Théorie de la démonstration

Transparents complémentaires / de transition

Pierre-Louis Curien (CNRS and University Paris 7)

cours M2 LMFI, 2009

Biblio :

- P.-L. Curien, Introduction to LL and ludics, part n [LL-lud-intro-n]
- P.-L. Curien, Sequential algorithms as bistable maps [obs-seq-bistable]
- K. Terui, Computational ludics [Terui]
- (connexe, optionnel) P.-L. Curien, Notes on game semantics [intro-game-sem]

(Par défaut, les références de pages renvoient aux transparents de la première partie du cours)

Première partie : logique classique et opérateurs de contrôle

Le cours enseigné cette année présente une lacune : je n'ai pas expliqué le lien entre calcul des séquents et machines abstraites. Voici quelques "turbo-explications". Je vous renvoie (optionnel !) à mes notes *Abstract machines, control, and sequents* pour plus de détails.

1. on peut lire une commande $\langle v \mid e \rangle$ comme un contexte e (c'est-à-dire un programme avec un trou) dont le trou est rempli par v, ce que l'on écrit aussi $e[v]$.
2. Dans une machine abstraite (SECD, CAM, machine de Krivine) pour le λ-calcul, le contexte est représenté comme une pile. Donc on a "pile = contexte (d'exécution)".
3. L'extension de la logique intuitionniste à la logique classique correspond via Curry-Howard à l'extension du λ-calcul avec des opérateurs de contrôle (Griffin, 1991) (voir exercice ci-dessous).

Exercice : Montrer que les codages suivants :

$$\begin{array}{lll} \kappa k.V = \mu\beta.\langle \star_\beta \mid \tilde{\mu}k.\langle V^\diamond \mid \beta \rangle \rangle & & \\ \star_e = (\tilde{\mu}(x, \beta^\bullet).\langle x^\diamond \mid e \rangle)^\bullet & (\beta \text{ non libre dans } e) \\ V \cdot e = \tilde{\mu}\alpha^\bullet.\langle (V, e^\bullet)^\diamond \mid \alpha \rangle & (\text{pile de sommet } V) \end{array}$$

satisfont les règles de réduction suivantes :

$$\begin{array}{lll} \langle \kappa k.V \mid e \rangle & \longrightarrow^* & \langle V[\star_e/k]^\diamond \mid e \rangle & \text{(capture)} \\ \langle \star_{e_1} \mid V \cdot e_2 \rangle & \longrightarrow^* & \langle V \mid e_1 \rangle & \text{(activation)} \end{array}$$

Deuxième partie : logique linéaire

La logique linéaire est une logique où les formules peuvent être considérées comme des ressources. Ainsi, le séquent

$$A \otimes A \otimes B \vdash C$$

se lira : pour avoir C, il faut "consommer" deux "unités" de A et une de B (C pourrait être une boisson à 1,2 euros, A une pièce de 50 centimes et B un pièce de 20 centimes).

Un nouveau connecteur apparaît, appelé "bien sûr. Le possesseur d'une ressource $!A$ est très "riche", car il peut utiliser A autant de fois qu'il veut.

Les deux décompositions fondatrices et l'isomorphisme fondateur de la logique linéaire sont

$$A \Rightarrow B = (!A) \multimap B \qquad A \multimap B = \overline{A} \parr B \qquad (!A) \otimes (!B) = !(A \& B)$$

(NB : La notation utilisée dans les travaux de logique linéaire (y compris dans [IntroLL]) est $A^{\perp}$.)

L'isomorphisme explique la terminologie : multiplicatif ($\otimes$), additif ($\&$), exponentiel (!) (cf. $e^a \times e^b = e^{a+b}$).

Réversible/irréversible versus additif/multiplicatif

La dichotomie qui nous a guidée dans la première partie est celle entre connecteurs irréversibles ($\otimes$, $\oplus$) et connecteurs réversibles ($\&$, $\parr$).

La dichotomie d'origine en logique linéaire est celle entre connecteurs multiplicatifs ($\otimes$ et son dual $\parr$) et additifs ($\&$ et son dual $\parr$). Mais la typographie, guidée par la distributivité entre $A \otimes (B \oplus C)$ et $(A \otimes B) \oplus (A \otimes C)$ anticipait déjà l'autre dichotomie.

Il faut lire maintenant (cf. transparent suivant) $\neg^{+} P$ comme $?\overline{P}$.

