Sémantique des programmes fonctionnels probabilistes à la lumière de la logique linéaire

Séminaire au Collège de France

Christine Tasson

23 janvier 2019

Sémantique dénotationnelle

Programmation fonctionnelle probabiliste

La logique linéaire

Call-By-Push-Value Probabiliste

Sémantique dénotationnelle

La **Syntaxe** décrit comment écrire les programmes.

```
let rec fact n = if n=0 then 1 else n*(fact (n-1))
```

La **Sémantique** décrit ce que les programmes calculent et comment.

La **sémantique opérationnelle** décrit l'exécution des programmes à l'aide d'un système de transitions (Landin 1966).

$$\texttt{fact 3} \to 3 * (\texttt{fact 2}) \to 3 * 2 * (\texttt{fact 1}) \to 3 * 2 * 1 * (\texttt{fact 0}) \to 6$$

La **sémantique dénotationnelle** interprète les programmes comme des fonctions qui calculent et qui peuvent avoir un effet sur un état mémoire. (Strachey et Scott 1969).

$$[\![\texttt{fact}]\!] : \begin{cases} \mathbb{N} & \to & \mathbb{N} \\ n & \mapsto & n! \end{cases}$$

La sémantique dénotationnelle donne des informations sur la relation entrées/sorties d'un programme, mais peut aussi donner des informations, par exemple comme le temps de exécution d'un programme,...

```
(function prose -> "Par ma foi ! il y a plus de 42 ans
                    que je dis de la" ^ prose ^ "sans que
                    j'en susse rien") "denotational semantics"
```

« Le but de la sémantique dénotationnelle est de donner une signification mathématique aux langages de programmation, afin d'obtenir

- des définitions rigoureuses indépendantes des détails d'implémentation,
- des objets mathématiques permettant de prouver les propriétés des programmes. »
- La sémantique dénotationnelle est un guide menant à l'introduction de nouvelles constructions en logique et dans les langages de programmation.

(Roberto M. Amadio and Pierre-Louis Curien. 1998. Domains and Lambda-Calculi. Cambridge University Press, New York, NY, USA.)

La correspondance de **Curry-Howard** entre *programmes* et *preuves*

λ-**calcul**	**Logique Intuitionniste**
Programme : Type	Preuve : Formule
$\Gamma \vdash M : A$	$\dfrac{\overset{\pi}{}}{\Gamma \vdash A}$

La correspondance de **Curry-Howard** entre *programmes* et *preuves*

λ-**calcul**	**Logique Intuitionniste**
Programme : Type	Preuve : Formule
$\Gamma \vdash M : A$	$\dfrac{\overset{\pi}{}}{\Gamma \vdash A}$

La correspondance de **Lambek** entre l'égalité β des *programmes* et l'égalité des *morphismes* dans les **Catégories Cartesiennes Closes**.

En sémantique catégorique,

- un type A est interprété par un objet $[\![A]\!]$
- un programme $x_1 : A_1, \ldots, x_n : A_n \vdash M : A$ est interprété par un morphisme $[\![M]\!] : [\![A_1]\!] \times \cdots \times [\![A_n]\!] \to [\![A]\!]$

Au produit cartésien est adjoint un objet des morphismes qui permet, entre autres, d'interpréter le type $A \Rightarrow B$

PCF, un langage de programmation fonctionnel typé

(D. Scott : A Type-Theoretical Alternative to ISWIM, CUCH, OWHY. Theor. Comput. Sci., 1993)

(G. Plotkin : LCF Considered as a Programming Language. Theor. Comput. Sci.,1977)

Syntaxe

$$M, N, P := \underbrace{x \mid \lambda x\, M \mid (M)\, N}_{\lambda\text{-calcul}} \mid \underbrace{0 \mid \mathtt{succ}\, M}_{\text{Arithmétique}} \mid \underbrace{\mathtt{if}\, M = 0\, \mathtt{then}\, N\, \mathtt{else}\, P}_{\text{Conditionnelle}} \mid \underbrace{\mathtt{fix}\, M}_{\text{Récursion}}$$

Sémantique opérationnelle en appel par nom (quelques règles)

$$\frac{}{(\lambda x\, M)N \to M\,[N/x]} \qquad \frac{}{\mathtt{fix}\ M \to (M)(\mathtt{fix} M)} \qquad \frac{M \to M'}{(M)N \to (M')N}$$

Sémantique dénotationnelle : Catégories Cartésiennes Closes enrichies

- `int` et l'arithmétique qui lui est associé
- `fix` et la partialité qu'il induit

Invariance de la sémantique : Si $M \to M'$, alors $[\![M]\!] = [\![M']\!]$.
Lemme d'adéquation : Pour tout n, si $[\![M]\!] = n$ alors $M \to^* n$.

Pleine Adequation

Équivalence observationnelle : Pour tout contexte à trou C dans PCF et pour tout n, $C[M] \to^* n \Leftrightarrow C[N] \to^* n$

Théorème de pleine adéquation : (Full Abstraction)

$$[\![M]\!] = [\![N]\!] \text{ si et seulement si } \forall C,\ C[M] \to^* 0 \Leftrightarrow C[N] \to^* 0$$

- Domaines de Scott et fonctions continues

  (D. Scott : Domains for Denotational Semantics. ICALP 1982)
- dI-domaines et fonctions stables

  (G. Berry, Stable Models of Typed lambda-Calculi. ICALP 1978)
- Le modèle des algorithme séquentiels

  (G. Berry, P.-L. Curien : Sequential Algorithms on Concrete Data Structures. Theor. Comput. Sci.,1982)
- Sémantique des jeux

  (S. Abramsky, R. Jagadeesan, P. Malacaria : Full Abstraction for PCF. Inf. Comput., 2000)

  (J. Hyland, L. Ong : On Full Abstraction for PCF. Inf. Comput.,2000)

Programmation fonctionnelle probabiliste

Exemples de programmes

On ajoute à PCF deux primitives :

- `coin` correspond au lancé d'une pièce équilibrée : $\texttt{coin} \xrightarrow{1/2} 0$, $\texttt{coin} \xrightarrow{1/2} 1$
- `sample` simule une v.a. U avec loi de proba uniforme sur $[0, 1]$:

