# Un système de type avec polarités

Bruno Barras, Christine Paulin

Le but de ce projet est de modéliser un système de types permettant d'identifier des fonctions monotones croissantes ou décroissantes, comme dans le chapitre 2 de la thèse d'Andreas Abel intitulée "A Polymorphic Lambda-Calculus with Sized Higher-Order Types", disponible à partir de la page web de l'auteur (`http://www.tcs.informatik.uni-muenchen.de/~abel/`). On pourra s'en inspirer pour mener à bien ce projet.

On commencera par définir le système, puis dans une deuxième partie, on établira la décidabilité du jugement de typage. Indépendamment, une troisième partie définira la sémantique de ce système afin de prouver la correction des polarités.

## 1 Définition du système de types

### 1.1 Polarité, types, termes et environnements

L'idée est de considérer une extension du $\lambda$-calcul simplement typé dans lequel les type de fonctions peuvent être muni d'une "polarité", indiquant si celle-ci est monotone croissante (polarité $+$), décroissante (polarité -) ou éventiellement non monotone (polarité $\circ$). Ainsi $\tau_1 \to^+ \tau_2$ dénote les fonction croissantes de $\tau_1$ vers $\tau_2$.

Lorsque l'on compose une fonction de polarité $p_1$ avec une fonction de polarité $p_2$, on obtient une fonction de polarité $p_1 p_2$ défini par les règles suivantes : $\circ p = \circ$, $p \circ = \circ$, $++ = +$, $+- = -$, $-+ = -$ et $-- = +$.

1- Définir `polarity`, le type des polarités et `mul_pol`, la fonction de produit de polarité.

On considérera un ensemble de types simples avec comme seul type de base le type des entiers. Comme dit précédemment, les types fonctionnels seront munis d'une polarité.

2- Définir `typ`, le type représentant les types de notre système.

Les termes de notre calcul sera un $\lambda$-calcul muni de constantes. Les variables pourront être representées par des indices de de Bruijn, de façon à ce que la déclaration d'une variable $n$ s'obtienne en accédant au $n$-ième élément de l'environnement, qui pourra être codé comme une pile. On pourra toutefois aussi choisir une représentation des termes avec variables nommées.

Les $\lambda$-abstractions seront munies d'une polarité et du type de leur domaine afin de faciliter l'inférence de type. Le type des termes est donc :

```
Inductive trm : Set :=
| Var : nat -> trm
| Cst : cst -> trm
| App : trm -> trm -> trm
| Abs : polarity -> typ -> trm -> trm.
```

L'ensemble `cst` des constantes comprendra : 0, les fonctions successeur, addition, soustraction, et le récurseur sur les entiers de type $\tau$.
3- Définir le type `cst`.

Le type des environnements sera défini comme des listes de déclarations. Chaque déclaration de variable comportera un type et une polarité. Le type `option` dans la définition ci-dessous sera expliqué plus loin.

```
Require Import List.
Definition env := list (option (polarity * typ)).
```

### 1.2 Jugement de typage

Soit $\Delta$ un environnement, $t$ un terme et $\tau$ un type. On défini le jugement de typage $\Delta \vdash t : \tau$ exprimant (1) que le terme $t$ a le type $\tau$ dans l'environnement $\Delta$, et (2) que si l'on fait croître la valeur associée aux variables munie de la polarité $+$, et decroître les valeurs des variable munies de la polarité $-$, alors $t$ croît. Les règles de typage sont les suivantes :

$$\frac{\Delta(n)=(p,\tau) \quad p\in\{+,\circ\}}{\Delta\vdash n:\tau} \qquad \frac{}{\Delta\vdash c:\Sigma(c)}$$

$$\frac{\Delta;(p,\tau_1)\vdash m:\tau_2}{\Delta\vdash\lambda(p,\tau_1).m:\tau_1\to^p\tau_2} \qquad \frac{\Delta\vdash u:\tau_1\to^p\tau_2 \quad p^{-1}\Delta\vdash v:\tau_1}{\Delta\vdash u\ v\ :\tau_2}$$

La notation $p^{-1}\Delta$ est définie de la manière suivante : $+^{-1}\Delta = \Delta$, $-^{-1}\Delta$ consiste à composer $-$ avec les polarités des variables de $\Delta$, et $\circ^{-1}\Delta$ consiste à supprimer de $\Delta$ toutes les variable dont la polarité n'est pas $\circ$. Afin de ne pas devoir faire un relogement complexe des indices de de Bruijn, l'effacement d'une variable se fera en mettant `None` à son emplacement.
1- Écrire la fonction $\Sigma$, de type `cst`$\to$`typ`, associant à chaque constante son type le plus précis, (c'est-à-dire en employant les polarités $+$ et $-$ à chaque fois que cela est possible).
2- Définir un prédicat inductif `type` représentant le jugement de typage.
3- En se basant sur la signification du jugement de typage donnée ci-dessus, expliquer les règles de typage, en particulier celle de l'application.