Traduction dans LL

On décrit ici la dernière étape de la chaîne de traductions de la p.89. On remarque d'abord que l'image de l'avant-dernière traduction (de LKQ dans NJ, p. 53-54) atteint un sous-ensemble de l'ensemble des formules / types de NJ :

Commandes	Valeurs	Contextes
R	$A ::= X \mid R^A \mid A \times A$	R^A

que l'on traduit comme suit :

$$X_{LL} = X \qquad (R^A)_{LL} = ?\overline{P} \qquad (A \times B)_{LL} = (A_{LL}) \otimes (B_{LL})$$

et l'on a (exercice !) :

$$\left.\begin{array}{r} \Lambda \vdash c : R \\ \Lambda \vdash V : A \\ \Lambda \vdash e : R^A \end{array}\right\} \quad \Rightarrow \quad \left\{\begin{array}{l} !\Lambda_{LL} \vdash \\ !\Lambda_{LL} \vdash A_{LL} \\ !\Lambda_{LL} \vdash (R^A)_{LL} \end{array}\right.$$

Catégories (version “hyper-light”)

Une catégorie a des objets, et des morphismes $f, g, \ldots$ entre les objets (autrement dit un multi-graphe orienté, i.e. un graphe où il y a plusieurs flèches d'un nœud à un autre), plus des morphismes identité et une composition des morphismes qui satisfont (lois de monoïde) :

$$(f \circ g) \circ h = f \circ (g \circ h) \qquad f \circ id = f \qquad id \circ f = f$$

quand ces opérations sont *bien typées*, i.e. $f \circ g$ existe ssi le but de g est la source de f.

Voici trois catégories, dont les deux dernières nous intéressent ici (cf. Definition 6.9 de [LL-lud-intro-1]) :
– la catégorie des ensembles et des fonctions,
– la catégorie des espaces de cohérence et des fonctions linéaires,
– la catégorie des espaces de cohérence et des fonctions stabes.
Un foncteur est un morphisme F entre deux catégories $\mathbf{C}$, $\mathbf{C}'$, qui envoie les objets de $\mathbf{C}$ vers les objets de $\mathbf{C}'$, et les morphismes de $\mathbf{C}$ vers les morphismes de $\mathbf{C}'$, en respectant source, but, identités et compositions.

Adjonction

Si $\mathbf{C}$ et $\mathbf{C}'$ sont deux catégories, et $F : \mathbf{C} \to \mathbf{C}'$, $G : \mathbf{C}' \to \mathbf{C}$ sont deux foncteurs en sens inverses, alors G est adjoint à droite de F si pour tout objet A de $\mathbf{C}$, tout objet A' de $\mathbf{C}'$, il existe une bijection ζ entre

$$\underset{\text{(les morph. de } A \text{ dans } GA' \text{ dans } \mathbf{C})}{\mathbf{C}[A, GA']} \quad \text{et} \quad \underset{\text{(les morph. de } FA \text{ dans } A' \text{ dans } \mathbf{C}')}{\mathbf{C}'[FA, A']}$$

Ces bijections doivent être *naturelles* en A, A' ($\zeta((Gh') \circ f \circ g) = \ldots$ (compléter !))

L'adjonction stable / linéaire

On a (pour tous $(E, \frown)$, $(E', \frown)$) :

1. (Proposition 6.10 de [LL-lud-intro-1]) : $(\mathbf{Coh}[E, E'], \leq_s)$ (fonctions stables ordonnées par l'ordre stable) est isomorphe (en tant qu'ensemble ordonné) à $(D(!E \multimap E'), \subseteq)$;
2. (Exercice 6.12 de [LL-lud-intro-1]) : $(\mathbf{Coh_l}[E, E'], \leq_s)$ (fonctions linéaires ordonnées par l'ordre stable) est isomorphe (en tant qu'ensemble ordonné) à $(D(E \multimap E'), \subseteq)$.