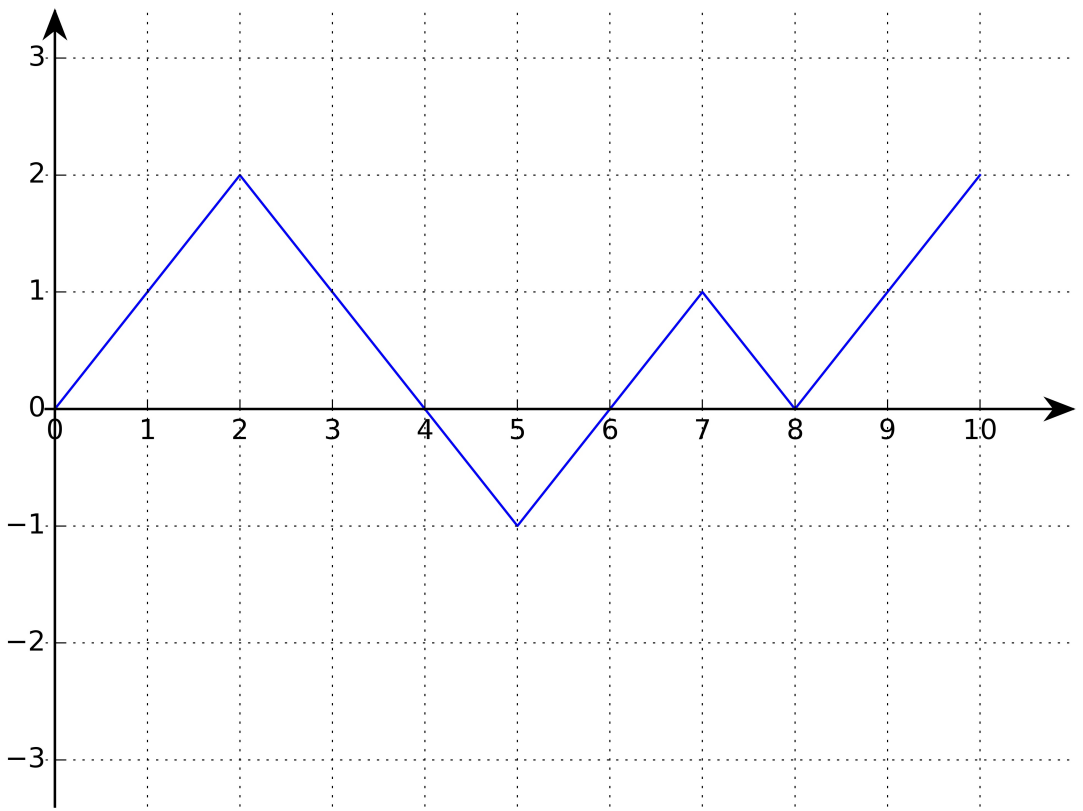
$$\mathbb{P}(U \leq x) = \int_{-\infty}^{x} \mathbb{1}_{[0,1]}(t)dt$$

Quelles distributions probabilistes peut-on simuler ?

1. Une marche aléatoire sur $\mathbb{N}$,
2. Une v.a. discrète de loi uniforme sur $\{0, \ldots, m\}$,
3. Une v.a. continue de densité exponentielle,
4. Un algorithme probabiliste qui cherche les zéros d'une fonction plus efficacement qu'un algorithme déterministe.

Exemples : Une marche aléatoire sur $\mathbb{N}$

```
let rec randomWalk n = if n = 0 then 0
        else if coin = 0 then randomWalk (n-1) + 1
                         else randomWalk (n-1) - 1
```



Exemples : une v.a. discrète de loi uniforme sur $\{0, \ldots, m\}$

Simuler une v.a. **uniforme sur** $\{0, \ldots, 2^n - 1\}$ à partir de `coin`

En utilisant n v.a. i.i.d. $X_i =$ `coin`, on construit une variable aléatoire $\sum_{i=0}^{n-1} X_i 2^i$ dont la loi est uniforme sur $\{0, \ldots, 2^n - 1\}$.

```
let rec unif2 n =
        if n = 0 then coin
        else  unif2 (n-1) + coin * 2**(n-1)
```

Simuler une v.a. **uniforme sur** $\{0, \ldots, m\}$ par méthode de rejet

```
let rec unif m =
        let x = unif2 (ceil (log2 (m+1))) in
                if x <= m then x else unif m
```

Exemples : une v.a. continue de densité exponentielle

`sample` simule une v.a. U avec loi uniforme sur $[0,1]$, on l'utilise pour programmer une v.a. X qui suit une loi exponentielle.

Méthode de la transformée inverse (Inverse transform sampling) :
On cherche T croissant tel que $T(U) = X$.

$$\begin{aligned}
\mathbb{P}(X \leq x) &= \int_0^x e^{-t} dt = 1 - e^{-x} \\
&= \mathbb{P}(T(U) \leq x) \\
&= \mathbb{P}(U \leq T^{-1}(x)) = \int_{-\infty}^{T^{-1}(x)} \mathbb{1}_{[0,1]}(u) du = T^{-1}(x)
\end{aligned}$$

On calcule alors : $T(u) = -log(1-u)$

Implémentation

```
let exp = let u = sample in -log(1-u)
```

Exemples : un algorithme probabiliste plus efficace

Méthode de rejet : On cherche les zéros d'une fonction `f:int->int` compris entre 0 et n.

```
let rec lasVegas =
        let k = (Random.int n) in
            if (f k)=0 then k else lasVegas
```

Proposition : S'il existe k tel que $f(k) = 0$, alors `lasVegas` termine presque sûrement sur un zéro.

