triangleq R$ it is also a simulation from w_1 to w_2^*.

This has been checked in Agda.
The second point is quite easy: it even holds if we replace w_2^* by an arbitrary w. Suppose that R is simulation from w_1^* to w, *i.e.*

$$\begin{array}{rcl}(s_1, s) \in R & \Rightarrow & \big(\forall a_1' \in A_1^*(s_1)\big)\big(\exists a \in A(s)\big) \\ & & \big(\forall d \in D(s,a)\big)\big(\exists d_1' \in D_1^*(s_1, a_1')\big) \\ & & \quad \big(s_1[a_1'/d_1'], s[a/d]\big) \in R\ .\end{array}$$

In particular, for any $a_1 \in A(s_1)$, we can define $a_1' \triangleq \mathrm{Call}\big(a_1, (\lambda d_1).\mathrm{Exit}\big)$, the strategy which plays a_1 and stops. In this case, $D_1^*(s_1, a_1') = D_1(s_1, a_1) \times \{*\}$, and $n_1^*\big(s_1, a_1', (d_1, *)\big) = n_1(s_1, a_1, d_1)$, so that we get

$$\begin{array}{rcl}(s_1, s) \in R & \Rightarrow & \big(\forall a_1 \in A_1(s_1)\big)\big(\exists a \in A(s)\big) \\ & & \big(\forall d \in D(s,a)\big)\big(\exists d_1 \in D_1(s_1, a_1)\big) \\ & & \quad \big(s_1[a_1/d_1], s[a/d]\big) \in R\end{array}$$

i.e. that R is a simulation from w_1 to w.

For the other point, we start by proving a variant of lemma 2.5.21 which doesn't involve equality:

- **Lemma 3.3.4:** if w_1 and w_2 are interfaces, a relation R from S_1 to S_2 is a simulation from w_1 to w_2 iff, for all s_1 in S_1 and $a_1 \in A_1(s_1)$, we have

$$R(s_1) \quad \subseteq \quad w_2^\circ\left(\bigcup_{d_1 \in D_1(s_1, a_1)} R\big(s_1[a_1/d_1]\big)\right).$$

proof (checked in Agda): this is a simple rewriting of the actual definition of simulation:

$(\forall s_1)(\forall s_2)\quad (s_1, s_2) \in R \Rightarrow (\forall a_1)(\exists a_2)(\forall d_2)(\exists d_1)\ \big(s_1[a_1/d_1], s_2[a_2/d_2]\big) \in R$
$\quad\Leftrightarrow\quad$ { logic }
$(\forall s_1)(\forall s_2)(\forall a_1)\quad s_2 \in R(s_1) \Rightarrow (\exists a_2)(\forall d_2)(\exists d_1)\ s_2[a_2/d_2] \in R(s_1[a_1/d_1])$
$\quad\Leftrightarrow\quad$ { definition of w_2° }
$(\forall s_1)(\forall a_1)(\forall s_2)\quad s_2 \in R(s_1) \Rightarrow s_2 \in w_2^\circ\big(\bigcup_{d_1} R(s_1[a_1/d_1])\big)$
$\quad\Leftrightarrow$
$(\forall s_1)(\forall a_1)\quad R(s_1) \subseteq w_2^\circ\big(\bigcup_{d_1} R(s_1[a_1/d_1])\big).$

☑

If we apply this lemma to a simulation from w_1 to w_2^* (*i.e.* a refinement), we obtain, using the notation from page 65: R is a refinement iff

$$R(s_1) \quad \lhd_{w_2} \quad \bigcup_{d_1 \in D_1(s_1, a_1)} R\big(s_1[a_1/d_1]\big)$$

for all $s_1 \in S_1$ and $a_1 \in A_1(s_1)$. We will now show that:

- **Lemma 3.3.5:** suppose that R is a simulation from w_1 to w_2^*, then

$$R(s_1) \quad \lhd_{w_2} \quad \bigcup_{d_1' \in D_1^*(s_1,a_1)} R\big(s_1[a_1'/d_1']\big)$$

for all $s_1 \in S_1$ and $a_1' \in A^*(s_1)$. This implies that if R is a simulation from w_1 to w_2^*, then it is also a simulation from w_1^* to w_2^* (by lemma 3.3.4).

proof (checked in Agda): let R be a simulation from w_1 to w_2^*, and suppose that $s_1 \in S_1$ and $a_1' \in A^*(s_1)$.

Let $s_2 \varepsilon R(s_1)$, we need to show that $s_2 \lhd \bigcup\{R\big(s_1[a_1'/d_1']\big) \mid d_1' \in D_1^*(s_1, a_1')\}$. We proceed by induction on $a_1' \in A^*(s_1)$:

- if $a_1' = \mathrm{Exit}$, the result is trivial: the RHS simplifies into $\bigcup\{R(s_1) \mid d \in \{*\}\}$ which is equal to $R(s_1)$. We have that $s_2 \varepsilon R(s_1)$ by hypothesis.
- if $a_1' = \mathrm{Call}(a_1, k)$, by lemma 3.3.4, we have

$$s_2 \quad \lhd_{w_2} \quad \bigcup_{d_1 \in D_1(s_1,a_1)} R\big(s_1[a_1/d_1]\big)\ . \tag{3-1}$$

For any $d_1 \in D(s_1, a_1)$, we can use the induction hypothesis on $s_1' \triangleq s_1[a_1/d_1]$ and $k(d_1)$ to get

$$R(s_1') \quad \lhd_{w_2} \quad \bigcup_{d_1'' \in D^*(s_1', k(d_1))} R\big(s_1'[k(d_1)/d_1'']\big)\ .$$

The RHS is included in $\bigcup_{d_1'} R\big(s_1[a_1'/d_1']\big)$, so that we get, by monotonicity:

$$R\big(s_1[a_1/d_1]\big) \quad \lhd_{w_2} \quad \bigcup_{d_1' \in D_1^*(s_1,a_1')} R\big(s_1[a_1'/d_1']\big)\ .$$

Since the above is true for any $d_1 \in D_1(s_1, a_1)$, we can conclude that

$$\bigcup_{d_1 \in D_1(s_1,a_1)} R\big(s_1[a_1/d_1]\big) \quad \lhd_{w_2} \quad \bigcup_{d_1' \in D_1^*(s_1,a_1')} R\big(s_1[a_1'/d_1']\big)\ .$$

By transitivity with (3-1), this finishes the proof that $s_2 \lhd \bigcup_{d_1'} R\big(s_1[a_1'/d_1']\big)$.

☑

Putting everything back together, we have shown that R is a simulation from w_1 to w_2^* iff it is a simulation from w_1^* to w_2^*. This concludes the proof proposition 3.3.1.

- **Lemma 3.3.6:** the operation "$\_^*$" is functorial, with the following action on simulations:

$$R \quad \mapsto \quad R^* \triangleq R\ .$$

proof: if one accepts the identity relation, this makes **Rel** into a category rather than a precategory, and a premonad M is functorial by putting $M(f) \triangleq \big((\mathbf{Id}_{M(B)})_\natural \cdot f\big)^\natural$ for any morphism f from A to B.

It is also possible to make a direct proof of this fact by proving that if R is a simulation from w_1 to w_2, then R is also a simulation from w_1^* to w_2^*.

☑

3.3.2 Refinements

Since $\_^*$ is a monad, we can construct its Kleisli category **Ref**: objects are objects in **Int**, *i.e.* interfaces; and a morphism from w_1 to w_2 in **Ref** is given by a morphism from w_1 to w_2^* in **Int**, or as we called them in section 2.6, a refinement from w_1 to w_2. That we have a monad guarantees that composition is well defined: in general, to compose f and g in a Kleisli category, take $f^\sharp \cdot g$ (or $\mu \cdot Mf \cdot g$ depending on the presentation of the monad). In this case, this is very simple since $R^\sharp$ is R. We thus compose relations as usual...
Note that the Kleisli construction works as well for precategories.

3.3.3 Demonic Iteration: a Comonad

Since Demonic iteration is dual to Angelic iteration, it is not surprising to have the dual statement to proposition 3.3.1 and lemma 3.3.6:

> ◇ **Proposition 3.3.7:** *in the category* **Int**, *"$\_^\infty$" is functorial; it is a comonad.*

proof: we start by checking that $\_^\infty$ is functorial: let R be a simulation from w_1 to w_2.[3] We will show that R is also a simulation from w_1^∞ to w_2^∞.
Suppose $(s_1, s_2) \varepsilon R$, let $a_1' \epsilon A^\infty(s_1)$. We will first define an action in $A_2^\infty(s_2)$ simulating a_1'. To this aim, define the following coalgebra for w_2:

- for any $s_2 \epsilon S_2$, put $X(s_2) \triangleq (\Sigma s_1 \epsilon S_1)\big(\Sigma r \epsilon R(s_1, s_2)\big)\ A_1^\infty(s_1)$;
- and $C \epsilon (s_2 \epsilon S_2) \to X(s_2) \to w_2(X, s_2)$: if $s_2 \varepsilon X$, we have $(s_1, s_2) \varepsilon R$ for some $s_1 \epsilon S_1$ and that we moreover have an action $a_1' \epsilon A_1^\infty(s_1)$. If $\mathrm{elim}(a_1')$ is of the form (a_1, k_1), we can find some $a_2 \epsilon A_2(s_2)$ simulating a_1 by R, *i.e.*

$$\big(\forall d_2 \epsilon D_2(s_2, a_2)\big)\big(\exists d_1 \epsilon D_1(s_1, a_1)\big)\ \big(s_1[a_1/d_1], s_2[a_2/d_2]\big) \varepsilon R\ .$$

  This implies that for all $d_2 \epsilon D_2(s_2, a_2)$, we can find a $d_1 \epsilon D_1(s_1, a_1)$. Thus, $k_1(d_1) \epsilon A_1^\infty(s_1[a_1/d_1])$ and this implies that for all d_2, we have that $s_2[a_2/d_2] \varepsilon X$. This allows to define C: put

$$C\big(s_2, (s_1, r, a_1')\big) \quad \triangleq \quad \Big(a_2, (\lambda d_2)\,.\,\big(s_1', r', k(d_1)\big)\Big)$$

  where a_2 is the action simulating a_1 and for any reaction $d_2 \epsilon D_2(s_2, a_2)$, if d_1 is the reaction corresponding to d_2 by the simulation, s_1' is the new state $s_1[a_1/d_1]$, and r' the proof that the new states $s_2[a_2/d_2]$ and s_1' are related.

In this coalgebra, we have: *(we abbreviate* coiter(X, C) *by* coiter*)*

$$\mathrm{elim}\Big(\mathrm{coiter}\big(s_2, (s_1, r, a_1')\big)\Big) = \Big(a_2, (\lambda d_2).\mathrm{coiter}\big(s_2[a_2/d_2], (s_1', r', k(d_1))\big)\Big)$$

where, by construction, a_2 simulates a_1.

[3]: This proof is meant to be implemented in a proof system, not so much for reading!

For any s_1, s_2, r and a'_1 as above, we define $a'_2 \triangleq \mathrm{coiter}\big(X, C, s_2, (s_1, r, a'_1)\big)$. We need to show that that this construction does simulate $a'_1 \in A_1^\infty(s_1)$:

$$\begin{array}{l}(\forall s_1, s_2)\big(\forall r \in R(s_1, s_2)\big)\\ \big(\forall a'_1 \in A_1^\infty(s_1)\big)\\ \Big(\forall d'_2 \in D_2^\infty\big(s_2, \mathrm{coiter}\big(s_2, (s_1, r, a'_1)\big)\big)\Big)\big(\exists d'_1 \in D_1^\infty(s_1, a'_1)\big)\\ \qquad \Big(s_1[a'_1/d'_1], s_2\big[\mathrm{coiter}\big(s_2, (s_1, r, a'_1)\big)/d'_2\big]\Big) \in R\ .\end{array} \tag{3-2}$$

We proceed by induction on d'_2:

- if $d'_2 = \mathrm{Nil}$, then the result is obvious: take $d'_1 \triangleq \mathrm{Nil}$. We have that $s_1[a'_1/\mathrm{Nil}] = s_1$ and $s_2[\mathrm{coiter}(\ldots)/\mathrm{Nil}] = s_2$ and the result holds by hypothesis.
- if d'_2 is of the form $\mathrm{Cons}(d_2, d''_2)$: we have by definition that $d_2 \in A_2(s_2, a_2)$ and that $d''_2 \in A_2^\infty\big(s_2[a_2/d_2], k(d_2)\big)$ where $k(d_2) = \mathrm{coiter}\big(s_2[a_2/d_2], (s'_1, r', k_1(d_1))\big)$ as above: a_2 simulates a_1, and if d_1 is the image of d_2 by the simulation, s'_1 is in fact $s_1[a_1/d_1]$.
  By induction hypothesis applied to:
  - $s'_1 \triangleq s_1[a_1/d_1]$,
  - $s'_2 \triangleq s_2[a_2/d_2]$,
  - the proof r' that $(s'_1, s'_2) \in R$,
  - $k_1(d_1)$,
  - and d''_2 which is indeed an element of $D_2^\infty\big(s'_2, \mathrm{coiter}\big(s'_2, (s'_1, r', k_1(d_1))\big)\big)$;

  we obtain a reaction d''_1 to $k_1(d_1)$ which satisfies

  $$\big(s'_1[k(d_1)/d''_1], s'_2[\mathrm{coiter}(\ldots)/d''_2]\big) \in R\ .$$

  It is straightforward to see that taking $d'_1 \triangleq \mathrm{Cons}(d_1, d''_1)$ makes (3-2) true.

  This completes the proof that R is a simulation from w_1^∞ to w_2^∞ and thus that $\_^\infty$ is functorial.

We now need to show that this endofunctor is indeed a comonad. We do not repeat what was done in the previous section for monads and the problem of identities; we only note that it will be enough for us to show that a relation is a simulation from w_1^∞ to w_2 iff it is a simulation from w_1^∞ to w_2^∞.

- let R be a simulation from w_1^∞ to w_2^∞; let's show that R is also a simulation from w_1^∞ to w_2: if $(s_1, s_2) \in R$ and $a'_1 \in A^\infty(s_1)$, we need to find an $a_2 \in A_2(s_2)$ simulating a'_1.
  By hypothesis, we can find an action $a'_2 \in A_2^\infty(s_2)$ simulating a'_1. If $\mathrm{elim}(a'_2)$ is of the form (a_2, k_2), take the action a_2 to simulate a'_1: we only need to show that

  $$\big(\forall d_2 \in D_2(s_2, a_2)\big)\big(\exists d'_1 \in A_1^\infty(s_1, a'_1)\big)\big(s_1[a'_1/d'_1], s_2[a_2/d_2]\big) \in R\ .$$

  For $d_2 \in A_2(s_2, a_2)$, the reaction $\mathrm{Cons}(d_2, \mathrm{Nil})$ is an element of $A_2^\infty(s_2, a'_2)$ and it thus has a corresponding reaction $d'_1 \in A_1^\infty(s_1, a'_1)$. This particular d'_1 does work because $s_2[a'_2/\mathrm{Cons}(d_2, \mathrm{Nil})] = s_2[a_2/d_2]$.

- for the other direction, suppose R is a simulation from w_1^∞ to w_2; we need to show that is it also a simulation from w_1^∞ to w_2^∞. The construction is *very similar* to the proof of functoriality of $\_^\infty$. We only sketch it: given $(s_1, s_2) \in R$

and an action $a'_1 \in A_1^\infty(s_1)$, we construct the coalgebra (X, C) as above. The only difference is that in the definition of C, we do not simulate the first action of a'_1 in order to get an action in $A_2(s_2)$, but use the simulation to simulate the whole a'_1. The rest can be copied almost word for word.

☑

3.4 A Right-Adjoint for the Tensor

We already saw in corollary 3.2.2 that **Int** is not cartesian closed: we cannot hope to have an exponential object $w_2{}^{w_1}$ in the usual categorical sense. The category **Int** enjoys however a weaker property which still allows to speak about "the object of simulations from w_1 to w_2": it is symmetric monoidal closed.

▷ **Definition 3.4.1:** if w_1 and w_2 are interfaces, define $w_1 \multimap w_2$ to be the interaction system on $S_1 \times S_2$ with components $(A_{\multimap}, D_{\multimap}, n_{\multimap})$:

$$\begin{array}{lcl} A_{\multimap}\big((s_1,s_2)\big) & \triangleq & \big(\Sigma\ f \in A_1(s_1) \to A_2(s_2)\big) \\ & & \big(\Pi\ a_1 \in A_1(s_1)\big) \\ & & \quad D_2\big(s_2, f(a_1)\big) \to D_1(s_1,a_1) \\ D_{\multimap}\big((s_1,s_2),(f,G)\big) & \triangleq & \big(\Sigma\ a_1 \in A_1(s_1)\big)\ D_2\big(s_2,f(a_1)\big) \\ n_{\multimap}\big((s_1,s_2),(f,G),(a_1,d_2)\big) & \triangleq & \Big(s_1\big[a_1/G_{a_1}(d_2)\big]\ ,\ s_2\big[f(a_1)/d_2\big]\Big)\ . \end{array}$$

The definition of $\_ \multimap \_$ may look very complex, but is *a posteriori* rather natural:

- An action in state (s_1, s_2) is given by:
  1) a function f translating actions from s_1 into actions from s_2;
  2) for any a_1, a function G_{a_1} translating reactions to $f(a_1)$ into reactions to a_1.
- A reaction to such a "translating mechanism" is given by:
  1) an action a_1 in $A_1(s_1)$ (which we want to simulate);
  2) 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 $a_2 \in A_2(s_2)$ obtained by applying f to a_1, and translate back d_2 into $d_1 \in D_1(s_1, a_1)$ by applying G_{a_1} to d_2. The next state is just the pair of states $s_1[a_1/d_1]$ and $s_2[a_2/d_2]$.

That this operation is indeed an object of simulations is justified by the following:

◇ **Proposition 3.4.2:** *for any interface w, "$w \multimap \_$" is right adjoint to "$\_ \otimes w$". In other words, there is a natural isomorphism*

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

proof: to emphasize the part of the formula being manipulated, we underline it. That R is a simulation from $w_1 \otimes w_2$ to w_3 takes the form[4]

$$(s_1,s_2,s_3)\ \varepsilon\ R \quad \Rightarrow \quad \big(\forall a_1 \in A_1(s_1)\big)\underline{\big(\forall a_2 \in A_2(s_2)\big)} \\ \underline{\big(\exists a_3 \in A_3(s_3)\big)}$$

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

$$
\begin{array}{l}
(\forall d_3 \in D_3(s_3, \underline{a_3})) \\
(\exists d_1 \in D_1(s_1, a_1))(\exists d_2 \in D_2(s_2, a_2)) \\
\quad (s_1[a_1/d_1], s_2[a_2/d_2], s_3[\underline{a_3}/d_3]) \varepsilon R\ .
\end{array}
$$

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

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

By first swapping the last two existential quantifiers, we can apply **AC** on $\forall d_3 \exists d_2$:

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

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

$$
\begin{array}{lll}
(s_1, s_2, s_3) \varepsilon R & \Rightarrow & (\forall a_1 \in A_1(s_1)) \\
& & (\exists f \in A_2(s_2) \to A_3(s_3)) \\
& & (\exists G \in (a_2 \epsilon A_2(s_2)) \to D_3(s_3, f(a_2)) \to D_2(s_2, a_2)) \\
& & (\forall a_2 \in A_2(s_2)) \\
& & (\forall d_3 \in D_3(s_3, f(a_2))) \\
& & (\exists d_1 \in D_1(s_1, d_1)) \\
& & \quad (s_1[a_1/d_1], s_2[a_2/G_{a_2}(d_3)], s_3[f(a_2)/d_3]) \varepsilon R
\end{array}
$$

which is equivalent to

$$
\begin{array}{lll}
(s_1, s_2, s_3) \varepsilon R & \Rightarrow & (\forall a_1 \in A_1(s_1)) \\
& & \left(\exists (f, G) \in \begin{array}{l} (\Sigma f \epsilon A_2(s_2) \to A_3(s_3)) \\ (\Pi a_2 \epsilon A_2(s_2))\, D_3(s_3, f(a_2)) \to D_2(s_2, a_2) \end{array}\right) \\
& & (\forall (a_2, d_3) \in (\Sigma a_2 \epsilon A_2(s_2))\ D_3(s_3, f(a_2))) \\
& & (\exists d_1 \in D_1(s_1, d_1)) \\
& & \quad (s_1[a_1/d_1], s_2[a_2/G_{a_2}(d_3)], s_3[f(a_2)/d_3]) \varepsilon R\ .
\end{array}
$$

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

Naturality is trivial as the isomorphism is given by $S_1 \times (S_2 \times S_3) \simeq (S_1 \times S_2) \times S_3$ (set isomorphism).

☑

In particular, since **skip** is neutral for the tensor, we have

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

which allows us to see simulations from w_1 to w_2 as an invariant predicate for the Angel in the interaction system $w_1 \multimap w_2$, see lemma 3.2.4.

3.5 A Dualizing Object

Some SMCC are equipped with an internal duality, which makes them particularly well-behaved: $\star$-autonomous categories ([13]). In addition to the closed structure, they require the existence of a *dualizing object:*

▷ **Definition 3.5.1:** a *dualizing object* in a symmetric monoidal closed category $\mathcal{C}$ is an object $\bot$ such that, for every object A, the canonical morphism from A to $(A \multimap \bot) \multimap \bot$ is an isomorphism.

The *canonical* morphism comes from the equivalence

$$\begin{array}{lcl} \mathcal{C}\big(A, (A \multimap \bot) \multimap \bot\big) & \simeq & \mathcal{C}\big(A \otimes (A \multimap \bot), \bot\big) \\ & \simeq & \mathcal{C}\big((A \multimap \bot) \otimes A, \bot\big) \\ & \simeq & \mathcal{C}\big(A \multimap \bot, A \multimap \bot\big) \quad . \end{array}$$

Specialized to our case, we have that

$$\begin{array}{lcl} R & : & \mathbf{Int}\big(w_1, (w_1 \multimap w_2) \multimap w_2\big) \\ R & \triangleq & \big\{(s_1, (s_1', s_2, s_2')) \mid s_1 =_{S_1} s_1' \ \wedge \ s_2 =_{S_2} s_2'\big\} \end{array}$$

is the canonical morphism from any w_1 to $(w_1 \multimap w_2) \multimap w_2$. *(This uses equality.)*

The notion of dualizing object has a very classical feeling (double negation); it is thus not very surprising (??) that the following holds only classically:

◇ **Proposition 3.5.2: (classically)** *in* **Int**, *the object* **skip** *is dualizing.*

proof: the canonical morphism from w to $(w \multimap \mathbf{skip}) \multimap \mathbf{skip}$ takes the form

$$ND \quad \triangleq \quad \big\{(s, ((s', *), *)) \mid s =_S s'\big\} \quad .$$

This relation is invertible in **Rel**, with inverse: *(where* DN *stands for "Double Negation")*

$$DN \quad \triangleq \quad \big\{(((s, *), *), s') \mid s =_S s'\big\} \quad .$$

Thus, to show that **skip** is dualizing, we only need to show that DN is a simulation from any $(w \multimap \mathbf{skip}) \multimap \mathbf{skip}$ to w. It is just an application of the definition from page 80 that $w^\perp$ is structurally isomorphic to $w \multimap \mathbf{skip}$. Thus, we need to show that $\mathbf{Eq}_S \simeq DN$ is a simulation from $w^{\perp\perp}$ to w.
Here are the components of $w^{\perp\perp}$:

$$\begin{array}{lcl} A^{\perp\perp}(s) & = & \Big(\big(a \epsilon A(s)\big) \to D(s, a)\Big) \to A(s) \\ D^{\perp\perp}(s, F) & = & \Big(\big(a \epsilon A(s)\big) \to D(s, a)\Big) \end{array}$$

$$n^{\perp\perp}(s,F,g) \quad = \quad s\big[F(g)/g\big(F(g)\big)\big] \ .$$

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

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

By applying the contraposition of the axiom of choice (page 31) on $\exists a \forall d$, this is equivalent to

$$\begin{array}{ll}(\forall s \in S) & \big(\forall F \in A^{\perp\perp}(s)\big)\big(\forall f \in \big(a \epsilon A(s)\big) \to D(s,a)\big) \\ & \big(\exists a \in A(s)\big)\big(\exists g \in D^{\perp\perp}(s,F)\big) \\ & \quad s[a/d] =_S s\big[F(g)/g\big(F(g)\big)\big] \ .\end{array}$$

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

$$\begin{array}{ll}(\forall s \in S) & \big(\forall f \in A^{\perp}(s)\big)\underline{\big(\forall F \in A^{\perp}(s) \to D^{\perp}(s,\_)\big)} \\ & \underline{\big(\exists g \in D^{\perp\perp}(s,F)\big)}\big(\exists a \in D^{\perp}(s,f)\big) \\ & \quad s[a/f(a)] =_S s\big[\underline{F(g)}/g\big(\underline{F(g)}\big)\big] \ .\end{array}$$

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

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

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

Thus, we can conclude that **Eq** is a simulation from $w^{\perp\perp}$ to w and thus that DN is a simulation from $(w \multimap \mathbf{skip}) \multimap \mathbf{skip}$ to w. This concludes the proof that **skip** is a dualizing object in **Int**.

☑

This proposition has a very disturbing corollary: call an interaction system *simple* when the sets of reactions $w.D(s,a)$ do *not* depend on $a \in w.A(s)$;

- **Corollary 3.5.3: (classically)** any interaction system is isomorphic to a simple interaction system.

proof: just take $w' \triangleq w^{\perp\perp}$.

☑

We will latter see more properties of this duality in a classical setting. For now, we just mention that:

- **Lemma 3.5.4:** for all interfaces w_1 and w_2, we have
  1) $(\_ \boxplus \_)^{\perp} \approx \_^{\perp} \boxtimes \_^{\perp}$ (structural isomorphism);
  2) **classically:** $(\_ \boxtimes \_)^{\perp} \simeq \_^{\perp} \boxplus \_^{\perp}$ (isomorphism).

4 Interaction Systems and Topology

4.1 Constructive Sup-Lattices

In [6], Peter Aczel gives a description of constructive sup-lattices in CZF. We review those notions and show how interaction systems can be seen as a type theoretic reformulation of the notion of "set-presented sup-lattice".

Note: in this chapter, we assume equality in the underlying type theory.

4.1.1 Classical Notions

Recall that classically:

▷ **Definition 4.1.1:** a partial order $(S, \leqslant)$ with binary suprema $s \vee s'$ and a least element is called a *sup-lattice*. It is *complete* if it has arbitrary suprema $\bigvee U$ for all $U : \mathcal{P}(S)$.
A partial order $(S, \leqslant)$ is a lattice if it is both a sup-lattice and an inf-lattice.

And a simple lemma:

○ **Lemma 4.1.2:** let S be a set and $\leqslant$ a partial order on S, the following are equivalent:
1) $(S, \leqslant)$ is a complete sup-lattice;
2) $(S, \leqslant)$ is a complete lattice.

proof: define $\bigwedge U \triangleq \bigvee\{s \in S \mid s \text{ is a lower bound of } U\}$. It is easy to check that this is the infimum operation. The rest is trivial. ☑

4.1.2 Constructive Sup-Lattices

Predicatively, the above definition is not adequate: we would like to consider partial orders defined on a proper type $\mathcal{S}$. In such a case, it is not possible to state that an element A is the lowest upper bound of a predicate $\mathcal{U}$: the expression

$$(\forall B{:}\mathcal{S}) \qquad A \leqslant B \quad \Leftrightarrow \quad \Big((\forall C{:}\mathcal{S})\ C \varepsilon \mathcal{U} \Rightarrow C \leqslant B\Big)$$

is not a set because the second quantification over the $\mathcal{S}$ brings us beyond $\mathbf{\Pi}^1_1$ quantification. It is however easy to say that $A : \mathcal{S}$ is the lub of the family $(B_i)_{i\epsilon I}$ (where I : **Set**):

$$(\forall C{:}\mathcal{S}) \quad A \leqslant C \quad \Leftrightarrow \quad \big((\forall i\epsilon I)\ B_i \leqslant C\big)$$

which is an instance of $\mathbf{\Pi}^1_1$ quantification. We thus put:

▷ **Definition 4.1.3:** a partial order $(\mathcal{S}, \leqslant)$ is a complete sup-lattice if for any (set-indexed) family $(A_i)_{i\epsilon I}$, there is an element $\bigvee_{i\epsilon I} A_i$ such that:

$$B : \mathcal{S} \qquad \vdash \qquad \bigvee_{i\epsilon I} A_i \leqslant B \ \Leftrightarrow\ \big((\forall i\epsilon I)\ A_i \leqslant B\big)\ .$$

Lemma 4.1.2 doesn't hold anymore because the predicate $\{B \mid B \text{ is a lub of } \mathcal{U}\}$ is usually not set-based.

§ *Set-Generated Sup-Lattices.* In order to get a predicatively friendlier theory, it is traditional to restrict one's attention to sup-lattice having a set-indexed "basis":

▷ **Definition 4.1.4:** (adapted from [6])
a partial order $(\mathcal{S}, \leqslant)$ with set-indexed lubs is *set-generated* if there is a *set-indexed* predicate $G \subseteq \mathcal{S}$ s.t.
- for all $A : \mathcal{S}$ the predicate $A^{\downarrow G} \triangleq \{g\ \varepsilon\ G \mid g \leqslant A\}$ is set-indexed;
- for any $A : \mathcal{S}$, we have $\bigvee A^{\downarrow G} = A$.

We call $\{G_i \mid i\epsilon I\}$ a *generating family*.

The interest of this notion is that any set-generated sup-lattice is isomorphic to the collection of (pre) fixpoints for a closure operator on $\mathcal{P}(G)$. The idea is to define $F(U) = \{g\varepsilon G \mid g \leqslant \bigvee U\}$. (This is possible because as a predicate over G, U is set-indexed.) The details can be found in [6], theorem 6.3.

Type theoretically, any interaction system gives rise to a set-generated sup-lattice in the following way: if w is an interaction system on S, define the proper type

$$\mathcal{O}_w \quad \triangleq \quad \{U : \mathcal{P}(S) \mid w^*(U) \subseteq U\}$$

and take inclusion as a partial order.

§ *Set-Presented Sup-Lattices.* We saw, on page 62, an example of predicate transformer which couldn't be represented by an interaction system. It is easy to check that this predicate transformer $U \mapsto U^{\diamond\diamond}$ is a closure operator, so that it is equal to its reflexive transitive closure. This shows that not every set-generated sup-lattice arises as some $\mathcal{O}_w$. The next notion gives what is missing:

▷ **Definition 4.1.5:** a *presentation* for a sup-lattice $(\mathcal{S}, \leqslant)$ with generating set G is a *set-indexed* relation $\lhd$ between G and $\mathcal{P}(G)$ s.t.

$$g \leqslant \bigvee U \quad \Leftrightarrow \quad (\exists U' \subseteq U)\ g \lhd U'$$

for any $g \epsilon G$ and $X \subseteq G$.

Notice that the RHS quantification "$\exists Y$" is predicative because we quantify over a set-indexed relation.

In [6], Peter Aczel proves that

◇ **Proposition 4.1.6:** *every set-generated sup-lattice arises from a closure operator on a $\mathcal{P}(S)$ for some set S.*
Moreover, the sup-lattice is set presented iff the corresponding closure operator is set based.

The first point is just the remark from the previous paragraph. The second point shows that a sup-lattice is set-presented iff it is isomorphic to some $\mathcal{O}_w$ for some interaction system w.
Let's check the interesting direction:

○ **Lemma 4.1.7:** for any interaction system w in S, $(\mathcal{O}_w, \subseteq)$ is a set-presented sup-lattice.

proof: for any family $\{U_i \mid i \in I\}$ of fixpoints of w^*, define

$$\bigvee_{i\in I} U_i \quad \triangleq \quad w^*\left(\bigcup_{i\in I} U_i\right) .$$

Let's check that this defines a sup-lattice structure for $\mathcal{O}_w$: suppose $U_i \subseteq V$ for all $i \in I$. We need to check that $\bigvee_i U_i \subseteq V$: we trivially have that $\bigcup_i U_i \subseteq V$, which by monotonicity implies that $w^*(\bigcup_i U_i) \subseteq w^*(V)$. However, since $V : \mathcal{O}_w$, we know that $V = w^*(V)$, so that we obtain $\bigvee_i U_i \subseteq V$.

The sup-lattice $\mathcal{O}_w$ is set-generated by using $\{w^*\{s\} \mid s \in S\}$ as a generating family. It is set-presented by using the relation:

$$\blacktriangleleft_w \quad = \quad \left\{ \left(w^*(s), \bigcup_{d\in D^*(s,a')} w^*(s[a'/d']) \right) \;\middle|\; s \in S, a' \in A^*(s) \right\}$$

which satisfies the condition. For the first direction, we have

$$(\exists U \subseteq V)\ w^*(s) \blacktriangleleft_w U \quad \Rightarrow \quad w^*(s) \subseteq V$$

because $w^*(s) \blacktriangleleft_w U$ implies $s \lhd_w U$, which implies $s \lhd_w V$ and thus $w^*(s) \subseteq V$ (recall that V is open, *i.e.* $V = w^*(V)$). For the second direction,

$$w^*(s) \subseteq V \quad \Rightarrow \quad (\exists U \subseteq V)\ w^*(s) \blacktriangleleft_w U ,$$

we have that $w^*(s) \subseteq V$ implies $s \lhd_w V$, which easily implies that $w^*(s) \blacktriangleleft_w U$ for some $U \subseteq V$.

☑

4.1.3 Morphisms

The notion of morphism between sup-lattices is the traditional one: morphisms are maps commuting with lubs. Rather than looking at this condition in the context of set-generated / set-presented sup-lattices, we postpone the discussion about morphisms to section 4.2.2 where we look at a specific example of sup-lattice: collection of open sets in a "basic topology". Analysis of sup-lattice morphisms between set-generated sup-lattices can be extracted from [36].

4.2 Interaction Systems and Topology

The most popular sup-lattices are probably the sup-lattices arising as collections of open sets of a topological space. Let's start by recalling the main ideas of constructive topology.

4.2.1 Constructive Topology

Abstract topology is a very classical domain, and both the principles of excluded middle and the axiom of choice are used quite heavily. It is however possible to develop non-trivial parts of topology in a constructive framework.

§ *Pointfree Topology.* The first observation is that many topological results can be rephrased to mention open sets rather than actual points. *Pointfree topology* is a systematic analysis of topology along this line. Working directly with open sets allows in particular to remove many occurrences of the axiom of choice, giving a more constructive theory. Examples of topological theorems having received a constructive treatment include Tikhonov, Hahn-Banach and Heine-Borel theorems, Stone's representation theorem or other representation theorems.

Pointfree topology can be seen as the study of the dual of the category of *frames* (complete Heyting algebras). Taking the dual is just a technical artifact to get morphisms in the "topological" direction: a continuous function from X_1 to X_2 is a lub-preserving function from $\mathcal{O}(X_2)$ to $\mathcal{O}(X_1)$ where $\mathcal{O}(X)$ is the lattice of open sets of the topology X. This category is called the category of *locales*. Further motivation for "pointless" topology can be found in [56].

§ *Formal Topology.* However, the theory of locales is still impredicative. *Formal topology* is the study of locales in a predicative setting. To achieve this goal, one considers a topology as given by a base of open sets S. Any element of S is a *basic open*. Together with this base is given a *covering* predicate: $s \lhd U$ (for some $s \in S$ and $U : \mathcal{P}(S)$). It's intuitive meaning is that "the basic open s is covered by the basic opens in U". As we saw in the previous section, formal topology is thus the study of frames arising from set-generated sup-lattices.
In order for this relation to generate a distributive sup-lattice, it should satisfies (among others):

$$\frac{s \lhd U \qquad s \lhd V}{s \lhd U \downarrow V}\ \text{convergence}$$

where $U \downarrow V \triangleq \{s \in S \mid (\exists s' \varepsilon U)\, s \lhd \{s'\} \ \wedge\ (\exists s' \varepsilon V)\, s \lhd \{s'\}\}$.

The convergence condition expresses that for any pair of coverings of S, it is possible to find a covering refining both of them. Classically, we just take the collection of binary intersections between the two coverings.

The last component of a formal topology is the *positivity predicate* **Pos**. The intuitive meaning of $\mathbf{Pos}(s)$ is "the basic open s is not empty". It should in particular satisfy:

$$\frac{s \lhd U}{s \lhd U^{+}}\ \text{positivity}$$

where $\mathcal{U}^+ \triangleq \{s \varepsilon \mathcal{U} \mid \mathbf{Pos}(s)\}$. This asserts that only positive basic opens are important. An introduction to formal topology can be found in [76].

An interesting class of formal topologies is the class of "inductively generated" formal topologies. For those, the rules of convergence and positivity can be understood as "generating" rules rather than "admissible rules". Those are studied in details in [27], and, as can be easily inferred from the previous section, coincide with frames constructed from set-presented sup-lattices.

§ *Basic Topology.* In [79] and [36], Giovanni Sambin introduces a new structure for topology. The differences with the traditional approach are:

- the unary predicate **Pos** is replaced by a binary predicate: $\ltimes$, dual to $\lhd$;
- the notion of convergence is dropped;
- the positivity axiom is dropped.

The idea is to obtain a concise, completely symmetric core which can be extended at will in order to approximate the classical theory.
Formally:

▷ **Definition 4.2.1:** let S be a set, a *basic topology on* S is a pair of operators $\mathcal{A}$ and $\mathcal{J}$ on $\mathcal{P}(S)$ such that:

- $\mathcal{A}$ is a closure operator;
- $\mathcal{J}$ is an interior operator;
- $\mathcal{A}$ and $\mathcal{J}$ are related via the *compatibility* condition:

$$\frac{\mathcal{A}(\mathcal{U}) \between \mathcal{J}(V)}{\mathcal{U} \between \mathcal{J}(V)} \text{ compatibility .}$$

The notation $s \lhd \mathcal{U}$ is synonym to $s \varepsilon \mathcal{A}(\mathcal{U})$ and $s \ltimes V$ is synonym to $s \varepsilon \mathcal{J}(V)$.

It can be enlightening to look at the definition of $\mathcal{A}$ and $\mathcal{J}$ in the case of a traditional (classical) topological space: if S is a base for a topological space, and if $\mathcal{U}$ is a collection of basic opens, we have:

- $s \varepsilon \mathcal{A}(\mathcal{U})$ iff $s \subseteq \bigcup \mathcal{U}$, or s is covered by $\mathcal{U}$;
- $s \varepsilon \mathcal{J}(\mathcal{U})$ iff for some $x \epsilon s$, all basic neighborhoods of x are members of $\mathcal{U}$.

From such a basic topology, we can define a lattice of open sets (predicates $\mathcal{U}$ s.t. $\mathcal{A}(\mathcal{U}) \subseteq \mathcal{U}$) and a lattice of closed sets (subsets V s.t. $V \subseteq \mathcal{J}(V)$).[1] Those lattices are set-generated, but generally speaking not set-presented.

§ *Basic Continuity.* The main topological notion is probably the notion of continuous function. How can we express the fact that a "function" from $(S, \mathcal{A}, \mathcal{J})$ to $(S', \mathcal{A}', \mathcal{J}')$ is continuous? Classically, a continuous function is a function whose inverse image (the angelic update of its graph) sends opens to opens. This implies that a continuous function arises as a relation R between (basic) opens: the meaning of $(s, s') \varepsilon R$ is "the open s is included in the inverse image of s'". Since in a basic topology, the notions of closed and open sets are "independent", we also add a dual clause stating that the inverse image of a closed set is a closed set:

[1]: The fact that open subsets are stable w.r.t. a closure operation and closed subsets are stable w.r.t. an interior operator is justified in [78].

▷ **Definition 4.2.2:** ([36]) if $(S_1, \mathcal{A}_1, \mathcal{J}_1)$ and $(S_2, \mathcal{A}_2, \mathcal{J}_2)$ are basic topologies, a relation $R \subseteq S_1 \times S_2$ is *continuous* if the following two conditions hold:
1) $\langle R \rangle \cdot \mathcal{A}_2 \subseteq \mathcal{A}_1 \cdot \langle R \rangle$;
2) $\langle R^\sim \rangle \cdot \mathcal{J}_1 \subseteq \mathcal{J}_2 \cdot \langle R^\sim \rangle$.

It should be noted that in general, the two conditions are independent.

The shape of condition *2* may look strange, but the reason is that the definition makes $\langle R^\sim \rangle$ send open sets to open sets and $[R^\sim]$ send closed set to closed sets.[2] In the traditional case where R comes from a real function, this is irrelevant as both the Angelic and Demonic updates of a functional relation are equal.

One of the problems with this definition is that continuous relations are relations on *bases* for topological spaces. It is possible for two (extensionally) different relations to represent the same continuous function. In order to deal with this, we need the following notion of equality:

▷ **Definition 4.2.3:** if R and T are two continuous relations from $(S_1, \mathcal{A}_1, \mathcal{J}_1)$ to $(S_2, \mathcal{A}_2, \mathcal{J}_2)$, they are *topologically equal,* if $\mathcal{A}_1 \cdot \langle R \rangle(s_2) = \mathcal{A}_1 \cdot \langle T \rangle(s_2)$ for all $s_2 \in S_2$. We write $R \approx T$.
This forms the category of basic topologies with continuous relations between them, which we call **BTop**.

We refer to [36] for the proof that this forms a category.

4.2.2 Topology and Interaction

§ *Execution and Compatibility.* We know from lemma 2.5.14 that $w^{*\circ}$ and $w^{\perp\infty\circ}$ are respectively a closure and an interior operator on $\mathcal{P}(S)$. We also saw in section 2.6.4 that the relations $\lhd_w$ and $\ltimes_w$ are linked by the "execution formula":

$$\frac{\textsc{Init} \lhd_w \textsc{Goal} \qquad \textsc{Init} \ltimes_w V}{\textsc{Goal} \ltimes_w V}\ \text{execution}\ .$$

This expresses the soundness of interaction between a *client program* and a *server program*. Specialized when INIT is a singleton,[3] we obtain the exact form of Sambin's compatibility rule:

$$\frac{s \lhd_w \textsc{Goal} \qquad s \ltimes_w V}{\textsc{Goal} \ltimes_w V} \quad i.e. \quad \frac{w^{*\circ}(\textsc{Goal}) \between w^{\perp\infty\circ}(V)}{\textsc{Goal} \between w^{\perp\infty\circ}(V)}\ .$$

It is thus natural to put:

▷ **Definition 4.2.4:** if w is an interaction system on S, define:
- $\mathcal{A}_w : \mathcal{P}(S) \to \mathcal{P}(S)$ with $\mathcal{A}_w(U) \triangleq w^{*\circ}(U)$;
- $\mathcal{J}_w : \mathcal{P}(S) \to \mathcal{P}(S)$ with $\mathcal{J}_w(U) \triangleq w^{\perp\infty\circ}(U) = w^{\bullet\infty}(U)$.

The previous remarks show that:

○ **Lemma 4.2.5:** if w is an interaction system on S, then $(S, \mathcal{A}_w, \mathcal{J}_w)$ is a basic topology. Moreover, this basic topology is "set-presented", or "(co)inductively generated".

---

[2]: condition *2* is easily seen to be equivalent to $\mathcal{J}_1 \cdot [R^\sim] \subseteq [R^\sim] \cdot \mathcal{J}_2$ using lemma 2.5.11.

[3]: Note however that equality is not needed to define execution when INIT is a singleton.

§ *Continuity and Interaction.* Since interaction systems are nothing but representations for (co)inductively generated basic topologies, we may hope that the notion of continuous relation corresponds to a notion of simulation. This is indeed the case: continuous relations are exactly refinements modulo saturation (sections 2.6.2 and 2.6.5).

◦ **Lemma 4.2.6:** if w_h and w_l are interaction systems and R a refinement from w_h to w_l (*i.e.* R is a simulation from w_h to w_l^*), then we have:

$$\langle R\rangle \cdot \mathcal{J}_l \quad \subseteq \quad \mathcal{J}_h \cdot \langle R\rangle \ .$$

proof: suppose $V \subseteq S_l$, we need to show that $\langle R\rangle \cdot \mathcal{J}_l(V) \subseteq \mathcal{J}_h \cdot \langle R\rangle(V)$. Since $\mathcal{J}_h \cdot \langle R\rangle(V)$ is a greatest fixpoint of the operator $X \mapsto \langle R\rangle(V) \cap w^\bullet(X)$, it is sufficient to show that $\langle R\rangle \cdot \mathcal{J}_l(V)$ is a post-fixpoint for this same operator:

- $\langle R\rangle \cdot \mathcal{J}_l(V) \subseteq \langle R\rangle(V)$ because $\mathcal{J}_l(V) \subseteq V$;
- $\langle R\rangle \cdot \mathcal{J}_l(V) \subseteq w^\bullet\langle R\rangle \cdot \mathcal{J}_l(V)$: suppose $s_h \varepsilon \langle R\rangle \cdot \mathcal{J}_l(V)$, and let $a_h \in A_h(s_h)$. We need to find a $d_h \in D_h(s_h, a_h)$ s.t. $s_h[a_h/d_h] \varepsilon V$. Because $s_h \varepsilon \langle R\rangle \cdot \mathcal{J}_l(V)$, we know that $(s_h, s_l) \varepsilon R$ for some $s_l \ltimes_l V$. By lemma 3.3.4, we know that $s_l \lhd_l \bigcup_{d_h} R(s_h[a_h/d_h])$ and by compatibility (since $s_l \ltimes_l V$), we can find a "final" state $s'_l \varepsilon \bigcup_{d_h} R(s_h[a_h/d_h])$ s.t. $s'_l \ltimes_l V$. This implies in particular that there is a reaction $d_h \in D_h(s_h, a_h)$ s.t. $s'_l \varepsilon R(s_h[a_h/d_h])$. We thus conclude that $s_h[a_h/d_h] \varepsilon \langle R\rangle \cdot \mathcal{J}_l(V)$.

☑

This proof is not strictly speaking predicative, as it uses the definition of $\mathcal{J}_w$ as $w^{\bullet\infty}$. It is possible to prove lemma 4.2.6 directly by introducing an appropriate coalgebra defining an action in $A_h^\infty(s_h)$ and proving that everything works. After proposition 2.5.18 and 3.3.7, the reader probably doesn't want to read such a proof.

We thus obtain:

◇ **Proposition 4.2.7:** *if w_h and w_l are interaction systems, then a relation $R \subseteq S_h \times S_l$ is a refinement from w_h to w_l iff it is a continuous relation from $(S_l, \mathcal{A}_l, \mathcal{J}_l)$ to $(S_h, \mathcal{A}_h, \mathcal{J}_h)$.*
Moreover, topological equality coincide with equality of saturations as defined in section 2.6.5.

proof: we first need to show that if R is a refinement, we have $\langle R^\sim\rangle \cdot \mathcal{A}_h \subseteq \mathcal{A}_l \cdot \langle R^\sim\rangle$ and $\langle R\rangle \cdot \mathcal{J}_l \subseteq \mathcal{J}_h \cdot \langle R\rangle$. The first point is given by lemma 3.3.5 and the second by lemma 4.2.6. For the converse, use lemma 2.5.21 and the definition of refinements.

That topological equality coincide with extensional equality of saturations holds by definition.

☑

We can conclude by:

◇ **Proposition 4.2.8:** *the operation $w \mapsto (S, \mathcal{A}_w, \mathcal{J}_w)$ is a full and faithful functor from $\mathbf{Ref}_{\approx}^{\mathrm{op}}$ to* **BTop**.