## 2 Décidabilité du typage

1- Écrire une fonction de type `env`$\to$`trm`$\to$`option typ`, qui retourne soit le type d'un terme bien formé de le contexte donné, soit `None` si et seulement si le terme est mal typé.
2- Prouver la propriété d'unicité du typage si un terme $t$ possède les types $\tau$ et $\tau'$ dans un même contexte $\Delta$, alors $\tau = \tau'$.
3- Prouver la décidabilité du typage :

```
Lemma decidabilite: forall e t,
  {ty|type e t ty} + {forall ty, ~ type e t ty}.
```

# 3 Correction des polarités

On supposera l'existence d'un modèle du $\lambda$-calcul. On ne cherchera pas à le définir.

```
Parameter lambda : Type.
Parameter mk_fun : (lambda -> lambda) -> lambda.
Parameter app : lambda -> lambda -> lambda.
Parameter beta_eq : forall f v, app (mk_fun f) x = f v.
```

Ce type `lambda` permet d'encoder tous les $\lambda$-termes.
1- Écrire les expressions représentant les $\lambda$-termes suivants : $\lambda x.x$, $\delta = \lambda x.x\,x$, $\delta\,\delta$, $0 = \lambda z\,f.z$, $S(n) = \lambda z\,f.f\,(n\,z\,f)$. Montrer que $\mathrm{fix}(f) = (\lambda x.f\,(x\,x))(\lambda x.f\,(x\,x))$ est un point-fixe de $f$, c'est-à-dire que $\mathrm{fix}(f) = f\,\mathrm{fix}(f)$.

L'idée de cette section est de construire un modèle de notre calcul. Les termes seront interprétés comme des objets de type `lambda`. Nous définirons ensuite un ordre pour chaque type $\tau$. Si $f$ et $g$ sont des fonctions de type $\tau_1 \to \tau_2$, alors $f \leq g \in \tau_1 \to \tau_2$ si $f(x) \leq g(x) \in \tau_2$ pour tout $x$.
2- Définir `leq:lambda->lambda->Prop`, l'ordre $\leq$ sur les entiers, codés comme ci-dessus. Ensuite, définir `ty_leq:typ→lambda→lambda→Prop` par récurrence sur le type. On aura $f \leq g \in \tau_1 \to^p \tau_2$ si pour tout $x$ et $y$ tels que $x \leq^p y \in \tau_1$, on a $f(x) \leq g(x) \in \tau_2$, $f(x) \leq f(y) \in \tau_2$ et $g(x) \leq g(y) \in \tau_2$. Autrement dit $f$ est inférieure à $g$ en $x$, et $f$ et $g$ sont des fonctions de polarité $p$. (La notation $f \leq^p g \in \tau$ étant définie par : $\leq^+ = \leq$, $\leq^- = \geq$ et $\leq^\circ = \leq \cap \geq$.)
3- Écrire `interp:(nat->lambda)->trm->lambda`, la fonction d'interprétation traduisant un terme dans `lambda`, la fonction servant de valuation pour interpréter les variable libres.
4- Les environnements seront interprétés comme des paires de valuations, dont les valeurs associées à chaque variables suivront la variance de cette variable : $\rho_1 \leq^p \rho_2 \in \Delta$ signifie que si $\Delta(n) = (q, \tau)$ alors $\rho_1(n) \leq^{pq} \rho_2(n)$.
5- Prouver la correction de la polarité : si $\Delta \vdash m : \tau$ et $\rho_1 \leq^p \rho_2 \in \Delta$, alors $[\![m]\!]_{\rho_1} \leq^p [\![m]\!]_{\rho_2} \in \tau$. Pour le cas de l'application, on utilisera le lemme : $\rho_1 \leq^p \rho_2 \in \Delta \Rightarrow \rho_1 \leq^{pq} \rho_2 \in q^{-1}\Delta$.