En effet, si f admet z zéros, alors :

- `lasVegas` termine en 1 étape avec probabilité $\frac{z}{n}$,
- `lasVegas` termine en p étapes avec probabilité $(1 - \frac{z}{n})^{p-1}\frac{z}{n}$,
- `lasVegas` termine en un nombre quelconque d'étapes avec probabilité

$$\mathbf{Prob}^{\infty}(\texttt{lasVegas}, \texttt{Success}) = \sum_{p}(1 - \tfrac{z}{n})^{p-1}\tfrac{z}{n} = 1$$

PCF Probabiliste

Syntaxe

$$M, N, P \quad := \quad \underbrace{x \mid \lambda x\, M \mid (M)\, N}_{\lambda\text{-calcul}} \mid \underbrace{0 \mid \mathtt{succ}\, M}_{\text{Arithmétique}} \mid \underbrace{\mathtt{if}\, M = 0\, \mathtt{then}\, N\, \mathtt{else}\, P}_{\text{Conditionnelle}}$$
$$\mid \underbrace{\mathtt{fix}\, M}_{\text{Récursion}} \mid \underbrace{\mathtt{coin}}_{\text{Probabilités}}$$

Sémantique opérationnelle en appel par nom : $\dfrac{}{(\lambda x\, M)N \xrightarrow{1} M\,[N/x]}$

$$\frac{}{\mathtt{coin} \xrightarrow{1/2} 0} \qquad \frac{}{\mathtt{coin} \xrightarrow{1/2} 1} \qquad \frac{M \xrightarrow{p} M'}{(M)N \xrightarrow{p} (M')N}$$

Matrice de transition :

$$\mathbf{Prob}(M, M') = \begin{cases} p & \text{si } M \xrightarrow{p} M' \\ 1 & \text{si } M \text{ normal et } M = M' \\ 0 & \text{sinon.} \end{cases}$$

Sémantique dénotationnelle dans le style monadique

(C. Jones, G. Plotkin : A Probabilistic Powerdomain of Evaluations. LICS 1989)

On peut donner une sémantique dénotationnelle à l'effet probabiliste grâce à une monade (cours du 12 décembre 2018) :

$$\mathcal{V}(A) = \left\{ (x_a) \text{ t.q. } \sum_a x_a = 1 \right\} \qquad \text{(Distributions de probabilité)}$$

Un programme probabiliste `M: int -> int` sera dénoté par une fonction

$$[\![\mathtt{M}]\!] : \mathbb{N}_\perp \to \mathcal{V}(\mathbb{N}_\perp)$$

Si l'entrée est probabiliste, on utilise le relèvement de la monade, qui correspond à un opérateur de mise en mémoire :

$$[\![\texttt{let n=x in M}]\!] : \quad (x_n) \in \mathcal{V}(\mathbb{N}_\perp) \mapsto \left(\sum_n [\![M]\!]_{n,q} x_n \right)_q \in \mathcal{V}(\mathbb{N}_\perp)$$

Cette sémantique suit une stratégie d'évaluation en appel par valeur.

Sémantique dénotationnelle dans le style direct

(V. Danos, T. Ehrhard : Probabilistic coherence spaces as a model of higher-order probabilistic computation. Inf. Comput., 2011)

Définis par J.-Y. Girard, les espaces cohérents probabilistes forment un modèle des langages fonctionnels probabilistes.

La sémantique d'un type A est donnée par : $[\![A]\!] = (|A|, \mathrm{P}(A))$ où

- $|A|$ un ensemble de valeurs élémentaires,
- $\mathrm{P}(A)$ un ensemble de suites indexées par les valeurs élémentaires.

Par exemple, $|\mathtt{int}| = \mathbb{N}$ et $\mathrm{P}(\mathtt{int}) = \{(x_n) \text{ t.q. } \sum x_n \leq 1\}$

Un programme probabiliste `M: int -> int` est interprété par une **série** $[\![\mathtt{M}]\!] : \mathrm{P}(\mathtt{int}) \to \mathrm{P}(\mathtt{int})$:

$$[\![\mathtt{M}]\!] : \quad (x_n) \mapsto \left(\sum_{[n_1,\ldots,n_k] \in \mathcal{M}_{\mathrm{fin}}(\mathbb{N})} [\![M]\!]_{[n_1,\ldots,n_k],q} \prod_{i=1}^{k} x_{n_i} \right)_q$$

Cette sémantique suit une stratégie d'évaluation en appel par nom.

Résultats

(T. Ehrhard, M. Pagani, C. T. : Proba. coh. spaces are fully abstract for proba. PCF. POPL 2014)

Théorème d'invariance de la sémantique :

$$\text{Si } \mathcal{P} \vdash M : \sigma, \text{ alors } [\![M]\!] = \sum_{\mathcal{P} \vdash M' : \sigma} \mathbf{Prob}(M, M')[\![M']\!]$$

Lemme d'adéquation : Si $\vdash M : \texttt{int}$, alors $[\![M]\!]_n = \mathbf{Prob}^{\infty}(M, n)$

Théorème de pleine adéquation pour **Pcoh** et pPCF :

$[\![M]\!] = [\![N]\!]$ si et seulement si $\forall C,\ \mathbf{Prob}(C[M], 0) = \mathbf{Prob}(C[N], 0)$

Résultats

(T. Ehrhard, M. Pagani, C. T. : Proba. coh. spaces are fully abstract for proba. PCF. POPL 2014)

Théorème d'invariance de la sémantique :

$$\text{Si } \mathcal{P} \vdash M : \sigma, \text{ alors } [\![M]\!] = \sum_{\mathcal{P} \vdash M' : \sigma} \mathbf{Prob}(M, M')[\![M']\!]$$

Lemme d'adéquation : Si $\vdash M : \texttt{int}$, alors $[\![M]\!]_n = \mathbf{Prob}^{\infty}(M, n)$

Théorème de pleine adéquation pour **Pcoh** et pPCF :

$$[\![M]\!] = [\![N]\!] \text{ si et seulement si } \forall C,\ \mathbf{Prob}(C[M], 0) = \mathbf{Prob}(C[N], 0)$$

Si $\exists a\, [\![M]\!]_a \neq [\![M']\!]_a$, alors on construit $\texttt{test}_\texttt{a}$ qui va les distinguer.