4.2.3 More Basic Topologies

§ *Positivity.* The definition of basic topology puts very little constraint on the operators $\mathcal{A}$ and $\mathcal{J}$. The operators $\mathcal{A}_w$ and $\mathcal{J}_w$ have a much stronger relationship: they are dual to each other. **Classically,** by lemmas 2.5.17 and 2.5.18, we have

$$\mathcal{A}_w \quad = \quad \complement \cdot \mathcal{J}_w \cdot \complement \,. \tag{4-1}$$

A direct consequence is that classically, any basic topology generated from an interaction system will satisfy the *positivity axiom*. Recall that the positivity predicate **Pos** found in formal topologies can be defined as $\mathcal{J}(S)$.

- **Lemma 4.2.9: (classically)** if w is an interaction system on S, then, for any $U : \mathcal{P}(S)$, we have $U \lhd_w U \cap \mathcal{J}_w(S)$, *i.e.* $U \lhd U^+$ and the positivity axiom holds.

proof: define $U^+ \triangleq U \cap \mathcal{J}_w(S)$, let $s \in S$, let's show that $s \lhd_w \{s\}^+$:

- if $s \lhd_w \emptyset$, then we have $s \lhd_w \{s\}^+$ by monotonicity;
- if not, then we have $s \in \complement\mathcal{A}_w(\emptyset) = \mathcal{J}_w(S)$ by the remark (4-1). This means that $s \in \{s\}^+$, and so $s \lhd_w \{s\}^+$.

This implies that for any $U : \mathcal{P}(S)$, $U \lhd_w U^+$.

☑

↬ Remark 15: for example, the basic topology on S (containing at least one element) with $\mathcal{A}(U) = U$ and $\mathcal{J}(U) = \emptyset$ cannot be generated from an interaction system. No $(S, \mathcal{A}_w, \mathcal{J}_w)$ can be a counter-example to the positivity axiom.

§ *Extending the Execution Formula.* We will now see that it is possible to use the machinery of interaction systems and refinements in order to generate more basic topologies. The idea is simple: use different interaction systems to generate $\mathcal{A}$ and $\mathcal{J}$.

- **Lemma 4.2.10:** let w_h and w_l be two interaction systems, suppose R is a refinement from w_h to w_l, then we have:
  - $\langle R \rangle \cdot \mathcal{J}_l \cdot [R^\sim]$ is an interior operator on S_h;
  - $\mathcal{A}_h$ is compatible with $\langle R \rangle \cdot \mathcal{J}_l \cdot [R^\sim]$.

  In other words, $\big(S_h, \mathcal{A}_h, \langle R \rangle \cdot \mathcal{J}_l \cdot [R^\sim]\big)$ is a basic topology.

proof: let's first show that $\langle R \rangle \cdot \mathcal{J}_l \cdot [R^\sim]$ is an interior operator:

- it is contractive:

$$\langle R \rangle \cdot \mathcal{J}_l \cdot [R^\sim](U) \quad \subseteq \quad \langle R \rangle \cdot [R^\sim](U) \quad \subseteq \quad U$$

  where the first inclusion follows from $\mathcal{J}_l$ being contractive and the second from the fact that $\langle R \rangle \cdot [R^\sim]$ is an interior operator (lemma 2.5.11).
- moreover, we have:

$$\langle R \rangle \cdot \mathcal{J}_l \cdot [R^\sim] \;\subseteq \langle R \rangle \cdot \mathcal{J}_l \cdot \mathcal{J}_l \cdot [R^\sim] \;\subseteq \langle R \rangle \cdot \mathcal{J}_l \cdot [R^\sim] \cdot \langle R \rangle \cdot \mathcal{J}_l \cdot [R^\sim]$$

  where the first inclusion follows from $\mathcal{J}_l$ being an interior operator and the second from $[R^\sim] \cdot \langle R \rangle$ being a closure operator (lemma 2.5.11).

To show that this operator is compatible with $\mathcal{A}_h$, suppose that $s_h \lhd_h U$ and that $s_h \varepsilon \langle R\rangle \cdot \mathcal{J}_l \cdot [R^\sim](V)$, *i.e.* that $s_l \ltimes_l [R^\sim](V)$ for some s_l s.t. $(s_h, s_l) \varepsilon R$. We need to show that $U \between \langle R\rangle \cdot \mathcal{J}_l \cdot [R^\sim](V)$.
By hypothesis, we have that $s_l \varepsilon R \cdot \mathcal{A}_h(U)$, so that, because R is a simulation from w_h^* to w_l^*, $s_l \varepsilon \mathcal{A}_l \cdot R(U)$ (lemma 2.5.21). We can then use compatibility on $s_l \ltimes_l [R^\sim](V)$ and $s_l \lhd_l \langle R^\sim\rangle(U)$ to obtain a "final" state $s'_l \varepsilon \langle R^\sim\rangle(U)$ and $s'_l \ltimes_l [R^\sim](V)$. We have $(s'_h, s'_l) \varepsilon R$, with $s'_h \varepsilon U$ and $s'_l \ltimes_l [R^\sim](V)$, which implies:

- $s'_h \varepsilon U$;
- $s'_h \varepsilon \langle R\rangle \cdot \mathcal{J}_l \cdot [R^\sim](V)$.

This concludes the proof.

☑

The interactive reading of this lemma is simple: a basic topology is given by a way to specify servers (using $\mathcal{J}$) and a way to specify clients (using $\mathcal{A}$) on the same set of states. Compatibility, or "execution" is just here to ensure that servers and programs have a sound way to communicate (sections 2.6.3 and 2.6.4). Lemma 4.2.10 formalizes the following remark: if R is a refinement from w_h to w_l, then a client for w_h and a server for w_l can communicate "via" R: for related states,

1) a client request in w_h can be translated into a (sequence of) request(s) in w_l;
2) this request in w_l can be answered to by the server, in w_l;
3) this (sequence of) answer(s) can be translated back in w_h.

The Demon can translate the client's requests and the client can translate the Demon's responses: this is all that is necessary to conduct interaction.

Since $\langle R\rangle \cdot \mathcal{J} \cdot [R^\sim]$ is an interior operator whenever $\mathcal{J}$ is, it is natural to ask if we can weaken the condition of R being a refinement. The answer is no, at least classically speaking:

- **Lemma 4.2.11: (classically)** if w_h and w_l are interaction systems, and if R is a relation between S_h and S_l s.t. $\langle R\rangle \cdot \mathcal{J}_l \cdot [R^\sim]$ is compatible with $\mathcal{A}_h$, then R is a refinement from w_h to w_l.

proof: suppose we have compatibility between $\mathcal{A}_h$ and $\langle R\rangle \cdot \mathcal{J}_l \cdot [R^\sim]$:

$$\frac{(s_h, s_l) \varepsilon R \qquad s_h \lhd_h U \qquad s_l \ltimes_l [R^\sim](V)}{(\exists s'_h, s'_l)\ (s'_h, s'_l) \varepsilon R \ \wedge\ s'_h \varepsilon U \ \wedge\ s'_l \ltimes_l [R^\sim](V)}\ .$$

In order to show that R is a refinement from w_h to w_l, it is sufficient to prove that $s_h \lhd_h U \Rightarrow R(s_h) \lhd_l R(U)$ (lemma 3.3.5). Suppose $s_h \lhd_h U$ and $(s_h, s_l) \varepsilon R$; by contradiction, suppose $s_l \lhd_l R(U)$ doesn't hold.

$\neg s_h \lhd_l R(U)$
$\quad \Leftrightarrow$
$s_h \varepsilon \complement \cdot \mathcal{A}_l \cdot R(U)$
$\quad \Leftrightarrow$ { $\complement \cdot \mathcal{A}_l = \mathcal{J}_l \cdot \complement$ }
$s_h \ltimes_l \complement \cdot \langle R^\sim\rangle(U)$
$\quad \Rightarrow$ { compatibility applied to $(s_h, s_l) \varepsilon R$, $s_h \lhd U$ and $s_h \ltimes \complement \cdot \langle R^\sim\rangle(U)$: }
{ for some s'_h and s'_l, we have }
$(s'_h, s'_l) \varepsilon R$ and $s'_h \varepsilon U$ and $s'_l \ltimes_l \complement \cdot \langle R^\sim\rangle(U)$
$\quad \Rightarrow$

$(s'_h, s'_l) \varepsilon R$ and $s'_h \varepsilon U$ and $s'_l \varepsilon \complement \cdot \langle R^\sim \rangle(U)$
$\Leftrightarrow$
$(s'_h, s'_l) \varepsilon R$ and $s'_h \varepsilon U$ and $s'_l \not\varepsilon \langle R^\sim \rangle(U)$
$\Rightarrow$
contradiction!

☑

4.3 Localization

The collection of open subsets corresponding to an interaction system forms a set-presented sup-lattice with finite glbs (given by intersection). The actual notion of formal topology requires this lattice to be a frame. What is missing is *infinite distributivity*:

$$\bigvee_{i \epsilon I} (V \wedge U_i) \quad = \quad V \wedge \left(\bigvee_{i \epsilon I} U_i \right) .$$

In [27], the authors identify a restriction on *axiom sets*[4] to generate formal (distributive) topologies. Since their notion of inductive generation corresponds exactly to our reflexive and transitive closure, we can reuse their work.

> *Let* $(S, \leqslant)$ *be a preordered set, an interaction system on* S *is localized if the following holds:*
>
> $$s' \leqslant s \ , \ a \in A(s) \quad \Rightarrow \quad s' \varepsilon w^\circ \left(\bigcup_{d \epsilon D(s,a)} \{s[a/d]\} \downarrow \{s'\} \right)$$
>
> *where* $U \downarrow V \triangleq U^\downarrow \cap V^\downarrow$ *and* $U^\downarrow \triangleq \{s \in S \mid (\exists s' \varepsilon U)\ s \leqslant s'\}$.

By monotonicity, this implies in particular that the relation $\geqslant$ is a simulation from w to itself.

One result from [27] is that if $(w, \leqslant)$ is localized and if we extend the rules generating the reflexive and transitive closure of w with

$$\frac{s' \varepsilon U \qquad s \leqslant s'}{s \lhd U} \ \leqslant\text{-compat}$$

(*i.e.* we take the "*down* transitive closure" rather than the *reflexive* transitive closure), then the resulting sup-lattice of open sets will be distributive.

The preorder $\leqslant$ aims at representing *a priori* the notion of inclusion between basic opens: $s \leqslant s'$ intuitively means "$s \subseteq s'$". Note that we can always add such a preorder *a posteriori* by considering $s \leqslant s'$ iff $s \lhd \{s'\}$. This preorder is just the saturation of the identity which appears implicitly in the convergence axiom on page 88.

We now explore the meaning of localization in terms of interaction systems. The goal is to understand this notion to give meaning to the notions of *point* and *continuous maps*.

[4]: a notion equivalent to interaction system when equality is present

4.3.1 Localized Interaction Systems

§ *Self-Simulations.* The first step is to add a notion of preorder to the set of states. This order should be well-behaved w.r.t. to the parent interaction system. The most natural thing is to ask it to be a refinement:

▷ **Definition 4.3.1:** an *interaction system with self-simulation* on S is given by a pair (w, R) where:
- w is an interaction system on S;
- R is a refinement from w to itself.

We have:

○ **Lemma 4.3.2:** if R is a refinement from w to itself, then so is the reflexive and transitive closure of R.

proof: this is a consequence of the following facts: the identity is a refinement, refinements compose, and refinements are closed under arbitrary unions.

☑

Thus, we can always assume that the self simulation is a preorder, and we call it "$\geqslant$". The meaning of $s \leqslant s'$ is thus "s refines s'". We write $U^{\downarrow}$ for the down-closure of U, *i.e.* $U^{\downarrow} \triangleq \langle\leqslant\rangle(U)$.

○ **Lemma 4.3.3:** for any interaction system with self-simulation, we have

$$s \lhd_w U \quad \Rightarrow \quad \{s\}^{\downarrow} \lhd_w U^{\downarrow} .$$

proof: direct application of lemma 2.5.21 and the fact that $\geqslant$ is a refinement from w to w iff $\geqslant$ is a simulation from w^* to w^*.

☑

↬ Remark 16: for any given interaction system on S, there is a whole range of possible choices for the self simulation: the simplest one is the empty relation, or its reflexive transitive closure (the equality relation). Since refinements are closed under arbitrary unions, we also know (impredicatively) that there is a largest self-refinement from w to itself. This biggest refinement turns out to have a concise description: define

$$R_w \quad \triangleq \quad \big(\mathcal{J}_w(S) \times S\big) \cup \big(S \times \mathcal{A}_w(\emptyset)\big) .$$

Thus, $(s_1, s_2) \in R_w$ iff the Angel can avoid deadlocks from s_1 or the Demon can deadlock the Angel from s_2. It is quite easy to check that this relation is indeed a refinement. (Even if it is generally speaking not a linear simulation.)

§ *Localized Self-Simulations.* We now add a strong condition on the self-simulations in order to make the lattice of open sets distributive. We postpone the discussion about the computational relevance of this notion to section 4.3.2. The definition we use is a slight generalization of the notion of localization found in [27]:

▷ **Definition 4.3.4:** let $(w, \geqslant)$ be an interaction system with self-simulation; we say that $(w, \geqslant)$ is *localized* if the following holds:

$$s_1 \leqslant s_2 \ , \ a_2 \in A(s_2) \quad \Rightarrow \quad s_1 \lhd_w \bigcup_{d_2 \in D(s_2, a_2)} \{s_2[a_2/d_2]\} \downarrow \{s_1\} \ .$$

This is more general than the original definition appearing on page 94: the preorder $\geqslant$ needs not be a linear simulation, but only a refinement. Note also that this notion doesn't need the equality relation since $\{s\}^{\downarrow} = \{s' \mid s' \leqslant s\}$.

○ **Lemma 4.3.5:** suppose $(w, \geqslant)$ is localized, then we have:

$$s_1 \leqslant s_2 \ , \ a'_2 \in A^*(s_2) \quad \Rightarrow \quad s_1 \lhd_w \bigcup_{d'_2 \in D^*(s_2, a'_2)} \{s_2[a'_2/d'_2]\} \downarrow \{s_1\} \ .$$

This is reminiscent of lemma 3.3.5, and the proof is almost identical.
proof: let $s_1 \leqslant s_2$ and $a'_2 \in A^*(s_2)$; we proceed by induction on a'_2:

- if $a'_2 = \text{Exit}$, then the result is trivial: $s_1 \lhd \{s_2\} \downarrow \{s_1\} = \{s_1\}^{\downarrow}$.
- if a'_2 is of the form $\text{Call}(a_2, k_2)$, by localization, we know that

$$s_1 \quad \lhd \quad \bigcup_{d_2 \in D(s_2, a_2)} \{s_2[a_2/d_2]\} \downarrow \{s_1\} \ . \tag{4-2}$$

Let $s'_1 \ \varepsilon \ \bigcup_{d_2} \{s_2[a_2/d_2]\} \downarrow \{s_1\}$. This implies in particular, $s'_1 \leqslant s_2[a_2/d_2]$ for some $d_2 \in D(s_2, a_2)$. We can apply the induction hypothesis for $s'_1 \leqslant s_2[a_2/d_2]$ and $k_2(d_2) \in A^*(s_2[a_2/d_2])$ to get

$$s'_1 \quad \lhd \quad \bigcup_{d'_2 \in D^*(s_2[a_2/d_2], k_2(d_2))} \{s_2[a_2/d_2][k_2(d_2)/d'_2]\} \downarrow \{s'_1\} \ . \tag{4-3}$$

It is easy to check that

$$\bigcup_{d'_2 \in D^*(s_2[a_2/d_2], k_2(d_2))} \{s_2[a_2/d_2][k_2(d_2)/d'_2]\} \quad \subseteq \quad \bigcup_{d'_2 \in D^*(s_2, a'_2))} \{s_2[a'_2/d'_2]\}$$

and since $\{s'_1\}^{\downarrow} \subseteq \{s_1\}^{\downarrow}$, we can conclude that the RHS side of (4-3) is included in $\bigcup_{d'_2 \in D^*(s_2, a'_2))} \{s_2[a'_2/d'_2]\} \downarrow \{s_1\}$. By monotonicity, from (4-3), we get

$$s'_1 \quad \lhd \quad \bigcup_{d'_2 \in D^*(s_2, a'_2))} \{s_2[a'_2/d'_2]\} \downarrow \{s_1\} \ .$$

Since this is true for any $s_1 \ \varepsilon \ \bigcup_{d_2} \{s_2[a_2/d_2]\} \downarrow \{s_1\}$, we finally get

$$\bigcup_{d_2 \in D(s_2, a_2)} \{s_2[a_2/d_2]\} \downarrow \{s_1\} \quad \lhd \quad \bigcup_{d'_2 \in D^*(s_2, a'_2))} \{s_2[a'_2/d'_2]\} \downarrow \{s_1\} \ .$$

By transitivity with (4-2), we conclude that

$$s_1 \quad \lhd \quad \bigcup_{d'_2 \in D^*(s_2, a'_2))} \{s_2[a'_2/d'_2]\} \downarrow \{s_1\} \ .$$

☑

This allows to prove the following:

◦ **Lemma 4.3.6:** if $(w, \geqslant)$ is a localized interaction system, then $s \lhd_w U$ implies $s \lhd_w U \downarrow \{s\}$. More generally, if $U \lhd_w V$ then $U \lhd_w U \downarrow V$.

proof: suppose $s \lhd_w U$, *i.e.* there is some $a' \in A^*(s)$ s.t. whenever $d' \in D^*(s, a')$, we have $s[a'/d'] \varepsilon U$. Because $\leqslant$ is reflexive, we can apply the preceding lemma and get $s \lhd_w \bigcup_{d'} \{s[a'/d']\} \downarrow \{s\}$. By monotonicity, we obtain $s \lhd_w U \downarrow \{s\}$.

The second point follows easily from the first one.

☑

§ *Convergence and Distributivity.* We can now check that convergence is satisfied for localized interaction systems. The importance of this will be that convergence implies infinite distributivity of the binary $\wedge$ over arbitrary $\bigvee$ (lemma 4.3.9).

◦ **Lemma 4.3.7:** if $(w, \geqslant)$ is a localized interaction system, then

$$s \lhd_w U \ , \ s \lhd_w V \quad \Rightarrow \quad s \lhd_w U \downarrow V \ .$$

proof: by applying lemma 4.3.6, we know that $s \lhd_w U \downarrow \{s\}$. By lemma 4.3.3, we also know that $\{s\}^\downarrow \lhd_w V^\downarrow$, which implies that $U \downarrow \{s\} \lhd_w V^\downarrow$. By the second point of lemma 4.3.6 we can conclude that $U \downarrow \{s\} \lhd_w (U \downarrow \{s\}) \downarrow V^\downarrow = U \downarrow V \downarrow \{s\}$.

We thus have the following sequence:

$$s \quad \lhd_w \quad U \downarrow \{s\} \quad \lhd_w \quad U \downarrow V \downarrow \{s\} \quad \subseteq \quad U \downarrow V$$

which allows to conclude.

☑

In order to get a distributive lattice of open sets, we need to make sure the preorder is "compatible" with the covering by adding the $\leqslant$-compat rule (page 94). Equivalently,

▷ **Definition 4.3.8:** if $(w, \geqslant)$ is an interaction system with self-simulation, we write $\mathcal{A}_{w\leqslant}$ for the following predicate transformer:

$$U \quad \mapsto \quad \mathcal{A}_w(U^\downarrow) \ .$$

It is easy to check that $\mathcal{A}_{w\leqslant}$ is a closure operator,[5] so that the collection of prefixpoints for $\mathcal{A}_{w\leqslant}$ forms a complete sup-lattice, which is denoted by $\mathcal{O}_{w\leqslant}$.

◦ **Lemma 4.3.9:** if $(w, \geqslant)$ is a localized interaction system, then the sup-lattice $\mathcal{O}_{w\leqslant}$ is distributive.

proof: the most important remark is that convergence is equivalent to

$$\mathcal{A}_{w\leqslant}(U) \cap \mathcal{A}_{w\leqslant}(V) \quad = \quad \mathcal{A}_{w\leqslant}(U \downarrow V) \ .$$

The actual proof is not difficult and can be found in [79].

☑

↬ REMARK 17: it is also possible to show that in this case, $\mathcal{O}_w$ is a Heyting algebra in the sense that for any open predicate U, the operation $U \wedge \_$ has a right adjoint $U \Rightarrow \_$: put $U \Rightarrow V \triangleq \lambda s.U(s) \to V(s)$. This is the same construction as in $\mathcal{P}(S)$, so that the only thing to do is show that this predicate is open. This uses localization.

[5]: because $\mathcal{A}_{w\leqslant} = \mathcal{A}_w \cdot \langle\leqslant\rangle$, with $\mathcal{A}_w$ and $\langle\leqslant\rangle$ both closure operators s.t. $\langle\leqslant\rangle \cdot \mathcal{A}_w \subseteq \mathcal{A}_w \cdot \langle\leqslant\rangle$.

4.3.2 Computational Interpretation

We already argued in sections 2.6 that the notion of *refinement* does have a natural interpretation as *components, i.e.* programs providing an interface (given by their domain) relying on other interfaces (given by the codomain). Localization can be seen as the following requirement on an interaction system: the Demon should be able to answer *concurrent requests*. The situation where several clients (Angels) want to connect to a single server (Demon) is very common. In such a case, the server must find a way to answer *all* the clients concurrently, or at least *simulate* such a concurrent interaction in a sequential way.

§ *Localization.* If we spell out the definition of localization in details, we get: a preorder "$\geqslant$" is localized if

$$
\begin{array}{lll}
s' \leqslant s & \Rightarrow & \big(\forall a \in A(s)\big) \\
& & \big(\exists a' \in A^*(s')\big) \qquad (4\text{-}4) \\
& & \big(\forall d' \in D^*(s', a')\big) \\
& & \big(\exists d \in D(s, a)\big) \qquad (4\text{-}5) \\
& & \quad s'[a'/d'] \leqslant s[a/d] \quad \wedge \quad \underline{s'[a'/d'] \leqslant s'} \,.
\end{array}
$$

That "$\geqslant$" is localized is thus a strengthening of "$\geqslant$" being a refinement. It requires that if $s' \leqslant s$, then we can simulate any action from s by a sequence of actions from s', *and guarantee that the final simulating state gets finer*.

In order to understand this from a computational point of view, consider the following situation: let $(w, \geqslant)$ be localized. We allow the Demon to have an *internal,* or *hidden* state, different from the *external,* or *visible* state. Think of the internal state as the "real" state of the system. It is natural to require that the internal state is finer than the external state, so that the Demon can carry interaction from the visible state using the internal state of the system.

Suppose now that the internal state is s'; suppose also that $s' \ltimes V$, *i.e.* the Demon can maintain an invariant from s'. Suppose that the visible state is s, *i.e.* we have $s' \leqslant s$. Interaction between the server and a client in state s can be conducted in the following way:

- the visible state, as viewed by a client, is s; the server is internally in state s'.
- a client connects to the server and sends a request $a \in A(s)$;
- the Demon (server) needs to answer this request by some $d \in D(s, a)$, but he needs to carry out internal interaction:
  - *a)* he first translates the request $a \in A(s)$ into a (general) request $a' \in A^*(s')$ from his internal state according to (4-4);
  - *b)* because $s' \ltimes_w V$, the Demon can answer this (general) request by a (general) response $d' \in D^*(s', a')$;
  - *c)* the Demon can now translate back this (general) request into a (single) request $d \in D(s, a)$ according to (4-5);
- the client receives response $d \in D(s, a)$;
- the internal state of the Demon is now $s'[a'/d']$ and the visible state is $s[a/d]$, we have both $s'[a'/d'] \leqslant s[a/d]$ (consistency between internal and visible states) and $s'[a'/d'] \leqslant s'$.

The last point shows that if the client has more requests, she can ask them (because $s'[a'/d'] \leqslant s[a/d]$), but it also shows that *the Demon can answer another requests from visible initial state* s (because $s'[a'/d'] \leqslant s' \leqslant s$). This means that if there originally was a second client wanting to connect in state s, the server can now answer her request. In other words, the Demon can simulate concurrent interaction in a sequential way.

We can summarize this by a "generalized execution formula":

$$\frac{s_1 \lhd U_1 \ \ldots \ s_n \lhd U_n \qquad s \leqslant s_1, \ldots, s_n \qquad s \ltimes V}{(\exists s'_1, \ldots, s'_n, s') \quad s'_1 \ \varepsilon \ U_1 \ \ldots \ s'_n \ \varepsilon \ U_n \quad s' \leqslant s'_1, \ldots, s'_2 \quad s' \ltimes V}$$

which can be written in a more concise way as:

$$\frac{\mathcal{A}(U_1) \downarrow \ldots \downarrow \mathcal{A}(U_n) \ \between \ \mathcal{J}(V)}{U_1 \downarrow \ldots \downarrow U_n \ \between \ \mathcal{J}(V)} \quad .$$

This is a rather direct consequence of the simple execution formula and convergence.

One difference between the simple execution formula and the generalized one is that there is no canonical way to obtain the final state for interaction. Different strategies for interaction may yield different traces of interaction and different final states. For example, the server may start interaction on the left and then proceed on the right, or vice and versa; he may even interleave pieces of interaction on both sides. (The same applies to the proof of lemma 4.3.7.)

The computational interpretation is thus that with a localized interaction system, a server program can be turned into a "a concurrent virtual server" which can simulate independent parallel interaction with several clients in a sequential way.

§ *Points.* Now that localization has been given a computational interpretation, it is relatively easy to interpret the notion of *formal point* in terms of interaction.

▷ **Definition 4.3.10:** ([79]) if $(w, \geqslant)$ is a localized interaction system, a *formal point* in $(w, \geqslant)$ is given by a subset $\alpha : \mathcal{P}(S)$ s.t.

- α is not empty: $S \between \alpha$;
- α is closed: $\alpha = \mathcal{J}_w(\alpha)$ (or equivalently, $\alpha \subseteq \mathcal{J}_w(\alpha)$);
- α is *convergent*: if $s_1 \ \varepsilon \ \alpha$ and $s_2 \ \varepsilon \ \alpha$ then $s_1 \downarrow s_2 \between \alpha$.

It is obvious that a subset α is closed iff it is of the form $\mathcal{J}(V)$ for some subset $V \subseteq S$. As we saw in section 2.6.3, a non-empty closed subset can be seen as a specification for a server program where the initial state predicate is trivial (always true). We saw above that in a localized interaction system, such a server program could be changed into a "virtual concurrent server program", where by concurrent, we means that the server can answer requests to several clients at the same time, *provided the clients' states are "compatible" with the server's state* (see the generalized execution formula). The additional condition of convergence can be seen as a strengthening of that: it requires that for any finite number of initial, independent requests, there is a compatible server state. It makes it possible to conduct interaction as follows:

0) the server specification is given by $S \ltimes V$;

1) there are many clients waiting to connect to the server, their specifications are given by $s_1 \lhd \textsc{Goal}_1, \ldots, s_n \lhd \textsc{Goal}_n$;

2) by convergence of the formal point, the server can choose a state $s \in s_1 \downarrow \ldots \downarrow s_n$ s.t. $s \ltimes V$;

3) interaction between the server and clients goes on as described in the previous section, according to the generalized execution formula;

...) when this is finished (or even before), new clients may connect to the server, but they have to respect the server's state: when the server is in state s', the client specification needs to be $s \lhd \textsc{Goal}$ with $s' \leqslant s$.

Thus, a *formal point* is given by a specification for a server which can respond to any finite number of initial requests and then go on forever as any other normal server program.

§ *Continuity.* We now at look at the notion of *continuous map* between formal spaces. We recall the definition that can be found for example in [36], and briefly give a possible computational interpretation in terms of interaction. The situation is not entirely clear, and more work is probably needed before reaching a satisfactory explanation.

▷ **Definition 4.3.11:** if $(w_h, \geqslant)$ and $(w_l, \geqslant)$ are localized interaction systems, a refinement from w_h to w_l is *continuous* if it is both:

- *convergent*: $R(s_1) \downarrow R(s_2) \lhd_l R(s_1 \downarrow s_2)$ for all $s_1, s_2 \in S_h$;
- and *total*: $S_l \lhd_l R(S_h)$.

This is just the definition of formal continuous map, with the arrows reversed.

Just like in the concrete interpretation of the condition of convergence for formal points, a refinement is *convergent* if it can refine several independent states concurrently. This means that if a low-level state s_l refines s_1 and s_2, it can also refine (modulo low-level interaction) a state finer than both s_1 and s_2. Using the generalized execution formula outlined above, this means that it is possible to simulate s_1 and s_2 concurrently. This can be extended to any finite number of concurrent independent states.

The meaning of totality is subtle. Our current understanding is the following: it means that from any low-level state, the Angel can always conduct a finite low-level interaction to reach "a high level state", or in other words, reach a low-level state refining a high-level state. What is slightly disturbing is that this property will never be used if we conduct interaction in the natural way, *i.e.* use the refinement as a black box between high-level and low-level (using the generalized execution formula). The condition requires the refinement to be strong enough so that in *any* situation, the low-level Angel can prevent infinite low-level interaction. It seems to be related to some kind of *productivity* condition.

4.4 A non-Localized Example: Geometric Linear Logic

We now give an example of interaction system which is naturally non-localized. The point of departure is an elegant topological completeness result for geometric theories. If we replace the notion of geometric theory by an obvious notion of "linear geometric theory", we obtain a non localized topological semantics.

4.4.1 Geometric Logic

To start with, we recall the construction of [15] and [24], associating a localized interaction system to any geometric theory. It is well-known that we can interpret intuitionistic logic in any topological space.[6] Since being true in this particular model is equivalent to being provable, the construction of this localized interaction system can be seen as an elementary proof of completeness with respect to this semantic. We skip all the details about the general notion of topological model and only show how to construct the particular interaction system.

§ *Geometric Theories.* Fix a first order language $\mathcal{L}$. As usual, terms are constructed from variables (x, x_1, ...), parameters (constants, a, a_1, ...) and function symbols. Formulas are built as usual.

▷ **Definition 4.4.1:** a *geometric formula* is a formula of the form

$$\Big((\exists \vec{x})\ A^1_1 \wedge \cdots \wedge A^1_{n_1}\Big) \quad \vee \quad \cdots \quad \vee \quad \Big((\exists \vec{x})\ A^k_1 \wedge \cdots \wedge A^k_{n_k}\Big)$$

where all the A^j_i are atomic formulas.

A *geometric theory* is a (not necessarily finite) set of implications between geometric formulas.

The empty disjunction is written $\bot$ and the empty conjunction is written $\top$.

○ **Lemma 4.4.2:** for every geometric theory, we can find an equivalent geometric theory (in the sense that intuitionistic provability coincide for both theories) where all axioms have the form:

$$\bigwedge_l A_l \quad \to \quad \bigvee_j (\exists \vec{x}) \bigwedge_i B^j_i$$

where the A_i's and B^j_i's are atomic.

proof: we only need to "expand" axioms according to the following:

- $F_1 \vee F_2 \ \to\ G$ gives $\{F_1 \to G\ ,\ F_2 \to G\}$;
- $(\exists \vec{x})\, F \ \to\ G$ gives $\big\{F[\vec{t}/\vec{x}] \to G \mid t \text{ is a term}\big\}$.

☑

An interesting case of geometric theory is given by any collection of Horn clauses, where all axioms have the form $A_1 \wedge \cdots \wedge A_n \to B$ and the LHS or the RHS can be empty.

§ *A Generic Topological model for Geometric Theories.* To any geometric theory $\mathcal{T}$, we can associate an interaction system in the following way:

- *states* are given by finite sets of closed atomic formulas (called *facts*);
- if F is such a set, then an action in that state, also called a *question,* is given by a closed instance of an axiom of $\mathcal{T}$ with all its LHS formulas in F (*i.e.* an action is a pair (σ, Ax), where σ is a closed substitution);

[6]: More generally in any Heyting algebra.

- a answer to such an action, also called an *answer*, is given by a vector $\vec{u}$ of terms, together with a RHS disjunct $(\exists \vec{x})\ \bigwedge_i B_i^{j_0}$ of Ax (where the length of $\vec{u}$ and $\vec{x}$ have to match);
- the new state is then $F \cup \left\{B_i^{j_0}[\vec{u}/\vec{x}] \mid i = 1, \ldots\right\}$.

The intuition is straightforward:

- a state is a *state of knowledge* about the world: it contains the facts the Angel knows to be true;
- an action is a question the Angel can ask: since (an instance of) the LHS is true, then the RHS is bound to be true;
- the answer to this question is one (instance of) a RHS disjunct;
- the new state is obtained by adding the new knowledge she got from the answer.

Note that in the case of Horn clauses, the RHS is trivial (singleton); the interaction system is thus of the form $\langle \nu \rangle$ for a *transition* system ν.

Consider the artificial example where we have (among others) the following axiom: $A \wedge B \to (D \wedge E) \vee F$. From the state $\{A, B, C\}$, this is a possible question. It can result in two different new states:

- $\{A, B, C, D, E\}$ if the answer is $D \wedge E$;
- $\{A, B, C, F\}$ if the answer is F.

We can define a "refinement" order on states by putting $F \leqslant G$ if $G \subseteq F$. This means that F is *finer* than G, or that F contains more knowledge that G. It is quite easy to see that this order is localized (definition 4.3.4) because the next state function is antitonic. The lattice $\mathcal{O}_{\mathcal{T},\subseteq}$ we obtain from the reflexive and transitive closure of this interaction system gives rise to a formal topology.

This topology is interesting for the following reason:

◇ **Proposition 4.4.3:** *if $\mathcal{T}$ is a geometric theory and if Γ is a finite set of atomic formulas, then we have*

$$\Gamma \vdash_{\mathcal{T}} \bigvee_j \bigwedge_i (\exists \vec{x}\ B_i^j)$$

iff

$$\Gamma \lhd_{\mathcal{T},\leqslant} \bigcup_{\vec{t}\ \text{closed}} \left\{ \left\{ B_i^j[\vec{t}/\vec{x}] \mid i = 1, \ldots \right\} \mid j = 1, \ldots \right\} .$$

This gives an elementary/predicative proof of completeness for geometric logic in the following sense: we know that that for *any* formal topology, $\Gamma \vdash_{\mathcal{T}} F$ implies $\Gamma \lhd_{\mathcal{T}} F^*$ where F^* is the interpretation of F. (Soundness of the interpretation.) The interaction system we have just defined shows completeness.

4.4.2 Linear Geometric Logic

The interaction system in the previous system was localized. The intuitive reason was that states were states of *knowledge*. In particular, the state could only get finer as new knowledge is added to it. One of the guiding intuition about linear logic (refer to section 5.1 for a more thorough introduction) is that it is a logic of *resources* rather than a logic of truth. It is thus natural to look at the previous construction in a linear setting.

§ *Linear Geometric Theories.* Fix a first order language language $\mathcal{L}$.

▷ **Definition 4.4.4:** a *linear geometric formula* is a formula of the form:

$$\Big((\exists \vec{x})\ A^1_1 \otimes \cdots \otimes A^1_{n_1}\Big) \quad \oplus \cdots \oplus \quad \Big((\exists \vec{x})\ A^k_1 \otimes \cdots \otimes A^k_{n_k}\Big)$$

where all the A^j_i's are atomic.

A *linear geometric theory* is a (non necessarily finite) set of linear implications between linear geometric formulas.

We write $\mathbf{0}$ for the empty disjunction ($\oplus$) and $\mathbf{1}$ for the empty conjunction ($\otimes$).

Just like above, we have

○ **Lemma 4.4.5:** any linear geometric theory is equivalent to a linear geometric theory in which all the axioms have the form

$$\bigotimes_l A_l \quad \multimap \quad \bigoplus_j (\exists \vec{x_j}) \bigotimes_i B^j_i\ .$$

The proof is similar to that of lemma 4.4.2.

§ *Pretopologies and Intuitionistic Phase Spaces.* If we want to extend the previous section to deal with linear logic, we need to find a notion corresponding to topological semantics. This has already been done under two different names: pretopologies ([77]) and intuitionistic phase spaces ([69]). In traditional phase semantics, a formula is interpreted by a *fact*, that is, a subset of a monoid equal to its biorthogonal (see [39]). Since in the intuitionistic world we cannot rely on the biorthogonal, we replace it by a closure operator satisfying some mild conditions. Since we want to keep some topological intuition, we will adopt Sambin's terminology.

▷ **Definition 4.4.6:** a *pretopology* is given by

- a commutative monoid $(S, \cdot, \mathbf{1})$;
- a closure operator on $\mathcal{P}(S)$ satisfying $\mathcal{A}(U) \cdot \mathcal{A}(V) \subseteq \mathcal{A}(U \cdot V)$.[7]

An *open set* in a pretopology is a subset U of S s.t. $U = \mathcal{A}(U)$. We write $\mathcal{O}$ for the collection of open set of a pretopology.

Just like in the previous sections, we will write $s \lhd U$ as a synonym for $s \in \mathcal{A}(U)$. A structure for a language $\mathcal{L}$ with respect to a pretopology $(S, \cdot, \mathbf{1}, \mathcal{A})$ is given by:

- a set D
- if f is an n-ary function symbol, a function $f^*: D^n \to D$;
- if A is an n-ary relation symbol, a function $A^*: D^n \to \mathcal{O}$.

For a valuation ρ, terms are interpreted by elements of D, and if F is a formula, its interpretation F^*_ρ is defined as:

- $\mathbf{1}^*_\rho \triangleq \mathcal{A}(\{\mathbf{1}\})$ and $\mathbf{0}^*_\rho \triangleq \mathcal{A}(\emptyset)$;
- $A(t_1, \ldots, t_n)^*_\rho \triangleq A^*({t_1}^*_\rho, \ldots, {t_n}^*_\rho)$ for atomic A;
- $(F_1 \otimes F_2)^*_\rho \triangleq \mathcal{A}({F_1}^*_\rho \cdot {F_2}^*_\rho)$;
- $(F_1 \oplus F_2)^*_\rho \triangleq \mathcal{A}({F_1}^*_\rho \cup {F_2}^*_\rho)$;
- $(F_1 \multimap F_2)^*_\rho \triangleq \{s \mid s \cdot {F_1}^*_\rho \subseteq {F_2}^*_\rho\}$;
- and $\big((\exists x)F\big)^*_\rho \triangleq \mathcal{A}\big(\bigcup_t F^*_{\rho, x=t}\big)$. *(Not part of the original definition from [69].)*

It is trivial to check that the interpretation of a formula is always an open set.

[7]: We extend the operation "$\cdot$" on subsets in the traditional way: $U \cdot V = \{s_1 \cdot s_2 \mid s_1 \in U, s_2 \in V\}$.

§ *Soundness.* Soundness w.r.t. this semantics states that (see [77] or [69]):

$$F_1, \ldots, F_n \vdash G \qquad \Rightarrow \qquad {F_1}^*_\rho \cdot \ldots \cdot {F_n}^*_\rho \subseteq G^*_\rho$$

for any valuation ρ on any structure on any pretopology.

Since we are dealing with deduction w.r.t. a given theory, we need to modify this slightly: if $\mathcal{T}$ is a theory, a $\mathcal{T}$-structure w.r.t. to a pretopology $(S, \cdot, \mathbf{1}, \mathcal{A})$ is a structure for this pretopology satisfying the additional condition $\mathbf{1} \lhd T^*_\rho$ for all axioms T of $\mathcal{T}$ and all valuations ρ. We can easily extend soundness:

$$F_1, \ldots, F_n \vdash_{\mathcal{T}} G \qquad \Rightarrow \qquad {F_1}^*_\rho \cdot \ldots \cdot {F_n}^*_\rho \subseteq G^*_\rho$$

for any valuation on any $\mathcal{T}$-structure on any pretopology:
$F_1, \ldots, F_n \vdash_{\mathcal{T}} G$
$\quad\Leftrightarrow\quad$ { for some $T_1, \ldots, T_m$ of finite multiplicity in $\mathcal{T}$ }
$T_1, \ldots, T_m, F_1, \ldots, F_n \vdash G$
$\quad\Rightarrow\quad$ { soundness }
${T_1}^*_\rho \cdot \ldots \cdot {T_m}^*_\rho \cdot {F_1}^*_\rho \cdot \ldots \cdot {F_n}^*_\rho \subseteq G^*_\rho$ for any valuation ρ
$\quad\Rightarrow\quad$ { $\mathbf{1} \lhd {T_i}^*_\rho$, *i.e.* $\{\mathbf{1}\} \subseteq {T_i}^*_\rho$ for all $i = 1, \ldots, m$ }
$\{\mathbf{1}\} \cdot \ldots \cdot \{\mathbf{1}\} \cdot {F_1}^*_\rho \cdot \ldots \cdot {F_n}^*_\rho \subseteq G^*_\rho$ for any valuation ρ
$\quad\Leftrightarrow\quad$ { $\mathbf{1}$ is neutral for "$\cdot$" }
${F_1}^*_\rho \cdot \ldots \cdot {F_n}^*_\rho \subseteq G^*_\rho$ for any valuation ρ

§ *Completeness.* In order to show completeness, one needs to find a particular model $\mathcal{M}$ satisfying "if $\Gamma \models F$ then $\Gamma \vdash F$". We will construct, in an elementary way, a pretopology together with a $\mathcal{T}$-structure satisfying this. One nice feature about this pretopology is that it makes no direct reference to provability. It only depends on the set of axioms of the theory!

Suppose $\mathcal{T}$ is a linear geometric theory; we construct an interaction system inspired by the previous section:[8]

- a state is a *finite multiset*[9] of closed atomic formulas;
- an action in state Γ is a closed instance of an axiom Ax such that its LHS is included[10] in Γ (*i.e.* an action is a pair (σ, Ax) where σ is a closed substitution for free variables);
- a reaction to such an action is given by a vector $\vec{u}$ of closed terms and a RHS disjunct $(\exists \vec{x}) \bigotimes_i B_i^{j_o}$ of the axiom (the length of $\vec{u}$ and $\vec{x}$ have to match);
- the new state is given by $\Gamma \cup \{B_i^{j_o}[\vec{u}/\vec{x}] \mid i = 1, \ldots\} \setminus \{A_l(\vec{t}) \mid l = 1, \ldots\}$.

The difference with the previous interaction system is typical of the intuitions of linear logic (see "Lafont's menu" interpretation of the linear connectives):

- a state is a finite set of *resources* the Angel has at her disposal;
- an action is an experiment she can conduct. In order to do so, she must have all the necessary resources for the experiment (the LHS of the axiom);

[8]: the case of propositional linear geometric theory is much simpler and contains all the interesting ideas. The reader is encouraged to forget about quantifiers...

[9]: list modulo reindexing, or finite set with finite multiplicities

[10]: where inclusion takes multiplicities into accounts: $[1,1,2] \subseteq [1,1,1,2,3]$ but $[1,1,2] \not\subseteq [1,2,3]$.

- a reaction is given by a possible outcome of the experiment (a RHS disjunct);
- the new state is given by removing the resources used to conduct the experiment and adding the results produced by the experiment.

Thus, an axiom of the form

$$A_1 \otimes \cdots \otimes A_n \quad \rightarrow \quad \left(B^1_1 \otimes \cdots \otimes B^1_{n_1}\right) \oplus \cdots \oplus \left(B^k_1 \otimes \cdots \otimes B^k_{n_k}\right)$$

is read as an experiment which uses resources A_l's and produces the $B^{j_0}_i$'s for one particular j_0.

Even if reverse inclusion is a self-simulation, as opposed to the previous case, it is not localized. It is very easy to find a linear geometric theory for which reverse inclusion does not satisfy lemma 4.3.6. Take for example the theory consisting of the single axiom $A \otimes A \multimap A$. Moreover, if we want to model provability (as in proposition 4.4.3), it doesn't make sense to close sets of states downward since this amounts to allowing weakening (*i.e.* to use affine logic rather than linear logic).

We have the following easy result:

▷ **Definition 4.4.7:** if P is a closed geometric linear formula $\bigoplus_j (\exists \vec{x}) \bigotimes A^j_i$, define the set of states $\widetilde{P}$ as

$$\widetilde{P} \quad \triangleq \quad \bigcup_{\vec{t} \text{ closed}} \left\{ \left[B^j_i[\vec{t}/\vec{x}]\rho\right]_{i=1,\ldots} \mid j = 1, \ldots \right\}$$

where ρ is a closed valuation for the free variables of P.

○ **Lemma 4.4.8:** if P is a closed linear geometric formula and if Γ is a finite multiset of closed atomic formulas, then we have

$$\Gamma \lhd_{\mathcal{T}} \widetilde{P} \quad \Rightarrow \quad \Gamma \vdash_{\mathcal{T}} P$$

where we write $\lhd_{\mathcal{T}}$ for the covering associated to the above interaction system.

proof: simple induction on the proof that $\Gamma \lhd_{\mathcal{T}} \widetilde{P}$. ☑

We now need to show that this interaction system with its covering operator corresponds to the semantics of a pretopology with a $\mathcal{T}$-structure. The beginning is easy: define

- S defined as above;
- $\cdot$ is the sum of multisets and $\mathbf{1}$ is the empty multiset $[]$;
- $\Gamma \, \varepsilon \, \mathcal{A}(U)$ is defined as $\Gamma \lhd U$.

We have:

○ **Lemma 4.4.9:** $(S, \cdot, \mathbf{1}, \mathcal{A})$ is a pretopology.

proof: the only thing to prove is that $\mathcal{A}(U) \cdot \mathcal{A}(V) \subseteq \mathcal{A}(U \cdot V)$. This is direct. ☑

Defining a structure for this pretopology is also very easy: take the domain D to be the set of closed terms, and interpret function symbols as themselves. The interpretation of relation symbols is given by:

$$A^*(t_1,\dots,t_n) \quad \triangleq \quad \mathcal{A}\big(A(t_1,\dots,t_n)\big) \ .$$

This definition allows to prove:

- **Lemma 4.4.10:** the semantics from the previous paragraph and the semantics associated to the above interaction system coincide: for any atomic formulas $A_1,\dots,A_n$ and closed linear geometric formula F, we have:

$$[A_1,\dots,A_n] \lhd \widetilde{F} \qquad \text{iff} \qquad [A_1,\dots,A_n] \subseteq F^*$$

proof: Easy...

☑

The last things is to check that this intuitionistic phase space validates the axioms:

- **Lemma 4.4.11:** in the structure over $(S_{\mathcal{L}},\cdot,\mathcal{A})$ just defined, all the axioms of $\mathcal{T}$ are validated.

Where by validated, we mean: F is validated if $\mathbf{1} \lhd_{\mathcal{T}} F^*$.

proof: we need to show $\mathbf{1} \lhd_{\mathcal{T}} F^*$. This is true as soon as $\mathbf{1} \in F^*$. Since F is an axiom of $\mathcal{T}$, it is necessarily of the form $A_1 \otimes \cdots \otimes A_n \multimap \bigoplus_j (\exists \vec{x_j}) \bigotimes_i B_i^j$ and thus

$$\begin{array}{lcll}
\mathbf{1} \in F_i^* & \Leftrightarrow & \mathbf{1}\cdot A_1^* \cdot \ldots \cdot A_n^* & \subseteq \quad \Big(\bigoplus_j (\exists \vec{x_j}) \bigotimes_i B_i^j\Big)^* \\
& \Leftrightarrow & A_1^* \cdot \ldots \cdot A_n^* & \subseteq \quad \mathcal{A}\Big(\bigcup_{j,\vec{t}} \mathcal{A}\big((B_1^j[\vec{t}/\vec{x}])^* \cdot \ldots\big)\Big) \\
& \Leftrightarrow & A_1^* \cdot \ldots \cdot A_n^* & \subseteq \quad \mathcal{A}\Big(\bigcup_{j,\vec{t}} (B_1^j[\vec{t}/\vec{x}])^* \cdot \ldots\Big) \\
& \Leftrightarrow & A_1^* \cdot \ldots \cdot A_n^* & \lhd_{\mathcal{T}} \quad \bigcup_{j,\vec{t}} (B_1^j[\vec{t}/\vec{x}])^* \cdot \ldots
\end{array}$$

which is easily seen to be true from the definition of $\lhd_{\mathcal{T}}$.

