Projet du cours « Compilation »
Jalon 3 : Typage de HOPIX

version numéro 1.0

1 Système de types

1.1 Syntaxe des types et des environnements de typage

On se donne la syntaxe suivante pour les types et les environnements de typage de *Hopix* :

$$\begin{array}{lcl} \tau & ::= & \alpha \mid T\,(\tau_1, \ldots, \tau_n) \mid \tau \to \tau \mid \tau_1 \star \ldots \star \tau_n \\ \sigma & ::= & \forall \overline{\alpha}.\tau \\ \Gamma & ::= & \bullet \mid \Gamma, (x : \sigma) \mid \Gamma, \alpha \mid \Gamma, (T(\overline{\alpha}) := \Sigma_{i\in[1..n]} K_i : \overline{\tau}_i) \mid \Gamma, (T(\overline{\alpha}) := \{\ell_1 : \tau_1; \ldots; \ell_n : \tau_n\}) \end{array}$$

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, comme les enregistrements et sommes.

On rappelle que la notation $\overline{\alpha}$ représente une séquence potentiellement vide de α. Pour simplifier les notations, on écrira τ pour un monotype, c'est-à-dire 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 ; $(T(\overline{\alpha}) := \Sigma_{i\in[1..n]} K_i : \overline{\tau}_i)$ signifie que le constructeur de type T est un type somme paramétré par les variables de type $\overline{\alpha}$ et que ses constructeurs de données sont les $\overline{K}_i$ de paramètres $\overline{\tau}_i$; et enfin, $(T(\overline{\alpha}) := \{\ell_1 : \tau_1; \ldots; \ell_n : \tau_n\})$ signifie que le constructeur de T est un type enregistrement paramétré par les variables de type $\overline{\alpha}$ et que ses étiquettes ℓ_i ont pour paramètres τ_i.

1.2 Opérations sur les types, les schémas de type et les environnements

Le module `HopixTypes` fournit un ensemble d'opérations utiles. On décrit les principales dans cette section.

Syntaxe interne des types Les types de l'AST de Hopix sont annotés avec les positions du code source, mais ces positions vont nous gêner lors de la vérification des types. Par exemple, déterminer si deux types sont égaux exige d'ignorer les positions.

Pour résoudre ce problème, on introduit une *syntaxe interne* des types (voir le type `HopixTypes.aty`) qui suit la même structure mais ne contient pas les positions.

De nombreuses opérations sont fournies sur ces types : affichage, substitution, calcul des variables de type libres, etc. Par ailleurs, les types prédéfinis du langage (**bool**, **int**, etc.) sont aussi déclarés dans ce module.

Opérations sur les schémas de type Soit un schéma de type de la forme $\forall\overline{\alpha}.\tau$. On peut *instancier* ce schéma en substituant les variables de type $\overline{\alpha}$ par des (mono)types $\overline{\tau}$. On dit que le type τ' est une instance du schéma de type $\forall\overline{\alpha}.\tau$ par la substitution $\overline{\alpha} \mapsto \overline{\tau}$ si :

$$\tau' = \tau[\overline{\alpha} \mapsto \overline{\tau}]$$

Cette opération est réalisée par la fonction `HopixTypes.instanciate_type_scheme`.

Unification Les opérations d'unification sont utiles pour la partie optionnelle d'inférence des types. Vous pouvez les ignorer dans un premier temps.

Opérations sur l'environnement de typage Le type `typing_environment` contient les informations utiles au typage : les définitions de types et les schémas de types des variables.

On vous fournit de nombreuses opérations sur les environnements de typage : introduction d'une définition de types, d'une variable de type, d'un identificateur de valeurs, recherche d'informations sur ces objets, etc.

L'environnement doit obéir à un invariant indispensable durant la phase de typage : les types qui apparaissent dans l'environnement doivent être *bien formés*. On vous fournit la fonction `check_well_formed_type`, qui décide le jugement de bonne formation du type τ sous l'environnement Γ. Celui-ci s'écrit

$$\Gamma \vdash \tau.$$

Assurez-vous de lire et comprendre la fonction `check_well_formed_type`.

Environnement de typage initial L'environnement de typage initial est aussi fourni : étudiez sa définition !

1.3 Règles de typage

Dans cette section, on vous donne quelques règles de types mais pas toutes. Vous devez spécifier ces règles manquantes avant de les implémenter.

Le système de type de Hopix est défini par plusieurs jugements de typage : il y a un jugement pour les définitions, un pour les expressions, et un pour les motifs.

Typage des définitions Les définitions sont toutes traitées suivant leur ordre d'apparition dans le programme, exceptées les définitions de fonctions mutuellement récursives qui doivent être traitées ensemble (voir la prochaine sous-section).