- $\texttt{test}_\texttt{a}$ dépend de paramètres à trouver tels que $\texttt{test}_\texttt{a}$ dans pPCF
- $[\![(\texttt{test}_\texttt{a})N]\!]$ est une série formelle avec un nombre fini de paramètres
- L'un des monômes de cette série formelle est proportionnel à $[\![N]\!]_a$
- 2 séries qui diffèrent par un monôme, diffèrent au voisinage de 0
- On en déduit des valeurs des paramètres tels que $\texttt{test}_\texttt{a}$ dans pPCF
- $[\![(\texttt{test}_\texttt{a})M]\!] \neq [\![(\texttt{test}_\texttt{a})M']\!]$, on conclut par adéquation

Un opérateur de mise en mémoire est bien utile :

Programme qui cherche un zéro de `f:int->int` compris entre `0` et `n` :

```
let rec lasVegas = let k = (Random.int n) in
                   if (f k)=0 then k else lasVegas
```

La programmation probabiliste en appel par nom n'est pas pratique :

```
lasVegas = fix (λlV .
            (λk . if (f k)=0 then k else lV) (unif n))
```

Opérateur de mise en mémoire

(J.-L. Krivine : Opérateur de mise en mémoire, Arch Math Logic 30(4), 1990.)

Sémantique dénotationnelle de l'opérateur de mise en mémoire :

$$[\![\texttt{let x = M in N}]\!] = \left(\sum_{k \in \mathbb{N}} [\![N]\!]_{\underbrace{[k,\ldots,k]}_{n},a} [\![M]\!]_k \right)_a$$

En termes de ressources, une valeur de retour de `M` est calculée, puis effacée ou copiée à chaque fois que `N` en a besoin.

En termes sémantiques,

$$[\![(\lambda x.N)M]\!] = \left(\sum_{[k_1,\ldots,k_n] \in \mathcal{M}_{\text{fin}}(\mathbb{N})} [\![N]\!]_{[k_1,\ldots,k_n],a} \prod_{i=1}^{n} [\![M]\!]_{k_i} \right)_a$$

L'opérateur de mémorisation agit par composition avec `copy` :

$$[\![\text{copy}]\!]_{k,[k,\cdots,k]} = 1 \qquad [\![\text{copy}]\!]_{k,[k_1,\cdots,k_n]} = 0, \text{ si } \exists i \; k_i \neq k$$

Elle existe dans les espaces cohérents probabilistes pour le type `int`.

PCF probabiliste avec opérateur de mise en mémoire

(T. Ehrhard, M. Pagani, C. T. : Full Abstraction for Probabilistic PCF, J. ACM, 2018.)

Syntaxe

$$M, N, P \quad := \quad \underbrace{x \mid \lambda x.M \mid (M)\, N}_{\lambda\text{-calcul}} \mid \underbrace{0 \mid \mathtt{succ}\, M}_{\text{Arithmétique}} \mid \underbrace{\mathtt{if}\, M\, \mathtt{then}\, N\, \mathtt{else}\, P}_{\text{Conditionnelle}}$$
$$\mid \underbrace{\mathtt{fix}\, M}_{\text{Récursion}} \mid \underbrace{\mathtt{coin}}_{\text{Probabilité}} \mid \underbrace{\mathtt{let\ k\ =\ M\ in\ N}}_{\text{Mise en mémoire}}$$

Sémantique opérationnelle

$$\frac{}{\mathtt{let\ k\ = n\ in\ N} \xrightarrow{1} \mathtt{N}[\mathtt{n}/\mathtt{k}]} \qquad \frac{\mathtt{M} \xrightarrow{p} \mathtt{M}'}{\mathtt{let\ k\ = M\ in\ N} \xrightarrow{p} \mathtt{let\ k\ = M'\ in\ N}}$$

Sémantique dénotationnelle les espaces cohérents probabilistes sont pleinement adéquats pour pour cette extension de PCF.

Quels types admettent un opérateur de mise en mémoire ?

On a besoin d'effacer, d'utiliser et de dupliquer des ressources.

(Peut-dupliquer un objet ? G. Munch-Maccagnoni le 19 décembre 2018)

A quoi sert la sémantique dénotationnelle ?

(S. Staton : Commutative Semantics for Probabilistic Programming, ESOP 2017)

Comment montrer l'équivalence observationnelle de :

```
let A = let x = M in          let B = let y = N in
        let y = N in                  let x = M in
        P                             P
```

Avec la sémantique opérationnelle, on peut avoir des difficultés.

Avec la sémantique dénotationnelle, on utilise le théorème de Fubini :

$$[\![A]\!] = \sum_{i=0}^{\infty} [\![M]\!]_i \sum_{j=0}^{\infty} [\![N]\!]_j \; [\![P]\!](i,j)$$

$$[\![B]\!] = \sum_{j=0}^{\infty} [\![N]\!]_j \sum_{i=0}^{\infty} [\![M]\!]_i \; [\![P]\!](i,j)$$

On conclut en appliquant le théorème d'adéquation :

$$\text{Si } [\![A]\!] = [\![B]\!], \text{ alors } \forall n, \forall C, \mathbf{Prob}\,(C[A], n) = \mathbf{Prob}\,(C[B], n)$$

La logique linéaire

Logique Linéaire, introduction

Une analyse sémantique : dans les espaces cohérents (J.-Y. Girard : The System F of Variable Types, Fifteen Years Later. Theor. Comput. Sci. 1986), l'interprétation du type $A \Rightarrow B$ admet une décomposition naturelle :

$$A \Rightarrow B = !A \multimap B$$

Une réconciliation logique : logique classique (Cours du 05 décembre 2018) et logique intuitionniste (Cours du 21 novembre 2018).

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \wedge B} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

Affaiblissement $\dfrac{\Gamma \vdash C}{\Gamma,\ A \vdash C}$ Contraction $\dfrac{\Gamma,\ A,\ A \vdash C}{\Gamma,\ A \vdash C}$

Logique Linéaire, introduction

Une analyse sémantique : dans les espaces cohérents (J.-Y. Girard : The System F of Variable Types, Fifteen Years Later. Theor. Comput. Sci. 1986), l'interprétation du type $A \Rightarrow B$ admet une décomposition naturelle :

$$A \Rightarrow B = !A \multimap B$$

Une réconciliation logique : logique classique (Cours du 05 décembre 2018) et logique intuitionniste (Cours du 21 novembre 2018).