En réunissant les deux, on obtient que ! définit un adjoint à droite du foncteur d'inclusion de $\mathbf{Coh}[E, E']$ dans $\mathbf{Coh_l}[E, E']$:

$$\subseteq \dashv !$$

Déréliction et promotion dans les espaces de cohérence

Pour interpréter ces deux règles de la logique linéaire, nous avons besoin de deux fonctions linéaires ($(E, \frown)$ quelconque) :

$$\epsilon : !E \to E \qquad \delta : !E \to !!E$$

définies respectivement (par leurs traces) comme suit :

$$\{(\{e\}, e) \mid e \in E\} \qquad \{(\cup X, X) \mid X \in !!E\}$$

Ces deux fonctions sont les ingrédients d'une structure de *comonade* (Definition 7.7, [LL-ludic-intro-1]). Voir le bas de la p. 30 et le haut de la p. 31 de [LL-lud-introI] pour l'interprétation des règles de déréliction et promotion, ainsi que pour les autres règles.

Affaiblissement et contraction dans les espaces de cohérence

Pour interpréter ces deux règles , nous avons besoin de deux autres fonctions linéaires ($(E, \frown)$ toujours quelconque, $1 = \{*\}$) :

$$e : !E \to 1 \qquad d : !E \to !E \otimes !E$$

définies respectivement comme suit :

$$\{(\emptyset, *)\} \qquad \{(x_1 \cup x_2, (x_1, x_2)) \mid x_1, x_2 \in !E\}$$

Ces deux fonctions sont les ingrédients d'une structure de *comonoïde* (voir Exercise 7.10 de [LL-ludic-intro-1]).

Structure versus propriété

En mathématiques, on peut distinguer les notions de :
- **structure** : un ensemble *muni* d'une loi associative etc... est un groupe ;
- **propriété** : un groupe *est* ou *n'est pas* commutatif.

Cette division conceptuelle recouvre la dichotomie additifs / mutliplicatifs :
- Le $\&$ et le $\oplus$ sont interprétés comme *le* produit et coproduit (ou somme) au sens catégorique, respectivement. Par exemple, un produit de deux objets A, B est un triplet formé d'un objet noté $A \times B$, de deux morphismes $\pi_1 : A \times B \to A$, $\pi_2 : A \times B \to B$ tels qu'il existe une unique opération

$$\frac{f : C \to A \qquad g : C \to B}{\langle f, g \rangle : C \to A \times B}$$

  satisfaisant (pour tous A', f, g tels que ci-dessus) : $\pi_1 \circ \langle f, g \rangle = f$ et $\pi_2 \circ \langle f, g \rangle = g$. Il est facile de voir qu'un autre produit (C', π'_1, π'_2) de A, B est forcément isomorphe à $A \times B$ dans le sens fort qu'il existe un isomorphisme unique (i.e. un morphisme inversible) $\iota : C' \to (A \times B)$ tel que $\pi_i \circ \iota = \pi'_i$ $(i = 1, 2)$. L'existence de produits est donc une *propriété*.
- Le $\otimes$ et le $\parr$ sont interprétés comme des produits tensoriels au sens catégorique (catégorie monoïdale, voir Definition 7.1 de [LL-ludic-intro-1]). Et là il s'agit de structure. Une *même* catégorie peut être munie de *différents* tenseurs (exemple au transparent suivant !)

Deux sortes d'interprétation pour le "bien sûr"

Dans la sémantique cohérente, on distingue
- le "bien sûr" *ensembliste* (celui d'origine, Definition 6.8 de [LL-ludic-intro-1]),
- le "bien sûr" *multi-ensembliste*, défini comme suit :
  - Si $(E, \frown)$ est un espace de cohérence, alors $!_m(E)$ est l'ensemble des multi-ensembles à support fini de E. (Formellement, un multi-ensemble à support fini de X est une fonction M de X dans $\mathbb{N}$ qui vaut 0 sauf sur un sous-ensemble fini de X, et on note $M = \{x, x, x, y, y, \ldots\}$ si $M(x) = 3$, $M(y) = 2, \ldots$.)
  - On pose $M_1 \frown M_2$ si l'union des supports de M_1 et de M_2 est une clique/configuration (finie) de E.

Les points forts respectifs (et exclusifs !) sont :
- Le ! ensembliste est *finitaire* en ce sens que si E est fini, alors $!E$ l'est. Il permet aussi d'interpréter $D(!E \multimap E')$ comme un ensemble de *fonctions* de $D(E)$ dans $D(E')$: on reste donc dans un univers mathématique "usuel".
- Le ! multi-ensembliste conduit à des résultats précis sur la relation syntaxe/sémantique : pour certains fragments de la logique linéaire, l'interprétation cohérente des réseaux de preuve est *injective* , ce qui permet de regarder ces réseaux sous un autre angle (voir travaux de Lorenzo Tortora de Falco) et de les plonger dans un ensemble plus grand (celui des cliques). Se pose alors la question de la surjectivité : quelles cliques sont des réseaux (voir travaux de Michele Pagani) ?