☑

We can now finish the proof of completeness as follows: if F is a geometric formula which is true in all pretopological $\mathcal{T}$-structures, then it is in particular true for the pretopology associated to $\lhd_{\mathcal{T}}$. By lemma 4.4.8 it is thus provable. The result is in fact slightly more general than that, since it allows the presence of a finite number of (closed) atomic formulas as hypothesis.

↬ Remark 18: since F needs not be closed, we need to first replace F by a closed instance where all free variables have been replaced by fresh parameters...

§ *Completeness, more.* A topos theoretic result (Barr's theorem) implies that higher order classical reasoning with the axiom of choice coincides with first order intuitionistic reasoning for geometric theories ([12], and [86] for a simpler reading). A similar result holds for linear geometric theories, namely, higher order classical reasoning coincides with first order intuitionistic reasoning.

◇ **Proposition 4.4.12:** *if* G *is a geometric formula and* Γ *a finite multiset of atomic formulas, then we have*

$$\Gamma \vdash_{\mathcal{T}} G \quad \Rightarrow \quad \gamma \lhd_{\mathcal{T}} \widetilde{G}$$

where $\vdash_{\mathcal{T}}$ *denotes higher order classical deduction.*

proof: take a (classical) cut free proof of the sequent $\Gamma \vdash_{\mathcal{T}} G$, *i.e.* a cut free-proof of some $!T_0, \Gamma \vdash G$.
Note that since all the formula on the RHS of the sequent are positive, the number of formula on the RHS of the sequent cannot decrease. (Some RHS formula may decompose into structurally simpler formula, but they cannot disappear.) In particular, if the RHS contains 2 atomic formulas, then we cannot close it! This means that all the sequents in the proof are actually intuitionistic sequents, and so the proof is intuitionistic.
That this semantics is complete w.r.t. higher order reasoning follows from cut elimination for higher order linear logic.
☑

§ *Going Back to Traditional Geometric Theories.* While getting the real exponentials of linear logic seems difficult,[11] we can still encode weakening and contraction on arbitrary formulas. This allows to get classical geometric theories from linear ones via an appropriate encoding.
Here is how we can allow weakening on an occurrence of a formula F: we start by adding a new relation symbol ω_F, whose arity is the same as the arity of F; we then replace the occurrence of F by ω_F and add the following axioms:

$$\begin{aligned} \omega_F &\multimap F \\ \omega_F &\multimap \mathbf{1} . \end{aligned}$$

Adding contraction for F is similar: we add a symbol χ_F and the axioms

$$\begin{aligned} \chi_F &\multimap F \\ \chi_F &\multimap \chi_F \otimes \chi_F . \end{aligned}$$

This allows to get back the usual geometric theories in the following way: for each relation symbol A, add the following axioms:

$$\begin{aligned} A &\multimap \mathbf{1} \\ A &\multimap A \otimes A . \end{aligned}$$

[11]: As expected, problems come from the *promotion* rule.

It is quite easy to see that if we translate a classical formula F on $\mathcal{L}$ into a linear formula $\widetilde{F}$ on $\mathcal{L}$ by replacing $\wedge$, $\vee$, $\rightarrow$ and $\exists$ respectively by $\otimes$, $\oplus$, $\multimap$ and $\exists$, we have that

$$\Gamma \vdash_{i,\mathcal{T}} F \quad \text{iff} \quad \widetilde{\Gamma} \vdash_{i,\widetilde{\mathcal{T}}} \widetilde{F} .$$

Where deduction on the left is intuitionistic logic and deduction on the right is intuitionistic linear logic.

Part II

Linear Logic

5 Linear Logic and the Relational Model

This second part is concerned with denotational semantics of linear logic. We will define additional structure on the category **Int** to extend it to a model for full linear logic. We start by introducing the syntax of linear logic and the corresponding notion of denotational model. We then briefly look at the simple category **Rel** of sets and relations.

§ *(New) Notation.* After several unsuccessful attempts, I decided to give up consistency of notation between the two parts of this thesis. Here is a list of the differences with the previous part:

- we remove the distinction between sets and subsets, and use the symbol "ϵ" for membership;
- the notion of "subset" is the classical one;
- sets are usually written with capital letters from the end of the alphabet (X, Y,... and S);
- when dealing with subsets of states, we use *small* letters from the end of the alphabet: $x \subseteq S$,...
- relations are denoted by *small* letters: $r \subseteq S_1 \times S_2$.

5.1 An Introduction to Linear Logic

Linear logic ([39]) can be seen as a refinement of traditional logic obtained by restricting the use of "structural rules" in the sequent calculus. Those structural rules are implicit in most presentations of the proof calculus: they deal with repetitions and erasing of formulas.

5.1.1 Intuitionistic Linear Logic

Intuitionistic linear formulas are constructed from the following grammar:

$$\begin{array}{rcl} F_1, F_2 & ::= & X \mid \bot \mid \mathbf{1} \mid F_1 \otimes F_2 \mid F_1 \multimap F_2 \\ & & \mid \top \mid \mathbf{0} \mid F_1 \oplus F_2 \mid F_1 \,\&\, F_2 \\ & & \mid\ !F_1 \end{array}$$

where X belongs to a set $\mathcal{X}$ of propositional variables.

An intuitionistic sequent is of the form $\Gamma \vdash F$, where Γ is a list of formulas and F a formula. Since we are only dealing with commutative linear logic, we allow shuffling the list transparently. The calculus is given by the following rules:

permutation:

- $\dfrac{\Gamma , G_2 , G_1 , \Delta \vdash F}{\Gamma , G_1 , G_2 , \Delta \vdash F}$ this rule is applied transparently.

Axiom and cut:

- axiom: $\dfrac{}{X \vdash X}$;
- cut: $\dfrac{\Gamma \vdash F_1 \qquad \Delta , F_1 \vdash F_2}{\Gamma , \Delta \vdash F_2}$.

Additive connectives:

- constants: $\dfrac{}{\Gamma \vdash \top}$ and $\dfrac{}{\Gamma , \mathbf{0} \vdash F}$;
- "plus":
  - right: $\dfrac{\Gamma \vdash F_1}{\Gamma \vdash F_1 \oplus F_2}$ and $\dfrac{\Gamma \vdash F_2}{\Gamma \vdash F_1 \oplus F_2}$;
  - left: $\dfrac{\Gamma , G_1 \vdash F \qquad \Gamma , G_2 \vdash F}{\Gamma , G_1 \oplus G_2 \vdash F}$;
- "with":
  - right: $\dfrac{\Gamma \vdash F_1 \qquad \Gamma \vdash F_2}{\Gamma \vdash F_1 \,\&\, F_2}$;
  - left: $\dfrac{\Gamma , G_1 \vdash F}{\Gamma , G_1 \,\&\, G_2 \vdash F}$ and $\dfrac{\Gamma , G_2 \vdash F}{\Gamma , G_1 \,\&\, G_2 \vdash F}$.

Multiplicative connectives:

- constants: $\dfrac{}{\vdash \mathbf{1}}$, $\dfrac{\Gamma \vdash F}{\Gamma , \mathbf{1} \vdash F}$;

  and $\dfrac{\Gamma \vdash}{\Gamma \vdash \bot}$, $\dfrac{}{\bot \vdash}$; *(in other words, $\bot$ is a notation for an empty RHS)*
- "tensor":
  - right: $\dfrac{\Gamma \vdash F_1 \qquad \Delta \vdash F_2}{\Gamma , \Delta \vdash F_1 \otimes F_2}$;
  - left: $\dfrac{\Gamma , G_1 , G_2 \vdash F}{\Gamma , G_1 \otimes G_2 \vdash F}$;
- "linear arrow":
  - right: $\dfrac{\Gamma , F_1 \vdash F_2}{\Gamma \vdash F_1 \multimap F_2}$;
  - left: $\dfrac{\Gamma_1 \vdash G_1 \qquad \Gamma_2 , G_2 \vdash F}{\Gamma_1 , \Gamma_2 , G_1 \multimap G_2 \vdash F}$.

Exponentials:

- weakening: $\dfrac{\Gamma \vdash F}{\Gamma , !G \vdash F}$;
- dereliction: $\dfrac{\Gamma , G \vdash F}{\Gamma , !G \vdash F}$;

- contraction: $\dfrac{\Gamma, !G, !G \vdash F}{\Gamma, !G \vdash F}$;
- promotion: $\dfrac{!\Gamma \vdash F}{!\Gamma \vdash !F}$ *(where* $!(G_1, \ldots, G_n) = !G_1, \ldots, !G_n$*).*

5.1.2 Classical Linear Logic

One can get to classical logic from intuitionistic logic by adding the principle of double negation: $\neg\neg F \to F$. Classical linear logic is obtained in the same way by adding the principle $F^{\perp\perp} \multimap F$. Just like traditional logic, because of the symmetries, there are many equivalent ways to present the calculus. We choose the following:

- only atoms can be negated; full negation is obtained by de Morgan laws;
- we use single sided sequents.

Besides that, classical linear formulas are not very different from intuitionistic ones:

$$\begin{array}{rcl} F_1, F_2 & ::= & X \mid X^{\perp} \mid \bot \mid \mathbf{1} \mid F_1 \otimes F_2 \mid F_1 \parr F_2 \\ & & \mid \top \mid \mathbf{0} \mid F_1 \oplus F_2 \mid F_1 \,\&\, F_2 \\ & & \mid\ !F_1 \mid ?F_1 \end{array}$$

where X belongs to a set $\mathcal{X}$ of propositional variables. Note the presence of negation on atoms, that the connective "$\multimap$" is replaced by "$\parr$" and the presence of a new unary connective: "$?\_$". A sequent is now of the form $\vdash \Gamma$ where Γ is a finite list of formulas. We have the following sequent calculus:

Axiom and cut:

- axiom: $\dfrac{}{\vdash X^{\perp}, X}$;
- cut: $\dfrac{\vdash \Gamma, F \qquad \vdash F^{\perp}, \Delta}{\vdash \Gamma, \Delta}$.

Additive connectives:

- constants: $\dfrac{}{\vdash \Gamma, \top}$ and *(no rule for* $\mathbf{0}$*);*
- *"plus":* $\dfrac{\vdash \Gamma, F_1}{\vdash \Gamma, F_1 \oplus F_2}$ *and* $\dfrac{\vdash \Gamma, F_2}{\vdash \Gamma, F_1 \oplus F_2}$;
- *"with":* $\dfrac{\vdash \Gamma, F_1 \qquad \vdash \Gamma, F_2}{\vdash \Gamma, F_1 \,\&\, F_2}$.

Multiplicative connectives:

- constants: $\dfrac{}{\vdash \mathbf{1}}$ and $\dfrac{\vdash \Gamma}{\vdash \Gamma, \bot}$;
- "tensor": $\dfrac{\vdash \Gamma, F_1 \qquad \vdash \Delta, F_2}{\vdash \Gamma, \Delta, F_1 \otimes F_2}$;
- "par": $\dfrac{\vdash \Gamma, F_1, F_2}{\vdash \Gamma, F_1 \parr F_2}$.

Exponentials:

- weakening: $\dfrac{\vdash \Gamma}{\vdash \Gamma, ?F}$;

- dereliction: $\dfrac{\vdash \Gamma , F}{\vdash \Gamma , ?F}$;
- contraction: $\dfrac{\vdash \Gamma , ?F , ?F}{\vdash \Gamma , ?F}$;
- promotion: $\dfrac{\vdash ?\Gamma , F}{\vdash ?\Gamma , !F}$ *(where* $?(G_1, \ldots, G_n) = ?G_1, \ldots, ?G_n$*)*.

We define the linear negation $F^\perp$ of a formula by the following de Morgan laws:

- $X^{\perp\perp} \triangleq X$;
- $(F_1 \oplus F_2)^\perp \triangleq F_1^\perp \,\&\, F_2^\perp$;
- $(F_1 \,\&\, F_2)^\perp \triangleq F_1^\perp \oplus F_2^\perp$;
- $(F_1 \otimes F_2)^\perp \triangleq F_1^\perp \parr F_2^\perp$;
- $(F_1 \parr F_2)^\perp \triangleq F_1^\perp \otimes F_2^\perp$;
- $(!F)^\perp \triangleq ?(F^\perp)$;
- $(?F)^\perp \triangleq !(F^\perp)$.

Moreover, the formula $F_1 \multimap F_2$ is defined as $F_1^\perp \parr F_2$.

This system, like the previous one, enjoys cut elimination: there is a rewriting procedure transforming any proof into a proof of the same sequent *which doesn't use the cut rule*.

5.2 Categorical Models of Linear Logic

A naive approach to making a denotational model of classical logic is simply to take a cartesian closed category (modeling the simply typed λ-calculus, *i.e.* intuitionistic logic) and require negation to be involutive: $\neg\neg F \simeq F$. However, such a category is trivial: it is given by a partial order, *i.e.* a boolean algebra (see for example the short note [75]). Linear logic brings some light on this and shows how to construct subtler, non-trivial denotational models which can be used (via appropriate encodings) as denotational models for classical logic. The key idea is to start with a category with a weaker closure property than cartesian closure to interpret multiplicative linear logic (MLL) and use a Kleisli construction over the exponentials to obtain a cartesian closed category.

5.2.1 Multiplicative Additive Linear Logic

Our first aim is to get a denotational model for linear logic without exponentials. Spelled out in details, we want a non trivial category $\mathcal{C}$ where:

- formulas are interpreted by objects;
- a proof of $F_1 \vdash F_2$ is interpreted by a morphism from F_1 to F_2.

§ *Multiplicative Connectives.* The easiest part of linear logic is "multiplicative intuitionistic linear logic": MILL. A model for MILL is simply a symmetric monoidal closed category $(\mathcal{C}, \otimes, \multimap)$. By the rule for the tensor on the left, we can replace sequents $G_1 , \cdots , G_n \vdash F$ by sequents $G_1 \otimes \ldots \otimes G_n \vdash F$. With that in mind, all the interpretation can be done inductively: we just replace the syntactical symbols by their semantical counterpart. For proofs, we use the canonical morphisms and their obvious compositions.

What is worth noticing is that we can use *any* object C of $\mathcal{C}$ to interpret $\bot$: we interpret an empty left-hand side by $\mathbf{1}$ (the neutral element for $\otimes$) and an empty right-hand side by C. Then, everything works "out of the box".

§ *Additive Connectives.* To be able to interpret the additive connectives, one needs the additional property that $\mathcal{C}$ has finite products and coproducts. For obvious reasons, we write the product $\_ \,\&\, \_$ (with terminal element $\top$) and the coproduct $\_ \oplus \_$ (with initial element $\mathbf{0}$). Just like above, everything works: "out of the box".

§ *Dualizing Object.* Getting a model for classical linear logic is less trivial. The idea is simple in itself, but has many consequences: we want a special object $\bot$ which is *dualizing*. Let's recall the definition from page 82

> *a dualizing object in an SMCC* $(\mathcal{C}, \otimes, \multimap)$ *is an object* $\bot$ *such that, for every object* A*, the canonical morphism from* A *to* $(A \multimap \bot) \multimap \bot$ *is an isomorphism.*

With such a dualizing object, we can interpret classical multiplicative linear logic (and additive if we have finite products and coproducts). We define $X^\bot \triangleq F \multimap \bot$ and $X \parr Y \triangleq X^\bot \multimap Y$; it is easy to see that $\parr$ defines a commutative tensor product which is dual to $\otimes$:

$$\begin{array}{rcl} (X \otimes Y)^\bot & = & (X \otimes Y) \multimap \bot \\ & \simeq & X \multimap (Y \multimap \bot) \\ & \simeq & \big((X \multimap \bot) \multimap \bot\big) \multimap (Y \multimap \bot) \\ & = & X^\bot \parr Y^\bot \,. \end{array}$$

Showing all the other isomorphisms in a purely categorical setting is an (un)interesting exercise in abstract nonsense.

5.2.2 Lafont's Exponentials

The challenge lies in the interpretation of the exponentials. Before giving the abstract definition, let's look at some of the properties we want for the objects $!X$.

1) for any morphism $f : \mathcal{C}(X, Y)$, there should be a morphism $!f$ in $\mathcal{C}(!X, !Y)$ by promotion / dereliction;
2) for any object X, there should be a morphism in $\mathcal{C}(!X, !X \otimes !X)$ by contraction / axiom;
3) for any object X, there should be a morphism in $\mathcal{C}(!X, \mathbf{1})$ by weakening;
4) for any object X, there should be a morphism in $\mathcal{C}(!X, X)$ by dereliction / axiom;
5) for any object X, there should be a morphism in $\mathcal{C}(!X, !!X)$ by promotion / axiom.

It is not difficult to see that those simple rules allows to infer the more general dereliction, contraction and weakening rules with appropriate compositions.

We generalize those observations to a categorical setting by requiring:

- $!\_$ is a functor (point *1*);
- any $!X$ is equipped with a $\otimes$-comonoid structure (points *2* and *3*);
- $!\_$ is a comonad (points *4* and *5*).

In the classical case, we can dualize everything for $?\_$ and ask it to be a monad sending objects to $\parr$-monoids.

This only takes into account the purely algebraic properties of !_ and ?_. Surprisingly enough, the only real logical property needed to have an exponential construction coherent with the logic is the following isomorphism (natural in X and Y):

$$!(X \,\&\, Y) \quad \simeq \quad !X \otimes !Y \ .$$

This isomorphism will imply among others that the Kleisli category over the !_ comonad is cartesian closed. Defining the exponentials in full generality requires a lot of bureaucracy: one needs to pay attention to small and subtle details. We refer to the survey paper [63] and all the references given there.

Lafont's exponentials are obtained by taking !X to be the free $\otimes$-comonoid associated to X. The advantage of this approach is that most of the technical details hold automatically. The disadvantage is that it might neither be easy nor even possible to construct this free exponential. More complex axiomatizations for exponentials are possible. The basic idea is to split the exponential !X into two parts:

1) we send X to an object $E(X)$ in the category $\mathbf{CoMon}(\mathcal{C}, \otimes)$ of $\otimes$-comonoids on $\mathcal{C}$;
2) we then send $E(X)$ back into $\mathcal{C}$ by applying a functor $U : \mathbf{CoMon}(\mathcal{C}, \otimes) \to \mathcal{C}$.

We ask that U and E are adjoint, which implies that $U \cdot E$ is a comonad. Lafont's exponentials are the special case when U is the forgetful functor from $\mathbf{CoMon}(\mathcal{C}, \otimes)$ to $\mathcal{C}$.

5.3 The Relational Model

After this crash course on categorical models for linear logic, let's get back to a more "concrete" situation: we recall the simplest (??) categorical model for linear logic, the relational model.

The category **Rel** of sets and relations has already been introduced on page 71. Recall some trivial results:

- disjoint union gives both product and coproduct, $\emptyset$ is both initial and terminal;
- cartesian product is a tensor product, with neutral object $\{*\}$;
- the singleton set $\{*\}$ is a dualizing object;
- the tensor (cartesian product of sets) is self-adjoint.

Checking those properties is quite direct and we omit the proofs.

5.3.1 Intuitionistic Multiplicative Additive Linear Logic

As the previous remarks showed, the category **Rel** can be made into a denotational model of intuitionistic multiplicative additive linear logic. We will not detail the interpretation of intuitionistic proofs, since it can easily be extracted from the following section (relational interpretation of classical proofs). Let's only mention that a proof π of a sequent $G_1, \ldots, G_n \vdash F$ is interpreted by a relation between $|G_1 \otimes \cdots \otimes G_n|$ and $|F|$, *i.e.* between $|G_1| \times \cdots \times |G_n|$ and $|F|$ (where $|F|$ represent the relation interpretation of the formula F).

5.3.2 Classical Multiplicative Additive Linear Logic

Since the object $\{*\}$ is dualizing in **Rel**, we can extend the category of sets and relations to a denotational model for *classical* MALL. Decide first on a valuation ρ from propositional variables to sets and define the interpretation $|F|$ of a formula F to be, as expected:

$$\begin{array}{llllll} |X| & \triangleq \rho(X) & \text{and} & |X^\perp| & \triangleq \rho(X) \\ |\top| & \triangleq \emptyset & \text{and} & |\mathbf{0}| & \triangleq \emptyset \\ |\bot| & \triangleq \{*\} & \text{and} & |\mathbf{1}| & \triangleq \{*\} \\ |F_1 \oplus F_2| & \triangleq |F_1| + |F_2| & \text{and} & |F_1 \,\&\, F_2| & \triangleq |F_1| + |F_2| \\ |F_1 \parr F_2| & \triangleq |F_1| \times |F_2| & \text{and} & |F_1 \otimes F_2| & \triangleq |F_1| \times |F_2| \end{array}$$

This interpretation is a little boring since for any formula F, we have $|F| = |F^\perp|$.

The interpretation of proofs comes directly from the categorical structure of **Rel**, but it is interesting to spell it out in details. For any proof π of a sequent $\vdash G_1, \ldots, G_n$ (denoted by "$\pi \vdash G_1, \ldots, G_n$"), define its interpretation $[\![\pi]\!]$, a subset of $|G_1| \times \ldots \times |G_n|$, inductively in the following manner:

- Axiom and cut:
  - axiom: if $\dfrac{}{\pi \vdash X^\perp, X}$, then $[\![\pi]\!] \triangleq \mathbf{Eq}_{\rho(X)}$;
  - cut: if $\dfrac{\pi_1 \vdash \Gamma, F \qquad \pi_2 \vdash F^\perp, \Delta}{\pi \vdash \Gamma, \Delta}$,
    then $[\![\pi]\!] \triangleq \big\{(\gamma,\delta) \mid \big(\exists a \in |F|\big)\ (\gamma,a) \in [\![\pi_1]\!] \ \wedge\ (a,\delta) \in [\![\pi_2]\!]\big\} = [\![\pi_2]\!] \cdot [\![\pi_1]\!]$.
- Additive connectives:
  - constants: if $\dfrac{}{\pi \vdash \Gamma, \top}$, then $[\![\pi]\!] = \emptyset$;
  - "plus": if $\dfrac{\pi_1 \vdash \Gamma, F_1}{\pi \vdash \Gamma, F_1 \oplus F_2}$, then $[\![\pi]\!] = \big\{\big(\gamma, \mathrm{inl}(a)\big) \mid (\gamma,a) \in [\![\pi_1]\!]\big\}$
    and similarly for $\dfrac{\pi_2 \vdash \Gamma, F_2}{\pi \vdash \Gamma, F_1 \oplus F_2}$;
  - "with": if $\dfrac{\pi_1 \vdash \Gamma, F_1 \qquad \pi_2 \vdash \Gamma, F_2}{\pi \vdash \Gamma, F_1 \,\&\, F_2}$
    then $[\![\pi]\!] \triangleq \big\{\big(\gamma, \mathrm{inl}(a_1)\big) \mid (\gamma,a_1) \in [\![\pi_1]\!]\big\} \cup \big\{\big(\gamma, \mathrm{inr}(a_2)\big) \mid (\gamma,a_2) \in [\![\pi_2]\!]\big\}$.
- Multiplicative connectives:
  - constants: if $\dfrac{}{\pi \vdash \mathbf{1}}$ then $[\![\pi]\!] \triangleq \{*\}$;
    and if $\dfrac{\pi_1 \vdash \Gamma}{\pi \vdash \Gamma, \bot}$ then $[\![\pi]\!] \triangleq \{(\gamma,*) \mid \gamma \in [\![\pi_1]\!]\}$;
  - "tensor": if $\dfrac{\pi_1 \vdash \Gamma, F_1 \qquad \pi_2 \vdash \Delta, F_2}{\pi \vdash \Gamma, \Delta, F_1 \otimes F_2}$
    then $[\![\pi]\!] \triangleq \big\{\big(\gamma,\delta,(a_1,a_2)\big) \mid (\gamma,a_1) \in [\![\pi_1]\!] \ \wedge\ (\delta,a_2) \in [\![\pi_2]\!]\big\}$;

- "par": if $\dfrac{\pi_1 \vdash \Gamma, F_1, F_2}{\pi \vdash \Gamma, F_1 \parr F_2}$
  then $[\![\pi]\!] \triangleq \{(\gamma, (a_1, a_2)) \mid (\gamma, a_1, a_2) \in [\![\pi_1]\!]\}$.

Thus, given a valuation from $\mathcal{X}$ to sets, we interpret a proof of $\vdash \Gamma$ by a subset of $|\Gamma|$.

5.3.3 Exponentials

Interpreting the exponentials in the relational model amounts to looking for the free $\times$-monoid in **Rel**. It is not too difficult to see that this construction is given by *finite multisets, i.e.* finite tuples modulo reindexing:

▷ **Definition 5.3.1:** a *finite family* over S is a family $(s_i)_{i \in I}$ where the set I is finite. Two finite families $(s_i)_{i \in I}$ and $(t_j)_{j \in J}$ are *equivalent up to reindexing* if there is an isomorphism σ from I to J satisfying $s_i = t_{\sigma i}$ for all $i \in I$.

A *finite multiset over* S is an equivalence class of finite lists modulo reindexing. We write $[s_i]_{i \in I}$ for the equivalence class containing $(s_i)_{i \in I}$. The collection of finite multisets over S is denoted by $\mathcal{M}_f(S)$.

Sum of multisets is defined as concatenation (see footnote 11 on page 24) of the underlying families; it is written $+$.

The following is rather easy to check:

○ **Lemma 5.3.2:**
- $\mathcal{M}_f(\_)$ is both a monad and a comonad in **Rel**;
- for every set S, $\mathcal{M}_f(S)$ is the free $\times$-monoid over S;
- for every set S, $\mathcal{M}_f(S)$ is the free $\times$-comonoid over S;
- in **Rel**, we have the isomorphism (natural in X and Y)

$$\mathcal{M}_f(X + Y) \quad \simeq \quad \mathcal{M}_f(X) \times \mathcal{M}_f(Y) \ .$$

Interpreting the logical rules is now straightforward, everything is bound by the categorical structure of **Rel**, and no improvisation is possible:

- weakening: if $\dfrac{\pi_1 \vdash \Gamma}{\pi \vdash \Gamma, ?F}$ then $[\![\pi]\!] \triangleq \{(\gamma, []) \mid \gamma \in [\![\pi_1]\!]\}$;
- dereliction: if $\dfrac{\pi_1 \vdash \Gamma, F}{\pi \vdash \Gamma, ?F}$ then $[\![\pi]\!] \triangleq \{(\gamma, [a]) \mid (\gamma, a) \in [\![\pi_1]\!]\}$;
- contraction: if $\dfrac{\pi_1 \vdash \Gamma, ?F, ?F}{\pi \vdash \Gamma, ?F}$ then
  $[\![\pi]\!] \triangleq \{(\gamma, \mu_1 + \mu_2) \mid (\gamma, \mu_1, \mu_2) \in [\![\pi_1]\!]\}$;
- promotion: if $\dfrac{\pi_1 \vdash ?G_1, \ldots, ?G_n, F}{\pi \vdash ?G_1, \ldots, ?G_n, !F}$ then
  we define $(\mu_1, \ldots, \mu_n, [a_1, \ldots, a_k]) \in [\![\pi]\!]$ iff each μ_i is of the form $\mu_{i,1} + \ldots + \mu_{i,k}$ and each $(\mu_{1,j}, \ldots, \mu_{n,j}, a_j) \in [\![\pi_1]\!]$.

The case of intuitionistic linear logic is very similar, except for the fact that sequents are two-sided.

5.3.4 Cut Elimination

This model enjoys the additional property that the interpretation of proofs is invariant under cut-elimination: if π reduces to π' by cut-elimination, then $[\![\pi]\!] = [\![\pi']\!]$:

> ◇ **Proposition 5.3.3:** *suppose π is a proof of $G_1, \ldots, G_n \vdash F$, and suppose π' is obtained from π by applying one step of the cut-elimination procedure (see [39]), then $[\![\pi]\!] = [\![\pi']\!]$.*

proof: the direct proof is at the same time easy and quite long; but it follows from the fact that the category **Rel** is a categorical model for linear logic. ☑

6 A Refinement of the Relational Model

In **Rel**, the mathematical object interpreting a proof is simply a set (more precisely, a subset of the interpretation of a formula); this is not very informative. There are two degeneracies: the fact that interpretation is stable by negation ($|F| = |F^\perp|$) and that *any* subset of $|F|$ is a candidate for a proof of F. We are thus as far from completeness as we can be. There are several models "based" on the relational model which give extra structure to the sets interpreting formulas and for which the interpretation of proofs gives subsets satisfying various healthiness properties.

The way to make this intuition precise is to say that a categorical model $\mathcal{C}$ is a refinement of **Rel** if there is a faithful "forgetful" functor $|\_|$ from $\mathcal{C}$ to **Rel** which commutes with all the linear constructions, *i.e.*

- $|F^\perp| = |F|$;
- $|F_1 \,\&\, F_2| = |F_1| + |F_2|$;
- $|F_1 \parr F_2| = |F_1| \times |F_2|$;
- $|!F| = \mathcal{M}_f(|F|)$.

If we look only at multiplicative additive linear logic (MALL), the very first model of linear logic, *coherent spaces* can be seen as a refinement of **Rel**: formulas are interpreted by adding a structure of graph to the relational model. In particular, the linear negation is interpreted by taking the complement of the graph. This removes the first degeneracy. Then, it is possible to show that the **Rel**-interpretation of a MALL proof always gives a *clique:* a subset of vertices pairwise connected (complete subgraph).

> ⚲ Remark 19: the exponentials from coherent spaces are not built on **Rel**: the *web* (set of vertices) of !G is not the collection of finite multisets over the web of G but the collection of finite cliques of G. We thus need the graph structure of G to construct the web of !G. It is possible to define a "non-uniform" variant of coherent spaces which uses **Rel** as a basis. This is what is done in [16], for the model of *hypercoherences*.

Another refinement of **Rel** of interest is given by finiteness spaces ([31]). There, an object is a set X together with a notion of "finiteness": a collection of subsets of X which can be considered "finite". Once again, linear negation is non-trivial and proofs are interpreted by finitary subsets.

We now show that **Int** is (not surprisingly after section 3.4 and 3.5) a categorical model of linear logic, and that it is a refinement of **Rel**.

6.1 Exponential

We have already seen that **Int** is symmetric monoidal closed (proposition 3.4.2) and that it has product and coproduct (lemmas 3.2.6 and 3.2.5). We can thus interpret intuitionistic multiplicative additive linear logic in the category **Int**. We now turn our attention to the exponential.

6.1.1 Multithreading

The connective $!\_$ is given, in the relational model, by taking *finite multisets*. We thus need an operation on interaction systems, taking w on S to $!w$ on $\mathcal{M}_f(S)$. The computational interpretation of this operation is linked with the notion of *multithreading*, *i.e.* the idea of executing several instances of a program in parallel. Since we are dealing with a synchronous tensor, it is not surprising that we get a notion of *synchronous multithreading*.

▷ **Definition 6.1.1**: if $w = (A, D, n)$ is an interaction system on S, define the interaction system $!w = (!A, !D, !n)$ on $\mathcal{M}_f(S)$ as follows:

$$!A(\mu) \quad \triangleq \quad \big(\Sigma(s_1, \ldots, s_n) \in \mu\big)\ A(s_1) \times \cdots \times A(s_n)$$

i.e. an action in state $\mu = [s_1, \ldots, s_n]$ (finite multiset of states) is given by an ordering $(s_{\sigma 1}, \ldots, s_{\sigma n})$ of μ, together with a parallel action $(a_1, \ldots, a_n)$;

$$!D\big(\_, \big((s_1, \ldots), (a_1, \ldots, a_n)\big)\big) \quad \triangleq \quad D(s_1, a_1) \times \cdots \times D(s_n, a_n)$$

i.e. a reaction to such an action is given by a parallel reaction for the a_i;

$$!n\big(\_, \big((s_1, \ldots), (a_1, \ldots)\big), (d_1, \ldots, d_n)\big) \quad \triangleq \quad \big[s_1[a_1/d_1], \ldots, s_n[a_n/d_n]\big]$$

i.e. the new state is simply the tuple of new states, quotiented by renaming. The interaction system $!w$ is called "*of course w!*", and the operation is called "*synchronous multithreading*".

This operation can be obtained in two steps by first taking $Ł(w)$ defined on $\textsc{List}(S)$ as follows:

$$\begin{array}{lcl} Ł(w).A\big((s_1, \ldots, s_n)\big) & \triangleq & A(s_1) \times \cdots \times A(s_n) \\ Ł(w).D\big((s_1, \ldots), (a_1, \ldots, a_n)\big) & \triangleq & D(s_1, a_1) \times \cdots \times D(s_n, a_n) \\ Ł(w).\big((s_1, \ldots), (a_1, \ldots), (d_1, \ldots, d_n)\big) & \triangleq & \big(s_1[a_1/d_1], \ldots, s_n[a_n/d_n]\big) \end{array}$$

and then noticing that this interaction system is compatible with the action of permutations: if σ is a permutation in $\mathfrak{S}_n$, then we have:

$$\begin{array}{l} \sigma \cdot Ł(w).n\big((s_1, \ldots, s_n), (a_1, \ldots, a_n), (d_1, \ldots, d_n)\big) \\ \qquad = \\ Ł(w).n\big(\sigma \cdot (s_1, \ldots, s_n), \sigma \cdot (a_1, \ldots, a_n), \sigma \cdot (d_1, \ldots, d_n)\big)\ . \end{array}$$

This allows to see $Ł(w)$ as acting on equivalence classes of lists, modulo reindexing.

It is quite obvious that $Ł(w)$ can be seen as a multithreaded version of w. It is just an arbitrary juxtaposition of several instances of w using the synchronous product:

$$Ł(w) \quad = \quad \bigoplus_{n \in \mathbf{N}} w^{n\otimes}$$

where $w^{n\otimes}$ is an abbreviation for $w \otimes \cdots \otimes w$. The actual $!w$ is a little subtler. For aesthetical reasons, we may write it as:

$$!w \quad = \quad \bigoplus_{n \in \mathbf{N}} \frac{w^{n\otimes}}{\mathfrak{S}_n}$$

which is reminiscent of the Taylor expansion of e^w (recall that the order of $\mathfrak{S}_n$ is $n!$). (This is quite informal but carries the appropriate intuition.)

That this operation is functorial is easy:

- **Lemma 6.1.2:** both $!\_$ and $Ł(\_)$ can be extended to endofunctors.

Recall that the action of $!\_$ is defined on morphisms as:

$$\big([s_1, \ldots, s_n], [s'_1, \ldots, s'_n]\big) \in !r \quad \text{iff} \quad (\exists \sigma \in \mathfrak{S}_n)\ (\forall i = 1, \ldots, n)\ (s_i, s'_{\sigma i}) \in r\ .$$

Notice also that there is an obvious bisimulation

$$Ł(w) \quad \underset{p}{\overset{\sigma}{\rightleftarrows}} \quad !w \tag{6-1}$$

where σ is the "$\in$" relation between a tuple and its equivalence class and p its converse. We have $\sigma \cdot p = \mathbf{Id}$, making this a retract.

6.1.2 Comonoid Structure

Each $!w$ is canonically equipped with a commutative $\otimes$-comonoid structure:

$$\begin{array}{lllll} e & \in & \mathbf{Int}(!w, \mathbf{1}) & \text{and} & m \in \mathbf{Int}(!w, !w \otimes !w) \\ e & \triangleq & \{([\,], *)\} & & m \triangleq \big\{\big(\mu + \nu, (\mu, \nu)\big) \mid \mu, \nu \in \mathcal{M}_f(S)\big\}\ . \end{array}$$

Checking that those relations e and m are indeed simulations is direct. That they satisfy the appropriate commutative diagrams follows from the fact that they do so in **Rel**. With this in mind, it is not too difficult to show that $!w$ is the free-comonoid for $\otimes$:

- **Lemma 6.1.3:** if we view $!\_$ as a functor from **Int** to $\mathbf{CoMon}(\mathbf{Int}, \otimes)$, then $!\_$ is right-adjoint to the forgetful "underlying object" functor from $\mathbf{CoMon}(\mathbf{Int}, \otimes)$ to **Int**.

proof: we need to show that there is a natural isomorphism

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

Going from left to right is easy:

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

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 $e_c \in \mathbf{Int}(w_c, \mathbf{1})$ 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 $!w$ in the following way:

- we start by extending comultiplication m_c to $\overline{m}_c : \mathbf{Int}\big(w_c, \text{Ł}(w_c)\big)$;
- we then compose that with $\text{Ł}(r) : \mathbf{Int}\big(\text{Ł}(w_c), \text{Ł}(w)\big)$;
- and finally compose that with $\sigma : \mathbf{Int}\big(\text{Ł}(w), !w\big)$, see (6-1).

We then check that this simulation respects the comonoid structures of w_c and $!w$. Define $\overline{m}_c \subseteq S_c \times \textsc{List}(S_c)$ by the following clauses:

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

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 $\text{Ł}(w_c)$. We have:

$$\begin{array}{l} \big(s_c, (s_{c,1}, \ldots, s_{c,n+m})\big) \in \overline{m} \\ \quad \Leftrightarrow \\ (\exists s_c^1, s_c^2 \in S_c)\ \big(s_c, (s_c^1, s_c^2)\big) \in m_c \wedge \big(s_c^1, (s_{c,1}, \ldots, s_{c,n})\big) \in \overline{m}_c \\ \qquad\qquad\qquad\qquad \wedge \big(s_c^2, (s_{c,n+1}, \ldots, s_{c,n+m})\big) \in \overline{m}_c \end{array} \tag{6-2}$$

by transitivity and

$$\begin{array}{l} \big(s_c, (s_{c,1}, \ldots, s_{c,i}, s_{c,i+1}, \ldots, s_{c,n})\big) \in \overline{m} \\ \quad \Leftrightarrow \\ \big(s_c, (s_{c,1}, \ldots, s_{c,i+1}, s_{c,i}, \ldots, s_{c,n})\big) \in \overline{m} \end{array} \tag{6-3}$$

by commutativity. (Both proofs are done by induction.)

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

$$\begin{array}{ccc} w_c & \xrightarrow{\widetilde{r}} & !w \\ & \searrow^{e_c} & \downarrow e \\ & & \mathbf{1} \end{array} \qquad \text{and} \qquad \begin{array}{ccc} w_c & \xrightarrow{c} & w_c \otimes w_c \\ \downarrow \widetilde{r} & & \downarrow \widetilde{r} \otimes \widetilde{r} \\ !w & \xrightarrow[m]{} & !w \otimes !w \end{array}$$

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

- $(s_{c,i}, s_i) \in r$ for all $i = 1, \ldots, n+m$
- and $\big(s_c, (s_{c,1}, \ldots, s_{c,n+m})\big) \in \widetilde{m}_c$.

That $\big(s_c, [s_1, \ldots, s_n], [s_{n+1}, \ldots, s_{n+m}]\big)$ 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 $(s_c^1, [s_1, \ldots, s_n]) \in \widetilde{r}$ and $(s_c^2, [s_{n+1}, \ldots, s_{n+m}]) \in \widetilde{r}$,

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

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

By using (6-2) and (6-3), it is trivial to show that the two conditions are in fact equivalent. This proves that the second diagram is commutative.

Since this is the same construction as in **Rel**, we can directly deduce that the operations just defined are inverse of each other.

☑

6.1.3 A Comonad

A direct consequence of lemma 6.1.3 is:

- **Lemma 6.1.4:** the functor "!_" is a comonad on **Int**.

Let's look at the actual structure of this comonad:

- the unit of this comonad $\varepsilon : !\_ \to \_$ is given by:

$$\varepsilon_w \quad = \quad \{([s], s) \mid s \in S\}$$

  which is obtained by taking the image of the identity along the natural bijection $\mathbf{CoMon}(\mathbf{Int}, \otimes)(!w, !w) \xrightarrow{\sim} \mathbf{Int}(!w, w)$;
- and the comultiplication $\delta : !\_ \to !!\_$ is given by

$$\delta_w \quad = \quad \left\{ \left(\sum_{i \in I} \mu_i, [\mu_i]_{i \in I} \right) \mid (\forall i \in I)\ \mu_i \in \mathcal{M}_f(S) \right\}$$

  which is obtained by taking the image of the identity along the natural bijection $\mathbf{Int}(!w, !w) \xrightarrow{\sim} \mathbf{CoMon}(\mathbf{Int}, \otimes)(!w, !!w)$, and then along the forgetful functor $\mathcal{U} : \mathbf{CoMon}(\mathbf{Int}, \otimes) \to \mathbf{Int}$ (whose action on morphism is the identity).

That those operations satisfy the appropriate commutativity properties follows from the fact that they do so in **Rel**. That ε_w is a simulation in $\mathbf{Int}(!w, w)$ is obvious, and that δ_w is a simulation in $\mathbf{Int}(!w, !!w)$ is not difficult.

Moreover, this operation satisfies the canonical isomorphism of linear logic:

- **Lemma 6.1.5:** for any interaction systems w_1 and w_2, we have a natural isomorphism $!(w_1 \,\&\, w_2) \approx !w_1 \otimes !w_2$.

proof: the direct proof that equality is a bisimulation is straightforward.

☑

6.2 Intuitionistic Linear Logic

6.2.1 Interpretation of Formulas

▷ **Definition 6.2.1:** a *valuation* ρ is a pair of maps $(|\rho|, \rho)$ where $|\rho|$ assigns to every propositional variable X a set $|\rho|(X)$ and ρ assigns to any propositional variable X an interaction system $\rho(X)$ on the set $|\rho|(X)$.

Fix, once and for all, a valuation ρ; this allows to define the interpretation of linear formulas as an interaction system in the following way:

▷ **Definition 6.2.2:** let φ be a linear formula; we define φ^*, the interpretation of φ by induction:

$$\begin{array}{llllll}
\mathbf{0}^* & \triangleq & (\emptyset, \mathbf{null}) & \text{and} \quad \top^* & \triangleq & (\emptyset, \mathbf{null}) \\
\mathbf{1}^* & \triangleq & (\{*\}, \mathbf{skip}) & \text{and} \quad \bot^* & \triangleq & (\{*\}, \mathbf{skip}) \\
X^* & \triangleq & \big(|\rho|(X), \rho(X)\big) & & & \\
(\varphi^\perp)^* & \triangleq & (\varphi^*)^\perp & & & \\
(\varphi_1 \oplus \varphi_2)^* & \triangleq & \varphi_1^* \oplus \varphi_2^* & \text{and} \quad (\varphi_1 \,\&\, \varphi_2)^* & \triangleq & \varphi_1^* \,\&\, \varphi_2^* \\
(\varphi_1 \otimes \varphi_2)^* & \triangleq & \varphi_1^* \otimes \varphi_2^* & \text{and} \quad (\varphi_1 \multimap \varphi_2)^* & \triangleq & \varphi_1^* \multimap \varphi_2^* \\
(!\varphi)^* & \triangleq & !(\varphi^*)\ . & & &
\end{array}$$

We usually denote φ^* by $(|\varphi|, \varphi)$, or even φ. The context is enough to remove possible confusion.

Thus, φ^* is an interaction system on the interpretation of φ in the relational model (with valuation $|\rho|$).

6.2.2 Interpretation of Proofs

The relational interpretation $[\![\pi]\!]$ of a proof π of a sequent $\Gamma \vdash \varphi$ is a relation between $|\Gamma|$ and $|\varphi|$. What is surprising, is that this relation satisfies more than that: even though $[\![\pi]\!]$ doesn't depend on the interaction systems interpreting the atoms, we have:

◇ **Proposition 6.2.3:** *if π is an intuitionistic proof of $\Gamma \vdash \varphi$, then the relational interpretation $[\![\pi]\!]$ is a simulation from $\bigotimes \Gamma^*$ to φ^*.*

proof: it follows from the categorical structure of **Int**. ☑

Moreover, since the interpretation is "just" the relational one, we also have (see proposition 5.3.3):

◇ **Proposition 6.2.4:** *this denotational model is invariant under cut-elimination.*

6.3 Classical Linear Logic

Since the category **Int** is $\star$-autonomous, it is not surprising that proposition 6.2.3 extends to classical linear logic. The computational meaning of the connective $\parr$ is however slightly subtler than $\otimes$.

6.3.1 The New Connectives

Classically, the multiplicative connective $\multimap$ is replaced by the connective $\parr$. This is the de Morgan dual of the tensor:

▷ **Definition 6.3.1:** if w_1 and w_2 are interaction systems, define $w_1 \parr w_2$ as:

$$w_1 \parr w_2 \quad \triangleq \quad \left(w_1^\perp \otimes w_2^\perp\right)^\perp .$$

We call $\parr$ the "par", or the *split synchronous tensor*.

Before giving some intuition about this new interaction system, let's check that the connective $\multimap$ is redundant in a classical setting:

○ **Lemma 6.3.2:** for any interaction systems w_1 and w_2, we have

$$w_1 \multimap w_2 \quad \simeq \quad (w_1 \otimes w_2^\perp)^\perp \quad \simeq \quad w_1^\perp \parr w_2 .$$

proof: we'll only show quickly the first isomorphism. The second follows from the definition of $\parr$ and involutivity of $\_^\perp$.

$$\begin{array}{lcl} (w_1 \otimes w_2^\perp)^\perp & \approx & (w_1 \otimes w_2^\perp) \multimap \perp \\ & \simeq & w_1 \multimap (w_2^\perp \multimap \perp) \\ & \approx & w_1 \multimap (w_2^{\perp\perp}) \\ & \simeq & w_1 \multimap w_2 . \end{array}$$

☑

Unfolding naively the definition of $\parr$ gives an unreadable interaction system:

$$A\big((s_1,s_2)\big) \quad = \quad \times \begin{array}{l} \left(\begin{array}{l} \big(a_1 \epsilon A_1(s_1)\big) \to D_1(s_1,a_1) \\ \times \big(a_2 \epsilon A_2(s_2)\big) \to D_2(s_2,a_2) \end{array} \right) \to A_1(s_1) \\ \left(\begin{array}{l} \big(a_1 \epsilon A_1(s_1)\big) \to D_1(s_1,a_1) \\ \times \big(a_2 \epsilon A_2(s_2)\big) \to D_2(s_2,a_2) \end{array} \right) \to A_2(s_2) \end{array}$$

$$D\big((s_1,s_2),(F_1,F_2)\big) \quad = \quad \begin{array}{r} \big(a_1 \epsilon A_1(s_1)\big) \to D_1(s_1,a_1) \\ \times \big(a_2 \epsilon A_2(s_2)\big) \to D_2(s_2,a_2) \end{array}$$

and

$$n\big((s_1,s_2),(F_1,F_2),(f_1,f_2)\big) \quad = \quad \Big(s_1\big[F_1(f_1,f_2)/f_1 \cdot F_1(f_1,f_2)\big] \ , \ s_2\big[F_2(f_1,f_2)/f_2 \cdot F_2(f_1,f_2)\big]\Big) .$$

It is however possible to get an intuition about this interaction system: let's take the point of view of the Demon. As definition 2.5.3 and lemma 2.5.4 show, this is

achieved by looking at the dual interaction system. In our case, since $\_^{\perp\perp} \simeq \mathbf{Id}$, it amounts to looking at the interaction system $(w_1 \parr w_2)^\perp \simeq w_1^\perp \otimes w_2^\perp$: we have, in simplified form

$$\begin{array}{lcl} A\big((s_1,s_2)\big) & = & (A_1 \to D_1) \times (A_2 \to D_2) \\ D\big((s_1,s_2),(f_1,f_2)\big) & = & A_1 \times A_2 \\ n\big((s_1,s_2),(f_1,f_2),(a_1,a_2)\big) & = & \big(s_1[a_1/f_1(a_1)]\ ,\ s_2[a_2/f_2(a_2)]\big) \end{array}$$

i.e. a Demon's strategy in $w_1 \parr w_2$, or equivalently an Angel move in $(w_1 \parr w_2)^\perp$ is given by a pair of strategies: one in w_1 and one in w_2. If we compare that with the usual synchronous tensor, seen from the Demon's perspective ($(w_1 \otimes w_2)^\perp$):

$$\begin{array}{lcl} A\big((s_1,s_2)\big) & = & (A_1 \times A_2 \to D_1) \times (A_1 \times A_2 \to D_2) \\ D\big((s_1,s_2),(f_1,f_2)\big) & = & A_1 \times A_2 \\ n\big((s_1,s_2),(f_1,f_2),(a_1,a_2)\big) & = & \big(s_1[a_1/f_1(a_1,a_2)]\ ,\ s_2[a_2/f_2(a_1,a_2)]\big) \end{array}$$

we see that in the latter, the Demon's strategies take as arguments *the two actions in w_1 and w_2*. This means that in a $\parr$, the Demon needs to make his choice of reaction in w_i *independently* of the action played by the Angel on w_j (with $i \neq j$), whereas in a $\otimes$ his choice of reaction may depend on both actions played by the Angel.

