Programmer = démontrer?
La correspondance de Curry-Howard aujourd'hui

Sixième cours

Des théorèmes gratuits: paramétricité et relations logiques

Xavier Leroy

Collège de France

2018-12-19

COLLÈGE
DE FRANCE
1530

Quiz

Dans le lambda-calcul polymorphe (système F), quels sont les termes qui ont les types suivants ?

$$\forall X.\ X$$

$$\forall X.\ X \to X$$

$$\forall X.\ X \to \texttt{int}$$

$$\forall X.\ X \to X \to X$$

Quiz

Dans le lambda-calcul polymorphe (système F), quels sont les termes qui ont les types suivants ?

$\forall X.\ X$ aucun (incohérence logique sinon !)

$\forall X.\ X \to X$

$\forall X.\ X \to \texttt{int}$

$\forall X.\ X \to X \to X$

Quiz

Dans le lambda-calcul polymorphe (système F), quels sont les termes qui ont les types suivants ?

$\forall X.\ X$	aucun (incohérence logique sinon !)
$\forall X.\ X \to X$	l'identité $\Lambda X.\lambda x.\ x$; d'autres ?
$\forall X.\ X \to \texttt{int}$	
$\forall X.\ X \to X \to X$	

Quiz

Dans le lambda-calcul polymorphe (système F), quels sont les termes qui ont les types suivants ?

$\forall X.\, X$	aucun (incohérence logique sinon !)
$\forall X.\, X \to X$	l'identité $\Lambda X.\lambda x.\, x$; d'autres ?
$\forall X.\, X \to \texttt{int}$	les fonctions constantes $\Lambda X.\lambda x.\, n$; d'autres ?
$\forall X.\, X \to X \to X$	

Quiz

Dans le lambda-calcul polymorphe (système F), quels sont les termes qui ont les types suivants ?

$\forall X.\, X$	aucun (incohérence logique sinon !)
$\forall X.\, X \to X$	l'identité $\Lambda X.\lambda x.\, x$; d'autres ?
$\forall X.\, X \to \texttt{int}$	les fonctions constantes $\Lambda X.\lambda x.\, n$; d'autres ?
$\forall X.\, X \to X \to X$	les booléens de Church, $\Lambda X.\lambda x.\lambda y.\, x$ et $\Lambda X.\lambda x.\lambda y.\, y$; d'autres ?

Quiz

Dans le lambda-calcul polymorphe (système F), quels sont les termes qui ont les types suivants ?

$\forall X.\, X$	aucun (incohérence logique sinon !)
$\forall X.\, X \to X$	l'identité $\Lambda X.\lambda x.\, x$; d'autres ?
$\forall X.\, X \to \texttt{int}$	les fonctions constantes $\Lambda X.\lambda x.\, n$; d'autres ?
$\forall X.\, X \to X \to X$	les booléens de Church, $\Lambda X.\lambda x.\lambda y.\, x$ et $\Lambda X.\lambda x.\lambda y.\, y$; d'autres ?

Comment montrer qu'il n'y a pas d'autres termes (clos et en forme normale) de ces types ?

Des théorèmes sur les listes

(Phil Wadler, *Theorems for free!*, 1991)

Dans système F étendu avec le type t^* des listes de t, les équations suivantes sont toujours vraies :

$$\mathtt{map}\, f\, (F\, x) = F\, (\mathtt{map}\, f\, x) \qquad \text{si } F : \forall X.\, X^* \to X^*$$

$$\mathtt{map}\, f\, (G\, x\, y) = G\, (\mathtt{map}\, f\, x)\, (\mathtt{map}\, f\, y) \qquad \text{si } G : \forall X.\, X^* \to X^* \to X^*$$

$$\mathtt{map}\, f\, (H\, x) = H\, (\mathtt{map}\, (\mathtt{map}\, f)\, x) \qquad \text{si } H : \forall X.\, X^{**} \to X^*$$

$$\mathtt{map}\, f\, (\Phi\, (p \circ f)\, x) = \Phi\, p\, (\mathtt{map}\, f\, x) \qquad \text{si } \Phi : \forall X.\, (X \to \mathtt{bool}) \to X^* \to X^*$$

avec $f : A \to B$, $p : B \to \mathtt{bool}$, et $\mathtt{map}\, f\, [x_1; \ldots; x_n] = [f\, x_1; \ldots; f\, x_n]$.

On peut montrer ces équations pour $F = \mathtt{rev}$ ou $G = \mathtt{append}$ ou $H = \mathtt{concat}$ ou $\Phi = \mathtt{filter}$; mais pourquoi sont-elles vraies pour toutes les fonctions ayant ces types polymorphes ?

Enseigner les nombres complexes

(Une fable due à John C. Reynolds)

Deux enseignants, deux groupes d'élèves. Au premier cours :

- Prof. Descartes : $\ll \mathbb{C} = \mathbb{R} \times \mathbb{R} \gg$. (représentation cartésienne)
  Définit l'injection $\mathbb{R} \to \mathbb{C}$, puis i, $+$, $\times$, $^{-1}$, $\bar{z}$, $|z|$.
- Prof. Bessel : $\ll \mathbb{C} = \mathbb{R}^+ \times \mathbb{R} \gg$. (représentation polaire)
  Définit l'injection $\mathbb{R} \to \mathbb{C}$, puis i, $+$, $\times$, $^{-1}$, $\bar{z}$, $|z|$.

Au cours suivant, les deux groupes d'élèves sont intervertis.

Ni Descartes ni Bessel ne commettent d'erreur mathématique, alors même qu'ils sont jugés (par les élèves) sur les définitions de l'autre.

Types abstraits

Ce qui empêche Descartes et Bessel de se tromper, c'est qu'à partir du deuxième cours, ils traitent les nombres complexes comme un type abstrait : un nom de type $\mathbb{C}$ et des opérations sur ce type.