On retrouve ces points forts/faibles dans les sémantiques de jeux, voir plus bas.

Troisième partie : séquentialité et bistabilité

Il s'agit de montrer un autre exemple d'interprétation de la logique linéaire (en fait, seulement de la logique linéaire intuitionniste), qui préfigure la *sémantique des jeux* (sujet devenu très actif, voir par exemple [intro-game-sem]), et, dans une certaine mesure, la ludique (dernière partie du cours).

De continu à stable

Soit $\mathbb{B} = \{V, F\}$. La disjonction logique est la fonction $or : \mathbb{B} \otimes \mathbb{B} \to \mathbb{B}$ définie par la bonne vieille table de vérité :

$$or(V,V) = V \qquad or(V,F) = V \qquad or(F,V) = V \qquad or(F,F) = F$$

Soit maintenant $\mathbb{B}_\perp = \{\perp, V, F\}$, avec l'ordre $x \leq y$ ssi $x = y$ ou $x = \perp$. Alors il y a *quatre* fonctions continues (= croissantes dans ce cas simple où il n'y a pas de chaîne infinie croissante) au sens de Scott de $\mathbb{B}_\perp \times \mathbb{B}_\perp$ dans $\mathbb{B}_\perp$ dont la restriction à $\mathbb{B}$ est *or* :

$$\begin{array}{ll} por(\perp, V) = V & por(V, \perp) = V \\ lor(\perp, V) = \perp & lor(V, \perp) = V \\ ror(\perp, V) = V & ror(V, \perp) = \perp \\ sor(\perp, V) = \perp & sor(V, \perp) = \perp \end{array}$$

(Noter que nécessairement $(\perp, F)$ et $(F, \perp)$ sont envoyés sur $\perp$: pourquoi ?)

Exercice : Montrer que seules les trois dernières sont stables.

L'intuition pour "rejeter" *por* est qu'elle ne peut pas être programmée dans un langage "séquentiel", car le texte d'un tel programme doit nécessairement commencer par (avec $z = x$ ou $z = y$) :

$$\lambda(x,y).\ if\ z\ \ldots\ then\ \ldots\ else\ \ldots$$

(Pour *por*, il faudrait un langage permettant $\lambda x.\ if\ (x\ or\ y)\ =\ \ldots.$)

De stable à séquentiel

Mais avec cette intuition, il y a aussi des fonctions stables à rejeter ! Soit

$$BK : \mathbb{B}_\perp \times \mathbb{B}_\perp \times \mathbb{B}_\perp \to \mathbb{B}_\perp$$

(B pour Berry, K pour Kleene, qui ont discuté cette fonction indépendamment) telle que :

$$BK(V, F, \perp) = V \qquad BK(F, \perp, V) = V \qquad BK(\perp, V, F) = V$$

(Ce que fait cette fonction ailleurs n'est pas important, du moment qu'elle reste croissante, ses valeurs sur ces trois éléments importent également peu du moment qu'elles sont $\neq \perp$; enfin on peut aussi partir avec $(F, V, \perp)$ et de même continuer la permutation circulaire.)

Exercice : Montrer que cette fonction est stable.

Et pourtant BK n'est pas séquentielle, i.e., il n'existe pas de programme de la forme $\lambda(x_1, x_2, x_3).\ if\ x_i\ \ldots$ ($i \in \{1, 2, 3\}$). S'il en existait un, alors la fonction serait *stricte* en l'un de ses arguments. Une fonction f à n arguments est dite stricte en son ième argument si :

pour tous $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ on a $f(x_1, \ldots, x_{i-1}, \perp, x_{i+1}, \ldots, x_n) = \perp$.