Both "$\otimes$" and "$\parr$" are thus operations of synchronous parallel composition. The difference between them is: we put two pairs Angel/Demon in parallel and,

- in a $\otimes$, the Angels share a single channel of communication; both Demons receive the actions from the two Angels and can make their choice of reaction accordingly;
- in a $\parr$, each Angel has her own channel of communication; the Demons must react on their channel independently of the other channel.

In both cases, states are updated synchronously. Another metaphor is to think that in a $\otimes$, there are two non-communicating Angels against a single Demon, while in the $\parr$, there is a single Angel against two non-communicating Demons.

It is possible to do exactly the same thing for synchronous multithreading:

▷ **Definition 6.3.3:** if w is an interaction system on S, we define $?w$ to be:

$$?w \quad = \quad \big(!(w^\perp)\big)^\perp .$$

The interaction system $?w$ is called "why not w?", and the operation is called "*split synchronous multithreading*".

If $!w$ is viewed as:

- several Angels send actions on a channel of communication;
- a single Demon responds to all of them;
- all the states are updated;

then $?w$ can be viewed as:

- a single Angel sends several actions on separate channels;
- on each channel, an independent Demon respond to his action;
- all the states are updated.

6.3.2 The Model

Since classical logic is symmetric, we can use single sided sequents $\vdash G_1, \ldots, G_n$. Just like in the previous section, the interpretation of proofs is the relational one (section 5.3). We start by interpreting formulas in the most obvious way: see definition 6.2.2, but use the multiplicative connective $\parr$ rather than $\multimap$. Since all the canonical isomorphisms do hold in the category **Int**, the model is not sensitive on the way formulas are constructed: using two or one sided sequents, having linear negation as a primitive operation or as a defined one, etc. The categorical structure of **Int** guarantees that the interpretation is correct in the following sense:

◇ **Proposition 6.3.4:** *if π is a classical proof of $\vdash G_1, \ldots, G_n$ then the relational interpretation $[\![\pi]\!]$ is an invariant property for $G_1^* \parr \cdots \parr G_n^*$: for any valuation we have*

$$[\![\pi]\!] \quad \subseteq \quad \left(G_1^* \parr \cdots \parr G_n^*\right)^\circ([\![\pi]\!]) \ .$$

Moreover, this model is invariant under cut elimination.

6.3.3 Adding a Non-Commutative Connective

Non-commutative linear logic is a refinement of linear logic where we also take into account the reindexing of formulas in a sequent. We just mention the existence of a non-commutative, multiplicative self-dual connective, similar to Christian Retoré's connective ([71] and [72]).

This connective, written "$\blacktriangleright$", lies somewhere between "$\otimes$" and "$\parr$":

- in "$w_1 \otimes w_2$", both Demons see the two Angels' actions;
- in "$w_1 \parr w_2$", each Demon sees one Angel's action;
- in "$w_1 \blacktriangleright w_2$", the Demon from w_2 sees both actions, but the Demon from w_1 only sees one action.

We take the Demon's point of view and look at actions in $(w_1 \otimes w_2)^\perp$ and $(w_1 \parr w_2)^\perp$, *i.e.* at the Demon's strategies in $w_1 \otimes w_2$ and $w_1 \parr w_2$:

$$\begin{array}{lll} \otimes_\perp : & (A_1 \times A_2) \to D_1 & \times \ (A_1 \times A_2) \to D_2 \\ \parr_\perp : & A_1 \to D_1 & \times \ A_2 \to D_2 \ . \end{array}$$

It is thus natural to put:

$$\blacktriangleright_\perp : \quad (A_1 \to D_1) \qquad \times \ (A_1 \times A_2) \to D_2$$

i.e. the Demon from w_2 can chose his reaction with the knowledge of the Angel's actions in w_1 and w_2. On the other hand, the Demon from w_1 only sees the action from the Angel in w_1.

▷ **Definition 6.3.5:** if w_1 and w_2 are interfaces, define $w_1 \blacktriangleright_\perp w_2$ on $S_1 \times S_2$ as:

$$A_{\blacktriangleright_\perp}\left((s_1, s_2)\right) \quad \triangleq \quad \begin{array}{l} \left((a_1 \epsilon A_1(s_1)) \to D_1(s_1, a_1)\right) \\ \times \\ \left(a_1 \epsilon A_1(s_1)\right) \to \left(a_2 \epsilon A_2(s_2)\right) \to D_2(s_2, a_2) \end{array}$$

and

$$D_{\blacktriangleright_\perp}\big((s_1,s_2),(f_1,f_2)\big) \quad \triangleq \quad A_1(s_1)\times A_2(s_2)$$

and

$$n_{\blacktriangleright_\perp}\big((s_1,s_2),(f_1,f_2),(a_1,a_2)\big) \quad \triangleq \quad \big(s_1[a_1/f_1(a_1)],s_2[a_2/f_2(a_1,a_2)]\big)\ .$$

The interface $w_1 \blacktriangleright w_2$ is defined as $(w_1 \blacktriangleright_\perp w_2)^\perp$.

We have:

- **Lemma 6.3.6:** the connective $\blacktriangleright$ is self-dual:

$$(w_1 \blacktriangleright w_2)^\perp \quad \simeq \quad w_1^\perp \blacktriangleright w_2^\perp\ .$$

proof: the complete formal proof is not very interesting and involves a lot of shuffling of quantifiers using **AC** and **CtrAC**. To show the isomorphism, it suffices to show that:

$$\begin{array}{lll}(s_1,s_2)\ \epsilon\ (w_1 \blacktriangleright w_2)^{\perp\circ}(r) & \Leftrightarrow & \big(\forall a_1\ \epsilon\ A_1(s_1)\big)\\ & & \big(\exists d_1\ \epsilon\ D_1(s_1,a_1)\big)\\ & & \big(\forall a_2\ \epsilon\ A_2(s_2)\big)\\ & & \big(\exists d_2\ \epsilon\ D_2(s_2,a_2)\big)\\ & & \quad (s_1[a_1/d_1],s_2[a_2/d_2])\ \epsilon\ r\\ & \Leftrightarrow & (s_1,s_2)\ \epsilon\ (w_1^\perp \blacktriangleright w_2^\perp)^\circ(r)\end{array}$$

for all $s_1\ \epsilon\ S_1$, $s_2\ \epsilon\ S_2$ and $r \subseteq S_1 \times S_2$.

☑

Note that the predicate transformer $(w_1 \blacktriangleright w_2)^\circ$ is reminiscent of a kind of sequential composition "w_1 followed by w_2":

$$\begin{array}{lll}(s_1,s_2)\ \epsilon\ (w_1 \blacktriangleright w_2)^\circ(r) & \Leftrightarrow & \big(\exists a_1\ \epsilon\ A_1(s_1)\big)\big(\forall d_1\ \epsilon\ D_1(s_1,a_1)\big)\\ & & \big(\exists a_2\ \epsilon\ A_2(s_2)\big)\big(\forall d_2\ \epsilon\ D_2(s_2,a_2)\big)\\ & & \quad \big(s_1[a_1/d_1],s_2[a_2/d_2]\big)\ \varepsilon\ r\ .\end{array}$$

6.4 Interpreting the Differential Lambda-calculus

We now give the details for interpreting the simply typed λ-calculus. This will also allow to find a natural semantics counterpart to the fact that simulations are closed under unions by showing that we can also interpret the *differential λ-calculus* of Thomas Ehrhard and Laurent Régnier ([32]).

6.4.1 Syntax

We start by giving a short introduction to the simply typed differential λ-calculus. The grammar generating (untyped) terms is given by:

$$\begin{array}{lll}t,u & ::= & x \ \mid\ (t)u \ \mid\ (\lambda x).t\\ & \mid & 0 \ \mid\ t+u \ \mid\ D\,t\cdot u\ .\end{array}$$

where we use Krivine's convention and write "$(t)u$" for the application of term t to u. The definition of β reduction is the usual one:

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

where $t[u/x]$ is extended in the obvious way:

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

Together with this reduction, we also have a notion of *differential reduction*. The definition is driven by the following intuition: the term $D\,t \cdot u$ represents the term t where *exactly one* occurrence of the first λ-bound variable has been replaced by u. Since there may be many occurrences of this first variable, we take the *sum* of all the possible terms resulting from replacing a single occurrence:

$$D(\lambda x.t)\ \cdot\ u \quad \rightsquigarrow_D \quad \lambda x\ .\ \left(\frac{\partial t}{\partial x} \cdot u\right)$$

where $\partial t/\partial x \cdot u$ represent the *linear substitution* of x by u in t:

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

§ *Equations.* Differential λ-terms are then quotiented by many equations, giving a real "differential" flavor to the theory. Dealing with those quotient in the syntax itself can be cumbersome, but since we are dealing with semantics, we do not bother with the details. The first equations deal with linearity conditions. We put:

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

The most important equation is probably the last one:

$$D(D\,t \cdot u) \cdot v \quad = \quad D(D\,t \cdot v) \cdot u\ .$$

§ *Typing.* We can extend the typing discipline for the λ-calculus to this new context with the following typing rules:

1) $\dfrac{}{\Gamma \vdash x : \tau}$ if $x : \tau$ appears in Γ;

2) $\dfrac{\Gamma \vdash t : \tau \to \tau' \qquad \Gamma \vdash u : \tau}{\Gamma \vdash (t)u : \tau'}$;

3) $\dfrac{\Gamma, x : \tau \vdash t : \tau'}{\Gamma \vdash \lambda x.t : \tau \to \tau'}$;

4) $\dfrac{}{\Gamma \vdash 0 : \tau}$;

5) $\dfrac{\Gamma \vdash t_1 : \tau \qquad \Gamma \vdash t_2 : \tau}{\Gamma \vdash t_1 + t_2 : \tau}$;

6) $\dfrac{\Gamma \vdash t : \tau \to \tau' \qquad \Gamma \vdash u : \tau}{\Gamma \vdash \mathrm{D}\, t \cdot u : \tau \to \tau'}$.

The only typing rules deserving some comment is rule *6*: if t is of type $\tau \to \tau'$, then $\mathrm{D}\, t \cdot u$ is still of type $\tau \to \tau'$. The reason is simply that there may still be free occurrences of the first abstracted variable in $\mathrm{D}\, t \cdot u$. This is also consistent with the differential calculus intuition, where we can see $\mathrm{D}t$ as the function(al) giving, for each point, the best linear approximation of t around it. Informally we have:

$$\mathrm{D}t \quad : \quad \tau \to (\tau \multimap \tau')$$

i.e. for any point x, $\mathrm{D}t(x)$ is a *linear* function approximating t around x. We can swap the two arguments and obtain

$$\widetilde{\mathrm{D}t} \quad : \quad \tau \multimap (\tau \to \tau') \ .$$

Our "$\mathrm{D}\, t \cdot u$" can be though of as the application $\widetilde{\mathrm{D}t}$ to u.

This calculus enjoys many interesting properties, among which we find Church Rosser and strong normalization. We refer to [32] and [33].

6.4.2 The Model

§ *The Simply Typed λ-calculus.* We start by recalling the standard way to encode the simply type λ-calculus into intuitionistic multiplicative exponential linear logic:

- an atomic type ι is interpreted by an atomic linear formula ι^*;
- the type $\tau \to \tau'$ is interpreted by $!(\tau^*) \multimap \tau'^*$;
- a context $x_1 : \tau_1, \ldots, x_n : \tau_n$ is interpreted by the context $x_1 : !\tau_1^*, \ldots, x_n : !\tau_n^*$.

The typing rules are translated as follows:

- $\dfrac{}{\Gamma \vdash x : \tau}$ where $x : \tau$ appears in Γ, is replaced by an appropriate sequence of weakening(s), a dereliction and an axiom;
- $\dfrac{\Gamma \vdash t : \tau \to \tau' \qquad \Gamma \vdash u : \tau}{\Gamma \vdash (t)u : \tau'}$ is replaced by a modus-ponens followed by a generalized contraction on the context;

- λ-abstraction is replaced by an instance of the $\multimap$-right rule.

To interpret those rules, we start by fixing a valuation ρ going from atomic types to interaction systems and interpret higher types as:

$$\begin{array}{lcll} \iota^* & \triangleq & \rho(\iota) & \text{if } \iota \text{ is atomic} \\ (\tau \to \tau')^* & \triangleq & !(\tau^*) \multimap \tau'^* & . \end{array}$$

In particular, the sets of states corresponding to types are given by:

$$\begin{array}{lcll} |\iota| & \triangleq & |\rho(\iota)| & \text{if } \iota \text{ is atomic} \\ |\tau \to \tau'| & \triangleq & \big(\mathcal{M}_f|\tau|\big) \times |\tau'| & . \end{array}$$

For the interpretation of typed terms, we work by induction on the typing inference: if $x_1 : \tau_1, \dots, x_n : \tau_n \vdash t : \tau$, then $[\![t]\!]$ will be a relation between $\mathcal{M}_f|\tau_1| \times \cdots \times \mathcal{M}_f|\tau_n|$ and $|\tau|$. Equivalently, $[\![t]\!]$ is a function from $\mathcal{M}_f|\tau_1| \times \cdots \times \mathcal{M}_f|\tau_n|$ to $\mathcal{P}|\tau|$. We sometimes write $\gamma = (\mu_1, \dots, \mu_n) \in \mathcal{M}_f|\tau_1| \times \cdots \times \mathcal{M}_f|\tau_n|$ as "$x_1 = \mu_1, \dots, x_n = \mu_n$" and use $\gamma(x)$ for the projection on the appropriate coordinate.

- for the axiom $\dfrac{}{\Gamma \vdash x : \tau}$ where $x : \tau$ appears in Γ,
  then $[\![x]\!]_\gamma \triangleq \begin{cases} \{s\} & \text{if } \gamma(x) = [s] \text{ and } \gamma(y) = [] \text{ whenever } y \neq x \\ \emptyset & \text{otherwise ;} \end{cases}$
- for an application $\dfrac{\Gamma \vdash t : \tau \to \tau' \qquad \Gamma \vdash u : \tau}{\Gamma \vdash (t)u : \tau'}$,
  we put $s \in [\![(t)u]\!]_\gamma$ iff $(\mu, s) \in [\![t]\!]_{\gamma_0}$ for some $\mu = [s_1, \dots, s_n] \in \mathcal{M}_f|\tau|$ s.t.
  - $s_i \in [\![u]\!]_{\gamma_i}$ for all $i = 1, \dots, n$,
  - and $\gamma = \gamma_0 + \gamma_1 + \cdots + \gamma_n$;
- for an abstraction $\dfrac{\Gamma, x : \tau \vdash t : \tau'}{\Gamma \vdash \lambda x.t : \tau \to \tau'}$,
  we put $(\mu, s) \in [\![\lambda x.t]\!]_\gamma$ iff $s \in [\![t]\!]_{\gamma, x=\mu}$.

Two things are worth noticing: first we do not use the interaction system structure of the atoms, but just the sets of states (we are still in the relational model); second, the definition is really by induction on the *structure of the term* rather than on the typing inference. That the term is typed is thus mostly irrelevant. Using a colimit construction, it is possible to construct a reflexive object in the category **Int** to devise a model for the untyped λ-calculus in the spirit of Engeler's model.

Since this is the interpretation of the proofs corresponding to the typing judgments, we obtain, as a direct corollary to proposition 6.2.3:

> ◇ **Proposition 6.4.1:** *suppose* $x_1 : \tau_1, \dots, x_n : \tau_n \vdash t : \tau$ *is a correct typing judgment, then, for any valuation* ρ *for the atomic types, we have that* $\_ \in [\![t]\!]\_$ *is a simulation from* $!\tau_1^* \otimes \cdots \otimes !\tau_n^*$ *to* τ^*.

In other words, $s \in [\![t]\!]_\gamma$ implies that s (in τ) simulates γ (in $!\tau_1^* \otimes \cdots \otimes !\tau_n$). The direct proof can be found in [55].

↬ REMARK 20: note that since the typed λ-calculus can be translated into *intuitionistic* MELL, the proof of this proposition is entirely constructive. However, this result is not as such predicative: the exponential is not a predicative operation since it uses equivalence classes. One way to get a predicative version of this result would be to enrich the notion of interaction system with a notion of internal equality on states. This would be a notion of "interaction system on setoids".

Moreover, this interpretation is invariant under β-reduction:

◇ **Proposition 6.4.2:** *the interpretation of λ-terms is invariant under β-reduction:*

$$[\![(\lambda x.t)u]\!]_\gamma \quad = \quad [\![t[u/x]]\!]_\gamma \ .$$

proof: Direct consequence of the fact that **Int** is a categorical model of linear logic. (So that the Kleisli category of !_ is cartesian closed).

☑

§ *The Differential λ-calculus.* Since we know (proposition 2.4.4) that a union of simulations is still a simulation, it is tempting to try to interpret the differential λ-calculus which comes with a notion of sum of terms. Everything does works without any problem. We extend the interpretation of terms in the following way:

- for $\dfrac{}{\Gamma \vdash 0 : \tau}$, we put $[\![0]\!]_\gamma \triangleq \emptyset$;
- for $\dfrac{\Gamma \vdash t_1 : \tau \qquad \Gamma \vdash t_2 : \tau}{\Gamma \vdash t_1 + t_2 : \tau}$, we use $[\![t_1 + t_2]\!]_\gamma \triangleq [\![t_1]\!]_\gamma \cup [\![t_2]\!]_\gamma$;
- for differentiation $\dfrac{\Gamma \vdash t : \tau \to \tau' \qquad \Gamma \vdash u : \tau}{\Gamma \vdash D\,t \cdot u : \tau \to \tau'}$,

  we use $(\mu, s') \in [\![D\,t \cdot u]\!]_\gamma$ iff $(\mu + [s], s') \in [\![t]\!]_{\gamma_1}$ for some $s \in [\![u]\!]_{\gamma_2}$ s.t. $\gamma = \gamma_1 + \gamma_2$.

The first thing to check is that the interpretation is well-behaved as far as the equations are concerned:

○ **Lemma 6.4.3:** we have:
- $[\![(0)t]\!]_\gamma = [\![\lambda x.0]\!]_\gamma = [\![D\,0 \cdot t]\!]_\gamma = [\![D\,t \cdot 0]\!]_\gamma = [\![0]\!]_\gamma = \emptyset$;
- $[\![(t_1 + t_2)u]\!]_\gamma = [\![(t_1)u + (t_2)u]\!]_\gamma = [\![(t_1)u]\!]_\gamma \cup [\![(t_2)u]\!]_\gamma$;
- $[\![\lambda x.(t_1 + t_2)]\!]_\gamma = [\![(\lambda x.t_1) + (\lambda x.t_2)]\!]_\gamma = [\![\lambda x.t_1]\!]_\gamma \cup [\![\lambda x.t_2]\!]_\gamma$;
- $[\![D(t_1 + t_2) \cdot u]\!]_\gamma = [\![D\,t_1 \cdot u + D\,t_2 \cdot u]\!]_\gamma = [\![D\,t_1 \cdot u]\!]_\gamma \cup [\![D\,t_2 \cdot u]\!]_\gamma$;
- $[\![D\,t \cdot (u_1 + u_2)]\!]_\gamma = [\![D\,t \cdot u_1 + D\,t \cdot u_2]\!]_\gamma = [\![D\,t \cdot u_1]\!]_\gamma \cup [\![D\,t \cdot u_2]\!]_\gamma$;
- $[\![D(D\,t \cdot u) \cdot v]\!]_\gamma = [\![D(D\,t \cdot v) \cdot u]\!]_\gamma$.

proof: the part about 0 is quite direct. For the rest: (in order to be less verbose, we omit the "for some μ" appearing in the interpretation of an application)

→ $[\![(t_1 + t_2)u]\!]_\gamma = [\![(t_1)u + (t_2)u]\!]_\gamma$:
$s \in [\![(t_1 + t_2)u]\!]_\gamma$
$\Leftrightarrow$
$(\mu, s) \in [\![t_1 + t_2]\!]_{\gamma_0}$ and every $\mu_i \in [\![u]\!]_{\gamma_i}$ with $\gamma = \gamma_0 + \gamma_1 + \cdots$
$\Leftrightarrow$
$(\mu, s) \in [\![t_1]\!]_{\gamma_0}$ or $(\mu, s) \in [\![t_2]\!]_{\gamma_0}$ and every $\mu_i \in [\![u]\!]_{\gamma_i}$ with $\gamma = \gamma_0 + \gamma_1 + \cdots$

$\Leftrightarrow$
$s \in [\![(t_1)u]\!]_\gamma$ or $s \in [\![(t_2)u]\!]_\gamma$
$\Leftrightarrow$
$s \in [\![(t_1)u + (t_2)u]\!]_\gamma$

- $[\![\lambda x.(t_1 + t_2)]\!]_\gamma = [\![(\lambda x.t_1) + (\lambda x.t_2)]\!]_\gamma$:
$(\mu, s) \in [\![\lambda x.(t_1 + t_2)]\!]_\gamma$
$\Leftrightarrow$
$s \in [\![t_1 + t_2]\!]_{\gamma, x=\mu}$
$\Leftrightarrow$
$s \in [\![t_1]\!]_{\gamma, x=\mu}$ or $s \in [\![t_2]\!]_{\gamma, x=\mu}$
$\Leftrightarrow$
$(\mu, s) \in [\![\lambda x.t_1]\!]_\gamma$ or $(\mu, s) \in [\![\lambda x.t_2]\!]_\gamma$
$\Leftrightarrow$
$(\mu, s) \in [\![(\lambda x.t_2) + (\lambda x.t_2)]\!]_\gamma$

- $[\![D(t_1 + t_2) \cdot u]\!]_\gamma = [\![D\,t_1 \cdot u + D\,t_2 \cdot u]\!]_\gamma$:
$(\mu, s) \in [\![D(t_1 + t_2) \cdot u]\!]_\gamma$
$\Leftrightarrow$
$(\mu + [s'], s) \in [\![t_1 + t_2]\!]_{\gamma_1}$ and $s' \in [\![u]\!]_{\gamma_2}$ with $\gamma = \gamma_1 + \gamma_2$
$\Leftrightarrow$
$(\mu + [s'], s) \in [\![t_1]\!]_{\gamma_1}$ or $(\mu + [s'], s) \in [\![t_2]\!]_{\gamma_1}$ and $s' \in [\![u]\!]_{\gamma_2}$ with $\gamma = \gamma_1 + \gamma_2$
$\Leftrightarrow$
$(\mu, s) \in [\![D\,t_1 \cdot u]\!]_\gamma$ or $(\mu, s) \in [\![D\,t_2 \cdot u]\!]_\gamma$
$\Leftrightarrow$
$(\mu, s) \in [\![D\,t_1 \cdot u + D\,t_2 \cdot u]\!]_\gamma$

- $[\![D\,t \cdot (u_1 + u_2)]\!]_\gamma = [\![D\,t \cdot u_1 + D\,t \cdot u_2]\!]_\gamma$:
$(\mu, s) \in [\![D\,t \cdot (u_1 + u_2)]\!]_\gamma$
$\Leftrightarrow$
$(\mu + [s'], s) \in [\![t]\!]_{\gamma_1}$ for some $s' \in [\![u_1 + u_2]\!]_{\gamma_2}$ with $\gamma = \gamma_1 + \gamma_2$
$\Leftrightarrow$
$(\mu + [s'], s) \in [\![t]\!]_{\gamma_1}$ for some $s' \in [\![u_1]\!]_{\gamma_2}$ or $s' \in [\![u_2]\!]_{\gamma_2}$ with $\gamma = \gamma_1 + \gamma_2$
$\Leftrightarrow$
$(\mu, s) \in [\![D\,t \cdot u_1]\!]_\gamma$ or $(\mu, s) \in [\![D\,t \cdot u_2]\!]_\gamma$
$\Leftrightarrow$
$(\mu, s) \in [\![D\,t \cdot u_1 + D\,t \cdot u_2]\!]_\gamma$

- $[\![D(D\,t \cdot u) \cdot v]\!]_\gamma = [\![D(D\,t \cdot v) \cdot u]\!]_\gamma$:
by definition, we have $(\mu, s') \in [\![D(D\,t \cdot u) \cdot v]\!]_\gamma$ iff
there is $s_1 \in [\![v]\!]_{\gamma_1}$ and $s_2 \in [\![u]\!]_{\gamma_2}$ s.t. $(\mu + [s_1] + [s_2], s') \in [\![t]\!]_{\gamma_0}$
with $\gamma = \gamma_0 + \gamma_1 + \gamma_2$
By commutativity of "+", this is equivalent to $(\mu, s') \in [\![D(D\,t \cdot v) \cdot u]\!]_\gamma$.
☑

Now that we know the interpretation to be correct, it is quite easy to extend proposition 6.4.1:

◇ **Proposition 6.4.4:** *suppose* $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash t : \tau$ *is a correct typing judgment in the* differential λ*-calculus. For any valuation* ρ *for the atomic types, we have that* $\_ \in [\![t]\!]\_$ *is a simulation from* $!\tau_1^* \otimes \cdots \otimes !\tau_n^*$ *to* τ^*.

proof: since we already know that a union of simulations is a simulation and that the empty set is always a simulation, we only need to check that $[\![D\,t \cdot u]\!]$ is a simulation whenever $[\![t]\!]$ and $[\![u]\!]$ are.

Suppose we have $(\mu, s') \in [\![D\,t \cdot u]\!]_\gamma$, *i.e.* $(\mu + [s], s') \in [\![t]\!]_{\gamma_1}$ for some $s \in [\![u]\!]_{\gamma_2}$, with $\gamma = \gamma_1 + \gamma_2$. We need to show that (μ, s') (in $\tau \to \tau'$) simulates γ (in $!\Gamma$). Since $\gamma = \gamma_1 + \gamma_2$, it is enough to show that we can simulate (γ_1, γ_2) (in $!\Gamma \otimes !\Gamma$). By proposition 3.4.2, this is equivalent to showing that state s' (in τ') simulates state $(\gamma_1, \gamma_2, \mu)$ (in $!\Gamma \otimes !\Gamma \otimes !\tau$).
Let $a_{\gamma_1} \in !A_\Gamma(\gamma_1)$, $a_{\gamma_2} \in !A_\Gamma(\gamma_2)$ and $a_\mu \in !A_\tau(\mu)$. We need to find an action in $A_{\tau'}(s')$ to simulate $(a_{\gamma_1}, a_{\gamma_2}, a_\mu)$:

1) by induction hypothesis, we know that s (in τ) simulates γ_2 (in $!\Gamma$), so that we can find an action $a \in A_\tau(s)$ simulating a_{γ_2};

2) by induction, we know that s' (in τ') simulates $(\gamma_1, \mu + [s])$ (in $!\Gamma \otimes !\tau$), so that we can find an action $a' \in A_{\tau'}(s')$ simulating $\big(a_{\gamma_1}, (a_\mu, a)\big)$.
Since a simulates a_{γ_2}, by composition, a' simulates $\big(a_{\gamma_1}, (a_\mu, a_{\gamma_2})\big)$. By associativity and commutativity, it thus simulates $(a_{\gamma_1}, a_{\gamma_2}, a_\mu)$.

To translate back a reaction d' to a' into a reaction $(d_{\gamma_1}, d_{\gamma_2}, d_\mu)$, we proceed similarly:

2) by induction, we can translate d' into a reaction (d_{γ_1}, d_μ, d) to $\big(a_{\gamma_1}, (a_\mu, a)\big)$;

1) by induction, we can also translate the reaction d (in $D_\tau(s, a)$) into a reaction d_{γ_2} (in $!D_\Gamma(s, a_{\gamma_2})$).

We thus obtain reactions d_{γ_1}, d_{γ_2} and d_μ as desired. That the resulting next states are still related is obvious.

☑

As before, this interpretation is invariant under reduction:

◇ **Proposition 6.4.5:** *the interpretation of a differential* λ*-terms is invariant under* β*-reduction and differential reduction:*

$$\begin{aligned} [\![(\lambda x.t)u]\!]_\gamma &= [\![t[u/x]]\!]_\gamma \\ [\![D(\lambda x.t) \cdot u]\!]_\gamma &= [\![\lambda x.(\partial t/\partial x) \cdot u]\!]_\gamma \ . \end{aligned}$$

proof: this ought to be a consequence of the notion of categorical model for the differential λ calculus. Since we haven't developped such a notion in full generality, we check directly the result: see section 6.4.3.[1]

☑

[1]: Some work about a general notion of "differential" category has been done by Martin Hyland.

6.4.3 Invariance under Reduction

Here is, for posterity, the complete direct proof that the relational model for the differential λ-calculus is invariant under cut-elimination. This is the perfect example of "folkloric" proof: it is at the same time long, boring, easy, error prone, and true!

In order to simplify, we use λ-terms satisfying the Barendregt condition: no free variable is also bound in a term. By convention, if μ is a multiset, μ_i denotes an element of μ while μ^i denotes a "sub-multiset" of μ.

- **Lemma 6.4.6:**
  - if x is not free in t, then $\gamma(x) \neq []$ implies $[\![t]\!]_\gamma = \emptyset$;
  - if x is not free in t, then $[\![t]\!]_\gamma = [\![t]\!]_{\gamma,x=[]}$.

proof: simple induction. ☑

proof: (of proposition 6.4.5) just like in the proof of lemma 6.4.3, we omit the "for some μ" appearing in the interpretation of an application.

first part: $[\![(\lambda x.t)u]\!]_\gamma = [\![t[u/x]]\!]_\gamma$
Notice first the following equality:
$s \in [\![(\lambda x.t)u]\!]_\gamma$
$\Leftrightarrow$
$(\mu, s) \in [\![\lambda x.t]\!]_{\gamma^0}$ and every $\mu_i \in [\![u]\!]_{\gamma^i}$ with $\gamma = \gamma^0 + \gamma^1 + \cdots$
$\Leftrightarrow$
$s \in [\![t]\!]_{\gamma^0,x=\mu}$ and every $\mu_i \in [\![u]\!]_{\gamma^i}$ with $\gamma = \gamma^0 + \gamma^1 + \cdots$

→ variable: if $t = x$, then we need to show $[\![(\lambda x.x)u]\!]_\gamma = [\![u]\!]_\gamma$.
$s \in [\![(\lambda x.x)u]\!]_\gamma$
$\Leftrightarrow$
$s \in [\![x]\!]_{\gamma^0,x=\mu}$ and every $\mu_i \in [\![u]\!]_{\gamma^i}$ with $\gamma = \gamma^0 + \gamma^1 + \cdots$
$\Leftrightarrow$
$\gamma^0 = ([], \ldots, [])$ and $\mu = [s]$ and $\mu_1 = s \in [\![u]\!]_{\gamma^1}$ and $\gamma = [] + \gamma^1$
$\Leftrightarrow$
$s \in [\![u]\!]_\gamma$

→ variable (bis): if $t = y$, then we need to show $[\![(\lambda x.y)u]\!]_\gamma = [\![y]\!]_\gamma$.
$s \in [\![(\lambda x.y)u]\!]_\gamma$
$\Leftrightarrow$
$s \in [\![y]\!]_{\gamma^0,x=\mu}$ and every $\mu_i \in [\![u]\!]_{\gamma^i}$ with $\gamma = \gamma^0 + \gamma^1 + \cdots$
$\Leftrightarrow$
γ is "$(y = [s])$" and $\mu = []$ and $s \in [\![y]\!]_{\gamma,x=[]}$
$\Leftrightarrow$
$s \in [\![y]\!]_\gamma$

→ abstraction: if $t = \lambda y.t$, then we need to show $[\![(\lambda xy.t)u]\!]_\gamma = [\![\lambda y.t[u/x]]\!]_\gamma$.
By induction, we know that $[\![t[u/x]]\!] = [\![(\lambda x.t)u]\!]$,
we thus need to show $[\![(\lambda xy.t)u]\!]_\gamma = [\![\lambda y.(\lambda x.t)u]\!]_\gamma$.

$(\nu, s) \in [\![(\lambda xy.t)u]\!]_\gamma$
$\quad \Leftrightarrow$
$(\nu, s) \in [\![\lambda y.t]\!]_{\gamma^0, x=\mu}$ and every $\mu_i \in [\![u]\!]_{\gamma^i}$ with $\gamma = \gamma^0 + \cdots$
$\quad \Leftrightarrow$
$s \in [\![t]\!]_{\gamma^0, x=\mu, y=\nu}$ and every $\mu_i \in [\![u]\!]_{\gamma^i}$ with $\gamma = \gamma^0 + \cdots$

Similarly:
$(\nu, s) \in [\![\lambda y.(\lambda x.t)u]\!]_\gamma$
$\quad \Leftrightarrow$
$s \in [\![(\lambda x.t)u]\!]_{\gamma, y=\nu}$
$\quad \Leftrightarrow$
$(\mu, s) \in [\![(\lambda x.t)]\!]_{\gamma^0, y=\nu^0}$ and every $\mu_i \in [\![u]\!]_{\gamma^i, y=\nu^i}$
with $\gamma = \gamma^0 + \cdots$ and $\nu = \nu^0 + \cdots$.
$\quad \Leftrightarrow$
$s \in [\![t]\!]_{\gamma^0, y=\nu_0, x=\mu}$ and every $\mu_i \in [\![u]\!]_{\gamma^i, y=\nu^i}$
with $\gamma = \gamma^0 + \cdots$ and $\nu = \nu^0 + \cdots$. (6-4)

If $\nu = []$, equality holds.
If not, since y is not free in u (by Barendregt condition), by lemma 6.4.6, we have $\nu_i = []$ for every $i > 0$ (since $\mu_i \in [\![u]\!]_{\gamma^i, y=\nu_i}$). (6-4) simplifies into:
$s \in [\![t]\!]_{\gamma^0, y=\nu^0, x=\mu}$ and every $\mu_i \in [\![u]\!]_{\gamma^i, y=[]}$ with $\gamma = \gamma^0 + \cdots$.
We thus have equality.

- application: if $t = (t_1)t_2$, we need to show $[\![(\lambda x.(t_1)t_2)u]\!]_\gamma = [\![(t_1[u/x])t_2[u/x]]\!]_\gamma$.
By induction, we know that $[\![t_1[u/x]]\!] = [\![(\lambda x.t_1)u]\!]$,
we thus need to show $[\![(\lambda x.(t_1)t_2)u]\!]_\gamma = [\![(\lambda x.t_1)u((\lambda x.t_2)u)]\!]_\gamma$.
$s \in [\![(\lambda x.(t_1)t_2)u]\!]_\gamma$
$\quad \Leftrightarrow$
$s \in [\![(t_1)t_2]\!]_{\gamma^0, x=\mu}$ and every $\mu_i \in [\![u]\!]_{\mu_i}$ with $\gamma = \gamma^0 + \cdots$
$\quad \Leftrightarrow$
$(\nu, s) \in [\![t_1]\!]_{\gamma^{0,0}, x=\mu^0}$ and every $\nu_j \in [\![t_2]\!]_{\gamma^{0,j}, x=\mu^j}$ and every $\mu_i \in [\![u]\!]_{\gamma^i}$
with $\gamma = \gamma^0 + \cdots$ and $\gamma^0 = \gamma^{0,0} + \gamma^{0,1} + \cdots$ and $\mu = \mu^0 + \cdots$

Similarly:
$s \in [\![(\lambda x.t_1)u((\lambda x.t_2)u)]\!]_\gamma$
$\quad \Leftrightarrow$
$(\nu, s) \in [\![(\lambda x.t_1)u]\!]_{\gamma^0}$ and every $\nu_j \in [\![(\lambda x.t_2)u]\!]_{\gamma^j}$ with $\gamma = \gamma^0 + \cdots$
$\quad \Leftrightarrow$
$(\nu, s) \in [\![(\lambda x.t_1)u]\!]_{\gamma^0}$ and every $\nu_j \in [\![t_2]\!]_{\gamma^{j,0}, x=\mu^j}$ and every $\mu_i^j \in [\![u]\!]_{\gamma^{j,i}}$
with $\gamma = \gamma^0 + \cdots$ and every $\gamma^j = \gamma^{j,0} + \cdots$
$\quad \Leftrightarrow$
$(\nu, s) \in [\![t_1]\!]_{\gamma^{0,0}, x=\rho}$ and every $\rho_k \in [\![u]\!]_{\gamma^{0,k}}$ and every $\nu_j \in [\![t_2]\!]_{\gamma^{j,0}, x=\mu^j}$
and every $\mu_i^j \in [\![u]\!]_{\gamma^{j,i}}$
with $\gamma = \gamma^0 + \cdots$ and $\gamma^0 = \gamma^{0,0} + \cdots$ and every $\gamma^j = \gamma^{j,0} + \cdots$

In the last line, if one replaces ρ by the μ^0, it is easy to see that we have equality. (The only difference is that in the first case, the elements of μ are indexed by i, while in the second case, they are indexed by i, j.)

- zero: if $t = 0$, then we need to show that $[\![(\lambda x.0)u]\!]_\gamma = [\![0[u/x]]\!]_\gamma = [\![0]\!]_\gamma$.
  This holds trivially by lemma 6.4.3.

- If $t = t_1 + t_2$ then we need to show that $[\![(\lambda x.t_1 + t_2)u]\!]_\gamma = [\![t_1[u/x] + t_2[u/x]]\!]_\gamma$.
  By induction, we know that $[\![t_i[u/x]]\!] = [\![(\lambda x.t_i)u]\!]$,
  we thus need to show $[\![(\lambda x.t_1 r + t_2)u]\!]_\gamma = [\![(\lambda x.t_1)u + (\lambda x.t_2)u]\!]_\gamma$.
  This holds by lemma 6.4.3.

- differentiation: if $t = D\,t_1 \cdot t_2$,
  then we need to show that $[\![(\lambda x.\, D\,t_1 \cdot t_2)u]\!]_\gamma = [\![(D\,t_1 \cdot t_2)[u/x]]\!]_\gamma$,
  i.e. that $[\![(\lambda x.\, D\,t_1 \cdot t_2)u]\!]_\gamma = [\![D\,t_1[u/x] \cdot t_2[u/x]]\!]_\gamma$.
  By induction, it is enough to prove $[\![(\lambda x.\, D\,t_1 \cdot t_2)u]\!]_\gamma = [\![D(\lambda x.t_1)u \cdot (\lambda x.t_2)u]\!]_\gamma$.
  $(v, s) \in [\![(\lambda x.\, D\,t_1 \cdot t_2)u]\!]_\gamma$
  $\Leftrightarrow$
  $(v, s) \in [\![D\,t_1 \cdot t_2]\!]_{\gamma^0, x=\mu}$ and every $\mu_i \in [\![u]\!]_{\gamma^i}$ with $\gamma = \gamma^0 + \cdots$
  $\Leftrightarrow$
  $(v + [s'], s) \in [\![t_1]\!]_{\gamma^{0,1}, x=\mu^1}$ and $s' \in [\![t_2]\!]_{\gamma^{0,2}, x=\mu^2}$
  and every $\mu_i \in [\![u]\!]_{\gamma^i}$ with $\gamma = \gamma^{0,1} + \gamma^{0,2} + \gamma^1 + \cdots$

  Similarly:
  $(v, s) \in [\![D(\lambda x.t_1)u \cdot (\lambda x.t_2)u]\!]_\gamma$
  $\Leftrightarrow$
  $(v + [s'], s) \in [\![(\lambda x.t_2)u]\!]_{\gamma^1}$ with $s' \in [\![(\lambda x.t_2)u]\!]_{\gamma^2}$ and $\gamma = \gamma^1 + \gamma^2$
  $\Leftrightarrow$
  $(v + [s'], s) \in [\![t_1]\!]_{\gamma^{1,0}, x=\mu^1}$ and every $\mu_i^1 \in [\![u]\!]_{\gamma^{1,i}}$
  and $s' \in [\![t_2]\!]_{\gamma^{2,0}, x=\mu^2}$ and every $\mu_j^2 \in [\![u]\!]_{\gamma^{2,j}}$
  with $\gamma = \gamma^{1,0} + \cdots + \gamma^{2,0} + \cdots$
  It is easy to see that we have indeed equality.

  second part: $[\![D(\lambda x.t) \cdot u]\!]_\gamma = [\![\lambda x.(\partial t/\partial x) \cdot u]\!]_\gamma$.
  First notice the following equality:
  $(\mu, s) \in [\![D(\lambda x.t) \cdot u]\!]_\gamma$
  $\Leftrightarrow$
  $(\mu + [s'], s) \in [\![\lambda x.t]\!]_{\gamma^1}$ for some $s' \in [\![u]\!]_{\gamma^2}$ with $\gamma = \gamma^1 + \gamma^2$
  $\Leftrightarrow$
  $s \in [\![t]\!]_{\gamma^1, x=\mu+[s']}$ and $s' \in [\![u]\!]_{\gamma^2}$ with $\gamma = \gamma^1 + \gamma^2$

- variable: if $t = x$, then we need to show $[\![D(\lambda x.x) \cdot u]\!]_\gamma = [\![\lambda x.(\partial x/\partial x) \cdot u]\!]_\gamma$,
  i.e. that $[\![D(\lambda x.x) \cdot u]\!]_\gamma = [\![\lambda x.u]\!]_\gamma$.
  $(\mu, s) \in [\![D(\lambda x.x) \cdot u]\!]_\gamma$
  $\Leftrightarrow$
  $s \in [\![x]\!]_{\gamma^1, x=\mu+[s']}$ and $s' \in [\![u]\!]_{\gamma^2}$ with $\gamma = \gamma^1 + \gamma^2$
  $\Leftrightarrow$
  $\gamma^1 = []$ and $\mu = []$ and $s' = s$ and $s \in [\![u]\!]_{\gamma^2}$ with $\gamma = \gamma^1 + \gamma^2$
  $\Leftrightarrow$
  $s \in [\![u]\!]_\gamma$

On the other hand:
$(\mu, s) \in [\![\lambda x.u]\!]_\gamma$
$\quad \Leftrightarrow$
$s \in [\![u]\!]_{\gamma, x=\mu}$
$\quad \Leftrightarrow$ { since x is not free in u (by Barendregt condition) }
$s \in [\![u]\!]_\gamma$

- variable (bis): if $t = y$, then we need to show $[\![D(\lambda x.y) \cdot u]\!]_\gamma = [\![\lambda x.(\partial y/\partial x) \cdot u]\!]_\gamma$,
i.e. that $[\![D(\lambda x.y) \cdot u]\!]_\gamma = [\![0]\!]_\gamma$.
$(\nu, s) \in [\![D(\lambda x.y) \cdot u]\!]_\gamma$
$\quad \Leftrightarrow$
$s \in [\![y]\!]_{\gamma^1, x=\nu+[s']}$ and ...
By lemma 6.4.6, this is impossible (because x is not free in y).
We thus have that $[\![D(\lambda x.y) \cdot u]\!]_\gamma = \emptyset = [\![0]\!]_\gamma$.

- If $t = t_1 + t_2$, we need to show $[\![D(\lambda x.t_1 + t_2) \cdot u]\!]_\gamma = [\![\lambda x.(\partial t_1 + t_2/\partial x) \cdot u]\!]_\gamma$,
i.e. that $[\![D(\lambda x.t_1 + t_2) \cdot u]\!]_\gamma = [\![\lambda x.(\partial t_1/\partial x) \cdot u + (\partial t_2/\partial x) \cdot u]\!]_\gamma$.
By induction, we know that $[\![\lambda x.(\partial t_i/\partial x) \cdot u]\!] = [\![D(\lambda x.t_i) \cdot u]\!]$.
Thus, we need to show $[\![D(\lambda x.t_1 + t_2) \cdot u]\!]_\gamma = [\![D(\lambda x.t_1) \cdot u + D(\lambda x.t_2) \cdot u]\!]_\gamma$.
This follows from lemma 6.4.3.

- If $t = 0$, this is trivial.

- abstraction: if t is of the form $\lambda y.t$,
we need to show that $[\![D(\lambda xy.t) \cdot u]\!]_\gamma = [\![\lambda x.(\partial \lambda y.t)/(\partial x) \cdot u]\!]_\gamma$,
i.e. that $[\![D(\lambda xy.t) \cdot u]\!]_\gamma = [\![\lambda xy.(\partial t/\partial x) \cdot u]\!]_\gamma$.
$(\nu, \mu, s) \in [\![D(\lambda xy.t) \cdot u]\!]_\gamma$
$\quad \Leftrightarrow$
$(\mu, s) \in [\![\lambda y.t]\!]_{\gamma_1, x=\nu+[s']}$ and $s' \in [\![u]\!]_{\gamma^2}$ with $\gamma = \gamma^1 + \gamma^2$
$\quad \Leftrightarrow$
$s \in [\![t]\!]_{\gamma_1, x=\nu+[s'], y=\mu}$ and $s' \in [\![u]\!]_{\gamma^2}$ with $\gamma = \gamma^1 + \gamma^2$

Similarly:
$(\nu, \mu, s) \in [\![\lambda xy.(\partial t/\partial x) \cdot u]\!]_\gamma$
$\quad \Leftrightarrow$
$s \in [\![(\partial t/\partial x) \cdot u]\!]_{\gamma, x=\nu, y=\mu}$
$\quad \Leftrightarrow$
$(\nu, s) \in [\![\lambda x.(\partial t/\partial x) \cdot u]\!]_{\gamma, y=\mu}$
$\quad \Leftrightarrow$ { by induction }
$(\nu, s) \in [\![D(\lambda x.t) \cdot u]\!]_{\gamma, y=\mu}$
$\quad \Leftrightarrow$
$(\nu + [s'], s) \in [\![\lambda x.t]\!]_{\gamma^1, y=\mu^1}$ and $s' \in [\![u]\!]_{\gamma^2, y=\mu^2}$ with $\gamma = \gamma^1 + \gamma^2$ and $\mu = \mu^1 + \mu^2$
$\quad \Leftrightarrow$ { since y is not free in u (Barendregt's convention), $\mu^2 = []$ }
$(\nu + [s'], s) \in [\![\lambda x.t]\!]_{\gamma^1, y=\mu}$ and $s' \in [\![u]\!]_{\gamma^2, y=[]}$ with $\gamma = \gamma^1 + \gamma^2$
$\quad \Leftrightarrow$
$s \in [\![t]\!]_{\gamma^1, y=\mu, x=\nu+[s']}$ and $s' \in [\![u]\!]_{\gamma^2}$ with $\gamma = \gamma^1 + \gamma^2$
so that we have equality.