L'un et l'autre s'interdisent d'aller voir la représentation concrète des nombres complexes, p.ex. de projeter leur première composante, ce qui mènerait à des contradictions :

$$\mathtt{proj}_1(i) = \mathtt{proj}_1((0,1)) = 0 \qquad \text{pour Descartes}$$
$$\mathtt{proj}_1(i) = \mathtt{proj}_1((1,\pi/2)) = 1 \qquad \text{pour Bessel}$$

Morale de la fable :

> *Type structure is a syntactic discipline for enforcing levels of abstraction.*
>
> *(John C. Reynolds, 1983)*

Abstraction de types et polymorphisme

Comme observé par Reynolds (1974), un moyen de rendre un type abstrait est de rendre ses utilisateurs polymorphes en le nom de ce type.

Dans l'exemple des nombres complexes, les étudiants sont des fonctions polymorphes en le type $\mathbb{C}$:

$$\begin{aligned}\text{étudiant} : \forall\mathbb{C}.\ \{\ &\texttt{inj} : \mathbb{R} \to \mathbb{C};\ \ \texttt{i} : \mathbb{C};\\ &\texttt{conj} : \mathbb{C} \to \mathbb{C}; \texttt{module} : \mathbb{C} \to \mathbb{R}^{+};\\ &\texttt{add} : \mathbb{C} \to \mathbb{C} \to \mathbb{C};\ \ \texttt{mul} : \mathbb{C} \to \mathbb{C} \to \mathbb{C}\ \} \to t\end{aligned}$$

Le type $\mathbb{C}$ étant une variable de type, le seul moyen de construire et d'utiliser des valeurs de type $\mathbb{C}$ est via les opérations fournies en argument.

La paramétricité

Une intuition : le polymorphisme est paramétrique, c.à.d. deux instantiations $f[A]$ et $f[B]$ d'une fonction polymorphe $f : \forall X \ldots$ sur deux types différents implémentent le même algorithme.

Une intuition duale : les types abstraits sont indépendants de leurs représentations, c.à.d. deux implémentations d'un même type abstrait ne peuvent être distingués que via les constantes et les opérations fournies sur ce type.

Une technique de raisonnement : les relations logiques
($\approx$ interpréter les types par des relations entre valeurs).

Diverses applications : ≪théorèmes gratuits≫ (vrais pour toutes les fonctions d'un certain type), résultats de non-habitation, isomorphismes entre codages fonctionnels et types algébriques, etc.

I

Les relations logiques

Définissabilité et relations logiques (Plotkin)

G. Plotkin, *λ-definability and Logical Relations*, 1973 ;
G. Plotkin, *λ-definability in the full type hierarchy*, 1980.

Plotkin cherche à caractériser les fonctions qui sont définissables dans le lambda-calcul pur (1973) ou simplement typé (1980) auquel on ajoute éventuellement des constantes.

Par exemple : dans le cas simplement typé, on suppose un type de base o qui est interprété par l'ensemble $O = \{V, F\}$.

$$[\![o]\!] = O$$
$$[\![t_1 \to t_2]\!] = [\![t_1]\!] \to [\![t_2]\!] \quad \text{(ensemble de fonctions)}$$

Quelles sont les fonctions ensemblistes $O \to O$ définissables par des lambda-termes ?
sans constantes : l'identité $V \mapsto V,\ F \mapsto F$
avec constantes $\mathtt{V}$ et $\mathtt{F}$: toutes sauf la négation $V \mapsto F,\ F \mapsto V$

Les relations logiques

Idée de Plotkin : les fonctions définissables satisfont toutes certaines relations, celles qui sont compatibles avec l'abstraction et l'application de fonctions. Plotkin les appelle relations logiques.

Définition : une relation logique à n arguments est une famille de relations $R^t \subseteq [\![t]\!] \times \cdots \times [\![t]\!]$ indexée par un type t, telles que

$$R^{t \to s}\ (f_1, \ldots, f_n) \iff \forall x_1, \ldots, x_n,\ R^t\ (x_1, \ldots, x_n) \Rightarrow R^s\ (f_1\ x_1, \ldots, f_n\ x_n)$$

Autrement dit : des fonctions sont reliées ssi elles envoient des éléments reliés sur des résultats reliés.

Remarque : la relation logique est entièrement déterminée par les relations R^o sur les types de base o.

Les relations logiques

Seuls les cas $n = 1$ et $n = 2$ sont utilisés en pratique :

Relation logique unaire : un prédicat sur O qui s'étend ≪héréditairement≫ aux fonctions

$$R^o(x) \qquad \text{au choix}$$

$$R^{t\to s}\,(f) \stackrel{def}{=} \forall x,\ R^t\,(x) \Rightarrow R^s\,(f\ x)$$

Relation logique binaire : (ou juste ≪relation logique≫)

$$R^o(x_1, x_2) \qquad \text{au choix}$$

$$R^{t\to s}\,(f_1, f_2) \stackrel{def}{=} \forall x_1, x_2,\ R^t\,(x_1, x_2) \Rightarrow R^s\,(f_1\ x_1, f_2\ x_2)$$

Le théorème fondamental des relations logiques

Interprétation des termes :

$$\begin{aligned} [\![x]\!]\ \rho &= \rho(x) \\ [\![\lambda x.\, M]\!]\ \rho &= v \mapsto [\![M]\!]\ (\rho + (x \mapsto v)) \\ [\![M\ N]\!]\ \rho &= [\![M]\!]\ \rho\ ([\![N]\!]\ \rho) \end{aligned}$$

Si un terme est bien typé, ses interprétations dans des environnements reliés sont reliées.

Théorème

Si $\Gamma \vdash M : t$ *et* $R^{\Gamma(x)}\ (\rho_1(x), \rho_2(x))$ *pour tout* $x \in Dom(\Gamma)$, *alors* $R^t\ ([\![M]\!]\ \rho_1,\ [\![M]\!]\ \rho_2)$.