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}$$

$$\text{Affaiblissement } \frac{\Gamma \vdash C}{\Gamma, !A \vdash C} \qquad \text{Contraction } \frac{\Gamma, !A, !A \vdash C}{\Gamma, !A \vdash C}$$

$$!A \otimes !B = !(A \,\&\, B) \qquad e^A \cdot e^B = e^{A+B}$$

Logique Linéaire, introduction

Une analyse sémantique : dans les espaces cohérents (J.-Y. Girard : The System F of Variable Types, Fifteen Years Later. Theor. Comput. Sci. 1986), l'interprétation du type $A \Rightarrow B$ admet une décomposition naturelle :

$$A \Rightarrow B = !A \multimap B$$

Une réconciliation logique : logique classique (Cours du 05 décembre 2018) et logique intuitionniste (Cours du 21 novembre 2018).

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}$$

$$\text{Affaiblissement } \frac{\Gamma \vdash C}{\Gamma, !A \vdash C} \qquad \text{Contraction } \frac{\Gamma, !A, !A \vdash C}{\Gamma, !A \vdash C}$$

$$!A \otimes !B = !(A \,\&\, B) \qquad e^A \cdot e^B = e^{A+B}$$

Un contenu calculatoire : « donne moi autant de copies de l'entrée de type A que nécessaire et je te donnerai une sortie de type B. »

La logique linéaire

(J.-Y. Girard. Linear Logic. Theoretical Computer Science, 50, 1987.)

Fragment multiplicatif

$$\frac{}{A \vdash A} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \qquad \frac{\Gamma \vdash A \multimap B \qquad \Delta \vdash A}{\Gamma, \Delta \vdash B}$$

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}$$

Fragment exponentiel : les règles structurelles

Affaiblissement $\dfrac{\Gamma \vdash C}{\Gamma, !A \vdash C}$ Contraction $\dfrac{\Gamma, !A, !A \vdash C}{\Gamma, !A \vdash C}$

Relèvement $\dfrac{!\Gamma \vdash A}{!\Gamma \vdash !A}$ Co-unité $\dfrac{\Gamma, A \vdash C}{\Gamma, !A \vdash C}$

Sémantique catégorique de l'Exponentielle

(P.A. Mellies : Categorical semantics of linear logic. Panoramas et syntheses, 2009.)

Comonade

Co-unité : $!A \multimap A$ Comultiplication : $!A \multimap !!A$

Comonoïde

Effacer : $!A \multimap 1$ Dupliquer : $!A \multimap !A \otimes !A$

Encodage de l'appel par nom

$$A \Rightarrow B = !A \multimap B$$

Exemple : espaces cohérents probabilistes

$|[\![!\mathtt{int}]\!]| = \mathcal{M}_{\mathrm{fin}}(\mathbb{N})$

Si $x \in \mathrm{P}([\![\mathtt{int}]\!])$, alors $x^! \in \mathrm{P}([\![!\mathtt{int}]\!])$, où $x^!_{[k_1,\ldots,k_n]} = \prod_i x_{k_i}$

Les coalgèbres pour mettre en mémoire

(T. Ehrhard : Call-By-Push-Value from a Linear Logic Point of View, ESOP 2016.)

Une **coalgèbre** est un objet P et un morphisme $h : P \multimap !P$ compatible avec la counité et la comultiplication de la comonade.

Toute coalgèbre a une structure de **comonoïde**

$$\text{Effacer : } w^P : P \multimap \mathbf{1} \qquad \text{Dupliquer : } c^P : P \multimap P \otimes P$$

L'exponentielle est une coalgèbre.

Quelles constructions préservent les coalgèbres ?

$$\phi, \psi := \mathbf{1} \mid !\sigma \mid \phi \otimes \psi \mid \phi \oplus \psi \mid \zeta \mid \mathsf{Rec}\,\zeta \cdot \phi$$

Ce sont les types positifs de la logique linéaire !

Exemples : $\texttt{int} = \mathsf{Rec}\,\zeta \cdot \mathbf{1} \oplus \zeta$

Évaluation à la demande des sous-structures (Cours du 16 janvier 2019)

$$\texttt{lazy stream} = \mathsf{Rec}\,\zeta \cdot \texttt{int} \otimes\ !\zeta$$

Logique linéaire semi-polarisée

(J.-Y. Girard : A new constructive logic : classical logic, Math. Struct. in Comput. Science, 1991 ;

O. Laurent, L. Regnier : About translations of classical logic into polarized linear logic, LICS'03)

$$\frac{}{\mathcal{P}, \phi \vdash \phi} \qquad \frac{\mathcal{P}, \phi \vdash \sigma}{\mathcal{P} \vdash \phi \multimap \sigma} \qquad \frac{\mathcal{P} \vdash \phi \multimap \sigma \qquad \mathcal{P} \vdash \phi}{\mathcal{P} \vdash \sigma}$$

$$\frac{}{\mathcal{P} \vdash \mathbf{1}} \qquad \frac{\mathcal{P} \vdash \phi_1 \qquad \mathcal{P} \vdash \phi_2}{\mathcal{P} \vdash \phi_1 \otimes \phi_2} \qquad \frac{\mathcal{P} \vdash \phi_1 \otimes \phi_2}{\mathcal{P} \vdash \phi_i}$$

$$\frac{\mathcal{P} \vdash \sigma}{\mathcal{P} \vdash !\sigma} \qquad \frac{\mathcal{P} \vdash !\sigma}{\mathcal{P} \vdash \sigma} \qquad \frac{\mathcal{P}, !\sigma \vdash \sigma}{\mathcal{P} \vdash \sigma}$$

$$\frac{\mathcal{P} \vdash \phi_1 \oplus \phi_2 \qquad \mathcal{P}, \phi_1 \vdash \sigma \qquad \mathcal{P}, \phi_2 \vdash \sigma}{\mathcal{P} \vdash \sigma} \qquad \frac{\mathcal{P} \vdash \phi_i}{\mathcal{P} \vdash \phi_1 \oplus \phi_2}$$

$$\frac{\mathcal{P} \vdash \psi\,[\mathrm{Rec}\,\zeta \cdot \psi/\zeta]}{\mathcal{P} \vdash \mathrm{Rec}\,\zeta \cdot \psi} \qquad \frac{\mathcal{P} \vdash \mathrm{Rec}\,\zeta \cdot \psi}{\mathcal{P} \vdash \psi\,[\mathrm{Rec}\,\zeta \cdot \psi/\zeta]}$$

Logique linéaire semi-polarisée

(J.-Y. Girard : A new constructive logic : classical logic, Math. Struct. in Comput. Science, 1991 ;