- differentiation: if $t = D\,t_1 \cdot t_2$,
  we need to show $[\![D(\lambda x.\, D\,t_1 \cdot t_2) \cdot u]\!]_\gamma = [\![\lambda x.(\partial\, D\,t_1 \cdot t_2/\partial x) \cdot u]\!]$,
  i.e. $[\![D(\lambda x.\, D\,t_1 \cdot t_2) \cdot u]\!]_\gamma = [\![\lambda x.\, D((\partial t_1/\partial x) \cdot u) \cdot t_2]\!]_\gamma \cup [\![\lambda x.\, D\,t_1 \cdot (\partial t_2/\partial x) \cdot u]\!]_\gamma$
  $(\nu, \mu, s) \in [\![D(\lambda x.\, D\,t_1 \cdot t_2) \cdot u]\!]_\gamma$
  $\Leftrightarrow$
  $(\mu, s) \in [\![D\,t_1 \cdot t_2]\!]_{\gamma^1, x=\nu+[s']}$ and $s' \in [\![u]\!]_{\gamma^2}$ with $\gamma = \gamma^1 + \gamma^2$
  $\Leftrightarrow$
  $(\mu + [s''], s) \in [\![t_1]\!]_{\gamma^{1,1}, x=\nu^1}$ and $s'' \in [\![t_2]\!]_{\gamma^{1,2}, x=\nu^2}$ and $s' \in [\![u]\!]_{\gamma^2}$
  with $\gamma = \gamma^{1,1} + \gamma^{1,2} + \gamma^2$ and $\nu + [s'] = \nu^1 + \nu^2$ (6-5)

  On the other hand:
  $(\nu, \mu, s) \in [\![\lambda x.\, D((\partial t_1/\partial x) \cdot u) \cdot t_2]\!]_\gamma$
  $\Leftrightarrow$
  $(\mu, s) \in [\![D((\partial t_1/\partial x) \cdot u) \cdot t_2]\!]_{\gamma, x=\nu}$
  $\Leftrightarrow$
  $(\mu + [s''], s) \in [\![(\partial t_1/\partial x) \cdot u]\!]_{\gamma^1, x=\nu^1}$ and $s'' \in [\![t_2]\!]_{\gamma^2, x=\nu^2}$
  with $\gamma = \gamma^1 + \gamma^2$ and $\nu = \nu^1 + \nu^2$
  $\Leftrightarrow$
  $(\nu^1, \mu + [s''], s) \in [\![\lambda x.(\partial t_1/\partial x) \cdot u]\!]_{\gamma^1}$ and $s'' \in [\![t_2]\!]_{\gamma^2, x=\nu^2}$
  with $\gamma = \gamma^1 + \gamma^2$ and $\nu = \nu^1 + \nu^2$
  $\Leftrightarrow$ { by induction }
  $(\nu^1, \mu + [s''], s) \in [\![D\,\lambda x.t_1 \cdot u]\!]_{\gamma^1}$ and $s'' \in [\![t_2]\!]_{\gamma^2, x=\nu^2}$
  with $\gamma = \gamma^1 + \gamma^2$ and $\nu = \nu^1 + \nu^2$
  $\Leftrightarrow$
  $(\mu + [s''], s) \in [\![t_1]\!]_{\gamma^{1,1}, x=\nu^1+[s']}$ and $s' \in [\![u]\!]_{\gamma^{1,2}}$ and $s'' \in [\![t_2]\!]_{\gamma^2, x=\nu^2}$
  with $\gamma = \gamma^{1,1} + \gamma^{1,2} + \gamma^2$ and $\nu = \nu^1 + \nu^2$ (6-6)

  and
  $(\nu, \mu, s) \in [\![\lambda x.\, D\,t_1 \cdot (\partial t_2/\partial x) \cdot u]\!]_\gamma$
  $\Leftrightarrow$
  $(\mu, s) \in [\![D\,t_1 \cdot (\partial t_2/\partial x) \cdot u]\!]_{\gamma, x=\nu}$
  $\Leftrightarrow$
  $(\mu + [s''], s) \in [\![t_1]\!]_{\gamma^1, x=\nu^1}$ and $s'' \in [\![(\partial t_2/\partial x) \cdot u]\!]_{\gamma^2, x=\nu^2}$
  with $\gamma = \gamma^1 + \gamma^2$ and $\nu = \nu^1 + \nu^2$
  $\Leftrightarrow$
  $(\mu + [s''], s) \in [\![t_1]\!]_{\gamma^1, x=\nu^1}$ and $(\nu^2, s'') \in [\![\lambda x.(\partial t_2/\partial x) \cdot u]\!]_{\gamma^2}$
  with $\gamma = \gamma^1 + \gamma^2$ and $\nu = \nu^1 + \nu^2$
  $\Leftrightarrow$ { by induction }
  $(\mu + [s''], s) \in [\![t_1]\!]_{\gamma^1, x=\nu^1}$ and $(\nu^2, s'') \in [\![D\,\lambda x.t_2 \cdot u]\!]_{\gamma^2}$
  with $\gamma = \gamma^1 + \gamma^2$ and $\nu = \nu^1 + \nu^2$
  $\Leftrightarrow$
  $(\mu + [s''], s) \in [\![t_1]\!]_{\gamma^1, x=\nu^1}$ and $s'' \in [\![t_2]\!]_{\gamma^{2,1}, x=\nu^2+[s']}$ and $s' \in [\![u]\!]_{\gamma^{2,2}}$
  with $\gamma = \gamma^1 + \gamma^{2,1} + \gamma^{2,2}$ and $\nu = \nu^1 + \nu^2$ (6-7)

  It is easy to see that (6-6) implies (6-5) and that (6-7) implies (6-5).
  To see that (6-5) implies (6-6) or (6-7), note that s' is either in ν^1 or in ν^2. In the first case, we get (6-6) and in the second case, we get (6-7).
  This concludes the case of differentiation.

→ application: if $t = (t_1)t_2$,
we need to show $[\![D(\lambda x.(t_1)t_2) \cdot u]\!]_\gamma = [\![\lambda x.(\partial(t_1)t_2/\partial x) \cdot u]\!]$,
i.e. $[\![D(\lambda x.(t_1)t_2) \cdot u]\!]_\gamma = [\![\lambda x\ .\ ((\partial t_1/\partial x) \cdot u)t_2\ +\ (D\,t_1 \cdot (\partial t_2/\partial x) \cdot u)t_2]\!]_\gamma$,
i.e. $[\![D(\lambda x.(t_1)t_2) \cdot u]\!]_\gamma = [\![\lambda x.((\partial t_1/\partial x) \cdot u)t_2]\!]_\gamma \cup [\![\lambda x.(D\,t_1 \cdot (\partial t_2/\partial x) \cdot u)t_2]\!]_\gamma$.

$(\nu, s) \in [\![D(\lambda x.(t_1)t_2) \cdot u]\!]_\gamma$
$\Leftrightarrow$
$s \in [\![(t_1)t_2]\!]_{\gamma^1, x=\nu+[s']}$ and $s' \in [\![u]\!]_{\gamma^2}$ with $\gamma = \gamma^1 + \gamma^2$
$\Leftrightarrow$
$(\mu, s) \in [\![t_1]\!]_{\gamma^{1,0}, x=\nu^0}$ and every $\mu_i \in [\![t_2]\!]_{\gamma^{1,i}, x=\nu^i}$ and $s' \in [\![u]\!]_{\gamma^2}$
with $\gamma = \gamma^{1,0} + \cdots + \gamma^{1,i} + \gamma^2$ and $\nu + [s'] = \nu^0 + \cdots$ (6-8)

On the other hand:
$(\nu, s) \in [\![\lambda x.((\partial t_1/\partial x) \cdot u)t_2]\!]_\gamma$
$\Leftrightarrow$
$(\mu, s) \in [\![(\partial t_1/\partial x) \cdot u]\!]_{\gamma^0, x=\nu^0}$ and every $\mu_i \in [\![t_2]\!]_{\gamma^i, x=\nu^i}$
with $\gamma = \gamma^0 + \cdots$ and $\nu = \nu^0 + \cdots$
$\Leftrightarrow$
$(\nu^0, \mu, s) \in [\![\lambda x.(\partial t_1/\partial x) \cdot u]\!]_{\gamma^0}$ and every $\mu_i \in [\![t_2]\!]_{\gamma^i, x=\nu^i}$
with $\gamma = \gamma^0 + \cdots$ and $\nu = \nu^0 + \cdots$
$\Leftrightarrow$ (by induction)
$(\nu^0, \mu, s) \in [\![D\,\lambda x.t_1 \cdot u]\!]_{\gamma^0}$ and every $\mu_i \in [\![t_2]\!]_{\gamma^i, x=\nu^i}$
with $\gamma = \gamma^0 + \cdots$ and $\nu = \nu^0 + \cdots$
$\Leftrightarrow$
$(\nu^0 + [s'], \mu, s) \in [\![\lambda x.t_1]\!]_{\gamma^{0,1}}$ and $s' \in [\![u]\!]_{\gamma^{0,2}}$ and every $\mu_i \in [\![t_2]\!]_{\gamma^i, x=\nu^i}$
with $\gamma = \gamma^{0,1} + \gamma^{0,2} + \cdots$ and $\nu = \nu^0 + \cdots$
$\Leftrightarrow$
$(\mu, s) \in [\![t_1]\!]_{\gamma^{0,1}, x=\nu^0+[s']}$ and $s' \in [\![u]\!]_{\gamma^{0,2}}$ and every $\mu_i \in [\![t_2]\!]_{\gamma^i, x=\nu^i}$
with $\gamma = \gamma^{0,1} + \gamma^{0,2} + \cdots$ and $\nu = \nu^0 + \cdots$ (6-9)

and
$(\nu, s) \in [\![\lambda x.(D\,t_1 \cdot (\partial t_2/\partial x) \cdot u)t_2]\!]_\gamma$
$\Leftrightarrow$
$s \in [\![(D\,t_1 \cdot (\partial t_2/\partial x) \cdot u)t_2]\!]_{\gamma, x=\nu}$
$\Leftrightarrow$
$(\mu, s) \in [\![D\,t_1 \cdot (\partial t_2/\partial x) \cdot u]\!]_{\gamma^0, x=\nu^0}$ and every $\mu_i \in [\![t_2]\!]_{\gamma^i, x=\nu^i}$
with $\gamma = \gamma^0 + \cdots$ and $\nu = \nu^0 + \cdots$
$\Leftrightarrow$
$(\mu + [s''], s) \in [\![t_1]\!]_{\gamma^{0,1}, x=\nu^{0,1}}$ and $s'' \in [\![(\partial t_2/\partial x) \cdot u]\!]_{\gamma^{0,2}, x=\nu^{0,2}}$
and every $\mu_i \in [\![t_2]\!]_{\gamma^i, x=\nu^i}$
with $\gamma = \gamma^{0,1} + \gamma^{0,2} + \cdots + \gamma^i$ and $\nu = \nu^{0,1} + \nu^{0,2} + \cdots + \nu^i$
$\Leftrightarrow$
$(\mu + [s''], s) \in [\![t_1]\!]_{\gamma^{0,1}, x=\nu^{0,1}}$ and $(\nu^{0,2}, s'') \in [\![\lambda x.(\partial t_2/\partial x) \cdot u]\!]_{\gamma^{0,2}}$
and every $\mu_i \in [\![t_2]\!]_{\gamma^i, x=\nu^i}$
with $\gamma = \gamma^{0,1} + \gamma^{0,2} + \cdots + \gamma^i$ and $\nu = \nu^{0,1} + \nu^{0,2} + \cdots + \nu^i$
$\Leftrightarrow$ { by induction hypothesis }
$(\mu + [s''], s) \in [\![t_1]\!]_{\gamma^{0,1}, x=\nu^{0,1}}$ and $(\nu^{0,2}, s'') \in [\![D(\lambda x.t_2) \cdot u]\!]_{\gamma^{0,2}}$
and every $\mu_i \in [\![t_2]\!]_{\gamma^i, x=\nu^i}$

with $\gamma = \gamma^{0,1} + \gamma^{0,2} + \cdots + \gamma^{i}$ and $\nu = \nu^{0,1} + \nu^{0,2} + \cdots + \nu^{i}$

$\Leftrightarrow$

$(\mu + [s''], s) \in [\![t_1]\!]_{\gamma^{0,1}, x=\nu^{0,1}}$ and $(\nu^{0,2} + [s'], s'') \in [\![\lambda x.t_2]\!]_{\gamma^{0,2,1}}$ and $s' \in [\![u]\!]_{\gamma^{0,2,2}}$
and every $\mu_i \in [\![t_2]\!]_{\gamma^i, x=\nu^i}$
with $\gamma = \gamma^{0,1} + \gamma^{0,2} + \cdots + \gamma^{i}$ and $\nu = \nu^{0,1} + \nu^{0,2} + \cdots + \nu^{i}$ and $\gamma^{0,2} = \gamma^{0,2,1} + \gamma^{0,2,2}$

$\Leftrightarrow$

$(\mu + [s''], s) \in [\![t_1]\!]_{\gamma^{0,1}, x=\nu^{0,1}}$ and $s'' \in [\![t_2]\!]_{\gamma^{0,2,1}, x=\nu^{0,2}+[s']}$ and $s' \in [\![u]\!]_{\gamma^{0,2,2}}$
and every $\mu_i \in [\![t_2]\!]_{\gamma^i, x=\nu^i}$
with $\gamma = \gamma^{0,1} + \gamma^{0,2,1} + \gamma^{0,2,2} + \cdots + \gamma^{i}$ and $\nu = \nu^{0,1} + \nu^{0,2} + \cdots + \nu^{i}$ (6-10)

It is immediate that (6-9) implies (6-8).
It is also direct that (6-10) implies (6-8).

To see that (6-8) implies (6-9) or (6-10): in (6-7), we have either

- first case: $s' \in \nu^0$, in which case (6-8) is of the form
  $(\mu, s) \in [\![t_1]\!]_{\gamma^{1,0}, x=\nu^0+[s']}$ and every $\mu_i \in [\![t_2]\!]_{\gamma^{1,i}, x=\nu^i}$ and $s' \in [\![u]\!]_{\gamma^2}$
  with $\gamma = \gamma^{1,0} + \cdots + \gamma^{1,i} + \gamma^2$ and $\nu = \nu^0 + \cdots$
  which implies (6-9).
- second case: there is some i_0 such that $s' \in \nu^{i_0}$.
  If we rename μ_{i_0} into s'', (6-8) has the form:
  $(\mu + [s''], s) \in [\![t_1]\!]_{\gamma^{1,0}, x=\nu^0}$ and every $\mu_i \in [\![t_2]\!]_{\gamma^{1,i}, x=\nu^i}$
  and $s'' \in [\![t_2]\!]_{\gamma'', x=\nu''+[s']}$ and $s' \in [\![u]\!]_{\gamma^2}$ with $\gamma = \gamma'' + \gamma^{1,0} + \cdots + \gamma^{1,i} + \gamma^2$
  and $\nu = \nu'' + \nu^0 + \cdots$
  which easily implies (6-10).

This concludes the case of application, and so, concludes the proof of the second part of proposition 6.4.5.

☑

7 An Abstract Version: Predicate Transformers

We argued at the beginning of section 2.5 that predicate transformers are appropriate to model *abstract specifications* for programs. We also argued in sections 2.5.1 and 2.5.6 that interaction systems can be seen as concrete representations for predicate transformers. It is thus natural to look at the notion of predicate transformer as a denotational model for linear logic. We know by proposition 2.5.23 that the two categories are equivalent, so that predicate transformers do form a denotational model for full linear logic. It is however interesting to unfold the details since the resulting model is both concise and elegant.

7.1 A Denotational Model

7.1.1 Multiplicative Additive Linear Logic

It is quite straightforward to unfold proposition 2.5.23 and the linear logic connectives to act on predicate transformers rather than on interaction systems.

§ *The connectives.* By the isomorphism $\mathcal{P}(S_1 + S_2) \simeq \mathcal{P}(S_1) \times \mathcal{P}(S_2)$, we can define:

▷ **Definition 7.1.1:** if P_1 and P_2 are (monotonic) predicate transformers on S_1 and S_2, define the predicate transformer $P_1 \oplus P_2$ on $S_1 + S_2$ as:

$$P_1 \oplus P_2\big((x_1, x_2)\big) \quad \triangleq \quad \big(P_1(x_1), P_2(x_2)\big)\ . \qquad \textit{(where } x_1 \subseteq S_1 \textit{ and } x_2 \subseteq S_2\textit{)}$$

Because of the definition of linear negation, the fact that $\oplus$ is self-dual will be a triviality!

The constants are very simple:

▷ **Definition 7.1.2:** define the constants **0** and **skip** as:

$$\begin{array}{rcl} \mathbf{0} & : & \mathcal{P}(\emptyset) \to \mathcal{P}(\emptyset) \\ & & \emptyset \quad \mapsto \emptyset \end{array} \qquad \text{and} \qquad \begin{array}{rcl} \mathbf{skip} & : & \mathcal{P}(\{*\}) \to \mathcal{P}(\{*\}) \\ & & x \quad \mapsto x \end{array}$$

We also write $\bot$ for the predicate transformer **skip**.

The tensor $P_1 \otimes P_2$ of two predicate transformers is a little more complex but is the most "natural" predicate transformer to define on $S_1 \times S_2$. The first remark is that $P_1 \otimes P_2(x_1 \times x_2)$ is most naturally defined as $P_1(x_1) \times P_2(x_2)$. With that in mind, we define $P_1 \otimes P_2$ as the smallest predicate transformer "generated" by this:

▷ **Definition 7.1.3:** if P_1 and P_2 are (monotonic) predicate transformers on S_1 and S_2, define the predicate transformer $P_1 \otimes P_2$ on $S_1 \times S_2$ as:

$$P_1 \otimes P_2(r) \quad \triangleq \quad \bigcup_{x_1 \times x_2 \subseteq r} P_1(x_1) \times P_2(x_2) \ . \qquad \textit{(where } r \subseteq S_1 \times S_2\textit{)}$$

This operation has already been considered in the refinement calculus to model parallel execution of independent pieces of programs, see [10].

As opposed to the case of interaction systems, where the definition of the linear arrow is quite complex, $\multimap$ takes a very simple form in the context of predicate transformers, especially if one has some realizability intuition about it:[1]

▷ **Definition 7.1.4:** if P_1 and P_2 are (monotonic) predicate transformers on S_1 and S_2, define the predicate transformer $P_1 \multimap P_2$ on $S_1 \times S_2$ as:

$$(s_1, s_2) \in P_1 \multimap P_2(r) \quad \Leftrightarrow \quad (\forall x_1 \subseteq S_1)\ s_1 \in P_1(x_1) \Rightarrow s_2 \in P_2\big(r(x_1)\big) \ . \qquad \textit{(where } r \subseteq S_1 \times S_2\textit{)}$$

The most interesting definition is probably linear negation $P^\perp$, which can be defined as the implication $P \multimap \perp$. However, in the setting of monotonic predicate transformers, the definition can be simplified into:

▷ **Definition 7.1.5:** if P is a (monotonic) predicate transformer on S, define $P^\perp$ to be the following predicate transformer on S:

$$P^\perp(x) \quad \triangleq \quad \complement \cdot P \cdot \complement(x) \ . \qquad \textit{(where } \complement \textit{ denotes complementation with respect to } S\textit{)}$$

§ *Link with Interaction Systems.* We have the following representation lemma:

○ **Lemma 7.1.6:** the operations defined above on predicate transformers correspond to the operations with same name on interaction systems:

$$\begin{array}{lcl} \mathbf{0}^\circ & = & \mathbf{0} \\ \mathbf{skip}^\circ & = & \mathbf{skip} \\ (w_1 \oplus w_2)^\circ & = & w_1^\circ \oplus w_2^\circ \\ (w_1 \otimes w_2)^\circ & = & w_1^\circ \otimes w_2^\circ \\ (w^\perp)^\circ & = & (w^\circ)^\perp \end{array}$$

proof: let's only check the interesting points: tensor and negation.

→ tensor, $\subseteq$ direction:

$(s_1, s_2) \in (w_1 \otimes w_2)^\circ(r)$

$\quad \Leftrightarrow \quad$ { definition of $\_^\circ$ }

$\exists a \in (w_1 \otimes w_2).A\big((s_1, s_2)\big) \forall d \in (w_1 \otimes w_2).D\big((s_1, s_2), a\big)$

[1]: Note that like the definition of $\otimes$, the definition of $\multimap$ is impredicative in the sense that it uses quantification on subsets.

$(w_1 \otimes w_2).n\big((s_1, s_2), a, d\big) \in r$
$\quad \Leftrightarrow \quad$ { definition of $\otimes$ on interaction systems }
$\big(\exists a_1 \in A_1(s_1)\big)\big(\exists a_2 \in A_2(s_2)\big)\big(\forall d_1 \in D_1(s_1, a_1)\big)\big(\forall d_2 \in D_2(s_2, a_2)\big)$
$\big(s_1[a_1/d_1], s_2[a_2/d_2]\big) \in r$
$\quad \Leftrightarrow$
$\big(\exists a_1 \in A_1(s_1)\big)\big(\exists a_2 \in A_2(s_2)\big)$
$\big\{s_1[a_1/d_1] \mid d_1 \in D_1(s_1, a_1)\big\} \times \big\{s_2[a_2/d_2] \mid d_2 \in D_2(s_2, a_2)\big\} \subseteq r$
$\quad \Rightarrow \quad$ { definition of $\otimes$ on predicate transformer, }
{ with the fact that $s_1 \in w_1^\circ\big(\{s_1[a_1/d_1] \mid d_1 \in D_1(s_1, a_1)\}\big)$ }
$(s_1, s_2) \in w_1^\circ \otimes w_2^\circ(r)$

- tensor, $\supseteq$ direction:
$(s_1, s_2) \in w_1^\circ \otimes w_2^\circ(r)$
$\quad \Rightarrow \quad$ { definition: for some $x_1 \times x_2 \subseteq r$ }
$s_1 \in w_1^\circ(x_1)$ and $s_2 \in w_2^\circ(x_2)$
$\quad \Leftrightarrow$
$\big(\exists a_1 \in A_1(s_1)\big)\big(\forall d_1 \in D_1(s_1, a_1)\big)\ s_1[a_1/d_1] \in x_1$
and $\big(\exists a_2 \in A_2(s_2)\big)\big(\forall d_2 \in D_2(s_2, a_2)\big)\ s_2[a_2/d_2] \in x_2$
$\quad \Rightarrow \quad$ { because $x_1 \times x_2 \subseteq r$ }
$\big(\exists a_1 \in A_1(s_1)\big)\big(\exists a_2 \in A_2(s_2)\big)\big(\forall d_1 \in D_1(s_1, a_1)\big)\big(\forall d_2 \in D_2(s_2, a_2)\big)$
$\big(s_1[a_1/d_1], s_2[a_2/d_2]\big) \in r$
$\quad \Leftrightarrow$
$(s_1, s_2) \in (w_1 \otimes w_2)^\circ(r)$

- negation: we have already seen (lemma 2.5.4) that $w^{\perp\circ} = w^\bullet$. We thus need to show that $w^\bullet = \mathbb{C} \cdot w^\circ \cdot \mathbb{C}$
$s \in \mathbb{C} \cdot w^\circ \cdot \mathbb{C}(x)$
$\quad \Leftrightarrow$
$\neg\big(\exists a \in A(s)\big)\big(\forall d \in D(s, a)\big)\ s[a/d] \in \mathbb{C}x$
$\quad \Leftrightarrow$
$\big(\forall a \in A(s)\big)\big(\exists d \in D(s, a)\big)\ \neg s[a/d] \in \mathbb{C}x$
$\quad \Leftrightarrow$
$\big(\forall a \in A(s)\big)\big(\exists d \in D(s, a)\big)\ s[a/d] \in x$
$\quad \Leftrightarrow$
$s \in w^\bullet(x)$

☑

We can also check that the definition of the linear arrow corresponds to the "real" linear arrow:

- **Lemma 7.1.7**: for any predicate transformers P_1 and P_2, we have

$$P_1 \multimap P_2 \quad = \quad (P_1 \otimes P_2^\perp)^\perp \ .$$

From this, we can conclude that

$$(w_1 \multimap w_2)^\circ \quad = \quad w_1^\circ \multimap w_2^\circ \ .$$

proof:

$$
\begin{array}{l}
(s_1, s_2) \in P_1 \multimap P_2(r) \\
\quad \Leftrightarrow \\
(\forall x_1 \subseteq S_1)\ s_1 \in P_1(x_1) \Rightarrow s_2 \in P_2\big(r(x_1)\big) \\
\quad \Leftrightarrow \quad \{ \text{ logic } \} \\
(\forall x_1 \subseteq S_1)\ s_1 \notin P_1(x_1) \vee s_2 \in P_2\big(r(x_1)\big) \\
\quad \Leftrightarrow \quad \{ \text{ logic: } \Rightarrow \text{ direction: monotonicity of } P_2; \quad \} \\
\qquad\quad \{ \qquad\ \Leftarrow \text{ direction: specializing for } x_2 \triangleq r(x_1) \ \} \\
(\forall x_1 \subseteq S_1, x_2 \subseteq S_2)\ \big(r(x_1) \subseteq x_2\big) \Rightarrow s_1 \notin P_1(x_1) \vee s_2 \in P_2(x_2) \\
\quad \Leftrightarrow \quad \{ \text{ put } y \triangleq \complement x_2, \text{ equivalence } r(x_1) \subseteq \complement x_2 \text{ iff } x_1 \times x_2 \subseteq \complement r \ \} \\
(\forall x_1 \subseteq S_1, y \subseteq S_2)\ \big(x_1 \times y \subseteq \complement r\big) \Rightarrow s_1 \notin P_1(x_1) \vee s_2 \in P_2(\complement y) \\
\quad \Leftrightarrow \quad \{ \text{ logic } \} \\
\neg(\exists x_1 \subseteq S_1, y \subseteq S_2)\ \big(x_1 \times y \subseteq \complement r\big) \wedge s_1 \in P_1(x_1) \wedge s_2 \notin P_2(\complement y) \\
\quad \Leftrightarrow \\
\neg(\exists x_1 \subseteq S_1, y \subseteq S_2)\ \big(x_1 \times y \subseteq \complement r\big) \wedge s_1 \in P_1(x_1) \wedge s_2 \in P_2^{\perp}(y) \\
\quad \Leftrightarrow \quad \{ \text{ definition of } \otimes \text{ and } \_^{\perp} \ \} \\
(s_1, s_2) \in (P_1 \otimes P_2^{\perp})^{\perp}(r)
\end{array}
$$

The second point follows directly from the following facts

- for interaction systems, we have $w_1 \multimap w_2 \simeq (w_1 \otimes w_2^{\perp})^{\perp}$ (lemma 6.3.2);
- $\mathbf{Id}_S$ is an isomorphism from w to w' iff $w^{\circ} = w'^{\circ}$ (lemma 2.5.21).

☑

§ *Safety Properties, $\star$-Autonomy.* Even though it is equivalent to the category of interaction systems with simulations, the category of predicate transformers and forward data-refinements is slightly simpler. The reason is that we have replaced many isomorphisms (bisimilarity of interaction system) by plain (extensional) equality. For example, that negation is involutive is totally trivial. Let's first give another characterization of forward data-refinements (definition 2.5.20 on page 60).

▷ **Definition 7.1.8:** if P is a predicate transformer on S, a *safety property for* P is a subset $x \subseteq S$ satisfying $x \subseteq P(x)$. We write $\mathcal{S}(P)$ for the collection of safety properties for P.

Thus, "safety property" is just a synonym for "invariant predicate" (definition 2.5.15). If we have the intuition that a predicate transformer is an abstract specification (page 50), then a safety property is a set of states $x \subseteq S$ with the following property: for any program satisfying the specification,

> *if execution is started from a state in x, then execution will terminate, and the final state will also be in x.*

We have:

○ **Lemma 7.1.9:** a relation $r \subseteq S_1 \times S_2$ is a forward data-refinement from P_1 to P_2 iff $r \in \mathcal{S}(P_1 \multimap P_2)$.

proof: suppose first that r is a a safety property for $P_1 \multimap P_2$:

$$
\begin{array}{l}
s_2 \in r \cdot P_1(x) \\
\quad \Rightarrow \quad \{ \text{ for some } s_1, \ \} \\
(s_1, s_2) \in r \quad \wedge \quad s_1 \in P_1(x)
\end{array}
$$

$\Rightarrow$ { r is a safety property for $P_1 \multimap P_2$ }
$(s_1, s_2) \in P_1 \multimap P_2(r) \quad \wedge \quad s_1 \in P_1(x)$
$\Leftrightarrow$
$(\forall x)\ s_1 \in P_1(x) \Rightarrow s_2 \in P_2\big(r(x)\big) \quad \wedge \quad s_1 \in P_1(x)$
$\Rightarrow$
$s_2 \in P_2\big(r(x)\big)$
which shows that $r \cdot P_1 \subseteq P_2 \cdot r$.

Conversely, suppose $r \cdot P_1 \subseteq P_2 \cdot r$, and let $(s_1, s_2) \in r$. We are going to show that $(s_1, s_2) \in P_1 \multimap P_2(r)$. If $s_1 \in P_1(x)$, we have $a_2 \in r \cdot P_1(x)$, which implies by hypothesis that $s_2 \in P\big(r(x)\big)$.

☑

Since we know by proposition 2.5.23 that interaction systems with simulations and predicate transformers with forward data-refinements are weakly equivalent categories, no confusion really arises from the following definition:

▷ **Definition 7.1.10:** an *interface* is a pair (S, P) where S is a set and P a monotonic predicate transformer on S. The category of interfaces with forward data-refinements is called **Int**..

Moreover:

○ **Lemma 7.1.11:** the operations $\oplus$, $\otimes$ and $\_^\perp$ defined earlier can be lifted to functors on **Int**.

proof: easy. Let's look at the case of negation: first notice that we have the following:

$$\complement \cdot [r] \quad = \quad \langle r \rangle \cdot \complement \qquad \text{and} \qquad \complement \cdot \langle r \rangle \quad = \quad [r] \cdot \complement\ .$$

So that we can define the action of $\_^\perp$ on morphisms as $r^\perp \triangleq r^\sim$:
$\langle r^\sim \rangle \cdot P_1 \subseteq P_2 \cdot \langle r^\sim \rangle$
$\Rightarrow$
$\complement \cdot P_2 \cdot \langle r^\sim \rangle \cdot \complement \subseteq \complement \cdot \langle r^\sim \rangle \cdot P_1 \cdot \complement$
$\Leftrightarrow$
$\complement \cdot P_2 \cdot \complement \cdot [r^\sim] \subseteq [r^\sim] \cdot \complement \cdot P_1 \cdot \complement$
$\Leftrightarrow$
$P_2^\perp \cdot [r^\sim] \subseteq [r^\sim] \cdot P_1^\perp$
$\Rightarrow$
$\langle r \rangle \cdot P_2^\perp \cdot [r^\sim] \cdot \langle r \rangle \subseteq \langle r \rangle \cdot [r^\sim] \cdot P_1^\perp \cdot \langle r \rangle$
$\Rightarrow$ { by lemma 2.5.11, $[r^\sim] \cdot \langle r \rangle \supseteq \mathbf{Id}$ and $\langle r \rangle \cdot [r^\sim] \subseteq \mathbf{Id}$ }
$\langle r \rangle \cdot P_2^\perp \subseteq P_1^\perp \cdot \langle r \rangle$
$\Leftrightarrow$
$r^\sim$ is a forward data-refinement from $P_2^\perp$ to $P_1^\perp$.

☑

Just like the category of interaction systems, the category of interfaces is a model of MALL. However, in this new context, the result is much simpler to prove:

◇ **Proposition 7.1.12:** *the constructions just defined make* **Int** *into a $\star$-autonomous category.*

proof: let's start with the fact that $\multimap$ is right-adjoint to $\otimes$:

$$
\begin{array}{ll}
& r \in \mathbf{Int}(P_1 \otimes P_2, P_3) \\
\Leftrightarrow & \{ \text{ lemma 7.1.9 } \} \\
& r \in \mathcal{S}\big((P_1 \otimes P_2) \multimap P_3\big) \\
\Leftrightarrow & \{ \ P \multimap Q = (P \otimes Q^\perp)^\perp \text{ (lemma 7.1.7) } \} \\
& r \in \mathcal{S}\Big(\big((P_1 \otimes P_2) \otimes P_3^\perp\big)^\perp\Big) \\
\Leftrightarrow & \{ \text{ associativity of } \otimes \ \} \\
& r \in \mathcal{S}\Big(\big(P_1 \otimes (P_2 \otimes P_3^\perp)\big)^\perp\Big) \\
\Leftrightarrow & \{ \ P \multimap Q = (P \otimes Q^\perp)^\perp \text{ and } P^{\perp\perp} = P \ \} \\
& r \in \mathcal{S}\Big(P_1 \multimap (P_2 \multimap P_3)\Big) \\
\Leftrightarrow & \{ \text{ lemma 7.1.9 } \} \\
& r \in \mathbf{Int}(P_1, P_2 \multimap P_3)
\end{array}
$$

That $\perp \triangleq \mathbf{skip}$ is a dualizing object is easy: the canonical morphism from P to $P^{\perp\perp}$ is the identity, which is trivially an isomorphism!
We do not bother with the other bureaucratic conditions defining a $\star$-autonomy. They are trivially true.

☑

7.1.2 Exponentials

The definition of the exponential $!\_$ is a little subtler: intuitively, $!P$ should be a kind a arbitrary n-ary tensor $\mathbf{Id} \oplus P \oplus (P \otimes P) \oplus (P \otimes P \otimes P) \oplus \cdots$, quotiented by "shuffling". Just like $\mathcal{M}_f(S)$ is $\bigcup_n S^n$ modulo renaming (definition 5.3.1), so is $!P$ the predicate transformer $\bigoplus_n P^{n\otimes}$ modulo renaming. Define the "commutative product" $\bigotimes_{i\in I} x_i$ of a finite number of subsets of S:

▷ **Definition 7.1.13:** if $(x_i)_{i\in I}$ is a finite family of subsets of S, we define the following collection of multisets:

$$[s_i]_{i\in I} \in \bigotimes_{j\in J} x_j \quad \Leftrightarrow \quad (\exists \sigma : I \xrightarrow{\sim} J)\ (\forall i \in I)\ s_i \in x_{\sigma i} \ .$$

This is just the usual cartesian product modulo reindexing. With this definition, we can define the predicate transformer $!P$:

▷ **Definition 7.1.14:** if P is a monotonic predicate transformer on S, define $!P$ to be the following predicate transformer on $\mathcal{M}_f(S)$:

$$
\begin{array}{l}
[s_1, \ldots, s_n] \in !P(U) \\
\qquad \Leftrightarrow \\
(\exists x_1, \ldots, x_n \subseteq S) \quad \text{s.t.} \quad \Big(\bigotimes_{1\leqslant i\leqslant n} x_i\Big) \subseteq U \quad \wedge \quad (\forall 1 \leqslant i \leqslant n)\, s_i \in P(x_i) \ .
\end{array}
$$

Dually, define $?P \triangleq \big(!(P^\perp)\big)^\perp$.

We have:

○ **Lemma 7.1.15:** for any interaction system w on S,

$$(!w)^\circ \quad = \quad !(w^\circ) \ .$$

proof: the direct proof is straightforward. ☑

As a corollary to this, lemma 6.1.4 and proposition 2.5.23, we obtain:

- **Lemma 7.1.16:**
  - $!\_$ is a comonad on **Int**;
  - $!P$ is the free $\otimes$-comonoid on P;
  - $?\_$ is a monad on **Int**;
  - $?P$ is the free $\parr$-monoid on P;
  - $!(P_1 \,\&\, P_2) \simeq !P_1 \otimes !P_2$.

7.1.3 The Model

We can now detail the denotational model we obtain: start with a valuation of atomic formulas as predicate transformers and use the relational interpretation of proofs described in section 5.3.

- **Proposition 7.1.17:** *if π is a proof of the sequent $\vdash G_1, \ldots, G_n$, then $[\![\pi]\!]$ is a safety property in $G_1^* \parr \ldots \parr G_n^*$.*

Just like in the previous section, the interpretation $[\![\pi]\!]$ does *not* depend on the actual predicate transformers we use for the atoms, but just on their set of states. This remark will be the basis of the model for second order linear logic developed in chapter 8.

proof: this is in essence contained in proposition 7.1.12 and lemma 7.1.16.

For the sake of completeness, here is a direct proof that the interpretation of the tensor rule is functorial. More details can be read in [53].

If π is $\dfrac{\pi_1 \vdash \Gamma, F_1 \qquad \pi_2 \vdash \Delta, F_2}{\vdash \Gamma, \Delta, F_1 \otimes F_2}$

then $[\![\pi]\!] = \big\{ \big(\gamma, \delta, (s_1, s_2)\big) \mid (\gamma, s_1) \in [\![\pi_1]\!] \ \wedge \ (\delta, s_2) \in [\![\pi_2]\!] \big\}$.

Suppose that $[\![\pi_1]\!]$ is a safety property in $\Gamma \parr F_1$ and that $[\![\pi_2]\!]$ is a safety property in $\Delta \parr F_2$. We need to show that $[\![\pi]\!]$ is a safety property in $\Gamma \parr \Delta \parr (F_1 \otimes F_2)$.

$\big(\gamma, \delta, (s_1, s_2)\big) \in [\![\pi]\!]$

$\Rightarrow$

$(\gamma, s_1) \in [\![\pi_1]\!]$ and $(\delta, s_2) \in [\![\pi_2]\!]$

$\Rightarrow$ { $[\![\pi_1]\!]$ and $[\![\pi_2]\!]$ are safety properties in Γ, F_1 and Δ, F_2 }

$(\gamma, s_1) \in \Gamma, F_1([\![\pi_1]\!])$ and $(\delta, [\![\pi_2]\!]) \in \Delta, F_2([\![\pi_2]\!])$.

By contradiction, suppose $\big(\gamma, \delta, (s_1, s_2)\big) \notin \Gamma, \Delta, F_1 \otimes F_2([\![\pi]\!])$

$\Rightarrow$

$\big(\gamma, \delta, (s_1, s_2)\big) \in \Gamma^\perp \otimes \Delta^\perp \otimes (F_1 \otimes F_2)^\perp(\mathbb{C}[\![\pi]\!])$

$\Rightarrow$ { for some $u \times v \times r \subseteq \mathbb{C}[\![\pi]\!]$: }

$\gamma \in \Gamma^\perp(u) \wedge \delta \in \Delta^\perp(v) \wedge \underbrace{(s_1, s_2) \in (F_1 \otimes F_2)^\perp(r)}$

$\Rightarrow$

$\ldots \quad \wedge \Big((\forall x \times y \subseteq \mathbb{C}r) \ s_1 \in F_1^\perp(\mathbb{C}x) \vee s_2 \in F_2^\perp(\mathbb{C}y) \Big)$.

In particular, define $x = \langle \pi_1^{\smile} \rangle u$ and $y = \langle \pi_2^{\smile} \rangle v$. It is easy to show that $x \times y \subseteq \mathbb{C}r$, so that we have $s_1 \in F_1^{\perp}(\mathbb{C}x)$ or $s_2 \in F_2^{\perp}(\mathbb{C}y)$.
Suppose $s_1 \in F_1^{\perp}(\mathbb{C}x)$: we have $\gamma \in \Gamma^{\perp}(u)$ and $u \times \mathbb{C}x \subseteq \mathbb{C}[\![\pi_1]\!]$ (easy lemma), so by definition, $(\gamma, s_1) \in \Gamma^{\perp} \otimes F_1^{\perp}(\mathbb{C}[\![\pi_1]\!])$, *i.e.* $(\gamma, s_1) \notin \Gamma, F_1([\![\pi_1]\!])$! This is a contradiction.
Similarly, one can derive a contradiction from $s_2 \in F_2^{\perp}(\mathbb{C}y)$.
This finishes the proof that $[\![\pi]\!]$ is a safety property for $\Gamma, \Delta, F_1 \otimes F_2$.

We can make a simpler but more abstract proof by noting that if $r_i : \Gamma^{\perp} \multimap F_i$, then $r_1 \otimes r_2 : \Gamma_1 \otimes \Gamma_2 \multimap F_1 \otimes F_2 = \Gamma_1 \parr \Gamma_2 \parr (F_1 \otimes F_2)$.

☑

7.1.4 The Problem of Constants

This model, together with its interaction system variant, suffers from a small degeneracy: if the atoms are interpreted by trivial objects, then all the formulas will be trivial. This is in particular the case when the only atomic formulas are the constants: we obtain a subcategory of **Int** isomorphic to its relational counterpart.

- **Lemma 7.1.18:** if F is a linear formula without propositional variables, then its interpretation $F : \mathcal{P}(|F|) \to \mathcal{P}(|F|)$ is the identity.

proof: simple induction.

☑

Similarly, when adding atoms, one needs to be careful not to chose a too simple valuation:

- **Lemma 7.1.19:** suppose each atom is interpreted by an interface of the form $(S_i, \langle \mathbf{gr}(g_i) \rangle)$ where $g_i : S_i \xrightarrow{\sim} S_i$, then for any linear formula F, we have:
  - the interpretation of F is of the form $\langle \mathbf{gr}(f) \rangle$ where $f : |F| \xrightarrow{\sim} |F|$;
  - the interpretation of F is equal to the interpretation of $F^{\perp}$.

proof: simple induction.

☑

A similar phenomenon happens when the atoms are of the form $\langle r \rangle$ for a functional relation r. (In this case, $\langle r \rangle^{\perp} = [r] = \langle r \rangle$.)

↬ REMARK 21: note that even though the model is degenerated in the case of lemma 7.1.19 ($F = F^{\perp}$), it can still be of interest: for example, we can interpret an atom by:

$$\begin{array}{lcl} P & : & \mathcal{P}(\mathbf{B}) \to \mathcal{P}(\mathbf{B}) \\ P & \triangleq & \mathbf{gr}(\neg) \ . \end{array}$$

We have that $x \triangleq \{(\mathrm{True}, \mathrm{False}), (\mathrm{False}, \mathrm{True})\} \in \mathcal{S}(P \otimes P)$. This shows that a safety property in $P_1 \otimes P_2$ needs not contain a product of safety properties in P_1 and P_2.

7.1.5 Specification Structures

In [2], the authors define the notion of *specification structure,* a categorical notion bringing Hoare logic to the realm of categories. Recall the definition:

▷ **Definition 7.1.20:** if $\mathcal{C}$ is a category, a *specification structure on* $\mathcal{C}$ is given by the following data:
- for each object A of $\mathcal{C}$, a collection $P(A)$ of "properties over A";
- for any pair (A, B) of objects of $\mathcal{C}$, a relation $S_{A,B} \subseteq P_A \times \mathcal{C}(A, B) \times P_B$.

We write $\varphi\{f\}\psi$ for $(\varphi, f, \psi) \in S_{A,B}$ and require the following:

$$\varphi\{\mathbf{Id}_A\}\varphi$$
$$\varphi\{f\}\psi \text{ and } \psi\{g\}\theta \quad \Rightarrow \quad \varphi\{g \cdot f\}\theta$$

for all objects A, B and C, morphisms $f \in \mathcal{C}(A, B)$ and $g \in \mathcal{C}(B, C)$ and properties $\varphi \in P_A$, $\psi \in P_B$ and $\theta \in P_C$.

A specification structure on $\mathcal{C}$ forms a category $\mathcal{C}_S$ by taking:
- for objects, pairs (A, φ) where $\varphi \in P_A$;
- and for morphisms from (A, φ) to (B, ψ), the collection of morphisms f in $\mathcal{C}(A, B)$ s.t. $\varphi\{f\}\psi$.

○ **Lemma 7.1.21:** there is a faithful functor from any specification structure $\mathcal{C}_S$ to $\mathcal{C}$:

$$\begin{array}{lcl} (A, \varphi) & \mapsto & A \\ f & \mapsto & f\,. \end{array}$$

Conversely, for any faithful functor $F : \mathcal{D} \to \mathcal{C}$, there is a specification structure $\mathcal{C}_S$ equivalent to $\mathcal{D}$ s.t. F is equivalent to the faithful functor defined above.

proof: easy. For the second point, define the specification structure $\mathcal{C}_S$ by taking:
- $P_A \triangleq \{\varphi \epsilon \mathcal{D} \mid F(\varphi) = A\}$;
- and $\varphi\{f\}\psi$ iff $\big(\exists \alpha \in \mathcal{D}(\varphi, \psi)\big)\, F_\alpha = f$.

☑

The notion of specification structure can be extended to take into account some of the structure of $\mathcal{C}$. In particular, if $\mathcal{C}$ is a model of linear logic, we can require $\mathcal{C}_S$ to have a compatible structure. (See [2].)

○ **Lemma 7.1.22:** the category **Int** can be seen as a specification structure compatible with the linear structure of the category **Rel**:
- if A is a set, P_A is the collection of predicate transformers on A;
- if r is a relation between A and B, $P\{r\}Q$ iff $r \cdot P \subseteq Q \cdot r$.

Thus, the denotational model presented above can be seen as a particular instantiation of the theory sketched in the second section of [2]. However, it would be unfair to reduce **Int** to that: defining a concrete specification structure on **Rel** which is compatible with the linear structure is not a trivial exercise. Moreover the category **Int** is particularly interesting because it is constructed from concrete, well-known ingredients.

While we are mentioning [2], it could be interesting to compare the two approaches to obtain "deadlock freeness" (see chapter 6 in this thesis and section 5 in [2]). In essence, a morphism in the case of [2] is a process *with a guarantee that it will communicate with all possible (or interesting) processes.* The guarantee is necessary because in their context, the composition of deadlock free processes is not necessarily deadlock free. In our case, the problem is irrelevant since the composition of Angel-deadlock free processes is Angel-deadlock free.[2] A more thorough investigation about the relationship between the category **Int** and the work of [2] would probably be interesting but is yet to be done.

7.1.6 Injectivity of the Commutative Product

This small section is irrelevant to the purpose of interaction systems or linear logic. The idea is simple: the usual cartesian product $\prod_i A_i$ enjoys the following trivial injectivity property: if $A_1 \times \cdots \times A_n = B_1 \times \cdots \times B_n \neq \emptyset$, then $A_i = B_i$ for all i's. One can say that $\prod : \textsc{List} \cdot \mathcal{P}(S) \to \mathcal{P} \cdot \textsc{List}(S)$ is "almost injective". The operation $\bigotimes : \mathcal{M}_f \cdot \mathcal{P}(S) \to \mathcal{P} \cdot \mathcal{M}_f(S)$ used in the definition of the exponential can be seen as a commutative version of cartesian product, and it does enjoy the same injectivity property.[3]

- **Lemma 7.1.23:** if $(A_i)_{i \in I}$ and $(B_i)_{i \in I}$ are two finite families of non-empty subsets of S, and if $\bigotimes_i A_i = \bigotimes_i B_i$ then $(A_i)_{i \in I}$ and $(B_i)_{i \in I}$ are equivalent up to reindexing. (The multisets $[A_i]_{i \in I}$ and $[B_i]_{i \in I}$ are equal.)

The proof goes as follows: suppose $\bigotimes_i A_i = \bigotimes_j B_j = P$,

- $(A_i)_i$ and $(B_j)_j$ have one set in common: $A_{i_0} \in (A_i)_i$ and $A_{i_0} = B_{j_0} \in (B_j)_j$;
- we define an operation of *division* such that $\left(\bigotimes_i A_i\right)/A_{i_0} = \bigotimes_{i \neq i_0} A_i$;
- this implies that $\bigotimes_{i \neq i_0} A_i = P/A_{i_0} = P/B_{j_0} = \bigotimes_{j \neq j_0} B_j$;
- a trivial induction concludes the proof.