Corollaire

Si $\vdash M : t$ *alors* $R^t([\![M]\!], [\![M]\!])$.

Retour sur le problème de définissabilité

Toute fonction $f : O \to O$ définissable (i.e. de la forme $[\![M]\!]$ pour un certain terme M) doit donc être reliée avec elle-même au type $o \to o$:

$$R^{o \to o}\ (f, f) \quad \text{c.à.d.} \quad \forall x_1, x_2,\ R^o(x_1, x_2) \Rightarrow R^o(f\ x_1, f\ x_2)$$

Prenant $R^o = \{(V, F)\}$, on voit que les fonctions suivantes ne sont pas définissables :

- la fonction constante $f(x) = V$ puisque $(f(V), f(F)) = (V, V) \notin R^o$
- la fonction constante $f(x) = F$ puisque $(f(V), f(F)) = (F, F) \notin R^o$
- la négation $f(V) = F,\ f(F) = V$ puisque $(f(V), f(F)) = (F, V) \notin R^o$

Relations logiques syntaxiques

(R. Statman, *Logical relations and the typed λ-calculus*, Inf&Control 65, 1985)

Statman reformule les relations logiques sans référence à un modèle ensembliste du lambda-calcul typé, juste comme des relations entre lambda-termes (syntaxiques) modulo β-équivalence :

$$R^o\ (M_1, \ldots, M_n) = \quad \text{au choix}$$

$$R^{t \to s}\ (M_1, \ldots, M_n) \stackrel{def}{=} \forall N_1, \ldots, N_n,\ R^t\ (N_1, \ldots, N_n) \Rightarrow R^s\ (M_1\ N_1, \ldots, M_n\ N_n)$$

C'est Statman qui donne le nom de ≪théorème fondamental des relations logiques≫ au résultat suivant :

Théorème

Si $\vdash M : t$, alors $R^t(M, \ldots, M)$ pour toute relation logique R.

Lien avec la normalisation forte

Statman remarque que la démonstration de Tait (1967) de la normalisation forte du lambda-calcul simplement typé est un cas particulier de relation logique unaire.

On définit l'ensemble $RED(t)$ des termes réductibles de type t par récurrence sur t :

$$RED(o) = \{M \mid M \text{ est fortement normalisable}\}$$
$$RED(t \to s) = \{M \mid \forall N \in RED(t), M\ N \in RED(s)\}$$

On démontre alors :

1. Si $\vdash M : t$ alors $M \in RED(t)$.
2. Si $M \in RED(t)$ alors M est fortement normalisable.

RED est une relation logique unaire et (1) découle du théorème fondamental des relations logiques.

Extension aux types abstraits

(John C. Reynolds, *Types, abstraction and parametric polymorphism*, 1983)

Reynolds observe que les relations logiques permettent de raisonner sur un type abstrait et ses multiples implémentations, à condition de permettre à un même nom de type d'être interprété par différents ensembles.

Dans l'exemple des nombres complexes cartésiens et polaires :

$$\begin{aligned} [\![R]\!]_1 &= \mathbb{R} \\ [\![C]\!]_1 &= \mathbb{R} \times \mathbb{R} \\ [\![t \to s]\!]_1 &= [\![t]\!]_1 \to [\![s]\!]_1 \end{aligned} \qquad \begin{aligned} [\![R]\!]_2 &= \mathbb{R} \\ [\![C]\!]_2 &= \mathbb{R}^+ \times \mathbb{R} \\ [\![t \to s]\!]_2 &= [\![t]\!]_2 \to [\![s]\!]_2 \end{aligned}$$

Une relation logique (binaire) est alors une famille $R^t \subseteq [\![t]\!]_1 \times [\![t]\!]_2$ t.q.

$$R^{t \to s}\ (f_1, f_2) \stackrel{def}{=} \forall x_1 \in [\![t]\!]_1, x_2 \in [\![t]\!]_2,\ R^t\ (x_1, x_2) \Rightarrow R^s\ (f_1\ x_1, f_2\ x_2)$$

Extension aux types abstraits

Le théorème fondamental (que Reynolds appelle le théorème d'abstraction) reste vrai : les interprétations d'un terme bien typé dans des environnements reliés sont reliées.

Théorème

Si $\Gamma \vdash M : t$ *et* $R^{\Gamma(x)}\ (\rho_1(x), \rho_2(x))$ *pour tout* $x \in Dom(\Gamma)$, *alors* $R^t\ (\llbracket M \rrbracket\ \rho_1,\ \llbracket M \rrbracket\ \rho_2)$.

Indépendance vis-à-vis des représentations

Application aux nombres complexes : on prend pour Γ la signature des opérations, ρ_1 leur implémentation en représentation cartésienne, ρ_2 leur implémentation en représentation polaire.

$$\Gamma = \texttt{inj} : R \to C;\ \texttt{i} : C;\ \texttt{conj} : C \to C; \ldots$$
$$\rho_1 = \{\texttt{inj} \mapsto \lambda x.\,(x, 0);\ \texttt{i} \mapsto (0, 1);\ \texttt{conj} \mapsto \lambda(x, y).\,(x, -y); \ldots\}$$
$$\rho_2 = \{\texttt{inj} \mapsto \lambda x.\,(x, 0);\ \texttt{i} \mapsto (1, \pi/2);\ \texttt{conj} \mapsto \lambda(r, \theta).\,(r, -\theta); \ldots\}$$

Pour ce qui est des relations logiques sur les types de base, on prend

$$R^R\,(x_1, x_2) = (x_1 = x_2)$$
$$R^C\,((x, y), (r, \theta)) = (x = r\cos\theta \wedge y = r\sin\theta)$$

Indépendance vis-à-vis des représentations

Il suffit alors de montrer que les opérations dans les deux représentations sont bien reliées :

$$R^{\Gamma(op)}\ (\rho_1(op), \rho_2(op)) \quad \text{pour tout } op \in Dom(\Gamma)$$

Le théorème fondamental nous garantit alors que tous les calculs de type R donnent les mêmes résultats avec les deux implémentations des nombres complexes :

$$\Gamma \vdash M : R \Longrightarrow [\![M]\!]\ \rho_1 = [\![M]\!]\ \rho_2$$

Plus généralement : deux implémentations d'un type abstrait reliées par une relation logique sont observationnellement équivalentes, même si leurs types de représentation diffèrent.