O. Laurent, L. Regnier : About translations of classical logic into polarized linear logic, LICS'03)

$$\frac{}{\mathcal{P}, \phi \vdash \phi} \qquad \frac{\mathcal{P}, \phi \vdash \sigma}{\mathcal{P} \vdash \phi \multimap \sigma} \qquad \frac{\mathcal{P} \vdash \phi \multimap \sigma \qquad \mathcal{P} \vdash \phi}{\mathcal{P} \vdash \sigma}$$

$$\frac{}{\mathcal{P} \vdash \mathbf{1}} \qquad \frac{\mathcal{P} \vdash \phi_1 \qquad \mathcal{P} \vdash \phi_2}{\mathcal{P} \vdash \phi_1 \otimes \phi_2} \qquad \frac{\mathcal{P} \vdash \phi_1 \otimes \phi_2}{\mathcal{P} \vdash \phi_i}$$

$$\frac{\mathcal{P} \vdash \sigma}{\mathcal{P} \vdash !\sigma} \qquad \frac{\mathcal{P} \vdash !\sigma}{\mathcal{P} \vdash \sigma} \qquad \frac{\mathcal{P}, !\sigma \vdash \sigma}{\mathcal{P} \vdash \sigma}$$

$$\frac{\mathcal{P} \vdash \phi_1 \oplus \phi_2 \qquad \mathcal{P}, \phi_1 \vdash \sigma \qquad \mathcal{P}, \phi_2 \vdash \sigma}{\mathcal{P} \vdash \sigma} \qquad \frac{\mathcal{P} \vdash \phi_i}{\mathcal{P} \vdash \phi_1 \oplus \phi_2}$$

$$\frac{\mathcal{P} \vdash \psi\,[\mathrm{Rec}\,\zeta \cdot \psi/\zeta]}{\mathcal{P} \vdash \mathrm{Rec}\,\zeta \cdot \psi} \qquad \frac{\mathcal{P} \vdash \mathrm{Rec}\,\zeta \cdot \psi}{\mathcal{P} \vdash \psi\,[\mathrm{Rec}\,\zeta \cdot \psi/\zeta]}$$

CURRY-HOWARD

Un nouveau langage :

$$\frac{}{\mathcal{P}, x : \phi \vdash x : \phi} \qquad \frac{\mathcal{P}, x : \phi \vdash M : \sigma}{\mathcal{P} \vdash \lambda x^{\phi}\, M : \phi \multimap \sigma} \qquad \frac{\mathcal{P} \vdash M : \phi \multimap \sigma \qquad \mathcal{P} \vdash N : \phi}{\mathcal{P} \vdash \langle M \rangle N : \sigma}$$

$$\frac{}{\mathcal{P} \vdash () : \mathbf{1}} \qquad \frac{\mathcal{P} \vdash M_1 : \phi_1 \qquad \mathcal{P} \vdash M_2 : \phi_2}{\mathcal{P} \vdash (M_1, M_2) : \phi_1 \otimes \phi_2} \qquad \frac{\mathcal{P} \vdash M : \phi_1 \otimes \phi_2}{\mathcal{P} \vdash \pi_i M : \phi_i}$$

$$\frac{\mathcal{P} \vdash M : \sigma}{\mathcal{P} \vdash \mathsf{thunk}(M) : !\sigma} \qquad \frac{\mathcal{P} \vdash M : !\sigma}{\mathcal{P} \vdash \mathsf{force}(M) : \sigma} \qquad \frac{\mathcal{P}, x : !\sigma \vdash M : \sigma}{\mathcal{P} \vdash \mathsf{fix}\, x^{!\sigma}\, M : \sigma}$$

$$\frac{\mathcal{P} \vdash M : \phi_1 \oplus \phi_2 \qquad \mathcal{P}, x_1 : \phi_1 \vdash M_1 : \sigma \qquad \mathcal{P}, x_2 : \phi_2 \vdash M_2 : \sigma}{\mathcal{P} \vdash \mathsf{case}(M, x_1 \cdot M_1, x_2 \cdot M_2) : \sigma}$$

$$\frac{\mathcal{P} \vdash M : \phi_i}{\mathcal{P} \vdash \mathsf{in}_i M : \phi_1 \oplus \phi_2}$$

$$\frac{\mathcal{P} \vdash M : \psi\,[\mathsf{Rec}\,\zeta \cdot \psi/\zeta]}{\mathcal{P} \vdash \mathsf{fold}(M) : \mathsf{Rec}\,\zeta \cdot \psi} \qquad \frac{\mathcal{P} \vdash M : \mathsf{Rec}\,\zeta \cdot \psi}{\mathcal{P} \vdash \mathsf{unfold}(M) : \psi\,[\mathsf{Rec}\,\zeta \cdot \psi/\zeta]}$$

CURRY-HOWARD

Un ~~nouveau~~ langage : Call By Push Value

(P. Levy : Call by push value : a functional/imperative synthesis, Sem. Struct. in Comput., 2004.)

$$\frac{}{\mathcal{P}, x : \phi \vdash x : \phi} \qquad \frac{\mathcal{P}, x : \phi \vdash M : \sigma}{\mathcal{P} \vdash \lambda x^{\phi}\, M : \phi \multimap \sigma} \qquad \frac{\mathcal{P} \vdash M : \phi \multimap \sigma \qquad \mathcal{P} \vdash N : \phi}{\mathcal{P} \vdash \langle M \rangle N : \sigma}$$

$$\frac{}{\mathcal{P} \vdash () : \mathbf{1}} \qquad \frac{\mathcal{P} \vdash M_1 : \phi_1 \qquad \mathcal{P} \vdash M_2 : \phi_2}{\mathcal{P} \vdash (M_1, M_2) : \phi_1 \otimes \phi_2} \qquad \frac{\mathcal{P} \vdash M : \phi_1 \otimes \phi_2}{\mathcal{P} \vdash \pi_i M : \phi_i}$$

$$\frac{\mathcal{P} \vdash M : \sigma}{\mathcal{P} \vdash \mathsf{thunk}(M) : !\sigma} \qquad \frac{\mathcal{P} \vdash M : !\sigma}{\mathcal{P} \vdash \mathsf{force}(M) : \sigma} \qquad \frac{\mathcal{P}, x : !\sigma \vdash M : \sigma}{\mathcal{P} \vdash \mathsf{fix}\, x^{!\sigma}\, M : \sigma}$$