→ We first have:

$$\bigotimes_i A_i \subseteq \bigotimes_j B_j \quad \Rightarrow \quad (\forall j)(\exists i)\ A_i \subseteq B_j \ . \tag{7-1}$$

By contradiction, suppose that $(\exists j)(\forall i)\ \neg(A_i \subseteq B_j)$. Let j_0 be such a j. We have that $(\forall i)(\exists a_i \in A_i)\ a_i \notin B_{j_0}$. This implies that $[a_i]_i \in \bigotimes_i A_i$, but $[a_i]_i$ cannot be in $\bigotimes_j B_j$! Contradiction.

We can deduce that:

$$\bigotimes_i A_i = \bigotimes_j B_j \quad \Rightarrow \quad (\exists i, j)\ A_i = B_j \ .$$

By (7-1), we can construct an infinite chain $A_{i_1} \supseteq B_{j_1} \supseteq \ldots \supseteq A_{i_n} \supseteq B_{j_n} \ldots$ Since there is only a finite number of A_i's and B_j's, there must be a cycle. This implies that some $A_{i_n} = B_{j_n}$.

---

[2]: This gives yet another argument for distinguishing the program from its environment.

[3]: For the category inclined reader, this "commutative product" is the usual distributivity law from the monad $\mathcal{M}_f$ to the monad $\mathcal{P}_f$ in the category **Set**, just like $\prod$ is the distributivity law between the monads List and $\mathcal{P}$.

→ We can now define the operation of division, and prove its property:

▷ **Definition 7.1.24:** for any $E \subseteq \mathcal{M}_f(S)$, define:
1) for $a \in S$: $E/a = \{\mu \mid \mu + [a] \in E\}$;
2) for $A \subseteq S$: $E/A = \bigcap_{a \in A} E/a$.

It satisfies

○ **Lemma 7.1.25:** for any $A_0, \ldots, A_N$ non-empty subsets of S, we have:

$$\left(\bigotimes_{0 \leqslant i \leqslant N} A_i \right) / A_0 \quad = \quad \bigotimes_{1 \leqslant i \leqslant N} A_i \ .$$

proof: the $\supseteq$ inclusion is immediate. Let's show the converse inclusion.
Let $[a_1, \ldots, a_N] \in \left(\bigotimes_{0 \leqslant i \leqslant N}\right)/A_0$. We prove that $[a_i]_i \in \bigotimes_{1 \leqslant i \leqslant N} A_i$ by contradiction. Suppose that $[a_i]_i \notin \bigotimes_{1 \leqslant i \leqslant N} A_i$.
Let $a \in A_0$, we have $[a, a_1, \ldots, a_N] \in \bigotimes_{0 \leqslant i \leqslant N} A_i$. Since $[a_i]_i \notin \bigotimes_{1 \leqslant i \leqslant N}$, one of the a_i must be in A_0. Without loss of generality, we can suppose $a_1 \in A_0$, $a \in A_1$ and $a_i \in A_i$ for all $i \geqslant 2$.

Since $a_1 \in A_0$, we have $[a_1, a_1, a_2, \ldots, a_N] \in \bigotimes_{0 \leqslant i \leqslant N} A_i$, *i.e.* $a_{\sigma i} \in A_i$ for some bijection $\sigma : \{0, \ldots, N\} \to \{0, \ldots, N\}$. (We put $a_0 \triangleq a_1$.)

Define (k_i) by induction as follows:
- $k_0 = \sigma(0)$;
- $k_{i+1} = \sigma(k_i)$.

Let $K = \min\{i \mid k_i = 0 \text{ or } k_i = 1\}$. It is not difficult to show that such a K exists.
Put now $I = \{k_0, \ldots, k_K\}$.
Now, rearrange the columns of the following table:

$$\begin{array}{cccccccc} \overbrace{\phantom{A_0\ A_1\ \ldots\ A_l\ \ldots\ A_{l'}\ \ldots\ A_N}}^{\{0,\ldots,N\}} & & & & & & & \\ A_0 & A_1 & \ldots & A_l & \ldots & A_{l'} & \ldots & A_N \\ a_{\sigma 0} & a_{\sigma 1} & \ldots & a_1 & \ldots & a_1 & \ldots & a_{\sigma N} \\ \underbrace{\phantom{a_{\sigma 0}\ a_{\sigma 1}\ \ldots\ a_1\ \ldots\ a_1\ \ldots\ a_{\sigma N}}}_{\{1,1,\ldots,N\}} & & & & & & & \end{array}$$

into

$$\begin{array}{ccccccc} \overbrace{\phantom{A_0\ A_{k_0}\ \ldots\ A_{k_{K-1}}\ A_{k_K}}}^{\{0\}\cup I=\{0,k_0,\ldots,k_K\}} & & & & & \overbrace{}^{\{1,\ldots,N\}\setminus I=\bar{I}} & \\ A_0 & A_{k_0} & \ldots & A_{k_{K-1}} & A_{k_K} & \ldots\ A_l & \ldots \\ a_{k_0} & a_{k_1} & \ldots & a_{k_K} & a_1 & \ldots\ a_1 & \ldots \\ & \underbrace{\phantom{a_{k_0}\ a_{k_1}\ \ldots\ a_{k_K}}}_{\{1\}\cup I=\{1,k_0,\ldots,k_K\}} & & & & \underbrace{}_{\{1,\ldots,N\}\setminus I=\bar{I}} & \end{array}$$

From this (right hand part), we can deduce that $[a_i]_{i \in \bar{I}} \in \bigotimes_{i \in \bar{I}} B_i$. By hypothesis, we also have that $[a_i]_{i \in I} \in \bigotimes_{i \in I} A_i$ (because $1 \notin I$).
This implies that $[a_i]_{1 \leqslant i \leqslant N} \in \bigotimes_{1 \leqslant i \leqslant N} A_i$! Contradiction.

☑

→ The proof of lemma 7.1.23 is now immediate: by induction on N.
- $N = 0$: trivial;
- $N > 0$: suppose $\bigotimes A_i = \bigotimes B_i$. This implies that $[A_i]_{i \leqslant N}$ and $[B_i]_{i \leqslant N}$ are in fact of the form $[C] + [A_i]_{i<N}$ and $[C] + [B_i]_{i<N}$.
Apply lemma 7.1.25 to get $\bigotimes_{i<N} A_i = \bigotimes_{i<N} B_i$, and then the induction hypothesis to get $[A_i]_{i<N} = [B_i]_{i<N}$.

From this, we can easily conclude that $[C] + [A_i]_{i<N} = [C] + [B_i]_{i<N}$.
To finish on this operation of commutative product, let's mention that inclusion of products does *not* enjoy a the same property: $\bigotimes_i A_i \subseteq \bigotimes_i B_i$ *does not* imply that $[A_i]_i \subseteq [B_i]_i$ (pointwise). For example, if we write the binary commutative product with a $*$, we have $(A \cap B) * (A \cup B) \subseteq A * B$!

7.2 A Nice Restriction: Finitary Predicate Transformers

The collection of *all* monotonic predicate transformers on a set S is huge.[4] It is thus natural to see if we can find subcategories of **Int** which are still models of full linear logic.

The first idea is to require predicate transformers to commute with arbitrary unions. By proposition 2.5.8, we know that this amounts to considering predicate transformers of the form $\langle r \rangle$ for some relation r. The problem is that we also want $\langle r \rangle^\perp = [r]$ to be of this form. This implies that r is functional, and just like lemma 7.1.19, we obtain a degenerate model where the interpretation of F is equal to the interpretation of $F^\perp$.

A less demanding property is to ask for the predicate transformers to satisfy:

- $P(\emptyset) = \emptyset$;
- $P(S) = S$.

It is easy to check that all the constructions respect this property. It is however not entirely satisfactory for the simple reason that the full set of states is *always* a safety property. In particular, the formula $P \multimap P$ has at least two non-empty safety properties as soon as the set of states has cardinality two.

A more interesting property is to require that the predicate transformers are both Scott continuous and "cocontinuous". It turns out that this notion corresponds exactly to the notion of *finitary interaction system*.

▷ **Definition 7.2.1:** an interaction system $w = (A, D, n)$ on S is *finitary* if $A(s)$ and $D(s, a)$ are finite for all $s \in S$ and $a \in A(s)$.

The goal of this section is to prove that:

◇ **Proposition 7.2.2:** *for any monotonic predicate transformer* P *on* S*, the following are equivalent:*

1) P *commutes with directed intersections and unions;*[5]
2) P *is of the form* w° *for a finitary* w*;*
3) P *is continuous for the Cantor topology on* $\mathcal{P}(S)$.

[4]: The situation is even worse for interaction systems, since the collection of interaction systems over a set S forms a proper class!

[5]: where a directed intersection is the intersection of a "codirected" set U: whenever $x, y \in U$, there is a $z \in U$ s.t. $z \subseteq x$ and $z \subseteq y$.

Remark that monotonicity isn't implied by Cantor continuity: complementation *is* Cantor continuous, but hardly ever monotonic!

proof:

- we start with *2* $\Rightarrow$ *1*: suppose w is a finitary interaction system. Scott continuity is equivalent to $s \in P(x) \Rightarrow s \in P(x_0)$ for some finite $x_0 \subseteq x$. Suppose $s \in w^\circ(x)$, *i.e.* we have some $a \in A(s)$ s.t. $s[a/d] \in x$ for all $d \in D(s,a)$. Thus, we have that $\{s[a/d] \mid d \in D(s,a)\}$ is finite, included in x and its image contains s. This shows that w° is Scott continuous. For cocontinuity, it suffices to note that P is cocontinuous iff $P^\perp$ is continuous.

- We now show that *2* $\Rightarrow$ *3*, *i.e.* that w° is Cantor continuous as soon as w is finitary.

  Before that, let's give some notation: the Cantor topology on $\mathcal{P}(S)$ is given by the product topology when seeing $\mathcal{P}(S)$ as the product of S copies of the discrete topology on $\mathbf{B}$. A basis for this topology is given by the collections of all the $O_{x,y}$'s, where x and y are disjoint finite sets, and:

$$O_{x,y} \triangleq \{u \mid x \subseteq u \text{ and } y \cap u = \emptyset\} .$$

  An open set is simply an arbitrary union of such basic opens.[6] Note that a prebase is given by the simpler collection $\{O_{\emptyset,\{s\}}, O_{\{s\},\emptyset} \mid s \in S\}$.

  Suppose that w is finitary, let's show that w° (written w from now on) is continuous, *i.e.* that w^{-1} maps open set to open sets, or since w^{-1} commutes with arbitrary unions, that w^{-1} maps basic opens to opens: let $O_{x,y}$ be a basic open:

$u \in w^{-1}(O_{x,y})$
$\Leftrightarrow$
$w(u) \in O_{x,y}$
$\Leftrightarrow$
$x \subseteq w(u)$ and $y \cap w(u) = \emptyset$
$\Leftrightarrow$
$(\forall s \in x)(\exists a_s \in A(s))(\forall d \in D(s,a_s))\ s[a_s/d] \in u$ and
$(\forall s \in y)(\forall a \in A(s))(\exists d_a \in D(s,a))\ s[a/d_a] \notin u$
$\Rightarrow$ { define $x' = \{s[a_s/d] \mid s \in x, d \in D(s,a_s)\}$ }
{ and $y' = \{s[a/d_a] \mid s \in y, a \in A(s)\}$; }
{ both x' and y' are finite because x and y are finite and w is finitary }
$x' \subseteq u$ and $y' \cap u = \emptyset$
$\Leftrightarrow$
$u \in O_{x',y'}$.

  Define now the open set $U_{x,y}$ as:

$$U_{x,y} \triangleq \bigcup_{\substack{a_s:(s\in u)\to A(s) \\ d_s:(s\in y, a\in A(s))\to D(s,a)}} O_{x',y'}$$

  where x' and y' are defined as above. It is easy to check that $u \in w^{-1}(O_{x,y})$ iff $u \in U_{x,y}$, which shows that w° is Cantor continuous.

[6]: This topology coincide with the *Lawson topology* on the continuous domain $\mathcal{P}(S)$.

- The converse direction (*3* $\Rightarrow$ *2*) relies on the fact that the Cantor topology on $\mathcal{P}(S)$ is compact (by Tikhonov theorem): suppose P is Cantor continuous, and suppose $s \in S$. We want to find a finite set $A(s)$ and a family $\big(D(s,a)\big)_{a \in A(s)}$ of finite sets satisfying the condition

$$s \in P(U) \quad \Leftrightarrow \quad \big(\exists a \in A(s)\big)\ D(s,a) \subseteq U\ .$$

Once this is done, we can define an the corresponding interaction system by putting:

$$\begin{array}{lcl} A_P(s) & \triangleq & A(s) \\ D_P(s,a) & \triangleq & D(s,a) \\ n_P(s,a,s') & \triangleq & s'\ . \end{array}$$

Since P is Cantor continuous, we know that $P^{-1}\big(O_{\{s\},\emptyset}\big)$ and $P^{-1}\big(O_{\emptyset,\{s\}}\big)$ are open sets. We can write them as a union of (possibly infinitely many) basic opens:

$$P^{-1}\big(O_{\{s\},\emptyset}\big) \quad = \quad \bigcup \mathcal{U} \qquad \text{and} \qquad P^{-1}\big(O_{\emptyset,\{s\}}\big) \quad = \quad \bigcup \mathcal{U}'\ .$$

Since we have, for any subset u

$$\begin{array}{lcl} u \in P^{-1}\big(O_{\{s\},\emptyset}\big) & \Leftrightarrow & s \in P(u) \\ u \in P^{-1}\big(O_{\emptyset,\{s\}}\big) & \Leftrightarrow & s \notin P(u)\ , \end{array}$$

we have that $\mathcal{U} \cup \mathcal{U}'$ is a covering of $\mathcal{P}(S)$. By compactness, we can extract a finite covering from it: we write

- $\big(O_{x_i,y_i}\big)_{i \in I}$ for the subcovering of $\mathcal{U}$;
- and $\big(O_{x'_j,y'_j}\big)_{j \in J}$ for the subcovering of $\mathcal{U}'$.

We have:

$$\begin{array}{lcl} s \in P(u) & \Leftrightarrow & (\exists i \in I)\ u \in O_{x_i,y_i} \\ & \Leftrightarrow & (\exists i \in I)\ x_i \subseteq u \wedge u \cap y_i = \emptyset \end{array}$$

which shows we can take $A(s) \triangleq I$ and $D(s,i) \triangleq x_i$.

- Finally, we show that *1* $\Rightarrow$ *3*, *i.e.* that P is Scott continuous and Scott cocontinuous implies that P is Cantor continuous: we will show that the inverse image of a basic open is an open set: let $O_{x,y}$ be a basic open:

$u \in P^{-1}(O_{x,y})$

$\Leftrightarrow$

$x \subseteq P(u)$ and $y \cap P(u) = \emptyset$

$\Leftrightarrow$

$(\forall s \in x)\ s \in P(u)$ and $(\forall s \in y)\ s \notin P(u)$

$\Leftrightarrow$ { by Scott (co)continuity, where all x_s and y_s are finite }

$(\forall s \in x)(\exists x_s \subseteq u)\ s \in P(x_s)$ and $(\forall s \in y)(\exists y_s \subseteq \complement u)\ s \notin P(y_s)$

$\Rightarrow$ { define $x' \triangleq \bigcup_{s \in x} x_s$ and $y' \triangleq \bigcup_{s \in y} y_s$: both x' and y' are finite }

$x' \subseteq u$ and $y' \cap u = \emptyset$

$\Leftrightarrow$

$u \in O_{x',y'}$.

Now, define the open set $U_{x,y}$ as:

$$U_{x,y} \triangleq \bigcup_{\substack{(x_s)_{s\in x} \\ (y_s)_{s\in y}}} O_{x',y'}$$

where x' and y' are defined as above, and we quantify over families $(x_s)_{s\in x}$ and $(y_s)_{s\in y}$ of finite sets satisfying:

$$(\forall s \in x)\ x_s \subseteq u \wedge s \in P(x_s) \qquad \text{and} \qquad (\forall s \in y)\ y_s \subseteq \complement u \wedge s \notin P(y_s)\ .$$

It is straightforward to check that $u \in P^{-1}(O_{x,y})$ iff $u \in U_{x,y}$, thus showing that P is Cantor continuous.

☑

This full subcategory of **Int** is interesting in itself, but unfortunately it is not closed under the operation of second-order quantification (see chapter 8).

8 Second Order

As we already pointed out on page 152 (lemma 7.1.18), the interpretation of a linear formula (as an interaction system or as a predicate transformer) is trivial if the only atomic formulas are constants. To answer this problem, one needs to start with propositional variables. Since correctness of the model doesn't depend on those variables, it is tempting to interpret Π^1_1 logic and see if this introduces non-trivial objects. Once this is done (section 8.1), we extend the technology developed for this simple case to full second order quantification in sections 8.3 and 8.4. Both the interpretation of Π^1_1 and full second order follow closely the technology developed in [18] and [19].

Restriction: in all this chapter, we implicitly assume that the set of states are countable (possibly finite). The reason is that without this hypothesis, it is not possible to prove corollary 8.3.24. Most of the other result do hold for unrestricted interfaces.

8.1 PI-1 Logic

Π^1_1 logic is usually called "propositional logic": we start with propositional "variables" (which are not variable in any sense) and look at the resulting system. This can be seen as the restriction of second order logic to formulas of the form $(\forall \vec{X})\, \varphi(\vec{X})$.

8.1.1 Idea

Since the Π^1_1 formula $\varphi(\vec{X})$ really means $(\forall \vec{X})\, \varphi(\vec{X})$, we somehow want to form the predicate transformer "$(\forall \vec{P})\, \varphi(\vec{P})$". Since we certainly don't want to quantify over all predicate transformers over all sets (this is a proper class), we start by deciding on a *countably infinite* set I to serve as the generic set of states. We do not assume *any* structure on I.

One difference with the relational model (or with the coherent spaces model of [38]) is that the cardinality of I is important: it represents the maximal cardinality of our objects in the model. It is sensible, for pragmatic reasons, to restrict to countable interfaces, but if one wanted to model continuous states, then the set I should be taken of bigger cardinality. (See also remark 29 on page 179.)

Interpreting the formula $(\forall \vec{X})\, \varphi(\vec{X})$ is done in two steps. Suppose for example

that φ is the formula $X \multimap X$. We start by forming the predicate transformer

$$r \quad \mapsto \quad \bigcap_{P:\mathcal{P}(I)\to\mathcal{P}(I)} P \multimap P(r)$$

from $I \times I$ to $I \times I$. Taking the intersection is coherent with the main intuition since a safety property for this predicate transformer will be a safety property for *all* the predicate transformers of the form $P \multimap P$.

The second idea, is to "simplify" the above predicate transformer by quotienting the set of states by renaming: since I doesn't carry any structure, any permutation of I would do as well. For the particular example of $X \multimap X$, the quotiented set of states will contain only two elements, representing respectively $\{(i,i) \mid i \in I\}$ and $\{(i,j) \mid i \neq j\}$.

Notation: in order to model the different propositional variables, we use tuples of sets and predicate transformers. We use the vector notation to denote a tuple: $\vec{X}$ is a tuple of sets of the form $(X_1, \ldots, X_n)$ etc. Everything is lifted pointwise. For example, the notation $\vec{f} : \vec{X} \to \vec{Y}$ is just an abbreviation for $f_1 : X_1 \to Y_1$, ..., $f_n : X_n \to Y_n$.

8.1.2 State Spaces, Permutations

The set of of states $|\varphi|$ of a $\mathbf{\Pi}^1_1$ formula φ is simply given by its relational interpretation (see section 5.3) in which all the atoms are interpreted by the set I:

- $|\mathbf{0}| = \emptyset$ and $|\mathbf{1}| = \{*\}$;
- $|X_i| = I$;
- $|\varphi^\perp| = |\varphi|$;
- $|\varphi_1 \oplus \varphi_2| = |\varphi_1| + |\varphi_2|$;
- $|\varphi_1 \otimes \varphi_2| = |\varphi_1| \times |\varphi_2|$;
- $|!\varphi| = \mathcal{M}_f(|\varphi|)$.

If all the propositional variables of φ are in $(X_1, \ldots, X_n)$, then the product $\mathfrak{S}_I^n$ of n copies of the group $\mathfrak{S}_I$ of *finite permutations*[1] of I acts on $|\varphi|$. Suppose $\vec{\sigma} \in \mathfrak{S}_I^n$, we define the action $[\vec{\sigma}]_\varphi : |\varphi| \to |\varphi|$

- if φ is the propositional variable X_i, then $[\vec{\sigma}]_{X_i}(s) = \sigma_i(s)$;
- if φ is $\psi^\perp$, then $[\vec{\sigma}]_\varphi = [\vec{\sigma}]_\psi$;
- if φ is $\psi_1 \oplus \psi_2$, then $[\vec{\sigma}]_\varphi\big(\mathrm{inl}(a_1)\big) = \mathrm{inl}\big([\vec{\sigma}]_{\psi_1}(a_1)\big)$
  and $[\vec{\sigma}]_\varphi\big(\mathrm{inr}(a_2)\big) = \mathrm{inr}\big([\vec{\sigma}]_{\psi_2}(a_2)\big)$;
- if φ is $\psi_1 \otimes \psi_2$, then $[\vec{\sigma}]_\varphi\big((a_1, a_2)\big) = \big([\vec{\sigma}]_{\psi_1}(a_1), [\vec{\sigma}]_{\psi_2}(a_2)\big)$;
- if φ is $!\psi$, then $[\vec{\sigma}]_\varphi\big([a_1, \ldots, a_k]\big) = \big[[\vec{\sigma}]_\psi(a_1), \ldots, [\vec{\sigma}]_\psi(a_k)\big]$.

Two elements a and b of $|\varphi|$ are *equivalent up to renaming* if there is a finite permutation $\vec{\sigma} : \mathfrak{S}_I^n$ such that $[\vec{\sigma}]_\varphi(a) = b$. This is trivially an equivalence relation, and we write $a \approx_\varphi b$, or simply $a \approx b$ if φ is clear from the context. Note that the equivalence class of an element $a \in |\varphi|$ is simply the orbit of this element under the action of the group $\mathfrak{S}_I^n$.

> ↬ Remark 22: the definition of the group action makes it clear that the restriction to *finite* permutations is not really a restriction: since an element of $|\varphi|$ can be seen as a *finite* tree with leafs in I, if $[\vec{\sigma}]_\varphi(a) = b$ for an arbitrary permutation, then we can find a finite permutation $\vec{\sigma}'$ in $\mathfrak{S}_I^n$ such that $[\vec{\sigma}']_\varphi(a) = b$.

[1]: a permutation is finite if it only changes a finite number of elements

8.1.3 The Model

We can now define formally the interpretation of a formula.

▷ **Definition 8.1.1:** a *valuation* for a formula φ is a map ρ from the propositional variables of φ to predicate transformers on I. We write φ_ρ for the obvious predicate transformer on $|\varphi|$ defined by interpreting any propositional variable X by $\rho(X)$.

This definition allows to give a preliminary candidate for the interpretation of φ:

▷ **Definition 8.1.2:** let φ be a linear formula with atoms. Define $\widetilde{\varphi}$, a predicate transformer on $|\varphi|$ as:

$$\widetilde{\varphi}(x) \quad \triangleq \quad \bigcap_\rho \varphi_\rho(x) \qquad \textit{where } \rho \textit{ runs over all valuations of } \varphi \ .$$

This huge intersection (over a set of cardinality $\aleph_2!$) has the tendency to remove any local asperity. As a result, the predicate transformer $\widetilde{\varphi}$ is very well-behaved. In particular, we have

○ **Lemma 8.1.3:** the predicate transformer $\widetilde{\varphi}$ is compatible with $\approx_\varphi$.

Compatibility simply means that the predicate transformer sends $\approx$-closed subsets to $\approx$-closed subsets.

proof: this relies on the following trivial fact: if ρ is a valuation and $\vec{\sigma}$ and $\vec{\tau}$ are permutations, then the valuation $\vec{\sigma}\cdot\rho\cdot\vec{\tau}$, defined as $(\vec{\sigma}\cdot\rho\cdot\vec{\tau})(X_i)(x) = \sigma_i\cdot\rho(X_i)\cdot\tau_i(x)$ satisfies $\varphi_{\vec{\sigma}\cdot\rho\vec{\tau}} = [\vec{\sigma}]\cdot\varphi_\rho\cdot[\vec{\tau}]$. This is a direct induction...

↬ REMARK 23: to be precise, $[\vec{\sigma}]$ is not a predicate transformer, but only a function. We can lift if to a predicate transformer by taking the update of the converse of its graph.

Let $x \subseteq |\varphi|$ be $\approx$-closed, suppose $a \in \widetilde{\varphi}(x)$ and $a \approx b$, *i.e.* $a = [\vec{\sigma}](b)$ for some permutation $\vec{\sigma}$. We need to show that $b \in \widetilde{\varphi}(x)$:

$a \in \widetilde{\varphi}(x)$
$\quad\Leftrightarrow$ { definition }
$a \in \bigcap_\rho \varphi_\rho(x)$
$\quad\Rightarrow$ { for any valuation ρ, $\vec{\sigma}\cdot\rho\cdot\vec{\sigma}^{-1}$ is also a valuation }
$a \in \bigcap_\rho \varphi_{\vec{\sigma}\cdot\rho\cdot\vec{\sigma}^{-1}}(x)$
$\quad\Leftrightarrow$ { fact above }
$a \in \bigcap_\rho [\vec{\sigma}]\cdot\varphi_\rho\cdot[\vec{\sigma}^{-1}](x)$
$\quad\Leftrightarrow$ { $[\vec{\sigma}]$ commutes with intersections and $[\vec{\sigma}^{-1}](x) = x$ (because x is $\approx$-closed) }
$a \in [\vec{\sigma}]\bigcap_\rho \varphi_\rho(x)$
$\quad\Leftrightarrow$ { because $a = [\vec{\sigma}](b)$ }
$b \in \bigcap_\rho \varphi_\rho(x)$
$\quad\Leftrightarrow$ { definition }
$b \in \widetilde{\varphi}(x)$.

☑

This lemma makes it sound to define the final interpretation of a formula φ as:

▷ **Definition 8.1.4:** if φ is a linear formula with propositional variables, let $|\varphi|_\approx$ be the collection of $\approx$-equivalence classes of $|\varphi|$; define the interpretation $[\![\varphi]\!]$ of φ to be the following predicate transformer on $|\varphi|_\approx$:

$$[\![\varphi]\!](x) \quad \triangleq \quad \Big(\widetilde{\varphi}\big(\bigcup x\big)\Big)_\approx .$$

↬ Remark 23: there is an obvious bijection between $\mathcal{P}(|\varphi|_\approx)$ and $\approx$-closed subsets of $|\varphi|$: $x \mapsto x_\approx \triangleq \{\{a\}_\approx \mid a \in U\}$ and $U \mapsto \bigcup U$. The definition of $[\![\varphi]\!]$ is just the composition of $\widetilde{\varphi}$ with those bijections.

The following is a direct generalization of proposition 7.1.17.

◇ **Proposition 8.1.5:** *let π be a proof of $\vdash G_1, \ldots, G_n$, then $|\pi|$ is a $\approx$-closed subset of $|G_n| \times \ldots \times |G_n|$, and moreover, $|\pi|_\approx$ is a safety property for $[\![G_1 \parr \ldots \parr G_n]\!]$.*

proof: the fact that $|\pi|$ is a safety property in $\widetilde{\varphi}$ is exactly proposition 7.1.17.

The only thing to prove is that $|\pi|$ is $\approx$-closed. This is a direct induction. Let's only look briefly at the case of promotion:

$$\frac{\pi_1 \vdash ?G_1, \ldots, ?G_n, F}{\pi \vdash ?G_1, \ldots, ?G_n, !F} \quad .$$

Let $(\mu_1, \ldots, \mu_n, [a_1, \ldots, a_k]) \in |\pi|$. By definition (page 118), we can partition each μ_i into $\mu_{i,1} + \ldots + \mu_{i,k}$ and for each j, we have $(\mu_{1,j}, \ldots, \mu_{n,j}, a_j) \in |\pi_1|$. Suppose now that $(\nu_1, \ldots, \nu_n, [b_1, \ldots, b_n]) \approx (\mu_1, \ldots, \mu_n, [a_1, \ldots, a_k])$. By definition of $\approx$, this means that $\mu_i \approx \nu_i$ for each i, and that $[a_1, \ldots, a_k] \approx [b_1, \ldots, b_k]$. Without loss of generality, we can assume that we have $a_j \approx b_j$ for all j.

For all i, we have $\mu_i \approx \nu_i$ and we have a partition $\mu_i = \mu_{i,1} + \ldots + \mu_{i,k}$, we can "transfer" this partition onto ν_i along $\approx$: $\nu_i = \nu_{i,1} + \ldots + \nu_{i,k}$ with $\mu_{i,j} \approx \nu_{i,j}$. This implies that we have $(\mu_{1,j}, \ldots, \mu_{n,j}, a_j) \approx (\nu_{1,j}, \ldots, \nu_{n,j}, b_j)$ for all j. By induction hypothesis, we can conclude that $(\nu_{1,j}, \ldots, \nu_{n,j}, b_j) \in |\pi_1|$. By definition of π, we obtain finally $(\nu_1, \ldots, \nu_n, [b_1, \ldots, b_k]) \in |\pi|$.

The other cases are much simpler...

☑

↬ Remark 24: as far as only Π^1_1 is concerned, we could use $\widetilde{\varphi}$ as the interpretation of φ but as we'll see in the sequel, if we really want to interpret $(\forall \vec{X})\, \varphi(\vec{X})$, we need this operation of quotient. As the following examples will show, this is also relevant from a computational point of view, since it simplifies greatly the final predicate transformer interpreting the formula.

8.1.4 Examples

With the machinery in place, we can look at a couple of examples and check that the resulting interpretations are not trivial.
Note the following equivalences:

- $(a, b) \notin P \multimap Q(r)$ iff $a \in P(x)$ but $b \notin Q\big(r(x)\big)$ for some subset x;
- $\big([a_1, \ldots, a_n], a\big) \notin !P \multimap Q(R)$ iff all $a_i \in P(x_i)$ but $a \notin Q\big(R(\bigotimes_i x_i)\big)$ for some subsets x_i.

Since we are only interested in the safety properties of $[\![\varphi]\!]$, we are mostly interested by the action of $\widetilde{\varphi}$ on $\approx$-closed subsets of $|\varphi|$. We will transparently switch between seeing an element $e \in |\varphi|_\approx$ as an equivalence class and as a ($\approx$-closed) subset of $|\varphi|$. Finally, since we are dealing with closed subsets, when $e \notin U$ (in $|\varphi|_\approx$) we may assume that $e \cap \bigcup U = \emptyset$ (in $|\varphi|$).

§ *The Empty Type.* Let's start with the most simple example: $\varphi \triangleq (\forall X)\, X$. The state space $|\varphi|$ is simply I, and any element is equivalent to itself! We thus obtain that $|\varphi|_\approx$ is just $\{*\}$. Since $\widetilde{\varphi}(x) = \emptyset$, we have:

$$\begin{array}{rcccc} [\![\varphi]\!] & : & \mathcal{P}(\{*\}) & \to & \mathcal{P}(\{*\}) \\ & & x & \mapsto & \emptyset\ . \end{array}$$

In other words, we have reinvented the predicate transformer corresponding to the interaction system **abort** (page 41). This is consistent with the intuition of the "empty" type since there is no safety property besides $\emptyset$.

§ *The Singleton Type.* Another easy example is the unit type $\varphi \triangleq (\forall X)\, X \multimap X$. We expect the interpretation $[\![\varphi]\!]$ to have a single non-empty safety property, corresponding to the interpretation of the axiom. The set of states $|\varphi|$ is equal to $I \times I$, and it is easy to check that $(i, j) \approx (i', j')$ iff $i = i' \wedge j = j'$ or $i \neq i' \wedge j \neq j'$. We obtain a two elements set $|\varphi|_\approx = \{e, d\}$:

- $e \triangleq \{(i, i) \mid i \in I\}$, with e standing for "equal";
- $d \triangleq \{(i, j) \mid i, j \in I\ ,\ i \neq j\}$, with d standing for "different"

Let's first show that $\widetilde{\varphi}(|\varphi|) \subseteq e$, *i.e.* that $[\![\varphi]\!](\{e, d\}) \subseteq \{e\}$. Suppose by contradiction that $(i, j) \in \widetilde{\varphi}(|\varphi|)$ for some $i \neq j$. In particular, we must have $(i, j) \in Q \multimap Q(|\varphi|)$ where Q is constantly equal to $\{i\}$. This is impossible because we have $i \in Q(\emptyset)$ but $j \notin Q\big(|\varphi|(\emptyset)\big) = Q(\emptyset) = \{i\}$.[2]

We now show that $r \subsetneq e$ implies that $\widetilde{\varphi}(r) = \emptyset$: suppose $(i, i) \notin r$ and let $(j, j) \in \widetilde{\varphi}(r)$ (we know that an element of $\widetilde{\varphi}(r)$ is necessarily of this form by the preceding remark). In particular, $(j, j) \in Q \multimap Q(r)$, where Q is defined as

$$Q(x) \quad \triangleq \quad \begin{cases} \{j\} & \text{if } i \in x \\ \emptyset & \text{otherwise}\ . \end{cases}$$

This is impossible because $j \in Q(\{i\})$ but $j \notin Q\big(r(\{i\})\big) = \emptyset$ since $i \notin r(\{i\})$.

We also know by lemma 7.1.9 that $e \subseteq Q \multimap Q(e)$ for any Q. This gives:

$$\widetilde{\varphi}(r) \quad = \quad \begin{cases} e & \text{if } e \subseteq r \\ \emptyset & \text{otherwise} \end{cases}$$

[2]: Recall that $|\varphi|$ being a relation, it sends a subset x to its direct image $\langle |\varphi|^\sim \rangle(x)$.

which allows to get the final $[\![\varphi]\!]$:

$$\begin{array}{lcccl} [\![\varphi]\!] & : & \mathcal{P}(\{e,d\}) & \to & \mathcal{P}(\{e,d\}) \\ & & x & \mapsto & \begin{cases} \{e\} & \text{if } e \in x \\ \emptyset & \text{otherwise .} \end{cases} \end{array}$$

The only non-empty safety property for $[\![\varphi]\!]$ is thus $\{e\}$, which makes it a sensible interpretation for the unit type. Note however that the predicate transformer is not trivial in the sense that is is *not* the identity, nor the (update of the) graph of a function (refer to lemmas 7.1.18 and 7.1.19).

§ *Linear Booleans.* Let's now look at types with more than a single inhabitant. The simplest such type is the type of booleans, which we expect to have *three* inhabitants: true, false, and their union. There are several ways to code the booleans inside second order linear logic. We'll start with $\varphi \triangleq (\forall X)\,(\mathbf{1} \,\&\, X) \multimap (\mathbf{1} \,\&\, X) \multimap X$. This is similar to the booleans from system-F, except that we do not use the full intuitionistic arrow (contraction is not needed).

The set $|\varphi|$ is equal to $(\{*\} + I) \times (\{*\} + I) \times I$ and its quotient by renaming is:

$$|\varphi|_{\approx} \quad = \quad \left\{ \begin{array}{c} (*,*,1), \\ (*,1,1),(1,*,1), \\ (*,1,2),(1,*,2), \\ (1,1,1), \\ (1,2,1),(2,1,1), \\ (1,1,2), \\ (1,2,3) \end{array} \right\}$$

(where for example, $(1,*,2)$ *is the orbit of* $(\mathrm{inr}(i),\mathrm{inl}(*),j)$ *with* $i \neq j$*)*

This set is finite, but not trivial anymore (10 elements).

The computation will go as follows:

$$\widetilde{\varphi}(|\varphi|) \quad \subseteq \quad (*,1,1) \cup (1,*,1) \cup (1,1,1) \tag{8-1}$$

$$\widetilde{\varphi} \cdot \widetilde{\varphi}(|\varphi|) \quad \subseteq \quad (*,1,1) \cup (1,*,1)\ . \tag{8-2}$$

This will show that all safety properties for $[\![\varphi]\!]$ are subsets of $\{(*,1,1),(1,*,1)\}$ and since both $(*,1,1) \in [\![\varphi]\!](\{(*,1,1)\})$ and $(1,*,1) \in [\![\varphi]\!](\{(1,*,1)\})$ (they are the interpretation of the two canonical proofs of φ), we can conclude.

Proof of (8-1)*:* let $(a,b,i) \in \widetilde{\varphi}(|\varphi|)$, we first show that a and b are of the form $\mathrm{inl}(*)$ or $\mathrm{inr}(i)$. Suppose by contradiction that a is of the form $\mathrm{inr}(j)$ with $i \neq j$. Define $Q(x) \triangleq \{j\} \cup x$. By hypothesis , we know that

$$(a,b,i) \in (\mathbf{1} \,\&\, Q) \multimap (\mathbf{1} \,\&\, Q) \multimap Q(|\varphi|)$$

which is impossible for the following reason:

- $a \in \mathbf{1} \,\&\, Q(\emptyset)$
- and $b \in \mathbf{1} \,\&\, Q(\{*\} + I)$
- but $i \notin Q\big(|\varphi|(\emptyset, \{*\} + I)\big) = Q(\emptyset) = \{j\}$.[3]

a is thus necessarily of the form $\mathrm{inl}(*)$ or $\mathrm{inr}(i)$ and similarly for b.

[3]: $|\varphi|$ is a ternary relation between $\{*\}+I$, $\{*\}+I$ and I; as the predicate transformer $\langle|\varphi|^{\sim}\rangle$, it has type $\mathcal{P}\big(\{*\}+I\big) \times \mathcal{P}\big(\{*\}+I\big) \to \mathcal{P}(I)$.

Suppose now that $a = b = \mathrm{inl}(*)$, then we can take $Q(x)$ constantly equal to $\emptyset$. It is impossible that $(a, b, i) \in \widetilde{\varphi}(|\varphi|)$ since $a = b \in \mathbf{1}$ & $Q(\emptyset)$ but $i \notin Q\big(|\varphi|(\emptyset, \emptyset)\big) = \emptyset$.

↬ REMARK 25: it is possible to show that this inclusion is in fact an equality.
To show that $(1, 1, 1) \subseteq \widetilde{\varphi}(|\varphi|)$, suppose that $i \in Q(x)$ and $i \in Q(y)$.
- If one of x or y is empty, then we have that $i \in Q(\emptyset)$ which implies that $i \in Q\big(|\varphi|(x, y)\big)$;
- if not, we have that $x \subseteq |\varphi|(x, y)$: if $j \in x$, then there is some j' in y (since $y \neq \emptyset$) and we have that $(j, j') \in |\varphi|$. We thus have that $i \in Q\big(|\varphi|(x, y)\big)$ by monotonicity.

We can conclude that $\widetilde{\varphi}(|\varphi|) = (*, 1, 1) \cup (1, *, 1) \cup (1, 1, 1)$.

Proof of (8-2)*:* by the previous remark, we just need to show that $(1, 1, 1)$ is *not* an element of $\widetilde{\varphi}\big((*, 1, 1) \cup (1, *, 1) \cup (1, 1, 1)\big)$.
Suppose it is not the case, *i.e.* suppose that $\big(\mathrm{inr}(i), \mathrm{inr}(i), i\big) \in \widetilde{\varphi} \cdot \widetilde{\varphi}(|\varphi|)$. This means that $\big(\mathrm{inr}(i), \mathrm{inr}(i), i\big) \in (\mathbf{1} \,\&\, Q) \multimap (\mathbf{1} \,\&\, Q) \multimap Q\big(\widetilde{\varphi}(|\varphi|)\big)$ for all predicate transformers Q on I. Let $j \neq j'$ be two elements of I, and define

$$Q(x) \quad \triangleq \quad \begin{cases} \{i\} & \text{if } j \in x \text{ or } j' \in x \\ \emptyset & \text{otherwise .} \end{cases}$$

We have that $\mathrm{inr}(i) \in (\mathbf{1} \,\&\, Q)\big(\{\mathrm{inr}(j)\}\big)$ and $\mathrm{inr}(i) \in (\mathbf{1} \,\&\, Q)\big(\{\mathrm{inr}(j')\}\big)$, but since $\big((*, 1, 1) \cup (1, *, 1) \cup (1, 1, 1)\big)\big(\mathrm{inr}(j), \mathrm{inr}(j')\big) = \emptyset$,[4] we have $i \notin Q\big(\widetilde{\varphi}(|\varphi|)\big)$. We can conclude that $(\mathrm{inr}(i), \mathrm{inr}(i), i) \notin \widetilde{\varphi} \cdot \widetilde{\varphi}(|\varphi|)$ and obtain

$$\widetilde{\varphi} \cdot \widetilde{\varphi}(|\varphi) \quad \subseteq \quad (*, 1, 1) \cup (1, *, 1) \ .$$

Since we have that $\mathrm{True} \triangleq \{(1, *, 1)\}$ and $\mathrm{False} \triangleq \{(*, 1, 1)\}$ are both safety properties for $[\![\varphi]\!]$, we can conclude that the only non-empty safety properties are:

1) True;
2) False;
3) and $\mathrm{True} \cup \mathrm{False}$.

It is possible to do a little more computation to give the exact predicate transformer $[\![\varphi]\!]$: it is the (smallest) predicate transformer generated by

$$[\![\varphi]\!](x) \quad = \quad \begin{cases} \{(1, 1, 1)\} & \text{if } \{(1, 1, 1), (1, 2, 1), (2, 1, 1)\} \subseteq x \\ \{(1, *, 1)\} & \text{if } (1, *, 1) \in x \\ \{(*, 1, 1)\} & \text{if } (*, 1, 1) \in x \end{cases} .$$

§ *Booleans.* The "real" booleans from system-F are slightly more complex: they are given by the formula $\varphi \triangleq (\forall X)\, !X \multimap !X \multimap X$. It is possible to show that the only safety properties for $[\![\varphi]\!]$ are still given by $\mathrm{True} \triangleq \{([1], [], 1)\}$ and $\mathrm{False} \triangleq \{([], [1], 1)\}$ (and their union). To show that, we proceed as above. We show:

$$\begin{array}{lcl} \widetilde{\varphi}(|\varphi|) & \subseteq & \bigcup\big\{(1^n, 1^m, 1) \mid n + m > 0\big\} \\ \widetilde{\varphi}\Big(\bigcup\big\{(1^n, 1^m, 1) \mid n + m > 0\big\}\Big) & \subseteq & ([1], [], 1) \cup ([], [1], 1) \ . \end{array}$$

(where $1^n = [1, \ldots, 1]$ *of length* n*)*
The computations are exactly the same as in the previous case. However, giving an explicit definition of the whole predicate transformer $[\![\varphi]\!]$ is quite difficult and involves combinatorics on multisets.

[4]: here again, $(*, 1, 1) \cup (1, *, 1) \cup (1, 1, 1) \subseteq (\{*\}+I) \times (\{*\}+I) \times I$, *i.e.* it is a ternary relation; as a predicate transformer, it has type $\mathcal{P}\big(\{*\}+I\big) \times \mathcal{P}\big(\{*\}+I\big) \to \mathcal{P}(I)$.

§ *"Swap" Booleans.* There is a second linear formula for booleans which has the advantage of involving only $\otimes$ and $\multimap$: $\varphi \triangleq (\forall X)\,(X \otimes X) \multimap (X \otimes X)$. The two canonical proofs are either the identity or the "swap", which, if we differentiate the occurrences of X linked by axioms, is the proof coming from $(\forall X)\,(X_1 \otimes X_2) \multimap (X_2 \otimes X_1)$.

The set $|\varphi|$ is $(I \times I) \times (I \times I)$, and $|\varphi|_\approx$ is slightly bigger than for the linear booleans (15 elements). We write (ij,kl) instead of $\big((i,j),(k,l)\big)$:

$$|\varphi|_\approx \quad = \quad \left\{ \begin{array}{c} \underline{(00,00)}, \\ (00,01),(00,\underline{10}),(\underline{01},00),(10,00), \\ (00,12),(01,02),(10,20),(01,20),(10,02),(12,00), \\ (00,11),\underline{(01,01)},\underline{(01,10)}, \\ \underline{(01,23)} \end{array} \right\}$$

Once again, there are only three non-empty safety properties:

1) True $\triangleq \{(00,00),(01,01)\}$;

2) False $\triangleq \{(00,00),(01,10)\}$;

3) their union.

To show that there are no bigger safety properties, it suffices to show that

$$\widetilde{\varphi}(|\varphi|) \quad \subseteq \quad (00,00) \cup (01,01) \cup (01,10)\ .$$

Let's first show that $\widetilde{\varphi}(|\varphi|) \subseteq (00,00) \cup (01,01) \cup (01,10) \cup (01,00) \cup (10,00)$: suppose that there is some other $(ij,kl) \in \widetilde{\varphi}(|\varphi|)$. This means exactly that $\{k,l\} \not\subseteq \{i,j\}$. Define Q to be constantly equal to $\{i,j\}$. We have trivially that $(i,j) \in Q \otimes Q(\emptyset)$, but we cannot have $(k,l) \in Q \otimes Q\big(|\varphi|(\emptyset)\big)$. It is thus impossible for (ij,kl) to be in $\widetilde{\varphi}(|\varphi|)$.

To eliminate the last two elements (ij,ii) and (ji,ii), define $Q(x) \triangleq x \cup \{j\}$. We do have $i \in Q(\{i\})$ and $j \in Q(\emptyset)$ so that $(i,j) \in Q \otimes Q(\emptyset)$. However, we do *not* have $(i,i) \in Q \otimes Q\big(|\varphi|(\emptyset)\big)$.

We now need to show that there are no smaller safety properties than True, False and True$\cup$False. For example, let's show that $\{(00,00)\}$ is not a safety property for $[\![\varphi]\!]$: we will show that $(ii,ii) \notin (Q \otimes Q) \multimap (Q \otimes Q)\big(\{(00,00)\}\big)$. Choose two elements $j \neq j'$ in I, and define Q as:

$$Q(x) \quad \triangleq \quad \left\{ \begin{array}{ll} \{i\} & j \in x \text{ or } j' \in x \\ \emptyset & \text{otherwise}\ . \end{array} \right.$$

We have $(i,i) \in Q \otimes Q(\{(j,j')\})$, but since $(00,00)\big(\{(j,j')\}\big) = \emptyset$, it is *not* the case that $(i,i) \in Q \otimes Q\big((00,00)\big(\{(j,j')\}\big)\big)$.

To show that $(ij,ij) \notin [\![\varphi]\!]\big(\{(01,01)\}\big)$, or that $(ij,ji) \notin [\![\varphi]\!]\big(\{(01,10)\}\big)$, use

$$Q(x) \quad \triangleq \quad \left\{ \begin{array}{ll} \{i,j\} & \text{if } k \in x \\ \emptyset & \text{otherwise} \end{array} \right.$$

for some arbitrary $k \in I$. We have $(i,j) \in Q \otimes Q\big(\{(k,k)\}\big)$ but $(01,01)\big(\{(k,k)\}\big) = \emptyset$. The same predicate transformer Q also shows that $\{(01,01),(01,10)\}$ is not a safety property.

↬ Remark 26: unfortunately, there are not many easy examples. For instance, due to the presence of exponentials, natural numbers (given by the formula $\varphi \triangleq (\forall X)\, X \multimap !(X \multimap X) \multimap X$) turn out to be much more difficult and requires non-trivial combinatorics on multisets.

8.2 Second Order in the Relational Model

Before diving into the full construction, let's review briefly second order in the relational model. The details we omit can be found in [18], [19] or [20].

8.2.1 Injections

Define the following notation: if f is a function from X to Y,
- $f^+ \triangleq \langle \mathbf{gr}(f)^\sim \rangle$, *i.e.* $f^+ : \mathcal{P}(X) \to \mathcal{P}(Y)$ and $f^+(x) \triangleq \{f(a) \mid a \in x\}$;
- $f^- \triangleq \langle \mathbf{gr}(f) \rangle$, *i.e.* $f^- : \mathcal{P}(Y) \to \mathcal{P}(X)$ and $f^-(y) \triangleq \{a \mid f(a) \in y\}$.