Extension au polymorphisme (système F)

(John C. Reynolds, *Types, abstraction and parametric polymorphism*, 1983)

À la fin de son article de 1983, Reynolds tente d'étendre les relations logiques au lambda-calcul polymorphe (système F).

Problème : l'imprédicativité. Un terme de type polymorphe $\forall X.\, t$ peut être instancié en $t\{X \leftarrow t'\}$ pour n'importe quel type t', y compris $t' = \forall X.\, t$. Exemple :

$$\text{si} \quad id : \forall X.\, X \to X \quad \text{alors} \quad id\ [\forall X.\, X \to X]\ id : \forall X.\, X \to X$$

On ne peut donc pas modéliser $\forall X.\, t$ en intersectant toutes les instantiations :

✘ $$[\![\forall X.\, t]\!] = \bigcap_{t' \text{ type}} [\![t\{X \leftarrow t'\}]\!]$$

✘ $$R^{\forall X.\, t}(x_1, x_2) = \forall t',\ R^{t\{X \leftarrow t'\}}(x_1, x_2)$$

Normalisation forte du système F

Girard (1972) avait rencontré le même problème pour montrer la normalisation forte du système F. Une extension naïve de la méthode de réductibilité donne une définition circulaire :

$$\text{✘} \quad RED(\forall X.\, t) = \{\, M \mid \forall t',\ M[t'] \in RED(t\{X \leftarrow t'\}) \,\}$$

L'idée de Girard est d'interpréter les variables de type X pas seulement par les ensembles $RED(t')$ pour un certain type t', mais par une classe plus large d'ensembles : les candidats de réductibilité.

Un ensemble U de termes est un candidat de réductibilité si

1. tout $M \in U$ est fortement normalisable ;
2. U est fermé par β-expansion : si $M \to_\beta M'$ et $M' \in U$ alors $M \in U$
3. U est fermé par certaines β-réductions.
   (Voir Girard, Lafont, Taylor, *Proofs and Types*, ch. 14)

Normalisation forte du système F

Réductibilité : (Φ : variable de type $\to$ candidat)

$$
\begin{aligned}
RED(o, \Phi) &= \{M \mid M \text{ est fortement normalisable}\} \\
RED(t \to s, \Phi) &= \{M \mid \forall N \in RED(t, \Phi), M\ N \in RED(s, \Phi)\} \\
RED(X, \Phi) &= \Phi(X) \\
RED(\forall X.\, t, \Phi) &= \{M \mid \forall t', \forall R \in \text{candidats}(t'), M[t'] \in RED(t,\ \Phi + X \mapsto R\}
\end{aligned}
$$

On démontre alors :

1. Si $M \in RED(t, \Phi)$ alors M est fortement normalisable.
2. $RED(t, \Phi)$ est un candidat de réductibilité.
3. Si $\vdash M : t$ alors $M \in RED(t, \Phi)$.

Relations logiques pour système F

La même idée s'applique aux relations logiques : il faut pouvoir interpréter les variables de types X par des relations «sémantiques», prises dans un ensemble beaucoup plus grand que celui des relations «syntaxiques» $\{R^t \mid t \text{ type}\}$.

$$R^o_\Phi(x_1, x_2) \qquad \text{au choix}$$

$$R^{t\to s}_\Phi\,(f_1, f_2) \overset{def}{=} \forall x_1, x_2,\ R^t_\Phi\,(x_1, x_2) \Rightarrow R^s_\Phi\,(f_1\,x_1, f_2\,x_2)$$

$$R^X_\Phi\,(x_1, x_2) \overset{def}{=} \Phi(X)\,(x_1, x_2)$$

$$R^{\forall X.\,t}_\Phi\,(x_1, x_2) \overset{def}{=} \forall U \in \mathcal{U},\ R^t_{\Phi + X\mapsto U}\,(x_1, x_2)$$

Reste à (1) fixer un modèle pour les types et les termes du système F, et (2) déterminer l'ensemble $\mathcal{U}$ des relations «sémantiques».

Modèles de système F

Modèle ensembliste, p.ex. Reynolds (1983) :
impossible pour des raisons de cardinalité.
(Reynolds, *Polymorphism is not set-theoretic*, 1984).

Modèles dans les domaines de Scott :
p.ex. Bruce, Meyer, Mitchell, 1990, utilisé par Wadler, 1991.
Les relations $\mathcal{U}$ doivent être admissibles (fermées par limites).

Modèles catégoriques :
p.ex. les espaces cohérents de Girard.

Approches purement syntaxiques, à la manière de Statman.
p.ex. Harper, *Practical foundations for prog. lang.* chap. 48.
Les relations $\mathcal{U}$ doivent être fermées par β-expansion et par équivalence observationnelle.

Plongements de système F et des relations dans une théorie des types.
Voir partie III.

II

Théorèmes gratuits et autres applications

Le type $\forall X.\, X$ est vide

Supposons $\vdash M : \forall X.\, X$.

Par le théorème fondamental, $R^{\forall X.\, X}\ (\llbracket M \rrbracket, \llbracket M \rrbracket)$.

