Interprétation des programmes – TP 2 : Le typage pour Hopix

Université Paris Diderot – Master 1

(2016-2017)

Cette séance de travaux pratiques a pour objectifs de :
— vous faire spécifier les règles de jugement d'un système de type pour HOPIX ;
— vous faire implémenter un *vérificateur de type* correspondant à ce système.

Exercice 1 (Spécification du système de type)
On se donne la syntaxe suivante pour les types et les environnements de typage de Hopix :

$$\begin{array}{lcl} \tau & ::= & \alpha \mid T\,(\tau_1, \ldots, \tau_n) \mid (\tau_1 \star \ldots \star \tau_n) \to \tau \\ \sigma & ::= & \forall \overline{\alpha}.\tau \\ \Gamma & ::= & \bullet \mid \Gamma\,(x : \sigma) \mid \Gamma\,\alpha \mid \Gamma\,(K : \sigma) \mid \Gamma\,(T(n) := \overline{K}) \end{array}$$

où τ est un type, α est une variable de type, σ est un schéma de type et T est un constructeur de type. Les constructeurs de type comprennent les types prédéclarés comme **int**, **bool** *ou* **ref** *ainsi que les types définis par le programmeur. On rappelle que la notation $\overline{\alpha}$ représente une séquence potentiellement vide de α. On écrira τ pour un schéma de type dont les paramètres $\overline{\alpha}$ sont vides. Dans la syntaxe des environnements de typage, $(x : \sigma)$ signifie que x a le schéma de type σ ; α signifie que la variable de type α est liée ; $(K : \sigma)$ signifie que le constructeur de données K a le schéma de type σ et enfin $(T(n) := \overline{K})$ signifie que le constructeur de type T a pour arité l'entier naturel n et pour constructeurs de données $\overline{K}$.*

Le jugement « $\Gamma \vdash e : \tau$ » se lit « Sous l'environnement de typage Γ, l'expression e est de type τ. » Des jugements similaires existent pour les programmes, les définitions, les motifs et les annotations de types.

Par exemple, la règle de typage pour les fonctions est :

$$\frac{\Gamma\overline{\alpha}(x_1 : \tau_1) \ldots (x_n : \tau_n) \vdash e : \tau \qquad \forall i, \Gamma\overline{\alpha} \vdash \tau_i \qquad \overline{\alpha}\#\Gamma}{\Gamma \vdash \backslash[\overline{\alpha}](x_1 : \tau_1, \ldots, x_n : \tau_2).e : \forall\overline{\alpha}.\tau_1 \star \ldots \star \tau_n \to \tau}$$

L'hypothèse "$\overline{\alpha}\#\Gamma$" signifie que les variables de type $\overline{\alpha}$ ne doivent pas apparaître dans Γ. L'hypothèse "$\Gamma\overline{\alpha} \vdash \tau_i$" correspond à la bonne formation des annotations de type écrites par l'utilisateur.

Pour une application, on utilise la règle suivante :

$$\frac{\Gamma \vdash a : \forall\alpha_1 \ldots \alpha_n.\tau'_1 \star \ldots \star \tau'_m \to \tau \qquad \forall i, \Gamma \vdash e_i : \tau'_i[\alpha_1 \mapsto \tau_1] \ldots [\alpha_n \mapsto \tau_n]}{\Gamma \vdash a[\tau_1, \ldots, \tau_n](e_1, \ldots, e_m) : \tau[\alpha_1 \mapsto \tau_1] \ldots [\alpha_n \mapsto \tau_n]}$$

1. *À quoi sert un système de type ?*
2. *En utilisant la sémantique opérationnelle décrite dans la spécification du jalon 2, écrire les règles de typage des expressions de* HOPIX.
3. *Dans les règles précédentes, quelle forme avez-vous donné au jugement de typage des motifs ?*
4. *En utilisant la sémantique opérationnelle décrite dans la spécification du jalon 2, écrire les règles de typage des programmes, des définitions, des branches et des motifs de* HOPIX.

□

Exercice 2 (Implémentation d'un vérificateur de type) *Un programme* HOPIX *totalement annoté est tel que :*

- *Toute variable apparaissant dans un motif est annotée par son type.*
- *Toute fonction récursive a un type de retour spécifié par le programmeur.*

1. *Complétez la fonction* `HopixTypechecker.check_program_is_fully_annotated`*. qui vérifie que le programme donné en entrée est totalement annoté.*
2. *Complétez la fonction* `HopixTypechecker.typecheck` *qui renvoie un environnement de typage contenant les types des définitions globales du programme pris en entrée si ce programme est bien typé et totalement annoté. Elle produit un message d'erreur dans le cas contraire.*

□