We have:

- **Lemma 8.2.1:** if f is an injection $X \hookrightarrow Y$, then
  - $f^- \cdot f^+ = \mathbf{Id}_{\mathcal{P}(X)}$;
  - $f^+ \cdot f^- \subseteq \mathbf{Id}_{\mathcal{P}(Y)}$, and more precisely, $f^+ \cdot f^-(y) = f^+(X) \cap y$.

One last thing about injections:

▷ **Definition 8.2.2:** an injection $\iota : X \hookrightarrow Y$ satisfying $\iota(a) = a$ for all $a \in X$ is called an *inclusion*. We write $X \subseteq Y$.

8.2.2 Stable Functors

Let **Inj** denote the category of sets and injections.

▷ **Definition 8.2.3:** a functor F from **Inj** to **Inj** is *stable* if:
1) F sends inclusions to inclusions;
2) F commutes with finite intersections;
3) F commutes with directed unions.

> ↬ Remark 27: the categorical definition of stable functor would read "commutes with pullbacks and directed limits". Since we are dealing with sets, it is natural to require that the functor preserves inclusions. When a functor preserves inclusions, points *2* and *3* are equivalent to the categorical definition.

The definition of stable functor is extended to functors of arbitrary arity. We can talk about a stable functor from $\mathbf{Inj}^n$ to **Inj**. For such a functor F, we write $F_{\vec{X}}(\vec{f})$, or simply $F(\vec{f})$ for the action of F on the (pointwise) injection $\vec{f} : \vec{X} \hookrightarrow \vec{Y}$.

Stable functors enjoy a very nice property:

- **Lemma 8.2.4:** let F be a stable functor from $\mathbf{Inj}^n$ to **Inj**, if $a \in F(\vec{X})$, then there is a finite $\vec{X}_0 \subseteq \vec{X}$ such that $a \in F(\vec{X}_0)$. Moreover, there is a smallest such $\vec{X}_0$, which depends only on a and F (and not on X). We write $|a|^F$ for this unique minimal set and we call it the *support of* a.

proof: see [19] or [18] for the easy proof. It follows from the fact that any set is the directed union of its finite subsets and that stable functor preserve directed union and binary intersections, and send inclusions to inclusions.

☑

We have:

◦ **Lemma 8.2.5:** let F be a functor from $\mathbf{Inj}^n$ to **Inj** preserving inclusions and let $f, g : \vec{X} \hookrightarrow \vec{Y}$; if the restriction of f and g coincide on $x \subseteq X$, then the restriction of $F(f)$ and $F(g)$ coincide on $F(x)$.
In particular, if F is stable, $a \in F(\vec{X})$ and if $\vec{f}$ and $\vec{g}$ are injections from $\vec{X}$ to $\vec{Y}$ which coincide on $|a|^F$, then $F(\vec{f})(a) = F(\vec{g})(a)$.

proof: that $\vec{f}$ and $\vec{g}$ coincide on x can be expressed by saying that the following diagram commutes: (ι denotes an inclusion)

$$\begin{array}{ccc} \vec{x} & \xrightarrow{\iota} & \vec{X} \\ {\scriptstyle\iota}\downarrow & & \downarrow{\scriptstyle\vec{g}} \\ \vec{X} & \xrightarrow{\vec{f}} & \vec{Y} \end{array}$$

This implies (because F is a functor preserving inclusions) that the following diagrams commutes:

$$\begin{array}{ccc} F(\vec{x}) & \xrightarrow{\iota} & F(\vec{X}) \\ {\scriptstyle\iota}\downarrow & & \downarrow{\scriptstyle F(\vec{g})} \\ F(\vec{X}) & \xrightarrow{F(\vec{f})} & F(\vec{Y}) \end{array}$$

i.e. $F(\vec{f})$ and $F(\vec{g})$ coincide on $F(|a|^F)$.

The second point is direct application of the first point.

☑

8.2.3 Trace of a Stable Functor

If F is a stable functor of arity $n+1$, its *trace* will be a functor of arity n. The idea is to use the operation from the previous section: we quotient by the action of the group $\mathfrak{S}_I$.

▷ **Definition 8.2.6:** if F is a stable functor of arity $n+1$; define the following n-ary functor $\mathcal{T}F$, called the *trace of* F:

- action on objects:

$$(\mathcal{T}F)(X_1, \ldots, X_n) \quad \triangleq \quad F(X_1, \ldots, X_n, I)_{\approx_F}$$

  where $a \approx_F b$ iff $a = F_{\vec{X},I}(\vec{\mathbf{Id}}, \sigma)(b)$ for some finite bijection $\sigma : \mathfrak{S}_I$;
- action on morphisms: suppose $\vec{f} : \vec{X} \to \vec{Y}$, define

$$\begin{array}{rccc} (\mathcal{T}F)_{\vec{X}}(\vec{f}) & : & (\mathcal{T}F)(\vec{X}) & \to & (\mathcal{T}F)(\vec{Y}) \\ & & \{a\}_\approx & \mapsto & \left\{ F_{\vec{X},I}(\vec{f}, \sigma)(a) \mid \sigma : \mathfrak{S}_I \right\} . \end{array}$$

It is trivial to check that the definition is sound.
We have the following lemma:

◦ **Lemma 8.2.7:** Suppose F is a stable functor of arity $n+1$; suppose moreover that $a \in F(X_1, \ldots, X_n, Y)$ where Y is an infinite set. If f is an injection from Y to Y, then we have $a \approx_F F_{\vec{X},Y}(\vec{\mathbf{Id}}, f)(a)$.

proof: we only look at the case when F is of arity 1, the other cases are similar. By lemma 8.2.4, we have $a \in F(|a|^F)$ where $|a|^F \subseteq_f Y$ The restriction of f to $|a|^F$ is an injection with finite support, so that we can extend it to a bijection $g{:}Y \xrightarrow{\sim} Y$. We can even ensure that this bijection is a finite permutation.
Since f and g coincide on $|a|^F$, we can apply lemma 8.2.5 and get $F(f)(a) = F(g)(a)$. Since we have that $a \approx_F F(g)(a)$, we can conclude.

☑

8.3 Open Formulas as Predicate Transformers

We can now lift the notions of stable functor and trace to take into account predicate transformers.

8.3.1 Rigid Embeddings

The first question to answer is the following: what is an "injection" between predicate transformers? The answer is given by the notion of *rigid embedding*:

▷ **Definition 8.3.1:** if (X, P) and (Y, Q) are interfaces, a *rigid embedding* from P to Q is an injection $f : X \hookrightarrow Y$ satisfying $P \cdot f^- = f^- \cdot Q$.
We write $f : (X, P) \hookrightarrow (Y, Q)$, or simply $f : P \hookrightarrow Q$. If f is an inclusion, we write $(X, P) \prec (Y, Q)$ and say that P is a *subobject* of Q.

As lemma 8.3.7 will show, a rigid embedding is a special case of embedding. Since we are only interested in *rigid* embeddings, we omit the adjective "rigid".

A subobject of (X, P) is entirely determined by the subset X_0 of X:

○ **Lemma 8.3.2:** we have, $(X_0, P_0) \prec (X, P)$ iff $P(x) \cap X_0 \subseteq P(x \cap X_0)$ for all $x \subseteq X$.

proof: let $\iota : (X_0, P_0) \prec (X, P)$ be an embedding,

$(\forall x \subseteq X)\ P_0 \cdot \iota^-(x) = \iota^- \cdot P(x)$
$\quad \Leftrightarrow \quad \{\ \iota^-(x) = X_0 \cap x\ \}$
$(\forall x \subseteq X)\ P_0(X_0 \cap x) = X_0 \cap P(x)$
$\quad \Rightarrow \quad \{$ in particular, for $x \cap X_0$, $P_0(X_0 \cap x) = X_0 \cap P(X_0 \cap x)\ \}$
$(\forall x \subseteq X)\ X_0 \cap P(X_0 \cap x) = X_0 \cap P(x)$
$\quad \Leftrightarrow$
$(\forall x \subseteq X)\ X_0 \cap P(x) \subseteq P(X_0 \cap x)$

This concludes the proof: for any embedding $(X_0, P_0) \prec (X, P)$, the predicate transformer P_0 is necessarily of the form $P_0(x_0) = X_0 \cap P(x_0)$. This allows to write $X_0 \prec (X, P)$ without fear of confusion.

☑

As opposed to the traditional case of coherent spaces, it is *not* the case that any subset of (X, P) can be made into a subobject of (X, P). In particular, an interface needs not be the limit of its finite subobjects. Here is an example of infinite interface with *no* proper subobject:

$$P \quad : \quad \mathcal{P}(\mathbf{N}) \quad \to \quad \mathcal{P}(\mathbf{N})$$

$$x \quad \mapsto \quad \begin{cases} \mathbf{N} & \text{if } x = \mathbf{N} \\ \emptyset & \text{otherwise .} \end{cases}$$

↯ REMARK 28: unfortunately, there is now easy way out of this:
- we cannot require that all finite subsets to be subobject (see remark 29 on page 179) as the collection of such interfaces is not closed by dual (see previous example);
- if we require *all* subsets to be subobjects, we obtain something quite degenerate where $x \subseteq P(X) \Rightarrow x \subseteq P(x)$, *i.e.* $\mathcal{S}(P) = \mathcal{P}(P(X))$.

On the other hand, any superset Y of X can be made into a superobject of (X, P): just define $Q(y) \triangleq P(y \cap X)$.

▷ **Definition 8.3.3:** the category **Emb** has *countable* interfaces as objects and rigid embeddings as morphisms.

This category enjoys many closure properties, similar to **Inj**. For example:

○ **Lemma 8.3.4:**
- any embedding can be factorized as an inclusion followed by an isomorphism;
- **Emb** has all filtered limits and pullbacks.

Since those properties will not be used in the sequel, the proof is omitted. We only mention that the construction are the same as the one used in **Inj** and that one just needs to check that they preserve embeddings between interfaces.

• **Corollary 8.3.5:** The class of interfaces satisfies:
- the relation "$\prec$" is a partial order;
- it is closed under finite glbs (intersection);
- it is closed under directed lubs (union);
- it is closed under "bounded" lowest upper bounds: if $X_0 \prec (X, P)$ and $X_1 \prec (X, P)$, then $X_0 \cup X_1 \prec (X, P)$.

What is more important is that embeddings interact well with the logical connectives:

○ **Lemma 8.3.6:** if $f_1 : (X_1, P_1) \hookrightarrow (Y_1, Q_1)$ and $f_2 : (X_2, P_2) \hookrightarrow (Y_2, Q_2)$, we have:
- $f_1 : P_1^\perp \hookrightarrow Q_1^\perp$ (and in particular, $X_0 \prec (X, P)$ iff $X_0 \prec (X, P^\perp)$);
- $f_1 \oplus f_2 : P_1 \oplus P_2 \hookrightarrow Q_1 \oplus Q_2$;
- $f_1 \otimes f_2 : P_1 \otimes P_2 \hookrightarrow Q_1 \otimes Q_2$;
- $!f_1 : !P_1 \hookrightarrow !Q_1$;

proof: let's only look at the case of linear negation: write $\complement$ for complementation,

$f : (X, P) \hookrightarrow (Y, Q)$

$\Leftrightarrow$ { definition }

$P \cdot f^- = f^- \cdot Q$

$\Leftrightarrow$

$\complement \cdot P \cdot f^- \cdot \complement = \complement \cdot f^- \cdot Q \cdot \complement$

$\Leftrightarrow$ { f^- commutes with complementation }

$\complement \cdot P \cdot \complement \cdot f^- = f^- \cdot \complement \cdot Q \cdot \complement$

$\Leftrightarrow$

$P^\perp \cdot f^- = f^- \cdot Q^\perp$
$\quad \Leftrightarrow \quad$ { definition }
$f : (X, P)^\perp \hookrightarrow (Y, Q)^\perp$

The other cases are in no way more difficult.

☑

The central fact is that linear negation is covariant. (We are not (yet) defining a denotational model: an embedding does *not* represent a proof.)

Finally, we have:

- **Lemma 8.3.7:** if $f : P \hookrightarrow Q$, then:
  - $\mathbf{gr}(f)$ is a simulation from P to Q;
  - $\mathbf{gr}(f)^\sim$ is a simulation from Q to P, it is left inverse to $\mathbf{gr}(f)$;
  - x is a safety property for P iff $f^+(x)$ is a safety property for Q;
  - if y is a safety property for Q, then $f^-(y)$ is a safety property for P.

proof: the only non trivial part is checking the second point. It follows from the fact that $\langle \mathbf{gr}(f)^\sim \rangle = [\mathbf{gr}(f)^\sim]$ and that r is a simulation from P to Q iff $P \cdot [r^\sim] \subseteq [r^\sim] \cdot Q$. (This point follows from the Galois connection $\langle r \rangle \vdash [r^\sim]$ (lemma 2.5.11).)

☑

8.3.2 Parametric Interfaces

We now have the technology needed to define "parametric interfaces". Those are meant to represent formulas with free variables. The idea is, going back to the model of system-F presented in [38], that a formula with free variable X is represented by a stable functor from **Emb** to **Emb**. We moreover require this functor to be split in two parts:

- one part acting on the sets (of states), *i.e.* the relational part;
- one part acting on interfaces on those sets.

Formally, this gives:

▷ **Definition 8.3.8:** an n-ary *parametric interface* is a pair $(|F|, F)$, where:

1) $|F|$ is a stable functor from $\mathbf{Inj}^n$ to **Inj**;
2) if $\vec{P}$ is an interface on $\vec{X}$, then $F(\vec{P})$ is an interface on $F(\vec{X})$;
3) if $\vec{f} : (\vec{X}, \vec{P}) \hookrightarrow (\vec{Y}, \vec{Q})$, then $|F|(\vec{f}) : \big(|F|(\vec{X}), F(\vec{P})\big) \hookrightarrow \big(|F|(\vec{Y}), F(\vec{Q})\big)$.

(where all the vectors have length n)

$|F|$ is called the *relational part* of $(|F|, F)$; we usually omit it.

Any such parametric interface trivially induces a functor from $\mathbf{Emb}^n$ to **Emb**. Moreover, we have the following (easy) property:

- **Lemma 8.3.9:** as a functor from $\mathbf{Emb}^n$ to **Emb**, an n-ary parametric interface commutes with pullbacks and directed limits.

Note however that it is very unlikely that any stable functor from $\mathbf{Emb}^n$ to **Emb** can be split into a relational part and a specification part.

We can now lift all the logical constructions on interfaces:

▷ **Definition 8.3.10:** if F and G are n-ary parametric interfaces, define the following parametric interfaces:

$$\begin{array}{lcl} F^{\perp}(\vec{P}) & \triangleq & F(\vec{P})^{\perp} \\ F \oplus G(\vec{P}) & \triangleq & F(\vec{P}) \oplus G(\vec{P}) \\ F \otimes G(\vec{P}) & \triangleq & F(\vec{P}) \otimes G(\vec{P}) \\ (!F)(\vec{P}) & \triangleq & !\big(F(\vec{P})\big) \ . \end{array}$$

(the relational part is defined pointwise in the obvious way)

That those operations yield parametric specifications follows from lemma 8.3.6.

8.3.3 Parametric Safety properties (Objects of Variable Type)

The aim is now to represent "parametric safety properties", *i.e.* if F is a parametric interface, then a "safety property" for it should be given by a family of safety properties for all the $F(P)$. We first introduce the more intuitive notion of *object of variable type* F and then show it is possible to simplify this to obtain the notion of *object of type* F, slightly less intuitive but easier to manipulate.

§ *Objects of Variable Type.* A safety property for F should be a safety property for *all* the interfaces $F(\vec{P})$. For technical reasons, this is not quite enough, and we need to require that such a safety property behaves well w.r.t. embeddings:

▷ **Definition 8.3.11:** let F be an n-ary interface; an *object of variable type* F, or a *parametric safety property for* F is given by a family $(t_{\vec{X}})_{\vec{X}}$ indexed by countable (possibly finite) sets such that:

1) $t_{\vec{X}} \subseteq |F|(\vec{X})$;

2) $t_{\vec{X}} = |F|(\vec{f})^{-}(t_{\vec{Y}})$ whenever $\vec{f} : \vec{X} \hookrightarrow \vec{Y}$; (*stability*)

3) for any predicate transformer $\vec{P}$ on $\vec{X}$, $t_{\vec{X}}$ is a safety property for $F(\vec{P})$.

We write $t :: F$ to mean that t is a parametric safety property for F.

A (boring) example of parametric safety property is the constantly empty family: this is a parametric safety properties for every parametric interface.

Considering families indexed by *all* countable sets may seem a little extreme. The next lemma shows that a parametric safety property is in fact determined by its value on finite sets[5] and section 8.3.4 shows that we can even restrict to the single value on the set I.

○ **Lemma 8.3.12:** if $t :: F$, then $a \in t_{\vec{X}}$ iff $a \in t_{|a|^F} \cap |F|(\vec{X})$. We thus have

$$t_{\vec{X}} = \left\{ a \in |F|(\vec{X}) \mid a \in t_{|a|^F} \right\} \ .$$

proof: this is an easy application of the stability condition in definition 8.3.11. ☑

Moreover, since $|F|(f)^{-}$ commutes with unions, the lattice structure of safety properties for a predicate transformer lifts pointwise to n-ary interfaces:

[5]: We haven't gained much, since the collection of finite sets is still a proper class!

◦ **Lemma 8.3.13:** the collection of parametric safety properties for a given parametric interface forms a complete sup-lattice.

§ *Monotonic Objects and Composition.* One problem when dealing with objects of variable type is that they are not closed under composition: if we follow section 7.1, a morphism from F to G is a safety property for $F \multimap G$. However, if $t :: F \multimap G$ and $t' :: G \multimap H$, then the pointwise composition $t' \cdot t$ needs not be an object of variable type $F \multimap H$. In order to cope with this problem, we introduce the weaker notion of monotonic object:

▷ **Definition 8.3.14:** let F be an n-ary interface; a *monotonic object of variable type* F is given by a family $(t_{\vec{X}})_{\vec{X}}$ indexed by countable (possibly finite) sets such that:

1) $t_{\vec{X}} \subseteq |F|(\vec{X})$;

2) $t_{\vec{X}} \subseteq |F|(\vec{f})^{-}(t_{\vec{Y}})$ whenever $\vec{f} : \vec{X} \hookrightarrow \vec{Y}$; (*weak stability*)

3) for any specification $\vec{P}$ on $\vec{X}$, $t_{\vec{X}}$ is a safety property for $F(\vec{P})$.

The only difference with definition 8.3.11 is that we relaxed the stability condition to an inclusion, rather than an equality. Every monotonic object can be thought of as a representation for a object of variable type via the following closure operation:

◦ **Lemma 8.3.15:** if t is a monotonic object of type F; define $\widehat{t}$ as:

$$\widehat{t}_{\vec{X}} \quad \triangleq \quad \bigcup_{\vec{X} \subseteq \vec{Y}} t_{\vec{Y}} \cap |F|(\vec{X}) \ .$$

We have $\widehat{t} :: F$.
Moreover, $\widehat{\ }$ is a closure operation: $\widehat{t}$ is the smallest object of variable F containing t.[6]

proof: the only non-trivial part is showing that $(\widehat{t})_{\vec{X}}$ is a safety property for any $F(\vec{P})$. Since safety properties are closed under arbitrary unions, it is enough to show that $t_{\vec{Y}} \cap |F|(\vec{X})$ is such a universal safety property whenever $\vec{X} \subseteq \vec{Y}$.

Let $\vec{X} \subseteq \vec{Y}$, and suppose $\vec{P}$ is a specification on $\vec{X}$. We can "extend" $\vec{P}$ to a specification $\vec{Q}$ on $\vec{Y}$ so that $(\vec{X}, \vec{P}) \prec (\vec{Y}, \vec{Q})$: define $\vec{Q}(\vec{y}) \triangleq \vec{P}(\vec{y} \cap \vec{X})$. This implies that $\big(|F|(\vec{X}), F(\vec{P})\big) \prec \big(|F|(\vec{Y}), F(\vec{Q})\big)$. By lemma 8.3.2, we know that

$$|F|(\vec{X}) \cap F(\vec{Q})(t_{\vec{Y}}) \quad \subseteq \quad F(\vec{Q})(|F|(\vec{X}) \cap t_{\vec{Y}}) \ . \tag{8-3}$$

We can now compute:

$t_{\vec{Y}} \cap |F|(\vec{X})$

$\quad \subseteq \quad$ { $t_{\vec{Y}}$ is a safety property in $F(\vec{Q})$ }

$F(\vec{Q})(t_{\vec{Y}}) \cap |F|(\vec{X})$

$\quad \subseteq \quad$ { remark (8-3) above }

$F(\vec{Q})\big(t_{\vec{Y}} \cap |F|(\vec{X})\big)$

$\quad \subseteq \quad$ { $\big(|F|(\vec{X}), F(\vec{P})\big) \prec \big(|F|(\vec{Y}), F(\vec{Q})\big)$, so we have $F(\vec{P})\big(|F|(\vec{X}) \cap \vec{y}\big) = F(\vec{Q})(\vec{y}) \cap |F|(\vec{X})$ }

$F(\vec{P})\big(t_{\vec{Y}} \cap |F|(\vec{X})\big)$

This concludes the proof.

☑

[6]: Here again, we are dealing with a union indexed by a proper class!

- **Lemma 8.3.16:** if t and t' are monotonic objects of type $F \multimap G$ and $G \multimap H$, then the (pointwise) composition $(t' \cdot t)_{\vec{X}} \triangleq (t')_{\vec{X}} \cdot (t)_{\vec{X}}$ is a monotonic object of type $F \multimap H$.
  However, the (pointwise) composition of object of variable types needs *not* yield an object of variable type, but only a monotonic object.

proof: simple if one keeps in mind lemma 7.1.9.

☑

Since the pointwise relational composition of two variable objects is only a monotonic object, we need to define composition as the closure of the relational composition:

- ▷ **Definition 8.3.17:** if F, G and H are n-ary parametric interfaces and if t and t' are respectively objects of variable type $F \multimap G$ and $G \multimap H$, define the composition $t' \cdot t$ as the family $(t' \cdot t)_{\vec{X}} \triangleq \widehat{t'_{\vec{X}} \cdot t_{\vec{X}}}$.
  (where composition on the right is plain relational composition)

We have:

- **Lemma 8.3.18:** if $t :: F \multimap G$ and $t' :: G \multimap H$, then $t' \cdot t :: F \multimap H$; moreover, we only need to consider finite extensions of $\vec{X}$ to compute the value $(t' \cdot t)_{\vec{X}}$:

$$(t' \cdot t)_{\vec{X}} \quad = \bigcup_{\vec{Y} \text{ finite extension of } \vec{X}} t'_{\vec{Y}} \cdot t_{\vec{Y}} \cap |F \multimap H|(\vec{X})$$

  where Y is a finite extension of X if $X \subseteq Y$ and $Y \setminus X$ is finite.[7]

proof: See [18] or [19]

☑

An important corollary is

- **Corollary 8.3.19:** if $\vec{X}$ is (pointwise) infinite, then $(t' \cdot t)_{\vec{X}} = t'_{\vec{X}} \cdot t_{\vec{X}}$.
  (where composition on the right is plain relational composition)

proof: let $\vec{Y_0}$ be a finite extension of $\vec{X}$ s.t. $(a, c) \in t'_{\vec{Y_0}} \cdot t_{\vec{Y_0}}$. We need to show that (a, c) is already in $t'_{\vec{X}} \cdot t_{\vec{X}}$.

$(a, c) \in t'_{\vec{Y_0}} \cdot t_{\vec{Y_0}}$

$\Leftrightarrow$

$\left(\exists b \in |G|(\vec{Y_0})\right)\ (a, b) \in t_{\vec{Y_0}} \wedge (b, c) \in t'_{\vec{Y_0}}$

$\Rightarrow$ { let $\vec{f} : \vec{Y_0} \hookrightarrow \vec{X}$ s.t. the restriction of $\vec{f}$ to $|a|^F \cup |c|^H$ is the identity. }
{ This is possible because $|a|^F \cup |c|^G$ is finite and $\vec{X}$ is infinite. }

$\left(\exists b \in |G|(\vec{Y_0})\right)\ (a, b) \in |F \multimap G|(\vec{f})^-(t_{\vec{Y_0}}) \wedge (b, c) \in |G \multimap H|(\vec{f})^-(t'_{\vec{Y_0}})$

$\Leftrightarrow$

$\left(\exists b \in |G|(\vec{Y_0})\right)\ \left(|F|(\vec{f})(a), |G|(\vec{f})(b)\right) \in t_{\vec{X}} \wedge \left(|G|(\vec{f})(b), |H|(\vec{f})(c)\right) \in t'_{\vec{X}}$

$\Leftrightarrow$ { by lemma 8.2.5, $|F|(\vec{f})(a) = a$ and $|H|(\vec{f})(c) = c$ }

$\left(\exists b \in |G|(\vec{Y_0})\right)\ \left(a, |G|(\vec{f})(b)\right) \in t_{\vec{X}} \wedge \left(|G|(\vec{f})(b), c\right) \in t'_{\vec{X}}$

[7]: The collection of finite extensions of X still forms a proper class!

$$
\begin{array}{ll}
& \Rightarrow \quad \{ \text{ put } b' \triangleq |G|(\vec{f})(b); \text{ we have } b' \in |G|(\vec{X}) \ \} \\
& \big(\exists b' \in |G|(\vec{X})\big)\ (a,b') \in t_{\vec{X}} \wedge (b',c) \in t'_{\vec{X}} \\
& \Leftrightarrow \\
& (a,c) \in t'_{\vec{X}} \cdot t_{\vec{X}}
\end{array}
$$

☑

8.3.4 "Universality"

We can now explain in what sense the infinite set I is "universal".

- **Lemma 8.3.20:** if $t :: F$, then $(t_{\vec{X}})$ is entirely determined by its value on $\vec{I}$ (countably infinite set).

proof: by lemma 8.3.12, we only need to know the value of $t_{\vec{X}}$ for finite $\vec{X}$'s. If $\vec{X}$ is finite, then we can find a pointwise injection $\vec{f} : \vec{X} \to \vec{I}$. By the stability condition, we get $t_{\vec{X}} = |F|(\vec{f})^-(t_{\vec{I}})$.

☑

We can thus replace the notion of "object of variable type" by the simpler notion:

- ▷ **Definition 8.3.21:** if F is a parametric interface of arity n, an *object of type* F is a set $t \subseteq |F|(\vec{I})$ satisfying:
  1) if $\vec{f} : \vec{I} \hookrightarrow \vec{I}$, then $|F|(\vec{f})^-(t) = t$; (stability)
  2) for any specification $\vec{P}$ on $\vec{I}$, then t is a safety property in $F(\vec{P})$.

Note in this definition that the $\vec{f}$ from the stability condition is an injection, and not an isomorphism: requiring t to be only closed by permutation (*i.e.* bijections) is not enough.

This definition is equivalent to the notion of object of variable type. We trivially have that if $(t_{\vec{X}})$ is an object of type F, then $t_{\vec{I}}$ is an object of type F. For the converse, if t is an object of type F, define the family $(t_{\vec{X}})$ as follows:

$$t_{\vec{X}} \quad \triangleq \quad \bigcup_{\vec{f}:\vec{X}\hookrightarrow\vec{I}} |F|(\vec{f})^-(t) \ . \tag{8-4}$$

The stability condition imposes such symmetries that we can replace the union above by an intersection!

- **Lemma 8.3.22:**
  1) if $f, g : X \hookrightarrow I$, then there is $h : I \hookrightarrow I$ s.t. $f = h \cdot g$ or $g = h \cdot f$;
  2) if $\vec{f}, \vec{g} : \vec{I} \hookrightarrow \vec{I}$, then there are $\vec{h}_{\vec{f}}, \vec{h}_{\vec{g}} : \vec{I} \hookrightarrow \vec{I}$ s.t. $\vec{h}_{\vec{f}} \cdot \vec{f} = \vec{h}_{\vec{g}} \cdot \vec{g}$;
  3) if $t_{\vec{X}}$ is defined as (8-4), then we have: $t_{\vec{X}} = \bigcap_{\vec{f}:\vec{X}\hookrightarrow\vec{I}} |F|(\vec{f})^-(t)$.

proof: for the first point, suppose that $Cf^+(X)$ and $Cg^+(X)$ are infinite: we can then define $h\big(f(x)\big) \triangleq g(x)$ and since there is necessarily an injection from $Cf^+(X)$ to $Cg^+(X)$ (because the latter is infinite), we can complete h using such an injection. We have trivially that $h : I \hookrightarrow I$ and $h \cdot f = g$.
If one of $Cf^+(X)$ and $Cg^+(X)$ is finite, we can compare their cardinalities. Suppose $\#\big(Cf^+(X)\big) \leqslant \#\big(Cg^+(X)\big)$, we then define $h\big(f(x)\big) \triangleq g(x)$, and we complete the definition of h by an injection from $Cf^+(X)$ to $Cg^+(X)$.
If $\#\big(Cg^+(X)\big) \leqslant \#\big(Cf^+(X)\big)$, proceed symmetrically.

For the second point, let's look at an example. Suppose $n = 2$ and $(f_1, f_2) : \vec{X} \hookrightarrow \vec{I}$; suppose that by the first point, we have $h_1 \cdot f_1 = g_1$ and $h_2 \cdot g_2 = f_2$. We can take $\vec{h}_{\vec{f}} \triangleq (h_1, \mathbf{Id})$ and $\vec{h}_{\vec{g}} \triangleq (\mathbf{Id}, h_2)$. It is easy to extend to arbitrary n.

For the last point, let $a \in t_{\vec{X}}$, *i.e.* $a \in |F|(\vec{f})^-(y)$ for some $\vec{f} : \vec{X} \hookrightarrow \vec{I}$. Let $\vec{g} : \vec{X} \hookrightarrow \vec{I}$. We know that there are $\vec{h}_{\vec{f}}, \vec{h}_{\vec{g}} : \vec{I} \hookrightarrow \vec{I}$ s.t. $\vec{h}_{\vec{f}} \cdot \vec{f} = \vec{h}_{\vec{g}} \cdot \vec{g}$, so that we have:

$$\begin{array}{rcll} |F|(\vec{f})^-(t) & = & |F|(\vec{f})^- \cdot |F|(\vec{h}_{\vec{f}})^-(t) & \{ \text{ because } \vec{h}_{\vec{f}} : \vec{I} \hookrightarrow \vec{I} \ \} \\ & = & |F|(\vec{h}_{\vec{f}} \cdot \vec{f})^-(t) & \{ \text{ by functoriality } \} \\ & = & |F|(\vec{h}_{\vec{g}} \cdot \vec{g})^-(t) & \{ \text{ by the second point } \} \\ & = & |F|(\vec{g})^- \cdot |F|(\vec{h}_{\vec{g}})^-(t) & \{ \text{ by functoriality } \} \\ & = & |F|(\vec{g})^-(t) & \{ \text{ because } \vec{h}_{\vec{g}} : \vec{I} \hookrightarrow \vec{I} \ \} \ . \end{array}$$

This shows that $a \in |F|(\vec{g})^-(t)$, and thus that $a \in \bigcap_{\vec{g}:\vec{I}\hookrightarrow\vec{I}} |F|(\vec{g})^-(t)$.

☑

We can now show:

- **Lemma 8.3.23:** if $(t_{\vec{X}})$ is defined as (8-4), then $(t_{\vec{X}})$ is an object of variable type F.

proof: we need to check the three conditions:

- $t_{\vec{X}} \subseteq |F|(\vec{X})$: trivial

- if $\vec{f} : \vec{X} \hookrightarrow \vec{Y}$, then $t_{\vec{X}} = |F|(\vec{f})^- t_{\vec{Y}}$:
  - "$\supseteq$ direction":

$$\begin{array}{rcl} |F|(\vec{f})^-(t_{\vec{Y}}) & = & |F|(\vec{f})^- \bigcup_{\vec{g}:\vec{Y}\hookrightarrow\vec{I}} |F|(\vec{g})^-(t) \\ & = & \bigcup_{\vec{g}:\vec{Y}\hookrightarrow\vec{I}} |F|(\vec{f})^- \cdot |F|(\vec{g})^-(t) \\ & = & \bigcup_{\vec{g}:\vec{Y}\hookrightarrow\vec{I}} |F|(\vec{g} \cdot \vec{f})^-(t) \\ & \subseteq & \bigcup_{\vec{h}:\vec{X}\hookrightarrow\vec{I}} |F|(\vec{h})^-(t) \\ & = & t_{\vec{X}} \end{array}$$

  - "$\subseteq$ direction": we use lemma 8.3.22 point *3*

$$\begin{array}{rcl} |F|(\vec{f})^-(t_{\vec{Y}}) & = & |F|(\vec{f})^- \bigcap_{\vec{g}:\vec{Y}\hookrightarrow\vec{I}} |F|(\vec{g})^-(t) \\ & = & \bigcap_{\vec{g}:\vec{Y}\hookrightarrow\vec{I}} |F|(\vec{f})^- \cdot |F|(\vec{g})^-(t) \\ & = & \bigcap_{\vec{g}:\vec{Y}\hookrightarrow\vec{I}} |F|(\vec{g} \cdot \vec{f})^-(t) \\ & \supseteq & \bigcap_{\vec{h}:\vec{X}\hookrightarrow\vec{I}} |F|(\vec{h})^-(t) \\ & = & t_{\vec{X}} \end{array}$$

→ if $\vec{P}$ is a specification on $\vec{X}$, then $t_{\vec{X}} \subseteq F(\vec{P})(t_{\vec{X}})$:

$a \in t_{\vec{X}}$

$\Leftrightarrow$ { definition }

$a \in \bigcup_{\vec{f}:\vec{X}\hookrightarrow\vec{I}} |F|(\vec{f})^-(t)$

$\Rightarrow$ { claim: $|F|(\vec{f})^-(t) \subseteq F(\vec{P})\big(|F|(\vec{f})^-(t)\big)$, see below }

$a \in \bigcup_{\vec{f}:\vec{X}\hookrightarrow\vec{I}} F(\vec{P})\big(|F|(\vec{f})^-(t)\big)$

$\Rightarrow$ { for any predicate transformer Q, $\bigcup Q \subseteq Q \bigcup$ }

$a \in F(\vec{P})\Big(\bigcup_{\vec{f}:\vec{X}\hookrightarrow\vec{I}} |F|(\vec{f})^-(t)\Big)$

$\Leftrightarrow$ { definition }

$a \in F(\vec{P})(t_{\vec{X}})$

Claim: $|F|(\vec{f})^-(t) \subseteq F(\vec{P})\big(|F|(\vec{f})^-(t)\big)$ for any $\vec{f} : \vec{X} \hookrightarrow \vec{I}$.

Define $\vec{P}_{\vec{f}} = \vec{f}^+ \cdot \vec{P} \cdot \vec{f}^-$. It is trivial to check that $\vec{f} : (\vec{X}, \vec{P}) \hookrightarrow (\vec{I}, \vec{P}_{\vec{f}})$, which implies that $|F|(\vec{f}) : F(\vec{P}) \hookrightarrow F(\vec{P}_{\vec{f}})$. By definition, it means that $F(\vec{P})\cdot|F|(\vec{f})^- = |F|(\vec{f})^-\cdot F(\vec{P}_{\vec{f}})$. Since $\vec{P}_{\vec{f}}$ is a specification on $\vec{I}$, we know that $t \subseteq F(\vec{P}_{\vec{f}})(t)$. By monotonicity of $|F|(\vec{f})^-$, this implies $|F|(\vec{f})^-(t) \subseteq |F|(\vec{f})^- \cdot F(\vec{P}_{\vec{f}})(t)$. By the previous equality, we can conclude that $|F|(\vec{f})^-(t) \subseteq F(\vec{P}) \cdot |F|(\vec{f})^-(t)$.

☑

- **Corollary 8.3.24:** the collection of objects of variable type F and the collection of objects of type F are in bijection.

Since lemma 8.3.19 ensures that plain composition for objects of types $F \multimap G$ and $G \multimap H$ is well-defined, we now replace the notion of object of variable type F by the (simpler) notion of object of type F. We now write $t :: F$ for the latter.

↬ Remark 29: in the purely relational model, instead of (8-4), one recovers the whole $(t_{\vec{X}})$ from $t_{\vec{I}}$ in the following way:

$$t_{\vec{X}} \triangleq \Big\{ a \in |F|(\vec{X}) \mid (\exists \vec{f} : |a|^F \hookrightarrow \vec{I})\ |F|(\vec{f})(a) \in t \Big\}$$

which has the advantage of working for set of arbitrary cardinality.
The reason we cannot use this definition is that it seems impossible to prove that $t_{\vec{X}}$ is a safety property in all the $F(\vec{P})$ for $\vec{P}$ a specification on $\vec{X}$ when $\vec{X}$ is not countable. The problem comes from the fact that $|a|^F$ needs not be a subobject of $(\vec{X}, \vec{P})$. We can overcome this problem when $\vec{X}$ is countable by the following trick: if $\vec{f} : |a|^F \hookrightarrow \vec{I}$, then we can find some $\vec{g} : \vec{X} \hookrightarrow \vec{I}$ extending f, and proceed as in the proof of lemma 8.3.23.

We have finally found a notion of object of type F which doesn't involve proper classes. What is almost magical is that this allows to recover an object of *variable* type indexed by *all* countable sets!

8.3.5 The Categories of n-ary Parametric Interfaces

Just like **Int** forms a category, the collection of n-ary parametric interfaces $\mathbf{PInt}^{(n)}$ can be equipped with a structure of category:

▷ **Definition 8.3.25:** for any natural number n, the category $\mathbf{PInt}^{(n)}$ is defined as follows:

- objects are n-ary parametric countable interfaces;
- a morphism from F to G is an object of type $F \multimap G$;

- if $t :: F \multimap G$ and $t' :: G \multimap H$, the composition $t' \cdot t :: F \multimap H$ is given by the relational composition of t' and t.

By convention, if F is of arity 0, an object of type F is simply a safety property for F.

This is indeed a category (by lemma 8.3.19 and the fact that $\mathbf{Id}_{|F|(\vec{I})} :: F \multimap F$). Moreover, just like in section 2.5, we get:

◇ **Proposition 8.3.26:** *for all* n, $\mathbf{PInt}^{(n)}$ *with the operations from definition 8.3.10 forms a denotational model for full linear logic.*

Checking formally everything is a lengthy and boring job: it amounts to lift all of section 7.1 pointwise.

8.4 Second Order Quantification

Now that we can model formulas with free propositional variables (represented by parametric interfaces), the goal is to give the semantical operation modeling quantification. For that, we use the operation of "trace" defined on page 170 for the relational model. We extend it to deal with predicate transformer in the same way as we did in section 8.1 by taking a huge intersection over all predicate transformers on I.

8.4.1 Trace of a Parametric Interface

The definition is just the same as definition 8.1.2, relativized to a single variable:

▷ **Definition 8.4.1:** let F be an interface of arity $n+1$; define $\widetilde{\mathcal{T}}F$, the *pre-trace of* F to be the following interface of arity n:

$$\begin{array}{lcl} |\widetilde{\mathcal{T}}F|(X_1,\ldots,X_n) & \triangleq & |F|(X_1,\ldots,X_n,I) \\ \widetilde{\mathcal{T}}F_{\vec{X}}(P_1,\ldots,P_n) & \triangleq & \bigcap_{Q \text{ specification on } I} F_{\vec{X},I}(P_1,\ldots,P_n,Q)\ . \end{array}$$

Define the *trace of* F to be the following interface of arity n, on the set $\mathcal{T}|F|$ (the relational trace of $|F|$, see page 170):

$$\mathcal{T}F_{\vec{X}}(\vec{P})(U) \quad \triangleq \quad \Big(\widetilde{\mathcal{T}}F_{\vec{X}}(\vec{P})\big(\bigcup U\big)\Big)_{\approx}\ .$$

That this definition is sound follows from:

○ **Lemma 8.4.2:** if F is an interface of arity $n+1$,

1) $\widetilde{\mathcal{T}}F$ is an interface of arity n;
2) the group $\mathfrak{S}_I$ of finite permutations acts on $|\widetilde{\mathcal{T}}F|(\vec{X}) = |F|(\vec{X},I)$ with the obvious definition: $[\sigma](a) = |F|_{\vec{X},I}(\mathbf{Id}^n,\sigma)(a)$;
3) if $U \subseteq |\widetilde{\mathcal{T}}F|(\vec{X})$ is $\approx$-closed, then so is $\widetilde{\mathcal{T}}F_{\vec{X}}(\vec{P})(U)$.

The proof of point *3* is completely similar to the proof of lemma 8.1.3.

Moreover, we have:

◦ **Lemma 8.4.3:** the operation $\mathcal{T}$ is a functor from $\mathbf{PInt}^{(n+1)}$ to $\mathbf{PInt}^{(n)}$.

The action of $\mathcal{T}$ on morphisms (embeddings) is defined as the action of the relational trace on underlying injections (page 170).
Finally, we have (where stability is point *1* in definition 8.3.21)

◦ **Lemma 8.4.4:** if $t \subseteq |F|(\vec{I}, I)$, then t is stable w.r.t F iff $t_{\approx}$ is stable w.r.t. $\mathcal{T}F$. This implies that $t :: F$ iff $t_{\approx} :: \mathcal{T}F$.

8.4.2 An Appropriate Adjunction

Following Lawvere insight, we justify the relevance of $\mathcal{T}$ as a sound interpretation of universal quantification by showing an adjunction between $\mathcal{T} : \mathbf{PInt}^{(n+1)} \to \mathbf{PInt}^{(n)}$ and $\mathcal{U} : \mathbf{PInt}^{(n)} \to \mathbf{PInt}^{(n+1)}$, the "useless variable functor". It is defined by:

$$\begin{array}{lcl} |\mathcal{U}(F)|(X_1, \ldots, X_n, X_{n+1}) & \triangleq & |F|(X_1, \ldots, X_n) \\ \mathcal{U}(F)(P_1, \ldots, P_n, P_{n+1}) & \triangleq & F(P_1, \ldots, P_n)\ . \end{array}$$

This adjunction is the semantical counterpart of the logical rule defining universal quantification: $\Gamma \vdash (\forall X)\, F(X)$ iff $\Gamma \vdash F(X)$ where X is not free in Γ. The functor $\mathcal{U}$ makes sure that we use a fresh variable. Checking that this is an adjunction amounts to checking that the collection of morphisms from $\mathcal{U}\,G$ to F is naturally isomorphic to the collection of morphisms from G to $\mathcal{T}F$.

§ *From* $\mathbf{PInt}^{(n+1)}(\mathcal{U}\,G, F)$ *to* $\mathbf{PInt}^{(n)}(G, \mathcal{T}F)$. Suppose t is a morphism from $\mathcal{U}\,G$ to F, *i.e.* that $t :: \mathcal{U}\,G \multimap F$. The obvious candidate for a morphism $\Lambda t :: F \multimap \mathcal{T}F$ is:

$$\Lambda t \quad \triangleq \quad \left\{ \left(b, \{a\}_{\approx}\right) \mid (b, a) \in t \right\}\ .$$

In the terminology of system-F, Λt is a type abstraction.

◦ **Lemma 8.4.5:** if $t :: \mathcal{U}\,G \multimap F$ then $\Lambda t :: G \multimap \mathcal{T}F$.

proof: we need to prove that Λt is stable and that $(\Lambda t)_{\vec{X}}$ is a safety property in any $(G \multimap \mathcal{T}F)(\vec{P})$ for specifications $\vec{P}$ on $\vec{X}$.

→ Stability: let $\vec{f} : \vec{X} \hookrightarrow \vec{I}$

$(b, \{a\}_{\approx}) \in |G \multimap \mathcal{T}F|(\vec{f})^{-}\Lambda t$

$\Leftrightarrow$

$\left(|G|(\vec{f})(b), |\mathcal{T}F|(\vec{f})(\{a\}_{\approx})\right) \in \Lambda t$

$\Leftrightarrow$

$\left(|G|(\vec{f})(b), \left\{|F|(\vec{f}, f)(a) \mid f \in \mathfrak{S}_I\right\}\right) \in \Lambda t$

$\Leftrightarrow$

$\left(|G|(\vec{f})(b), \{|F|(\vec{f}, \mathbf{Id})(a)\}_{\approx}\right) \in \Lambda t$

$\Leftrightarrow$

$\left(|G|(\vec{f})(b), |F|(\vec{f}, \mathbf{Id})(a)\right) \in t$

$\Leftrightarrow$

$\left(|\mathcal{U}\,G|(\vec{f}, \mathbf{Id})(b), |F|(\vec{f}, \mathbf{Id})(a)\right) \in t$

$\Leftrightarrow$

$(b, a) \in |\mathcal{U}\,G \multimap F|(\vec{f}, \mathbf{Id})^{-}(t)$

$\Leftrightarrow$ { t is stable for $\mathcal{U}\,G \multimap F$ }

$(b, a) \in t$

$\Leftrightarrow$

$(b, \{a\}_{\approx}) \in \Delta t$

- Λt is a universal safety property:

  $(b, \{a\}_{\approx}) \in \Lambda t$

  $\Leftrightarrow$

  $(b, a) \in t$

  $\Rightarrow$ { t is an object of type $\mathcal{U}\, G \multimap F$ }

  $(\forall \vec{P}, Q)\ (b, a) \in (\mathcal{U}\, G \multimap F)(\vec{P}, Q)(t)$

  $\Leftrightarrow$

  $(\forall \vec{P}, Q)\ (b, a) \in \big(G(\vec{P}) \multimap F(\vec{P}, Q)\big)(t)$

  $\Leftrightarrow$ { definition of $\multimap$ }

  $(\forall \vec{P}, Q)\ (\forall y)\ b \in G(\vec{P})(y) \Rightarrow a \in F(\vec{P}, Q)(\langle t \rangle y)$

  $\Leftrightarrow$

  $(\forall \vec{P})\ (\forall y)\ b \in G(\vec{P})(y) \Rightarrow (\forall P)\ a \in F(\vec{P}, Q)(\langle t \rangle y)$

  $\Leftrightarrow$ { definition of pre-trace }

  $(\forall \vec{P})\ (\forall y)\ b \in G(\vec{P})(y) \Rightarrow a \in (\widetilde{\mathcal{T}}\, F)(\vec{P})(\langle t \rangle y)$

  $\Rightarrow$ { claim (see below): $\langle t \rangle y \subseteq \bigcup \langle \Lambda t \rangle y$ }

  $(\forall \vec{P})\ (\forall y)\ b \in G(\vec{P})(y) \Rightarrow a \in (\widetilde{\mathcal{T}}\, F)(\vec{P})(\bigcup \langle \Lambda t \rangle y)$

  $\Leftrightarrow$ { definition of $\mathcal{T}$, and because $\bigcup \langle \Lambda t \rangle y$ is $\approx$-saturated }

  $(\forall \vec{P})\ (\forall y)\ b \in G(\vec{P})(y) \Rightarrow \{a\}_{\approx} \in (\mathcal{T}\, F)(\vec{P})(\langle \Lambda t \rangle y)$

  $\Leftrightarrow$ { definition of $\multimap$ }

  $(\forall \vec{P})\ (b, \{a\}_{\approx}) \in \big(G \multimap (\mathcal{T}\, F)\big)(\vec{P})(\Lambda t)$

  proof of the claim: $\langle t \rangle y \subseteq \bigcup \langle \Lambda t \rangle y$ for any y:

  $a \in \langle t \rangle y$

  $\Leftrightarrow$

  $(\exists b \in y)\ (b, a) \in t$

  $\Rightarrow$

  $(\exists b \in y)\ (b, \{a\}_{\approx}) \in \Lambda t$

  $\Rightarrow$ { for $\alpha = \{a\}_{\approx}$ }

  $(\exists b \in y)\ (\exists \alpha)\ (b, \alpha) \in \Lambda t \wedge a \in \alpha$

  $\Leftrightarrow$

  $(\exists \alpha \in \langle \Lambda t \rangle y)\ a \in \alpha$

  $\Leftrightarrow$

  $a \in \bigcup \langle \Lambda t \rangle y$

  ☑

§ *From* $\mathbf{PInt}^{(n)}(G, \mathcal{T}\, F)$ *to* $\mathbf{PInt}^{(n+1)}(\mathcal{U}\, G, F)$. For the converse, just do ... the opposite: if $t :: G \multimap \mathcal{T}\, F$, define:

$$Et \quad \triangleq \quad \big\{(b, a) \mid \big(b, \{a\}_{\approx}\big) \in t\big\} \ .$$

We have the expected result, namely:

- **Lemma 8.4.6:** if $t :: G \multimap \mathcal{T}\, F$ then $Et :: \mathcal{U}\, G \multimap F$.

proof: suppose that $t :: G \multimap \mathcal{T}F$, we need to prove that $Et :: \mathcal{U}G \multimap F$: *i.e.* , that Et is stable and is a universal safety property.

