Interprétation des programmes – TP 3 : Le typage simple pour Hopix

Université Paris Diderot – Master 1

(2015-2016)

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

$$\begin{array}{rcl} \tau & ::= & T\ \overline{\tau} \\ \Gamma & ::= & \bullet \mid \Gamma\ (x : \tau) \mid \Gamma\ (T = d_t) \end{array}$$

où τ est un type et T est un constructeur de type.

On rappelle que la notation $\overline{\tau}$ représente une séquence potentiellement vide de τ. Par ailleurs, la notation « $\tau \to \tau'$ » représente $\to (\tau, \tau')$.

Le jugement « $\Gamma \vdash e : \tau$ » se lit « Sous l'environnement de typage Γ, l'expression e est de type τ. »

Par exemple, la règle de dérivation de ce jugement pour les variables est :

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau}$$

Des jugements similaires existent pour les programmes, les définitions, et les motifs.

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 à un type de retour spécifié par le programmeur.*

1. *Complétez la fonction* `HopixTypechecker.SimpleTypes.check_program_is_fully_annotated`*. qui vérifie que le programme donné en entrée est totalement annoté.*
2. *Complétez la fonction* `HopixTypechecker.SimpleTypes.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 produit un message d'erreur dans le cas contraire.*

□

Exercice 3 (Implémentation d'un algorithme d'inférence de type) *Un algorithme d'inférence de type attend un programme qui n'est pas totalement annoté et produit une version totalement annotée et bien typée de ce programme s'il n'y a qu'une unique façon de compléter le programme d'entrée en un programme totalement annoté.*

Pour cela, on procède par étapes :

1. *On construit une version totalement annotée du programme où les types manquants sont remplacés par des variables de type.*
2. *En parcourant le programme issu de l'étape précédente, on génère une conjonction d'égalités entre types équivalente au bon typage du programme.*
3. *On simplifie tant que possible la conjonction d'égalités à l'aide de l'unification du premier ordre.*
4. *Si la simplification de l'étape précédente mène à une conjonction d'égalité de la forme $\alpha = \tau$ où τ ne contient pas de variables de type alors on a trouvé une solution au problème d'inférence de type et on peut utiliser ces égalités pour réécrire le programme en un programme totalement annoté sans variable de type.*

1. *Complétez la fonction* `HopixTypechecker.SimpleTypes.annotate`*. qui correspond à l'étape 1 de l'algorithme.*
2. *Complétez la fonction* `HopixTypechecker.SimpleTypes.generate_constraint`*. qui correspond à l'étape 2 de l'algorithme.*
3. *Complétez la fonction* `HopixTypechecker.SimpleTypes.solve_constraint`*. qui correspond à l'étape 3 de l'algorithme.*
4. *Complétez la fonction* `HopixTypechecker.SimpleTypes.elaborate`*. qui correspond à l'étape 4 de l'algorithme.*

□