$$\frac{\mathcal{P} \vdash M : \phi_1 \oplus \phi_2 \qquad \mathcal{P}, x_1 : \phi_1 \vdash M_1 : \sigma \qquad \mathcal{P}, x_2 : \phi_2 \vdash M_2 : \sigma}{\mathcal{P} \vdash \mathsf{case}(M, x_1 \cdot M_1, x_2 \cdot M_2) : \sigma}$$

$$\frac{\mathcal{P} \vdash M : \phi_i}{\mathcal{P} \vdash \mathsf{in}_i M : \phi_1 \oplus \phi_2}$$

$$\frac{\mathcal{P} \vdash M : \psi\,[\mathsf{Rec}\,\zeta \cdot \psi/\zeta]}{\mathcal{P} \vdash \mathsf{fold}(M) : \mathsf{Rec}\,\zeta \cdot \psi} \qquad \frac{\mathcal{P} \vdash M : \mathsf{Rec}\,\zeta \cdot \psi}{\mathcal{P} \vdash \mathsf{unfold}(M) : \psi\,[\mathsf{Rec}\,\zeta \cdot \psi/\zeta]}$$

Un ~~nouveau~~ langage : Call By Push Value

(P. Levy : Call by push value : a functional/imperative synthesis, Sem. Struct. in Comput., 2004.)

$$\frac{}{\mathcal{P}, x : \phi \vdash x : \phi} \qquad \frac{\mathcal{P}, x : \phi \vdash M : \sigma}{\mathcal{P} \vdash \lambda x^{\phi}\, M : \phi \multimap \sigma} \qquad \frac{\mathcal{P} \vdash M : \phi \multimap \sigma \qquad \mathcal{P} \vdash N : \phi}{\mathcal{P} \vdash \langle M \rangle N : \sigma}$$

$$\frac{}{\mathcal{P} \vdash () : \mathbf{1}} \qquad \frac{\mathcal{P} \vdash M_1 : \phi_1 \qquad \mathcal{P} \vdash M_2 : \phi_2}{\mathcal{P} \vdash (M_1, M_2) : \phi_1 \otimes \phi_2} \qquad \frac{\mathcal{P} \vdash M : \phi_1 \otimes \phi_2}{\mathcal{P} \vdash \pi_i M : \phi_i}$$

$$\frac{\mathcal{P} \vdash M : \sigma}{\mathcal{P} \vdash \mathsf{thunk}(M) : !\sigma} \qquad \frac{\mathcal{P} \vdash M : !\sigma}{\mathcal{P} \vdash \mathsf{force}(M) : \sigma} \qquad \frac{\mathcal{P}, x : !\sigma \vdash M : \sigma}{\mathcal{P} \vdash \mathsf{fix}\, x^{!\sigma}\, M : \sigma}$$

$$\frac{\mathcal{P} \vdash M : \phi_1 \oplus \phi_2 \qquad \mathcal{P}, x_1 : \phi_1 \vdash M_1 : \sigma \qquad \mathcal{P}, x_2 : \phi_2 \vdash M_2 : \sigma}{\mathcal{P} \vdash \mathsf{case}(M, x_1 \cdot M_1, x_2 \cdot M_2) : \sigma}$$

$$\frac{\mathcal{P} \vdash M : \phi_i}{\mathcal{P} \vdash \mathsf{in}_i M : \phi_1 \oplus \phi_2}$$

$$\frac{\mathcal{P} \vdash M : \psi\,[\mathsf{Rec}\,\zeta \cdot \psi/\zeta]}{\mathcal{P} \vdash \mathsf{fold}(M) : \mathsf{Rec}\,\zeta \cdot \psi} \qquad \frac{\mathcal{P} \vdash M : \mathsf{Rec}\,\zeta \cdot \psi}{\mathcal{P} \vdash \mathsf{unfold}(M) : \psi\,[\mathsf{Rec}\,\zeta \cdot \psi/\zeta]}$$

Call-By-Push-Value Probabiliste

Call by push value probabiliste

Types :

$$\begin{aligned} \textit{positifs :} \quad & \phi, \psi := \mathbf{1} \mid !\sigma \mid \phi \otimes \psi \mid \phi \oplus \psi \mid \zeta \mid \mathsf{Rec}\,\zeta \cdot \phi \\ \textit{généraux :} \quad & \sigma, \tau := \phi \mid \phi \multimap \sigma \end{aligned}$$

Programmes :

$$\begin{aligned} \textit{valeurs :} \quad & V, W := x \mid () \mid \mathsf{thunk}(M) \mid (V, W) \mid \mathsf{in}_1 V \mid \mathsf{in}_2 V \mid \mathsf{fold}(V) \\ \textit{programmes :} \quad & M, N := x \mid () \mid \lambda x^{\phi}\, M \mid \langle M \rangle N \\ & \quad \mid \mathsf{thunk}(M) \mid \mathsf{force}(M) \\ & \quad \mid (M, N) \mid \pi_1 M \mid \pi_2 M \\ & \quad \mid \mathsf{in}_1 M \mid \mathsf{in}_2 M \mid \mathsf{case}(M, x_1 \cdot N_1, x_2 \cdot N_2) \mid \mathsf{coin} \\ & \quad \mid \mathsf{fix}\, x^{!\sigma}\, M \mid \mathsf{fold}(M) \mid \mathsf{unfold}(M) \end{aligned}$$

Contextes de typage : $\mathcal{P} = (x_1 : \phi_1, \ldots, x_k : \phi_k)$

Call by push value probabiliste : Sémantique opérationnelle

Appel par valeur

$$\frac{}{\langle \lambda x^{\phi}\, M\rangle V \xrightarrow{1} M\,[V/x]} \qquad \frac{}{\mathsf{force}(\mathsf{thunk}(M)) \xrightarrow{1} M}$$

$$\frac{}{\mathsf{case}(\mathsf{in}_i\, V, x_1 \cdot M_1, x_2 \cdot M_2) \xrightarrow{\mathrm{w}} M_i\,[V/x_i]}$$