Interprétons X par la relation vide $\emptyset$. On a donc :

$$R^{X}_{X \mapsto \emptyset}\ (\llbracket M \rrbracket, \llbracket M \rrbracket) \qquad \text{c.à.d.} \qquad (\llbracket M \rrbracket, \llbracket M \rrbracket) \in \emptyset$$

C'est impossible, donc M n'existe pas.

Valeurs du type $\forall X.\, X \to X$

Supposons $\vdash M : \forall X.\, X \to X$.

Soit t un type et $x \in [\![t]\!]$. Montrons que $[\![M]\!]\, x = x$.

Par le théorème fondamental : $R^{\forall X.\, X \to X}\, ([\![M]\!], [\![M]\!])$.

En interprétant X par la relation $\overline{X} = \{(x, x)\}$, on obtient :

$$\forall y_1, y_2,\ (y_1, y_2) \in \overline{X} \Rightarrow ([\![M]\!]\, y_1, [\![M]\!]\, y_2) \in \overline{X}$$

On prend $y_1 = y_2 = x$ et on obtient $[\![M]\!]\, x = x$.

Ceci vaut pour tout x, et donc $[\![M]\!]$ est la fonction identité.

Dans certains modèles mais pas tout, on peut en déduire que $M =_{\beta\eta} \Lambda X.\lambda x : X.\, x$.

Valeurs du type $\forall X.\, X \to X \to X$

On se donne un type de base `bool` avec deux éléments V et F.

Supposons $\vdash M : \forall X.\, X \to X \to X$. Soit t un type et $x, y \in [\![t]\!]$.

On interprète X par la relation $\overline{X} \subseteq [\![t]\!] \times [\![bool]\!] = \{ (x, V); (y, F) \}$.

En conséquence du théorème fondamental on obtient :

$$\forall u_1, u_2, v_1, v_2.\ (u_1, u_2) \in \overline{X} \wedge (v_1, v_2) \in \overline{X} \Rightarrow ([\![M]\!]\ u_1\ v_1, [\![M]\!]\ u_2\ v_2) \in \overline{X}$$

On prend $u_2 = V$ et $v_2 = F$ ce qui force $u_1 = x$ et $v_1 = y$, et donc

$$([\![M]\!]\ x\ y, [\![M]\!]\ V\ F) \in \overline{X}$$

Si $[\![M]\!]\ V\ F = V$ alors $[\![M]\!]\ x\ y = x$, et ce pour tous x et y.
Si $[\![M]\!]\ V\ F = F$ alors $[\![M]\!]\ x\ y = y$, et ce pour tous x et y.

Relations fonctionnelles

Si $f : A \to B$ est une fonction (ensembliste), on peut interpréter une variable de type X par son graphe $\overline{f} \stackrel{def}{=} \{(a, b) \mid b = f(a)\}$.

Deux éléments x, y sont reliés au type X ssi $y = f(x)$.

Deux fonctions g, h sont reliées au type $X \to X$ ssi $\forall x,\ h(f\ x) = f(g\ x)$.

Deux fonctions g, h sont reliées au type $X \to X \to X$ ssi
$\forall x, y,\ h\ (f\ x)\ (f\ y) = f(g\ x\ y)$.

Valeurs du type $\forall X.\, X \to X \to X$ (autre méthode)

Supposons $\vdash M : \forall X.\, X \to X \to X$.

En interprétant X par le graphe d'une fonction $\bar{f}$, on obtient

$$\forall x, y,\ [\![M]\!]\ (f\ x)\ (f\ y) = f\ ([\![M]\!]\ x\ y)$$

Soit t un type, et $x, y \in [\![t]\!]$.

On définit $f : \texttt{bool} \to [\![t]\!]$ par $f(V) = x$ et $f(F) = y$.

Il vient

$$[\![M]\!]\ x\ y = f([\![M]\!]\ V\ F) = \begin{cases} x & \text{si } [\![M]\!]\ V\ F = V \\ y & \text{si } [\![M]\!]\ V\ F = F \end{cases}$$

Extensions : produits, sommes, listes

On peut facilement définir les relations logiques sur les types produits, sommes, et listes :

$$R^{t \times s} = \{((x, y), (x', y')) \mid (x, x') \in R^t \wedge (y, y') \in R^s\}$$

$$\begin{aligned} R^{t+s} = \{&(\mathtt{inj}_1(x), \mathtt{inj}_1(x') \mid (x, x') \in R^t\} \\ &\cup \{(\mathtt{inj}_2(y), \mathtt{inj}_1(y') \mid (y, y') \in R^s\} \end{aligned}$$

$$R^{t^*} = \{([x_1, \ldots, x_n],\ [x'_1, \ldots, x'_n] \mid (x_i, x'_i) \in R^t \text{ pour } i = 1, \ldots, n\}$$

Des théorèmes sur les listes

(Phil Wadler, *Theorems for free!*, 1991)

Les ≪théorèmes gratuits≫ de Wadler sur les listes se déduisent facilement du théorème fondamental, à l'aide de relations fonctionnelles.

$$\mathtt{map}\, f\, (\llbracket F \rrbracket\, x) = \llbracket F \rrbracket\, (\mathtt{map}\, f\, x) \qquad \text{si } F : \forall X.\, X^* \to X^*$$

$$\mathtt{map}\, f\, (\llbracket G \rrbracket\, x\, y) = \llbracket G \rrbracket\, (\mathtt{map}\, f\, x)\, (\mathtt{map}\, f\, y) \qquad \text{si } G : \forall X.\, X^* \to X^* \to X^*$$

$$\mathtt{map}\, f\, (\llbracket H \rrbracket\, x) = \llbracket H \rrbracket\, (\mathtt{map}\, (\mathtt{map}\, f)\, x) \qquad \text{si } H : \forall X.\, X^{**} \to X^*$$

$$\mathtt{map}\, f\, (\llbracket \Phi \rrbracket\, (p \circ f)\, x) = \llbracket \Phi \rrbracket\, p\, (\mathtt{map}\, f\, x) \qquad \text{si } \Phi : \forall X.\, (X \to \mathtt{bool}) \to X^* \to X^*$$

Notons que si X est interprété par la relation fonctionnelle $\overline{f}$,
alors la relation au type X^* est $\overline{\mathtt{map}\, f}$,
et la relation au type X^{**} est $\overline{\mathtt{map}\, (\mathtt{map}\, f)}$.