Les algorithmes séquentiels

Cette observation est le point de départ de la théorie des *algorithmes séquentiels* de Berry-Curien. Quelques étapes (les références sont prises de [obs-seq-bistable]) :

– 1978 : Construction du modèle [3,9,1]
– 1981 : Conception du langage de programmation CDS, “dérivé” du modèle [4,9]
– 1992 : (Cartwright-Felleisen-Curien) Ajout de l’idée de séquentialité *observable* [5,6]
– 1993 : (Lamarche) Décomposition linéaire pour les algorithmes séquentiels, i.e. définition d’un “bien sûr” *ensembliste* pour ce modèle [17,7]
– 2002 (Laird) Bistabilité [15]

Des espaces de cohérence aux structures de données séquentielles

Un espace de cohérence a :
– des événements e, et
– une relation de cohérence entre événements.
Une structure de donnée séquentielle (sds) a :
– des *cellules* (penser à un champ dans un “record”, ou au nom d’un registre),
– des *valeurs* (la valeur d’un champ, la valeur placée dans un registre),
C’est un niveau de détail plus fin : intuitivement, on récupère un espace de cohérence en prenant pour événements des paires (c, v), avec comme relation de cohérence :

$$(c_1, v_1) \frown (c_2, v_2) \quad \text{ssi} \quad (c_1 = c_2 \Rightarrow v_1 = v_2)$$

(on ne peut mettre qu’une seule valeur dans une cellule).

Formellement, il faut faire cela au niveau des *requêtes* (suites $q = c_1 v_1 \dots c_n v_n c$) et des *réponses* (suites $r = c_1 v_1 \dots c_n v_n c v$) :
– les événements sont les réponses,
– $r_1 \frown r_2$ ssi $r = q v_1$ et $r = q v_2$ $(v_1 \neq v_2)$;
– l’on voit qu’il existe aussi une structure supplémentaire, absente des espaces de cohérence, mais présente dans les structures d’événements (Nielsen, Plotkin, Winskel). On pose $r_1 < r_2$ si r_1 est un préfixe de r_2 : une “clique”, pour coïncider avec la définition de stratégie (Definition 3.2 de [obs-seq-bistable]) doit être également close par cette relation, dite de *causalité*.

Algorithme séquentiel versus fonction séquentielle

Prenons sor. Cette fonction peut être programmée de deux manières différentes, selon que $z = x$ ou $z = y$ dans $\lambda(x, y).\ if\ z\ \ldots$. Ces deux programmes peuvent être décrits en tant que stratégies $lsor$ et $rsor$ comme suit (cf. p. 5 de [obs-seq-bistable] pour le produit, et Definition 3.5 pour l'espace de fonctions) , avec la convention que les suffixes $1, 2, \epsilon$ font respectivement référence au type du premier argument, du second argument, et du résultat. On ne donne que les réponses maximales de la stratégie (pour l'ordre de causalité discuté plus haut) :

$$lsor = \{(?.\epsilon)(?.1)(V.1)(?.2)(V.2)(V.\epsilon)\ldots\}$$
$$rsor = \{(?.\epsilon)(?.2)(V.2)(?.1)(V.1)(V.\epsilon)\ldots\}$$

où les $\ldots$ énumèrent les réponses correspondant aux trois autres clauses de la définition de la disjonction sur $\mathbb{B}$, dans l'ordre 1-2 et dans l'ordre 2-1 respectivement.

Après $\bot$, le $\top$!

Si l'on se donne maintenant "un autre $\bot$", on peut *observer* la différence entre $lsor$ et $rsor$.

Soit $\mathbb{B}_\bot^\top = \{\bot, V, F, \top\}$. Alors l'exécution (cf. Definition 3.8 de [obs-seq-bistable]) fournit les valeurs suivantes en $(\bot, \top)$:

$$lsor(\bot, \top) = \bot \qquad rsor(\bot, \top) = \top$$

Les deux éléments $\bot$ et $\top$ jouent dans cet exemple un rôle parfaitement symétrique, celui d'un arrêt du calcul avec la remontée au top level d'une information d'"erreur". Mais $\bot$ est aussi utilisé pour dénoter une exécution qui boucle, d'où une différentiation :

– $\top$ représente un arrêt délibéré, typiquement "j'ai obtenu l'information que je souhaitais" (par exemple le fait que $rsor$ est stricte à droite).
– $\bot$ représente une attente d'information, qui peut durer éternellement !

Les deux ! de la sémantique des jeux

Le ! de Lamarche (Definition 3.6 de [obs-seq-bistable]) est “ensembliste” : les réponses de $!\mathrm{S}$ sont des énumérations (qui respectent la causalité) de stratégies finies de S.