Probabilités

$$\frac{M \xrightarrow{p} M'}{\langle M\rangle V \xrightarrow{p} \langle M'\rangle V} \qquad \frac{N \xrightarrow{p} N'}{\langle M\rangle N \xrightarrow{p} \langle M\rangle N'} \qquad \frac{M \xrightarrow{p} M'}{\mathsf{force}(M) \xrightarrow{p} \mathsf{force}(M')}$$

Appel par nom

Les valeurs sont des arbres dont les feuilles sont () et $\mathsf{thunk}(M)$ et les noeuds (V, W) ou $\mathsf{in}_i\, V$ (pas des fonctions !)

$$(A \Rightarrow B)^* = !A \multimap B \qquad ((\lambda x^{\sigma}\, M)N)^* = \langle \lambda x^{!\sigma}\, M\rangle \mathsf{thunk}(N)$$

Retour vers PCF (1/2 :Typage)

Types de PCF :

$$A, B := \mathtt{int} \mid A \Rightarrow B$$

Types de pCBPV :

$$\textit{positifs :} \quad \phi, \psi := \mathbf{1} \mid !\sigma \mid \phi \otimes \psi \mid \phi \oplus \psi \mid \zeta \mid \mathsf{Rec}\, \zeta \cdot \phi$$

$$\textit{généraux :} \quad \sigma, \tau := \phi \mid \phi \multimap \sigma$$

Encodage :

$$\mathtt{int}^* = \mathsf{Rec}\, \zeta \cdot \mathbf{1} \oplus \zeta \qquad (A \Rightarrow B)^* = !(A^*) \multimap B^*$$

Retour vers PCF (2/2 :Programmes)

Syntaxe de PCF :

$$M, N, P \quad := \quad x \,|\, \lambda x.M \,|\, (M)\, N \,|\, 0 \,|\, \mathtt{succ}\, M \,|\, \mathtt{if}\, M = 0\, \mathtt{then}\, N\, \mathtt{else}\, P$$
$$|\, \mathtt{fix}\, M \;|\; \mathtt{coin} \;|\, \mathtt{let\ k\ =\ M\ in\ N}$$

Encodage :

$$(A \Rightarrow B)^* = !(A^*) \multimap B^*$$

$$(\lambda x\ M)^* = \lambda x^{!\sigma}\ M^* \qquad \text{et} \qquad ((M)N)^* = \langle M^* \rangle \mathsf{thunk}(N^*)$$

$$\mathtt{int}^* = \mathsf{Rec}\,\zeta \cdot \mathbf{1} \oplus \zeta$$

$$0^* = \mathsf{in}_1() \qquad \text{et} \qquad (n+1)^* = \mathsf{in}_2 n,$$
$$(\mathtt{if}\ \ M = 0\ \ \mathtt{then}\ \ N\ \ \mathtt{else}\ \ P)^* = \mathsf{case}(M^*, y \cdot N^*, z \cdot P^*)$$
$$(\mathbf{let}\ \ k = M\ \ \mathbf{in}\ \ N\ )^* = \mathsf{case}(M^*, y \cdot N^*[0/k], z \cdot N^*[\mathbf{succ}(z)/k])$$

$$\mathtt{lazy\ stream} = \mathsf{Rec}\,\zeta \cdot \mathtt{int} \ \otimes\ !\zeta$$

$$(\mathbf{let}\ (x, \_) = s\ \ \mathbf{in}\ \ M\ )^* = \pi_1(\mathsf{force}(M^*))$$

Call by push value probabiliste : Sémantique dénotationnelle

(T. Ehrhard, C. T. : Probabilistic call by push value, Log. Meth. Comput. Sci., 2019.)

Les espaces cohérents probabilistes sont un modèle de la logique linéaire.

On utilise les morphismes de coalgèbres pour interpréter les valeurs.

Invariance de la sémantique : $[\![M]\!] = \sum_{M'} \mathbf{Prob}(M, M')[\![M']\!]$

Lemme d'adéquation : Si $\vdash M : \mathbf{1}$ alors $\mathbf{Prob}^{\infty}(M, ()) = [\![M]\!]$

- Propriétés des coalgèbres
- Relation logique (Cours du 19 décembre 2018)
- Step-indexing (Cours du 09 janvier 2019)

Théorème de pleine adéquation pour **Pcoh** et pCBPV :

$[\![M]\!] = [\![N]\!]$ si et seulement si $\mathbf{Prob}(C[M], ()) = \mathbf{Prob}(C[N], ())$

Pour aller plus loin

Directions de recherche (1/2)

Langages fonctionnels pour les probabilités et les statistiques : pour modéliser les modèles Bayésiens utilisés en *computational statistics* et *machine learning* (Church, Anglican, Stan, Tabular, Venture, Hakaru, WebPPL) avec des types récursifs et d'ordre supérieur, des distributions de probabilités continues

Travaux en sémantique dénotationnelle : difficile de construire une catégorie cartésienne close (ordre supérieur) capturant la théorie de la mesure (probabilités continues)

(T. Ehrhard, M. Pagani, C. T. : Measurable cones and stable, measurable functions : a model for probabilistic higher-order programming, POPL2018.)

(M. Vákár, O. Kammar, S. Staton : A domain theory for statistical probabilistic programming, POPL2019)

Pleine adéquation ? Logique linéaire ?

Certifier des programmes probabilistes en utilisant la sémantique dénotationnelle et un assistant de preuve.

Directions de recherche (2/2)

De nouvelles primitives à comprendre : (`http://webppl.org/`)

En Inférence Bayésienne, on se restreint à des traces d'exécution dont on peut observer qu'elles satisfont une condition.

- `score` pénalise certaines exécutions d'un programme probabiliste en fonction des observations que l'on peut faire ;
- `infer` est utilisé pour renormaliser la loi de distribution d'un programme

Programmation réactive probabiliste :

Extension de `ReactiveML` et de `Zelus` avec `sample`, `score` et `infer` qui permettent de simuler des systèmes (exemple du cartpole) en adaptant au fur et à mesure les paramètres grâce à de l'apprentissage au cours de l'exécution du modèle.

(G. Baudart, L. Mandel, M. Pouzet, Reactive Probabilistic Programming)