Valeurs du type $\forall X.\, X \to (X \to X) \to X$

Soit M tel que $\vdash M : \forall X.\, X \to (X \to X) \to X$.

On se donne un type de base `nat` interprété par $\mathbb{N}$.

Soit t un type, $f : [\![t]\!] \to [\![t]\!]$ une fonction sur t, et $a \in [\![t]\!]$.

On interprète le type X par la relation

$$\overline{X} = \{(0, a);\ (1, f\ a); \ldots; (n, f^n\ a); \ldots\}$$

La fonction *succ* et la fonction f sont reliées au type $X \to X$:
si $(n, x) \in \overline{X}$, alors $x = f^n\ a$, et donc $(succ\ n,\ f\ x) = (n+1, f^{n+1}\ a) \in \overline{X}$.

Par conséquent, $[\![M]\!]\ 0\ succ$ et $[\![M]\!]\ a\ f$ sont reliées au type X.

D'où $[\![M]\!]\ a\ f = f^n\ a$ où n est déterminé par $n = [\![M]\!]\ 0\ succ$.

M est donc le n-ième entier de Church.

Codages fonctionnels des types inductifs

Nous avons vu (cours du 28/11) que tout type inductif, et pas juste les entiers naturels, pouvait se coder dans système F sous forme de fonctions polymorphes $\forall X. \ldots \to X$.

L'exemple précédent montre que le codage fonctionnel des entiers naturels est bien isomorphe aux entiers naturels, en ce sens que toutes les valeurs de ce type sont bien l'image d'un entier par le codage.

Ce résultat s'étend aux codages fonctionnels de tous les types inductifs.
(Abadi, Plotkin, *A logic for parametric polymorphism*, 1993.)

Exercice : montrer que $\forall X.\ (A \to B \to X) \to X$ est isomorphe à $A \times B$.

Paramétricité et syntaxe abstraite d'ordre supérieur

Pour représenter des termes contenant des variables et des lieurs :

- Syntaxe abstraite du premier ordre :
  représenter explicitement les variables.
- Syntaxe abstraite d'ordre supérieur (*Higher-Order Abstract Syntax*) :
  utiliser le ≪lambda≫ du langage hôte.

Exemple : le lambda-calcul pur.

```
type lam =
  | Var of int
  | Lam of lam
  | App of lam * lam
```

Premier ordre
(indices de de Bruijn)

```
type lam =

  | Lam of (lam -> lam)
  | App of lam * lam
```

Ordre supérieur

Paramétricité et syntaxe abstraite d'ordre supérieur

```
type lam = Lam of (lam -> lam) | App of lam * lam
```

Problème : dans un langage comme OCaml il y a beaucoup d'expressions de type `lam` qui ne représentent pas un lambda-terme, comme

```
Lam (fun x -> Lam (fun y -> match x with Lam _ -> x | App _ -> y))
```

Washburn et Weirich (2008) ont conjecturé que le codage fonctionnel du type algébrique ci-dessus,

$$\forall X.\ ((X \to X) \to X) \to (X \to X \to X) \to X$$

ne contient pas de tels termes ≪exotiques≫, en raison de la paramétricité vis-à-vis de X.

Paramétricité et syntaxe abstraite d'ordre supérieur

R. Atkey, *Syntax for free*, TLCA 2009.

```
type lam =
  | Var of int
  | Lam of lam
  | App of lam * lam
```

Atkey (2009) démontre que le type

$$\forall X.\ ((X \to X) \to X) \to (X \to X \to X) \to X$$

est bien isomorphe aux termes clos représentés en indices de de Bruijn, c.à.d. au sous-ensemble du type algébrique `lam` ci-dessus où il n'y a pas de variables libres.

(Il faut utiliser des relations logiques de Kripke, une extension des relations logiques.)

III

La paramétricité en théorie des types

Quelle logique pour parler de paramétricité ?

Jusqu'ici nous avons décrit la paramétricité en logique mathématique usuelle, via un modèle $[\![\cdots]\!]$ qui interprète les types et les termes du langage (système F) dans cette logique.

Autre possibilité : développer une logique ≪sur mesure≫ pour raisonner sur les termes du système F et sur les relations logiques. Le théorème fondamental des relations logiques est un axiome.
(Abadi et Plotkin, *A logic for parametric polymorphism*. TLCA 1993.)

Troisième possibilité : utiliser un formalisme unifié (théorie des types, *Pure Type System*) permettant de décrire à la fois le langage polymorphe et sa logique de paramétricité.
(Bernardy et al, 2010–2015 ; Lasson et al, 2011, 2012.)

Rappel sur les Pure Type Systems

(Voir le 2e cours du 21/11, ≪Polymorphisme à tous les étages≫)

Univers :		$U \in \mathcal{U}$	
Termes, types :	$A, B, C ::=$	x	variables
		$\mid \lambda x : A.\, B$	abstractions
		$\mid A\ B$	applications
		$\mid \Pi x : A.\, B$	type dépendant de fonction
		$\mid U$	nom d'univers

Notation : $A \to B \stackrel{def}{=} \Pi x : A.\, B$ si x non libre dans B.

Un PTS donné s'obtient en fixant l'ensemble $\mathcal{U}$ des univers et deux relations entres univers, $\mathcal{A}$ (quel univers est de quel univers ?) et $\mathcal{R}$ (quels Π-types peut-on former ?).