Le ! de la sémantique des jeux de Hyland et Ong (HO) (voir [intro-game-sem]) est “multi-ensembliste”. Il y a par exemple, au sens HO, un nombre infini de stratégies qui calculent des fonctions de $\mathbb{B}$ dans $\mathbb{B}$:

$$\begin{aligned}\sigma_1 &= \{(?.\epsilon)(?.1)(V.1)(V.\epsilon), \ldots\}\\ \sigma_2 &= \{(?.\epsilon)(?.1)(V.1)(?.1)(V.1)(V.\epsilon), \ldots\}\\ &\vdots\end{aligned}$$

Comme pour la sémantique cohérente (cf. p.11 ci-dessus), ces choix mènent à des résultats assez différents :

– Le ! de Lamarche est finitaire.
– Le ! de Hyland et Ong induit une interprétation injective des preuves/programmes (en forme normale, potentiellement infinie), et la caractérisation de l’image de cette injection par la condition d’innocence et de bon parenthésage a été le point de départ d’une remarquable classification des langages séquentiels.

Complète adéquation

Les travaux sur la séquentialité et les jeux ont été en grande partie motivés par des problèmes de *complète adéquation*, entre un langage (de preuves, de programmation) et un modèle. Le modèle induit une équivalence. Le langage lui-même, muni de sa sémantique opérationnelle (de sa stratégie d'élimination des coupures) induit une autre équivalence, dit *observationnelle* : deux termes sont égaux si aucune expérience ne peut les distinguer. Une expérience est un contexte, ou une contre-preuve (cf. le dernier critère de correction étudié, Definition 1.15 de [LL-lud-intro-2]). Un modèle est complètement adéquat lorsque l'équivalence qu'il induit coïncide avec l'équivalence observationnelle.

On peut montrer (voir mon article *Definability and full abstraction*) qu'un modèle est complètement adéquat s'il est telle que deux éléments distincts du modèle peuvent être séparés par un morphisme *définissable*. Le langage est alors assez expressif pour séparer les points du modèle.

On peut citer les deux résultats suivants :

– approche "ensembliste" : le modèle de Berry-Curien est complètement adéquat pour un langage avec exceptions / opérateurs de contrôle (mais sans références) (Cartwright-Curien-Felleisen)
– approche "mutli-ensembliste" : le modèle HO est complètement adéquat pour un langage avec références (Abramsky-McCusker)

Quatrième partie : ludique

Dans la troisième partie, on a construit une sds/jeu pour chaque formule, ce qui permet d'interpréter une preuve de A par une stratégie du jeu interprétant A.

La ludique reprend tous ces aspects :

– preuves comme stratégies,
– élimination des coupures comme interaction entre stratégies ,
– observations à l'aide d'une constante $\top$ (notée ✠ par Girard), permettant un résultat de
– séparation, i.e. le fait que deux stratégies qui se comportent de la même façon (ce qui dans le langage de [obs-seq-bistable] s'expime comme "définissent la même fonction séquentielle observale") coïncident.

mais dans un cadre encore plus pur, *non typé*.

Typé versus non typé

Dans l'histoire des modèles du λ-calcul, ce sont d'abord les modèles non typés qui ont été l'objet de travaux (parce qu'ils représentaient un défi de cardinalité, s'agissant de plonger $D \to D$ dans D, afin de pouvoir donner un sens à la possibilité dans le λ-caclul sans types d'appliquer un terme à lui-même) (Longo, Meyer, Hindley,... début des annés 70).

Les premiers travaux sur la sémantique des types, et en particulier sur la complétude des algorihmes d'inférence de type à la base d'un langage fonctionnel comme CAML, reposaient sur un modèle non typé, dans lesquels les types étaient interprétés comme des sous-ensembles particuliers (Hindley, et d'autres).

Puis sont venues (fin des année 70) les sémantiques catégoriques, où il est plus naturel d'interpréter d'abord les types (comme des objets), puis les termes (comme des morphismes).

Des liens entre les deux approches étaient connus :

1. modèles non typés comme objets universels dans un modèle typé,
2. construction d'un modèle typé à partir d'un modèle non typé.

La ludique suit l'approche 2, en interprétant les types comme des *comportements* (voir [Terui]).

Références : livres d'Amadio-Curien (Domains and lambda-calculi), de Barendregt (Lambda-calculus : syntax and semantics)