- Stability: let $\vec{f}, f$ and $\vec{g}, g$ be injections $\vec{I} \hookrightarrow \vec{I}$:

$$
\begin{array}{l}
(b, a) \in |\,\mathcal{U}G \multimap F|(\vec{f}, f)^{-}(Et) \\
\quad \Leftrightarrow \\
\big(|\,\mathcal{U}G|(\vec{f}, f)(b), |F|(\vec{f}, f)(a)\big) \in Et \\
\quad \Leftrightarrow \\
\big(|G|(\vec{f})(b), \{|F|(\vec{f}, f)(a)\}_{\approx}\big) \in t \\
\quad \Leftrightarrow \\
\big(|G|(\vec{f})(b), \big\{|F|(\vec{f}, g \cdot f)(a) \mid g : \mathfrak{S}_I\big\}\big) \in t \\
\quad \Leftrightarrow \quad \{ \text{ since } g \cdot f \text{ is an injection, we can apply lemma 8.2.7; } \} \\
\qquad\quad \{ \text{ the proof of the "}\Leftarrow\text{" direction is similar to the proof of lemma 8.2.7 } \} \\
\big(|G|(\vec{f})(b), \big\{|F|(\vec{f}, g)(a) \mid g : \mathfrak{S}_I\big\}\big) \in t \\
\quad \Leftrightarrow \\
\big(|G|(\vec{f})(b), |\,\mathcal{T}F|(\vec{f})(\{a\}_{\approx})\big) \in t \\
\quad \Leftrightarrow \\
(b, \{a\}_{\approx}) \in |G \multimap \mathcal{T}F|(\vec{f})^{-}t \\
\quad \Leftrightarrow \quad \{ \text{ } t \text{ is stable for } G \multimap \mathcal{T}F \text{ } \} \\
(b, \{a\}_{\approx}) \in t \\
\quad \Leftrightarrow \\
(b, a) \in Et
\end{array}
$$

- Et is a universal safety property: let $\vec{P}, Q$ be specifications on $\vec{I}$.
  We will now prove that $Et \subseteq G(\vec{P}) \multimap F(\vec{P}, Q)(Et)$: suppose $(b, a) \in Et$ and let $b \in G(\vec{P})(\vec{y})$. We need to show that $a \in F(\vec{P}, Q)(\langle Et\rangle\vec{y})$:

$$
\begin{array}{l}
(b, a) \in Et \\
\quad \Leftrightarrow \quad \{ \text{ definition of } Et \text{ } \} \\
(b, \{a\}_{\approx}) \in t \\
\quad \Rightarrow \quad \{ \text{ since } t \subseteq G(\vec{P}) \multimap \mathcal{T}F(\vec{P})(t) \text{ and } b \in G(\vec{P})(\vec{y}), \text{ by definition of } \multimap\text{: } \} \\
\{a\}_{\approx} \in \mathcal{T}F(\vec{P})(\langle t\rangle\vec{y}) \\
\quad \Rightarrow \quad \{ \text{ in particular (definition of } \mathcal{T}F) \text{ } \} \\
a \in F(\vec{P}, Q)(\bigcup\langle t\rangle\vec{y}) \\
\quad \Rightarrow \quad \{ \text{ claim (see below): } \bigcup\langle t\rangle\vec{y} \subseteq \langle Et\rangle\vec{y} \text{ } \} \\
a \in F(\vec{P}, Q)(\langle Et\rangle\vec{y})
\end{array}
$$

  Proof of the claim: $\bigcup\langle t\rangle\vec{y} \subseteq \langle Et\rangle\vec{y}$ for any $\vec{y}$:

$$
\begin{array}{l}
a \in \bigcup\langle t\rangle\vec{y} \\
\quad \Leftrightarrow \\
(\exists a')\ a \approx a' \wedge \{a'\}_{\approx} \in \langle t\rangle\vec{y} \\
\quad \Leftrightarrow \\
(\exists a')\ (\exists b \in \vec{y})\ a \approx a' \wedge (b, \{a'\}_{\approx}) \in t \\
\quad \Leftrightarrow \quad \{ \text{ } \{a\}_{\approx} = \{a'\}_{\approx} \text{ since } a \approx a' \text{ } \} \\
(\exists b \in \vec{y})\ (b, \{a\}_{\approx}) \in t \\
\quad \Leftrightarrow \\
b \in \langle Et\rangle\vec{y}
\end{array}
$$

☑

Checking that the operations $\Lambda\_$ and $E\_$ are inverse to each other is easy. The isomorphism is trivially natural, which allows to conclude:

◇ **Proposition 8.4.7:** *for any* n, *the functor* $\mathcal{U} : \mathbf{PInt}^{(n)} \to \mathbf{PInt}^{(n+1)}$ *is left-adjoint to the functor* $\mathcal{T} : \mathbf{PInt}^{(n+1)} \to \mathbf{PInt}^{(n)}$.

8.4.3 Substitution

Proposition 8.4.7 justifies, categorically speaking, the introduction rule for the universal quantifier:

$$\frac{\Gamma \vdash F(Y)}{\Gamma \vdash (\forall X)\ F(X)} \text{ if } Y \text{ is not free in } \Gamma \ .$$

The dual rule for the existential quantifier uses the notion of *substitution:*

$$\frac{\Gamma \vdash F[G/X]}{\Gamma \vdash (\exists X)\ F(X)} \text{ where } G \text{ is a formula } .$$

Rather than doing this "unary" substitution, we define an operation representing the parallel substitution $F[G_1/X_1, \ldots, G_n/X_n]$:

▷ **Definition 8.4.8:** suppose $\vec{G}$ is a "parametric substitution" from $\mathbf{Emb}^n$ to $\mathbf{Emb}^k$ (*i.e.* $\vec{G}$ is of the form $(G_1, \ldots, G_n)$ where each G_i is a k-ary interface); define the following operation of *substitution* taking an n-ary interface F and returning a k-ary interface $F_{\vec{G}/}$:

- action on objects:

$$\begin{aligned} |F_{\vec{G}/}|(X_1, \ldots, X_n) &\triangleq |F|\big(|\vec{G}|(X_1, \ldots, X_n)\big) \\ F_{\vec{G}/}(P_1, \ldots, P_n) &\triangleq F\big(\vec{G}(P_1, \ldots, P_n)\big) \end{aligned}$$

  i.e. $F_{\vec{G}/}$ is just the composition $F \cdot \vec{G}$;
- action on morphisms: if $t :: F \multimap H$, then $t_{\vec{G}/} = t_{\vec{G}(I^k)}$, *i.e.*

$$t_{\vec{G}/} \triangleq \bigcup_{\vec{f}:|\vec{G}|(I^k)\hookrightarrow\vec{I}} |F \multimap H|(\vec{f})^-(t) \ .$$

Unfortunately, this operation is not functorial but only a *lax-functorial:*

○ **Lemma 8.4.9:** for any substitution $\vec{G}$ and morphisms $t :: F_1 \multimap F_2$ and $t' :: F_2 \multimap F_3$, we have:

- $t'_{\vec{G}/} \cdot t_{\vec{G}/} \subseteq (t' \cdot t)_{\vec{G}/}$;
- if $|G|(I^k)$ is infinite,[8] equality holds.

proof: if we unfold the definition of $t_{\vec{G}/}$ (using point *3* of lemma 8.3.22), we get:

- $(a, c) \in s_{\vec{G}/} \cdot t_{\vec{G}/}$ iff

$$\Big(\exists b \in |F_2|\big(|\vec{G}|(I^k)\big)\Big) \begin{cases} (\forall \vec{f}:|\vec{G}|(I^k) \hookrightarrow \vec{I})\ \big(|F_1|(\vec{f})(a), |F_2|(\vec{f})(b)\big) \in s \\ (\forall \vec{g}:|\vec{G}|(I^k) \hookrightarrow \vec{I})\ \big(|F_2|(\vec{g})(b), |F_3|(\vec{g})(c)\big) \in t \end{cases}$$

[8]: in the sense that it is a tuple of infinite sets

- $(a,c) \in (s \cdot t)_{\vec{G}/}$ iff

$$(\forall \vec{f}{:}|\vec{G}|(I^k) \hookrightarrow I)\ \big(\exists b \in |F_2|(\vec{I})\big)\ \begin{cases} \big(|F_1|(\vec{f})(a), b\big) \in s \\ \big(b, |F_3|(\vec{f})(c)\big) \in t \end{cases} .$$

Checking the inclusion is easy.

For the second point, let $(a,c) \in (s \cdot t)_{\vec{G}/}$; in particular, let $\vec{h} : |\vec{G}|(I^k) \xrightarrow{\sim} \vec{I}$ (this is possible since $|\vec{G}|(I^k)$ is infinite), we know that

$$\big(\exists b \in |F_2|(\vec{I})\big)\ \begin{cases} \big(|F_1|(\vec{h})(a), b\big) \in s \\ \big(b, |F_3|(\vec{h})(c)\big) \in t \end{cases} .$$

Let $b' \triangleq |F_2|(h)^-(b)$ (this is well defined because $|F_2|(\vec{h})$ is a bijection); it suffices to show (by definition of $s_{\vec{G}/} \cdot t_{\vec{G}/}$) that

$$\begin{cases} (\exists \vec{f}{:}|\vec{G}|(I^k) \hookrightarrow \vec{I})\ \big(|F_1|(\vec{f})(a), |F_2|(\vec{f})(b')\big) \in s \\ (\exists \vec{g}{:}|\vec{G}|(I^k) \hookrightarrow \vec{I})\ \big(|F_2|(\vec{g})(b'), |F_3|(\vec{g})(c)\big) \in t \end{cases} .$$

Take $f \triangleq h$ and $g \triangleq h \ldots$

☑

↬ REMARK 30: if $|G|(I^k)$ contains a finite set, then equality needs not hold and the inclusion can be strict. Use the following: F, G and H in $\mathbf{PInt}^{(1)}$, with $t :: F \multimap G$, $s :: G \multimap H$ and $K : \mathbf{Emb} \to \mathbf{Emb}$ where:
- $|F|(X) = |H|(X) = \{*\}$; $F(P) = H(P) = \mathbf{Id}$;
- $|G|(X) = X$; $G(P) = \mathbf{Id}_{\mathcal{P}(X)}$;
- $K(X,P) = (\emptyset, \mathbf{Id})$;
- $t = \{(*, i) \mid i \in I\}$;
- $s = \{(i, *) \mid i \in I\}$.

It is easy to check that t and s are objects of type $F \multimap G$ and $G \multimap H$ and that $s_{\vec{G}/} \cdot t_{\vec{G}/} = \emptyset$ whereas $(s \cdot t)_{\vec{G}/} = \{(*,*)\}$.

Notice that when dealing with interpretations of linear logic formulas, if the substitution is pointwise open (in the sense that each G_i is an open formula) or contains exponentials, $\vec{G}(I^k)$ is infinite, and equality thus holds.

§ *Comprehension.* Suppose F is an interface of arity $n+1$,

▷ **Definition 8.4.10:** define $\varepsilon^F \triangleq E\big(\mathbf{Id}_{|\mathcal{T}F|}(\vec{I})\big)$; we have $\varepsilon^F :: \mathcal{U}\mathcal{T}F \multimap F$.

In particular, for any k-ary interface G we can apply substitution on this and obtain a parametric interface of arity $n+k$:

$$\varepsilon^F{}_{\mathbf{Id}^n, G/} \quad :: \quad (\mathcal{U}\mathcal{T}F \multimap F)_{\mathbf{Id}^n, G/} \quad = \quad (\mathcal{U}^k \mathcal{T}F) \multimap (F_{\mathbf{Id}^n, G/}) .$$

This is the semantics counterpart of $(\forall X)\, F(X) \vdash F[G/X]$, the so-called "comprehension axiom".

○ **Lemma 8.4.11:** for any $\pi :: \mathcal{U}\Gamma \multimap F$, we have $(\varepsilon^F{}_{\mathbf{Id}^n, G/}) \cdot \Lambda\pi = \pi_{\mathbf{Id}^n, G/}$.

proof: we just show one side of the inclusion:

$(\gamma, a) \in \varepsilon^{F}_{\mathbf{Id}^n, G} \cdot \Lambda\pi$

$\Leftrightarrow$

$(\exists b)\ (\gamma, \{b\}_{\approx}) \in \Lambda\pi \wedge (\{b\}_{\approx}, a) \in \varepsilon^{F}_{\mathbf{Id}^n, G}$

$\Leftrightarrow$ { where $(\vec{f}, g) : I^n \times |G|(I^k) \hookrightarrow I^{n+1}$ }

$(\exists b)\ (\gamma, \{b\}_{\approx}) \in \Lambda\pi \wedge \big(\forall(\vec{f}, g)\big)\ \big(\mathcal{U}\,\mathcal{T}\,|F|(\vec{f}, g)(\{b\}_{\approx}), |F|(\vec{f}, g)(a)\big) \in \varepsilon^{F}$

$\Rightarrow$ { in particular for $\vec{f}$ the identity }

$(\exists b)\ (\gamma, \{b\}_{\approx}) \in \Lambda\pi \wedge \big(\forall g{:}|G|(I^k) \hookrightarrow I\big)\ \big(\{b\}_{\approx}, |F|(\mathbf{Id}^n, g)(a)\big) \in \varepsilon^{F}$

$\Leftrightarrow$ { definition of ε^F }

$(\exists b)\ (\gamma, \{b\}_{\approx}) \in \Lambda\pi \wedge \big(\forall g{:}|G|(I^k) \hookrightarrow I\big)\ \{b\}_{\approx} = \{|F|(\mathbf{Id}^n, g)(a)\}_{\approx}$

$\Rightarrow$

$\big(\forall g{:}|G|(I^k) \hookrightarrow I\big)\ \big(\gamma, \{|F|(\mathbf{Id}^n, g)(a)\}_{\approx}\big) \in \Lambda\pi$

$\Leftrightarrow$

$\big(\forall g{:}|G|(I^k) \hookrightarrow I\big)\ \big(\gamma, |F|(\mathbf{Id}^n, g)(a)\big) \in \pi$

$\Rightarrow$ { take any $g : |G|(I^k) \hookrightarrow I$ }

$\big(|\,\mathcal{U}\,\Gamma|(\mathbf{Id}^n, g)(\gamma), |F|(\mathbf{Id}^n, g)(a)\big) \in \pi$

$\Rightarrow$

$\big(\exists(\vec{f}, g) : \vec{I} \times |G|(I^k) \hookrightarrow \vec{I} \times I\big)\ \big(|\,\mathcal{U}\,\Gamma|(\vec{f}, g)(\gamma), |F|(\vec{f}, g)(a)\big) \in \pi$

$\Leftrightarrow$

$(\gamma, a) \in \pi_{\mathbf{Id}^n, G}$

The converse inclusion is similar.

☑

§ *Equations for Substitution.* We can now check that substitution does behave as expected:

1) $(\mathcal{T}\,F)_{\vec{G}/} = \mathcal{T}(F_{\vec{G}, \mathbf{Id}/})$: this means that $\mathcal{T}$ binds the last variable of the interface (it is unchanged by substitution);

2) $(\Lambda t)_{\vec{G}/} = \Lambda(t_{\vec{G}, \mathbf{Id}/})$ this is similar, at the level of morphisms (proofs);

3) $(Et)_{\vec{G}, \mathbf{Id}/} = E(t_{\vec{G}/})$: the operation E acts like the identity substitution

proof: let's check the last equality: let $t :: F \multimap \mathcal{T}\,G$. That the typing is correct follows directly from point *1*.

$(b, a) \in (\varepsilon t)_{\vec{G}, \mathbf{Id}/}$

$\Leftrightarrow$

$\big(\forall(\vec{f}, g){:}\vec{G}(\vec{I}) \times I \hookrightarrow \vec{I} \times I\big)\ |\,\mathcal{U}\,F \multimap G|(\vec{f}, g)(b, a) \in \varepsilon t$

$\Leftrightarrow$ { in particular, for g the identity function: }

$\big(\forall(\vec{f}){:}\vec{G}(\vec{I}) \hookrightarrow \vec{I}\big)\ \big(|F|(\vec{f})(b), |G|(\vec{f}, \mathbf{Id})(a)\big) \in \varepsilon t$

$\Leftrightarrow$

$\big(\forall(\vec{f}){:}\vec{G}(\vec{I}) \hookrightarrow \vec{I}\big)\ \big(|F|(\vec{f})(b), \{|G|(\vec{f}, \mathbf{Id})(a)\}_{\approx}\big) \in t$

$\Leftrightarrow$

$\big(\forall(\vec{f}){:}\vec{G}(\vec{I}) \hookrightarrow \vec{I}\big)\ \big(|F|(\vec{f})(b), |\,\mathcal{T}\,G|(\vec{f})\{a\}_{\approx}\big) \in t$

$\Leftrightarrow$

$\big(\forall(\vec{f}){:}\vec{G}(\vec{I}) \hookrightarrow \vec{I}\big)\ |F \multimap \mathcal{T}\,G|(b, \{a\}_{\approx}) \in t$

$\Leftrightarrow$

$(b, \{a\}_{\approx}) \in t_{\vec{G}/}$

$\Leftrightarrow$

$(b, a) \in E(t_{\vec{G}/})$

☑

To be complete, one also needs to check that substitution commutes with the logical connectives. For example, we have:

$$\begin{aligned}(F \otimes H)_{\vec{G}/} &= F_{\vec{G}/} \otimes H_{\vec{G}/}\\ (t \otimes t')_{\vec{G}/} &= t_{\vec{G}/} \otimes t'_{\vec{G}/}\end{aligned}$$

(where $t :: F \multimap G$ *and* $t' :: F' \multimap G'$ *and* $t \otimes t' :: F \otimes F' \multimap G \otimes G'$*).*

8.4.4 Subinvariance by Cut-Elimination

Interpreting second order linear logic is rather easy now: just lift all the logical connectives to parametric interfaces. The only mild difficulty is getting the handling of free variables right. Let $\vec{X}$ be a (finite) list of n unique variables names and suppose F is a second order linear formula, with all its free variables in $\vec{X}$. We can interpret the formula F by an n-ary interface $[\![F]\!]^{\vec{X}} : \mathbf{Emb}^n \to \mathbf{Emb}$ inductively:

- for first order constructions, use definition 8.3.10;
- if F is of the form $(\forall X)\ G$ for a new variable name X, put $[\![F]\!]^{\vec{X}} \triangleq \mathcal{T}[\![G]\!]^{\vec{X},X}$;
- for an existential quantifier, use $(\exists X)\ F = \left((\forall X)\ F^{\perp}\right)^{\perp}$.

Of course, the interesting part is interpreting proofs: for first order rules, follow section 7.1. This only leaves the two rules

$$\frac{\vdash \Gamma\ ,\ F}{\vdash \Gamma\ ,\ (\forall X)\ F}\ \text{where } X \text{ is not free in } \Gamma$$

$$\frac{\vdash \Gamma\ ,\ F[G/X]}{\vdash \Gamma\ ,\ (\exists X)\ F}\ \text{where } G \text{ is a formula.}$$

For the first rule, we use the $\Lambda\_$ operation (page 181):

- if the proof is $\dfrac{\pi' \vdash \Gamma\ ,\ F}{\pi \vdash \Gamma\ ,\ (\forall X)\ X}$, use $[\![\pi]\!] \triangleq \Lambda[\![\pi']\!]$

Correctness is fairly straightforward.

Interpreting the existential rule is slightly trickier, but there is no room for improvisation:

- if the proof is $\dfrac{\pi' \vdash \Gamma\ ,\ F[G/X]}{\pi \vdash \Gamma\ ,\ (\exists X)\ F}$ and all the free variables of G not in $\vec{X}$ appear in $\vec{Y}$, of length k, we have:
  - $[\![\Gamma, (\exists X)\ F]\!]^{\vec{X}} = [\![\Gamma]\!]^{\vec{X}} \parr \left(\mathcal{T}[\![F^{\perp}]\!]^{\vec{X},X}\right)^{\perp}$;
  - $[\![\Gamma, F[G/X]]\!]^{\vec{X},\vec{Y}} = \mathcal{U}^k[\![\Gamma]\!]^{\vec{X}} \parr [\![F]\!]^{\vec{X},X}_{\mathbf{Id}^n, G/}$.

  (Note that we need to "pad" the context with some $\mathcal{U}$*'s to get an interface of appropriate arity.)*

We have:

$\varepsilon^{F^{\perp}}_{\mathbf{Id}^n, G/} :: \mathcal{U}^k \mathcal{T}(F^{\perp}) \multimap (F^{\perp}_{\mathbf{Id}^n, G/})$

$\Leftrightarrow$ { the action of $\_^{\perp}$ on morphism is just the converse operation on relations: $r \mapsto r^{\sim}$ }

$\left(\varepsilon^{F^{\perp}}_{\mathbf{Id}^n, G/}\right)^{\sim} :: (F_{\mathbf{Id}^n, G}) \multimap \mathcal{U}^k \mathcal{T}(F^{\perp})^{\perp}$

$\Leftrightarrow$

$\left(\varepsilon^{F^{\perp}}_{\mathbf{Id}^n, G/}\right)^{\sim} :: [\![F[G/X]]\!]^{\vec{X},\vec{Y}} \multimap \mathcal{U}^k [\![(\exists X)\ F]\!]^{\vec{X}}$.

We can compose that with $[\![\pi']\!]$ and obtain

$$\left(\varepsilon^{F^{\perp}}_{\mathbf{Id}^n, G/}\right)^{\sim} \cdot [\![\pi']\!] \quad :: \quad \mathcal{U}^k [\![\Gamma, (\exists X)\ F]\!]^{\vec{X}}$$

which implies that $\left(\varepsilon^{F^\perp}_{\mathbf{Id}^n,G/}\right)^\sim \cdot [\![\pi']\!] :: [\![\Gamma, (\exists X)\ F]\!]^{\vec{X}}$.
We define $[\![\pi]\!]$ to be precisely this object. The previous computation showed correctness of the interpretation.

§ *Failure of Cut-Elimination.* We thus obtain a denotational model of second order linear logic: formulas are interpreted by parametric interfaces, and proofs are interpreted by objects of (variable) type, *i.e.* "parametric safety properties". We have shown (for a subtheory of first order linear logic, namely simply typed λ-calculus) that the first order interpretation is invariant under cut elimination and it is easy to extend that to any $\mathbf{PInt}^{(n)}$. However, invariance may fail when eliminating *second order* cut:

$$\dfrac{\dfrac{\pi \vdash \Gamma\ ,\ F}{\vdash \Gamma\ ,\ (\forall X)\ F} \qquad \dfrac{\pi' \vdash F^\perp[G/X]\ ,\ \Delta}{\vdash (\exists X)\ F^\perp\ ,\ \Delta}}{\vdash \Gamma\ ,\ \Delta}$$

reduces to

$$\dfrac{\pi[G/X] \vdash \Gamma\ ,\ F[G/X] \qquad \pi' \vdash F^\perp[G/X]\ ,\ \Delta}{\vdash \Gamma\ ,\ \Delta}$$

(where $\pi[G/X]$ is the obvious proof where X as been replaced by G)
The respective interpretations of those proofs are

- $\pi_1 \triangleq [\![\pi']\!] \cdot (\varepsilon^{F^\perp}_{\mathbf{Id}^n,G/})^\sim \cdot \Lambda[\![\pi]\!]$;
- and $\pi_2 \triangleq [\![\pi']\!] \cdot [\![\pi[G/X]]\!]$.

By lemma 8.4.11, we have that $\pi_1 = [\![\pi']\!] \cdot [\![\pi]\!]_{\mathbf{Id}^n,G/}$; and by lemma 8.4.9, we can easily show (by induction) that $[\![\pi[G/X]]\!] \subseteq [\![\pi]\!]_{\mathbf{Id}^n,G/}$. We thus obtain:

$$\pi_2 = [\![\pi']\!] \cdot [\![\pi[G/X]]\!] \quad \subseteq \quad [\![\pi']\!] \cdot [\![\pi]\!]_{\mathbf{Id}^n,G/} = \pi_1\ .$$

We cannot guarantee that this inclusion is an equality. It means that it is *a priori* possible for the interpretation of proofs to *decrease* during cut elimination. The problem is not too serious because equality will hold as soon as the formula G is "infinite" (for example when it contains free variables or exponentials). In particular, if one interprets system-F in this way, the only "finite" formula is the empty type $(\forall\alpha)\,\alpha$. It is not known at the moment if objects of variable type coming from real proofs may actually decrease during cut elimination.[9]

[9]: Note that this problem is independent of interfaces: it is a question on the relational model.

Conclusion

This work was concerned with the notion of *interaction system*, a graph like structure with a notion of signed transitions between states. Those transitions are alternating between "Angel" and "Demon" transitions; the latter depending on the former. The point of departure was constructive topology, since this structure adequately describes a formal space in type theory, or in constructive predicative mathematics. However, the main motivation remained the computational relevance of interaction systems to describe programming interfaces and programs fulfilling those interfaces. In this respect, most of the early intuitions about interaction systems can be attributed to Peter Hancock.

Prior to that, the abstract structure of interaction systems was studied in chapters 2 and 3, and this was put into application to show that the category of interaction systems forms a non trivial denotational model of linear logic (chapter 6). Some of the additional structure of interaction systems can be interpreted in a similar way by showing that they also model the operation of *differentiation* of the differential λ-calculus (section 6.4). However, if the computational content of interaction system is not needed, the presentation of the model can be simplified by replacing the notion of interaction system by the notion of *predicate transformers*. This is what is done in chapter 7 and the result is a concise and elegant denotational model of full propositional linear logic. This model is then extended to second order using traditional technology: this is the content of chapter 8.

Future Work

Many things remain to be done, ranging from very concrete to very abstract. One long term goal is to reconcile the first part ($\_^*$ monad and $\_^\infty$ comonad) with the second part (?_ monad and !_ comonad), for which no common ground has been found. In particular, as surprising as it may seem, the notion of sequential composition is not used anywhere in the second part!

From an abstract point of view, generalizing the notion of interaction systems or predicate transformers could be enlightening. Two directions are the following:

- consider that S has some structure (order, group or even small category). For example, it would be interesting to see if there is a notion of interaction systems allowing to represent functors from $\widehat{\mathrm{S}}$ to itself. (Where $\widehat{\mathrm{S}}$ is the category of presheaves over a small category.)
- consider S to be an object in a locally cartesian closed category, or more gen-

erally a category with families ([30]). Such a category has as internal language dependent type theory ([81] and [49]). In this context, an interaction system becomes the following:

- an object S in $\mathcal{C}$;
- an object A in $\mathcal{C}/S$;
- an object D in $\mathcal{C}/\Sigma_S A$;
- a morphism n in $\mathcal{C}(\Sigma_S\Sigma_A D\,,\,S)$.[1]

When $\mathcal{C}$ is **Set**, we recover the notion studied in this thesis. The notion of morphism would be given by a span (relation) with some morphisms giving the translations from A_1 to A_2 and from D_2 to D_1. Since the intuitionistic part of the category **Int** was developed in dependent type theory, it probably lifts to the context of category with families.

This point is closely related to the work on *containers* done in Nottingham (see [1]), but taking advantage of the linear structure of "dependent containers", *aka* interaction systems.

Such extensions could in particular shed some light on proposition 2.5.23 (equivalence between interaction systems and predicate transformers), which is highly surprising. It is not expected that such a thing will hold in more general contexts like the one described above.

Another direction would be to use the fact that **Int** is enriched over *complete* sup-lattices to give models for the *untyped* differential λ-calculus. There, one wants to interpret arbitrary Taylor expansions, which is impossible to in the finiteness space model. The desciption of the model for untyped λ-calculus can be found in [54].

On the purely logical side, it would be interesting to see if we can have a "weak" completeness result for multiplicative linear logic: something along the lines of "if x is a safety property in F for all valuation, then x is a union of interpretations of proofs of F".

Trying to see if we can make precise the intuition of "synchrony" used in the multiplicative/exponential connectives. Could process algebra and interaction system learn from each other?

From a purely concrete point of view, trying to use interaction systems (or a variant) to describe some concrete interface should also be done. Another long term goal would be to see if the technology of interaction systems can be used to help developing programs satisfying various specification.

etc.

etc.

etc.

[1]: Recall that it is customary, for an object B of $\mathcal{C}/A$, to write $\Sigma_A B$ for the codomain of B.

Bibliography

[1] Michael Abott, Thorsten Altenkirch, and Neil Ghani. Containers - constructing strictly positive types. To appear in the Journal for Theoretical Computer Science, Special issue on Applied Semantics, 2005. *190*

[2] Samson Abramsky, Simon J. Gay, and Rajagopal Nagarajan. A specification structure for deadlock-freedom of synchronous processes. *Theoretical Computer Science*, 222(1-2):1–53, 1999. *153*

[3] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Information and Computation*, 163(2):409–470, 2000. Result from 1993. *35*

[4] Peter Aczel. An introduction to inductive definitions. In *Handbook of mathematical logic*, pages 739–782, Amsterdam, 1977. North-Holland Publishing Co. Edited by Jon Barwise, With the cooperation of H. Jerome Keisler, Kenneth Kunen, Yiannis Nikolas Moschovakis and Anne Sjerp Troelstra, Studies in Logic and the Foundations of Mathematics, Vol. 90. *16, 35, 39*

[5] Peter Aczel. The type theoretic interpretation of constructive set theory: inductive definitions. In *Logic, methodology and philosophy of science, VII (Salzburg, 1983)*, volume 114 of *Studies in Logic and Foundation of Mathematics*, pages 17–49. North-Holland, Amsterdam, 1986. *28*

[6] Peter Aczel and Michael Rathjen. Notes on constructive set theory, 2001. Report 40, preprint from the Mittag-Leffler institute, Stockholm. *27, 52, 85*

[7] Thorsten Altenkirch and Thierry Coquand. A finitary subsystem of the polymorphic λ-calculus. In *Typed Lambda Calculi and Applications, TLCA 2001*, number 2044 in Lecture Notes in Computer Science, pages 22–28, 2001. *27*

[8] Ralph-Johan Back and Joakim von Wright. *Refinement calculus, A systematic introduction*. Graduate Texts in Computer Science. Springer-Verlag, New York, 1998. *50, 55*

[9] Ralph-Johan Back and Joakim von Wright. Encoding, decoding and data refinement. Technical Report 236, Turku Center for Computer Science, February 1999. *60*

[10] Ralph-Johan Back and Joakim von Wright. Product in the refinement calculus. Technical Report 235, Turku Center for Computer Science, February 1999. *146*

[11] Henk Pieter Barendregt. Lambda calculi with types. In *Handbook of logic in computer science, Vol. 2*, Oxford Science Publishing, pages 117–309. Oxford University Press, New York, 1992. *15, 28*

[12] Michael Barr. Toposes without points. *Journal of Pure and Applied Algebra*, 5:265–280, 1974. *107*

[13] Michael Barr. *$*$-autonomous categories*, volume 752 of *Lecture Notes in Mathematics*. Springer, Berlin, 1979. With an appendix by Po Hsiang Chu. *82*

[14] Stefano Berardi, Marc Bezem, and Thierry Coquand. On the computational content of the axiom of choice. *The Journal of Symbolic Logic*, 63(2):600–622, 1998. *31*

[15] Marc Bezem and Thierry Coquand. Newman's lemma—a case study in proof automation and geometric logic. *Bulletin of the European Association for Theoretical Computer Science. EATCS*, (79):86–100, 2003. *8, 35, 101*

[16] Pierre Boudes. Non-uniform hypercoherences. In Rick Blute and Peter Selinger, editors, *Electronic Notes in Theoretical Computer Science*, volume 69. Elsevier, 2003. *121*

[17] Douglas Bridges and Fred Richman. *Varieties of constructive mathematics*, volume 97 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 1987. *13*

[18] Alexandra Bruasse-Bac. *Logique linéaire indexée du second ordre*. Thèse de doctorat, Université Aix-Marseille II, U.F.R de sciences, 2001. *11, 161, 169, 176*

[19] Alexandra Bruasse-Bac. On phase semantics and denotational semantics: the second order. Unpublished, 2002. *161, 169, 176*

[20] Alexandra Bruasse-Bac and Thomas Ehrhard. Second order relational interpretation, 2004. Unpublished note. *169*

[21] Wilfried Buchholz and Kurt Schütte. *Proof theory of impredicative subsystems of analysis*, volume 2 of *Studies in Proof Theory. Monographs*. Bibliopolis, Naples, 1988. *27*

[22] Thierry Coquand. A semantics of evidence for classical arithmetic. *The Journal of Symbolic Logic*, 60(1):325–337, 1995. *35*

[23] Thierry Coquand. Formal topology with posets, 1996. Unpublished note. *18, 35*

[24] Thierry Coquand. A completeness proof for geometric logic, 2003. Unpublished note. *8, 35, 101*

[25] Thierry Coquand. Completeness theorems and lambda-calculus. In Pawel Urzyczyn, editor, *TLCA*, volume 3461 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2005. *27*

[26] Thierry Coquand and Catarina Coquand. The Agda proof assistant. `http://www.cs.chalmers.se/~catarina/agda/`, 2000. *17*

[27] Thierry Coquand, Giovanni Sambin, Jan Magnus Smith, and Silvio Valentini. Inductively generated formal topologies. *Annals of Pure and Applied Logic*, 124(1-3):71–106, 2003. *35, 62, 89, 94, 95*

[28] Edsger Wybe Dijkstra. *A discipline of programming*. Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1976. With a foreword by Charles Anthony Richard Hoare, Prentice-Hall Series in Automatic Computation. *50*

[29] Albert Grigorevich Drágalin. A completeness theorem for intuitionistic predicate logic. An intuitionistic proof. *Publicationes Mathematicae Debrecen*, 34(1-2):1–19, 1987. *35*

[30] Peter Dybjer. Internal type theory. In *TYPES '95: Selected papers from the International Workshop on Types for Proofs and Programs*, pages 120–134, London, UK, 1996. Springer-Verlag. *190*

[31] Thomas Ehrhard. Finiteness spaces. To appear in Mathematical Structures in Computer Science, 2004. *121*

[32] Thomas Ehrhard and Laurent Régnier. The differential lambda calculus. *Theoretical Computer Science*, 309(1):1–41, 2003. *130, 132*

[33] Thomas Ehrhard and Laurent Régnier. Uniformity and the taylor expansion of ordinary lambda-terms. Submitted for publication, 2004. *132*

[34] Solomon Feferman. Systems of predicative analysis. *The Journal of Symbolic Logic*, 29:1–30, 1964. *26*

[35] Paul Gardiner, Clare Martin, and Oege De Moor. An algebraic construction of predicate transformers. In Charles Carroll Morgan and Jim C. P. Woodcock, editors, *Mathematics of Program Construction*, volume 669 of *Lecture Notes in Computer Science*, pages 100–121, 1993. *53*

[36] Sylvia Gebellato and Giovanni Sambin. Pointfree continutity and convergence (the Basic Picture, IV), 2002. Draft. *87, 89, 100*

[37] Jean Yves Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. Thèse de doctorat, Université Paris VII, 1972. *28*

[38] Jean-Yves Girard. The system F of variable types, fifteen years later. *Theoretical Computer Science*, 45(2):159–192, 1986. *11, 161, 173*

[39] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1), 1987. *103, 111, 119*

[40] Timothy G. Griffin. The formulae-as-types notion of control. In *Conference Record 17th Annual ACM Symposium on Principles of Programming Languages, POPL'90, San Francisco, CA, USA, 17–19 Jan 1990*, pages 47–57. ACM Press, New York, 1990. *31*

[41] Andrzej Grzegorczyk. A philosophically plausible formal interpretation of intuitionistic logic. *Indagationes Mathematicae, Koninklijke Nederlandse Akadademie van Wetenschappen, Proceedings Series A 67*, 26:596–601, 1964. *35*

[42] Peter Hancock. *Ordinals and interactive programs*. PhD, Laboratory for Foundations of Computer Science, University of Edinburgh, 2000. *7*

[43] Peter Hancock and Pierre Hyvernat. Programming interfaces and basic topology. *Annals of Pure and Applied Logic*, 137(1):189–239, 2006. *12*

[44] Peter Hancock and Anton Setzer. Interactive programs in dependent type theory. In *Computer science logic (Fischbachau, 2000)*, volume 1862 of *Lecture Notes in Computer Science*, pages 317–331. Springer, Berlin, 2000. *35*

[45] Peter Hancock and Anton Setzer. Specifying interactions with dependent types. In *Workshop on subtyping and dependent types in programming, Portugal, July 7th 2000*, 2000. *35*

[46] Peter Hancock and Anton Setzer. Guarded induction and weakly final coalgebras in dependent type theory. 16 pp. To appear in proceedings of the workshop "From Sets and Types to Topology and Analysis. Towards Practicable Foundations for Constructive Mathematics", 12-16 May 2003, Venice International University (VIU), San Servolo, Venice, Italy., 2004. *35, 46*

[47] Hugo Herbelin. Strong sums + callcc implies proof-irrelevance, 2002. Proof in Coq, unpublished. *31*

[48] Martin Hofmann. *Extensional concepts in intensional type theory*. PhD thesis, Laboratory for Foundations of Computer Science, Edinburgh, 1995. available as a research report ECS-LFCS-95-327. *22*

[49] Martin 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*, pages 427–441, London, UK, 1995. Springer-Verlag. *190*

[50] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford Univ. Press, New York, 1998. *22*

[51] John Martin Elliott Hyland and Luke Chih-Hao Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 163(2):285–408, 2000. Result from 1993. *35*

[52] Pierre Hyvernat. Interactive programs in pure (martin-Löf) type theory, 2001. Master's thesis, under the supervision of Thierry Coquand. *35*

[53] Pierre Hyvernat. Predicate transformers and linear logic: yet another denotational model. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *18th International Workshop CSL 2004*, volume 3210 of *Lecture Notes in Computer Science*, pages 115–129. Springer-Verlag, September 2004. *12, 151*

[54] Pierre Hyvernat. Interaction systems and linear logic — ¿A different games semantics? submitted to Annals of Pure and Applied Logic, 2005. *12, 190*

[55] Pierre Hyvernat. Synchronous games, simulations and λ-calculus. In Dan R. Ghica and Guy McCusker, editors, *Games for Logic and Programming Languages, GaLoP (ETAPS 2005)*, pages 1–15, April 2005. *12, 133*

[56] Peter T. Johnstone. The point of pointless topology. *Bulletin of the American Mathematical Society*, 8(1):41–53, 1983. *88*

[57] Jean-Louis Krivine. Dependent choice, 'quote' and the clock. *Theoretical Computer Science*, 308(1-3):259–276, 2003. *31*

[58] Paul Blain Levy. Martin-Löf clashes with Griffin, 2003. Unpublished note. *31*

[59] Ingrid Lindström. A construction of non-well-founded sets within Martin-Löf's type theory. *The Journal of Symbolic Logic*, 54(1):57–64, 1989. *18*

[60] Maria Emilia Maietti and Silvio Valentini. Can you add power-sets to Martin-Löf's intuitionistic set theory? *MLQ. Mathematical Logic Quarterly*, 45(4):521–532, 1999. *28*

[61] Per Martin-Löf. *Notes on constructive mathematics*. Almqvist & Wiksell, Stockholm, 1970. *13*

[62] Per Martin-Löf. *Intuitionistic type theory*. Bibliopolis, Naples, 1984. Notes by Giovanni Sambin. *13*

[63] Paul André Melliès. Categorical models of linear logic revisited, 2002. To appear in Theoretical Computer Science,. *116*

[64] Markus Michelbrink. Interfaces as functors, programs as coalgebras - a final coalgebra theorem in intensional type theory. unpublished, available from `http://www.cs.swan.ac.uk/~csmichel/`, 2005. *18, 19, 35*

[65] Markus Michelbrink. Interfaces as games, programs as strategies. unpublished, available from `http://www.cs.swan.ac.uk/~csmichel/`, 2005. *38*

[66] Markus Michelbrink and Anton Setzer. State-dependent IO-monads in type theory. Submitted, 2004. *35*

[67] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3):267–310, 1983. *40*

[68] Bengt Nordström, Kent Petersson, and Jan Magnus Smith. *Programming in Martin-Löf's type theory. An introduction*. The Clarendon Press Oxford University Press, New York, 1990. *13, 22, 35*

[69] Mitsuhiro Okada. A uniform semantic proof for cut-elimination and completeness of various first and higher order logics. *Theoretical Computer Science*, 281(1-2):471–498, 2002. Selected papers in honour of Maurice Nivat. *103*

[70] Kent Petersson and Dan Synek. A set constructor for inductive sets in Martin-Löf's type theory. In *Proceedings of the 1989 Conference on Category Theory and Computer Science, Manchester, UK*, volume 389 of *Lecture Notes in Computer Science*. Springer Verlag, 1989. *17, 35, 39*

[71] Christian Retoré. *Réseaux et séquents ordonnés*. Thèse de doctorat, Université Paris 7, 1993. *129*

[72] Christian Retoré. Pomset logic: a non-commutative extension of classical linear logic. In *Typed lambda calculi and applications (Nancy, 1997)*, volume 1210 of *Lecture Notes in Comput. Sci.*, pages 300–318. Springer, Berlin, 1997. *129*

[73] John C. Reynolds. Towards a theory of type structure. In *Programming Symposium (Paris, 1974)*, pages 408–425. Lecture Notes in Computer Science, Volume 19. Springer, Berlin, 1974. *28*

[74] John C. Reynolds. Polymorphism is not set-theoretic. In *Semantics of data types (Valbonne, 1984)*, volume 173 of *Lecture Notes in Computer Science*, pages 145–156. Springer, Berlin, 1984. *28*

[75] Paul Ruet. Self-adjoint negation, 2002. prépublication IML, 2002-23. *114*

[76] Giovanni Sambin. Intuitionistic formal spaces—a first communication. In *Mathematical logic and its applications (Druzhba, 1986)*, pages 187–204. Plenum, New York, 1987. *89*

[77] Giovanni Sambin. Pretopologies and completeness proofs. *The Journal of Symbolic Logic*, 60(3):861–878, 1995. *103, 104*

[78] Giovanni Sambin. The Basic Picture, a structure for topology (the Basic Picture, I), 2001. Preprint n. 26, Dipartimiento di matematica, Università di Padova. *38*

[79] Giovanni Sambin. Basic topologies, formal topologies, formal spaces (the Basic Picture, III), 2002. Draft. *89, 97, 99*

[80] Giovanni Sambin and Silvio Valentini. Building up a toolbox for Martin Löf's type theory: subset theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, pages 221–244. Oxford University Press, New York, 1998. *19, 20*

[81] Robert A. G. Seely. Locally Cartesian closed categories and type theory. *Mathematical Proceedings of the Cambridge Philosophical Society*, 95(1):33–48, 1984. *190*

[82] Alfred Tarski. Fundamentale Begriffe der Methodologie der deduktiven Wissenschaften. I. *Monatshefte für Mathematik und Physik*, 37:361–404, 1930. *35*

[83] Anne Sjerp Troelstra and Dirk van Dalen. *Constructivism in mathematics. Volume I*, volume 121 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam, 1988. *13*

[84] Anne Sjerp Troelstra and Dirk van Dalen. *Constructivism in mathematics. Volume II*, volume 123 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam, 1988. *13*

[85] Wim Veldman. On the constructive contrapositions of two axioms of countable choice. In *The Luitzen Egbertus Jan Brouwer Centenary Symposium (Noordwijkerhout, 1981)*, volume 110 of *Studies in Logic and the Foundations of Mathematics*, pages 513–523. North-Holland, Amsterdam, 1982. *31*

[86] Gavin Charles Wraith. Intuitionistic algebra: some recent developments in topos theory. In *Proceedings of the International Congress of Mathematicians (Helsinki, 1978)*, pages 331–337, Helsinki, 1980. Academia Scientiarum Fennica. *107*

Index

Résumé en français :
Cette thèse, s'intéresse aux *systèmes d'interaction,* une notion visant à modéliser les interactions entre un système informatique et son environnement.

La première partie développe, dans le cadre de la théorie des types de Martin-Löf, la théorie de base des systèmes d'interaction et des constructions inductives et co-inductives qu'ils permettent. On trouve dans cette partie une étude des liens entre systèmes d'interaction et topologies formelles et une formulation (en terme de systèmes d'interaction) d'un théorème de complétude vis-à-vis d'une sémantique topologique des théories géométriques (linéaires).

Dans cette étude, la notion complètement standard de *simulation,* joue un rôle fondamental car elle permet de définir la notion de morphisme entre systèmes d'interaction. Ceci permet d'établir une équivalence entre la catégorie ainsi définie et une autre catégorie, beaucoup plus simple à décrire, celle des *transformateurs de prédicats.*

En traduisant dans ce nouveau vocabulaire les constructions précédentes, on observe que les transformateurs de prédicats forment un nouveau modèle de la logique linéaire, qui est décrit puis étendu au second ordre.

Enfin, les propriétés particulières des systèmes d'interaction / transformateurs de prédicats sont mises à profit pour donner une interprétation du λ-*calcul différentiel.* Cela suppose d'introduire du non déterminisme, ce que les systèmes d'interaction et les transformateurs de prédicats permettent de faire.

Mots-cls en franais : thorie de la dmonstration, logique linaire, smantique dnotationnelle, thorie des types dpendants, topologie constructive, interaction, simulation, transformateurs de prdicats, second-ordre

Titre original en anglais : A Logical Investigation of Interaction Systems

Résumé en anglais :

The topic of this thesis is the study of *interaction systems,* a notion modeling interactions between a program and its environment.

The first part develops the general theory of those interaction systems in Martin-Löf dependent type theory. It introduces several inductive and coinductive definitions of interest on those interaction systems. We study in particular the strong link between interaction systems and formal topology and give an application by formulating a completeness theorem (in terms of interaction systems) with respect to a topological semantics for (linear) geometric theories.

In all the thesis, a central notion is that of *simulations:* it allows to define the notion of morphism between interaction systems. It is possible to prove an equivalence between this category and the simpler category of *predicate transformers.*

We can then translate the constructions from the first part in this new context and obtain a new denotational model for linear logic. This model is then extended to second-order.

Finally, specific properties of interaction systems / predicate transformers are used to give a model of the *differential* λ-*calculus.* This presupposes the addition of non-determinism, which is fully supported by interaction systems / predicate transformers.

Mots-clés en anglais : proof-theory, linear-logic, denotational semantics, dependent type theory, constructive topology, interaction, simulation, predicate transformers, second order

Adresse du laboratoire : Institut mathématique de Luminy — UPR 9016, CNRS — 163 avenue de Luminy, Case 907 — 13 288 Marseille Cedex 9 — France