Paramétricité dans un PTS

(Bernardy, Jansson, Paterson, *Proofs for free*, JFP 2012)

Notations :

- Pour chaque variable x on se donne deux nouvelles variables x_1, x_2.
- A_i est le terme A où chaque variable libre x est remplacée par x_i.

Intuitions :

- Chaque type $A : U$ devient une relation $[\![A]\!] : A_1 \to A_2 \to \tilde{U}$ avec la forme générale d'une relation logique.
- Chaque terme $A : B$ devient une preuve que la relation $[\![B]\!]$ relie A_1 et A_2, i.e. $[\![A]\!] : [\![B]\!]\ A_1\ A_2$.
- En particulier, quand A et B sont clos, on aura $[\![A]\!] : [\![B]\!]\ A\ A$, ce qui démontre le théorème fondamental des relations logiques.

(Note : l'article traite les relations n-aires. Ici on prend $n = 2$.)

Types de fonctions

Cas particulier 1 : un type de fonction non dépendant, $A \to B$.
Condition habituelle : deux fonctions reliées envoient des arguments reliés sur des résultats reliés.

$$[\![A \to B]\!] = \lambda f_1 : A_1 \to B_1.\ \lambda f_2 : A_2 \to B_2.\ \forall x_1 : A_1.\ \forall x_2 : A_2.\ [\![A]\!]\ x_1\ x_2 \to [\![B]\!]\ (f_1\ x_1)\ (f_2\ x_2)$$

Cas particulier 2 : le ≪pour tout≫ du système F, $\forall X : \star.\ B$.
X est interprété par deux types X_1, X_2 et une relation $X : X_1 \to X_2 \to \texttt{Prop}$.

$$[\![\forall X : \star.\ B]\!] = \lambda f_1 : \forall X : \star.\ B_1.\ \lambda f_2 : \forall X : \star.\ B_2.\ \forall X_1 : \star.\ \forall X_2 : \star.\ \forall X : X_1 \to X_2 \to \texttt{Prop}.\ [\![B]\!]\ (f_1\ X_1)\ (f_2\ X_2)$$

Types de fonctions

Le cas général :

$$[\![\Pi x : A.\ B]\!] = \lambda f_1 : \Pi x : A_1.\ B_1.\ \lambda f_2 : \Pi x : A_2.\ B_2.\ \forall x_1 : A_1.\ \forall x_2 : A_2.\ \forall x : [\![A]\!]\ x_1\ x_2.\ [\![B]\!]\ (f_1\ x_1)\ (f_2\ x_2)$$

avec, par conséquent :

$$[\![x]\!] = x$$
$$[\![U]\!] = \lambda x_1 : U.\ \lambda x_2 : U.\ x_1 \to x_2 \to \tilde{U}$$

Abstractions, applications, contextes

En cohérence avec la traduction des types de fonctions :

$$[\![\lambda x : A.\, B]\!] = \lambda x_1 : A_1.\, \lambda x_2 : A_2.\, \lambda x : [\![A]\!]\, x_1\, x_2.\, [\![B]\!]$$

$$[\![A\ B]\!] = [\![A]\!]\, B_1\, B_2\, [\![B]\!]$$

Chaque variable libre x est traduite par 3 variables : deux interprétations x_1, x_2 et une relation x. D'où la traduction des contextes de typage Γ :

$$[\![\emptyset]\!] = \emptyset$$

$$[\![\Gamma, x : A]\!] = [\![\Gamma]\!],\ x_1 : A_1,\ x_2 : A_2,\ x : [\![A]\!]\, x_1\, x_2$$

Le théorème fondamental

Modulo des hypothèses techniques sur les univers U et leur traduction $\tilde{U}$:

Théorème

Si $\Gamma \vdash A : B$, *alors* $[\![\Gamma]\!] \vdash [\![A]\!] : [\![B]\!]\ A_1\ A_2$.

En corollaire, pour un terme clos : si $\vdash A : B$, alors $[\![B]\!]\ A\ A$ (A est relié à lui-même par la relation logique $[\![B]\!]$).

On retrouve les résultats sur système F en prenant $\tilde{\star} = \texttt{Prop}$: un type $A : \star$ devient une relation $[\![A]\!] : A_1 \to A_2 \to \texttt{Prop}$.

Extensions gratuites : constructeurs de types (F_ω)

Un constructeur de type comme $\texttt{list} : \star \to \star$ devient un transformateur de relations :

$$[\![\star \to \star]\!] = \lambda(F_1, F_2 : \star \to \star).\ \forall(X_1, X_2 : \star).\\ (X_1 \to X_2 \to \tilde{\star}) \to (F_1\, X_1 \to F_2\, X_2 \to \tilde{\star})$$

Par exemple, $[\![\star \to \star]\!]\ \texttt{list}\ \texttt{list}$ prend une relation $X_1 \to X_2 \to \texttt{Prop}$ entre deux types quelconques et rend une relation $\texttt{list}(X_1) \to \texttt{list}(X_2) \to \texttt{Prop}$.

Extensions gratuites : types dépendants (LF)

Un type dépendant comme $\texttt{even} : \texttt{nat} \to \star$ devient :

$$[\![\texttt{nat} \to \star]\!] = \lambda(F_1, F_2 : \texttt{nat} \to \star).\ \forall(n_1, n_2 : \texttt{nat}).\\ [\![\texttt{nat}]\!]\ n_1\ n_2 \to (F_1\ n_1 \to F_2\ n_2 \to \tilde{\star})$$

En supposant que $[\![\texttt{nat}]\!]$ est la relation identité, $[\![\texttt{nat} \to \star]\!]\ \texttt{even}\ \texttt{even}$ prend un entier n et renvoie une relation $\texttt{even}(n) \to \texttt{even}(n) \to \texttt{Prop}$.