Le jugement « $\Gamma \vdash d \Rightarrow \Gamma'$ » se lit « Sous l'environnement Γ, la définition d est bien formée et son introduction produit l'environnement Γ'. »

Par exemple, voici comment est typée une définition d'une valeur "simple" :

$$\frac{\Gamma \vdash \forall \overline{\alpha}.\tau \qquad \Gamma, \overline{\alpha} \vdash e : \tau \qquad \overline{\alpha} \notin FTV(\Gamma)}{\Gamma \vdash \textbf{let } x : \forall \overline{\alpha}.\tau = e \Rightarrow \Gamma, (x : \forall \overline{\alpha}.\tau)}$$

Ainsi pour pouvoir rajouter la définition de x dans l'environnement de typage, il faut vérifier que le schéma de type $\forall \overline{\alpha}.\tau$ écrit par l'utilisateur est bien formé et que l'expression e qui définit x est bien de type τ. Pour avoir le droit de généraliser les variables de type, il faut qu'elles n'apparaissent pas dans Γ.

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

Typage des fonctions mutuellement récursives Pour typer des fonctions $\overline{f}$ mutuellement récursives, on commence par vérifier que les annotations de types écrites par l'utilisateur sont bien formées. Ensuite, on produit tout de suite l'environnement Γ' dans lequel les fonctions $\overline{f}$ sont associées à leurs types *puis* on vérifie qu'effectivement les définitions de ces fonctions sont bien du type annoté par le programmeur.

Typage des expressions Le jugement de typage des expressions s'écrit :

$$\Gamma \vdash e : \tau$$

et se lit « Sous l'environnement Γ, l'expression e a le type τ. »

Par exemple, pour une application, on utilise la règle suivante :

$$\frac{\Gamma \vdash a : \tau_1 \to \tau \qquad \Gamma \vdash b : \tau_1}{\Gamma \vdash a\, b : \tau}$$

Typage des motifs Un motif bien formé étend un environnement Γ en un environnement Γ'. Le jugement de bon typage des motifs s'écrit :

$$\Gamma \vdash m : \tau \Uparrow \Gamma'$$

Par exemple, la règle de typage des motifs d'enregistrement s'écrit :

$$\frac{T(\overline{\alpha}) := \{\ell_1 : \tau_1; \ldots; \ell_n : \tau_n\} \in \Gamma_0 \qquad \forall i \in [1..n], \Gamma_{i-1} \vdash m_i : \tau_i[\overline{\alpha} \mapsto \overline{\tau}] \Uparrow \Gamma_i}{\Gamma_0 \vdash \{\ell_1 = m_1; \ldots; \ell_n = m_n\}[\overline{\tau}] : T(\overline{\tau}) \Uparrow \Gamma_n}$$

2 Implémentation d'un vérificateur de type

Votre vérificateur va s'appuyer sur des annotations de type écrites par le programmeur. En fonction de votre algorithme, certaines annotations vont être obligatoires, d'autres non.

Dans nos tests, nous allons supposer que :

— Toutes les définitions sont totalement annotées.
— Toutes les variables apparaissant dans un motif sont annotées par leur type.
— Toutes les fonctions anonymes sont annotées par leurs types.

mais il se peut que votre algorithme demande au programmeur d'écrire moins de types.

Dans tous les cas, vous devez avoir une idée précise des annotations de type qu'attend votre algorithme de vérification de bon typage. On vous donne donc la fonction `HopixTypechecker.check_program_is_fully_annotated` qui vérifie *avant de faire le typage* que le programmeur a bien écrit toutes les annotations nécessaires, conformément aux hypothèses ci-dessus. Si ce n'est pas le cas, cette fonction produit un message d'erreur à l'aide de la fonction `HopixTypechecker.type_error`. Si votre typeur est capable de demander moins d'annotations de type, vous pouvez modifier cette fonction pour rendre compte de cet allègement des annotations de type nécessaires.

Vous devez compléter la fonction `HopixTypechecker.typecheck` qui attend un environnement de typage, une liste de définitions (i.e. un programme) et qui produit l'environnement de typage qui correspond à l'introduction de ces définitions dans l'environnement, à la condition que ces dernières soient bien typées. Si le programme est mal typé alors vous devez utiliser la fonction `HopixTypechecker.type_error` pour produire un message d'erreur.

3 Travail à effectuer

La troisième partie du projet est la réalisation d'un vérificateur de types pour Hopix.
Le projet est à rendre **avant le** :

14 novembre 2022 à 19h59

Pour finir, vous devez vous assurer des points suivants :

— Le projet contenu dans cette archive **doit compiler**.
— Vous devez **être les auteurs** de ce projet.
— Il doit être rendu **à temps**.

Si l'un de ces points n'est pas respecté, la note de 0 vous sera affectée.

4 Log

24-10-2022 Version initiale.