Exercice : que devient $\texttt{vec} : \star \to \texttt{nat} \to \star$, le constructeur du type $\texttt{vec}\ A\ n$ des listes de A de longueur n ?

Traduction des univers

En général on peut traduire les univers à l'identique : $\tilde{U} = U$.

C'est le seul choix possible pour Agda et sa hiérarchie d'univers $\mathtt{Set}_i : \mathtt{Set}_{i+1}$.

Pour Coq, on a un univers $\mathtt{Prop}$ imprédicatif en plus de la hiérarchie $\mathtt{Set} = \mathtt{Type}_0 : \mathtt{Type}_1 : \cdots : \mathtt{Type}_i : \cdots$.

Il est plaisant de prendre $\overline{\mathtt{Prop}} = \overline{\mathtt{Set}} = \mathtt{Prop}$ pour obtenir de ≪vraies≫ relations ($A_1 \to A_2 \to \mathtt{Prop}$ et non $A_1 \to A_2 \to \mathtt{Type}$).

Mais il faut prendre $\overline{\mathtt{Type}_i} = \mathtt{Type}_i$ pour $i > 0$
(car sinon la règle de typage $\vdash \mathtt{Type}_i : \mathtt{Type}_{i+1}$ ne se traduit pas).

La paramétricité est anti-classique

(M. Lasson, 2012)

Autrement dit : la logique classique n'est pas paramétrique. En effet, si on considère les types A correspondant à des lois classiques :

$\forall X.\, X + (X \to \bot)$	tiers exclu
$\forall X.\forall Y.\, ((X \to Y) \to X) \to X$	loi de Peirce
$\forall X.\, ((X \to \bot) \to \bot) \to X$	élimination double négation

et qu'on suppose qu'il existe un terme $a : [\![A]\!]$ (un témoin de la relation logique pour A), on obtient une contradiction.

Par exemple pour l'élimination de la double négation :
on interprète X par deux types quelconque et la relation fausse partout.
$[\![(X \to \bot) \to \bot]\!]\ x_1\ x_2$ est vraie. $[\![X]\!]\ (f_1\ x_1)\ (f_2\ x_2)$ est fausse.
Donc $[\![(X \to \bot) \to \bot) \to X]\!]\ f_1\ f_2$ est fausse.

IV

Pour aller plus loin

Autres travaux sur la paramétricité

Nous avons vu les principes de base de la paramétricité. C'est un domaine de recherche encore très actif, avec des extensions dans de nombreuses directions. Notamment :

Récursion générale, non-terminaison :
la théorie s'applique encore mais les ≪théorèmes gratuits≫ sont plus faibles (parce que $\bot$ appartient à tous les types).
(A. M. Pitts, *Parametric Polymorphism and Operational Equivalence*, MSCS 2000.)

Types récursifs, état mutable contenant des fonctions, … :
autant de cas où la définition des relations R^t n'est pas bien fondée par récurrence sur le type t. Il faut trouver d'autres principes de bonne fondation, comme le *step-indexing*.
(Cours du 9 janvier 2019).

Autres travaux sur la paramétricité

Typage dynamique, typage graduel :
un test dynamique de type comme le `instanceof` de Java peut ≪casser≫ la paramétricité :

$$(\lambda x.\ \texttt{if}\ x\ \texttt{instanceof nat then 1 else 0}) : \forall X.\ X \to \texttt{nat}$$

D'autres formes de typage dynamique, comme le typage graduel, préservent une partie des propriétés de paramétricité.

Neis, Dreyer, Rossberg. *Non-parametric parametricity.* JFP 2011.

Ahmed et al. *Theorems for Free for Free : Parametricity, With and Without Types.* ICFP 2017.

Liens avec l'égalité et l'homotopie :
pour des termes clos, est-ce que $R^t\ (M, N)$ implique $M = N$? Peut-on le démontrer ? en faire un axiome ? On voit apparaître des liens avec la théorie homotopique des types.

Nuyts, Vezzosi, Devriese, *Parametric Quantifiers for Dependent Type Theory*, ICFP 2017.

Aussi : le cours du 23 janvier 2019.

Autres travaux sur la paramétricité

Paramétricité et invariance dans d'autres domaines :

la paramétricité de Reynolds est l'invariance par changement de représentations. On retrouve des idées de paramétricité dans d'autres domaines où les propriétés d'invariance sont cruciales, p.ex. le théorème de Noether en physique (existence d'une grandeur physique conservée).

Atkey, *From Parametricity to Conservation Laws, via Noether's Theorem*, POPL 2014.

Liens avec la différentiation formelle de programmes :

comment les programmes réagissent à un petit changement des valeurs d'entrées ? comme ils réagissent à un changement de représentation des données ?

Cai, Giarusso, Rendel, Ostermann, *A theory of changes for higher-order languages – incrementalizing λ-calculi by static differentiation*, PLDI 2014.

V

Bibliographie

Bibliographie

Deux articles fondateurs :

- John C. Reynolds, *Types, abstraction and parametric polymorphism*, 1983.
  `http://www.cse.chalmers.se/edu/year/2010/course/DAT140_Types/Reynolds_typesabpara.pdf`
- Phil Wadler, *Theorems for free!*, 1989.
  `http://homepages.inf.ed.ac.uk/wadler/papers/free/free.ps.gz`

Relations logiques :

- Approche syntaxique : Robert Harper, *Practical Foundations for Programming Languages*, 2016, chapitre 48.
- Approche par modèles : John C. Mitchell, *Foundations for Programming Languages* , 1996, chapitres 5 et 8.

Démonstrations de normalisation forte :

- J.-Y. Girard, Y. Lafont, P. Taylor, *Proofs and Types*, 2003, `http://www.paultaylor.eu/stable/Proofs+Types.html`, chapitres 6 et 14.