TP 3 (bonus) : Inférence de types pour FUN

TP original de Iona Pasca et Marc Lasson.

1 Description des fichiers de l'archive

— `expr.ml`, `expr_conv.ml`, `expr_lexer.mll`, `expr_parser.ml` : ces fichiers étaient déjà présents dans les parties précédentes de ce TP.

— `term.ml`, `term_conv.ml`, `term_lexer.mll`, `term_parser.mly` : ces fichiers implémentent les termes de FUN et leur conversion en chaînes de caractères.

— `term_type.ml` : ce module contient la définition des types décrits ci-dessous, ainsi qu'une fonction de conversion des types en chaînes de caractères.

— `infer.ml` : c'est ici que vous allez implémenter l'inférence de types pour FUN en vous aidant d'un moteur d'unification implémenté dans une partie précédente (il faudra l'intégrer à tout ça manuellement).

— `main.ml` : c'est le point d'entrée du programme, qui lance l'inférence de type sur une batterie d'exemples. Dans cette partie, c'est à vous de le coder. Vous êtes encouragé.es à y rajouter vos propres exemples pour tester vos fonctions.

— `Makefile` : le Makefile vous permet de compiler votre projet. Comme précédemment, il suffit d'ouvrir un terminal dans le dossier et de taper `make run` pour compiler et faire tourner le code.

Pour résumer, outre l'intégration du moteur d'unification, les seuls fichiers que vous aurez à modifier sont `infer.ml` et `main.ml`.

2 Engendrer et résoudre un problème d'unification

1. Écrire, dans `infer.ml`, une fonction prenant en entrée une valeur de type `Term.t` et renvoyant un problème d'unification.

Vous pouvez, pour vous simplifier la vie, engendrer ce problème dans une chaîne de caractères, à l'image des exemples qui sont utilisés pour les tests dans le fichier `main.ml` de votre code pour l'unification.

Voici quelques remarques pour vous guider.

Comme vu en cours, la grammaire pour les types est donnée par

$$\tau \quad ::= \quad \texttt{int} \;\Big|\; \tau_1 \to \tau_2 \ .$$

Afin de pouvoir engendrer un problème d'unification, il faut enrichir cette grammaire avec des *inconnues*, aussi appelées *variables d'unification*, autrement dit fabriquer des termes à partir de la grammaire des types donnée ci-dessus.

Dans `term_types.ml`, les types sont donc définis ainsi

```
type ty = Tint | Tarr of ty * ty | Tuvar of string
```

Vous pouvez choisir d'avoir des variables d'unification de la forme `Tuvar "X0"`, `Tuvar "X1"`,..., ce qui facilitera la fabrication de variables "fraîches".

Il s'agit ensuite d'écrire une fonction qui calcule un problème d'unification à partir d'une expression FUN. On suivra l'algorithme vu en cours, et disponible dans les notes de cours.

Par exemple, pour la règle de l'addition, si X_0 désigne le type de $e_1 + e_2$, on crée deux variables d'unification fraîches X_1 et X_2, et on renvoie le problème

$$\{X_0 = \texttt{int}, X_1 = \texttt{int}, X_2 = \texttt{int}\} \cup U_1 \cup U_2$$

où U_i est le problème d'unification que l'on calcule récursivement pour e_i et X_i.

La fonction calculant un problème d'unification prendra donc en argument l'expression FUN e à typer et une variable d'unification, correspondant au type de e. Cette fonction doit également avoir un troisième argument, un "environnement de typage enrichi", dans lequel on associe à chaque variable FUN (un `x`) une variable d'unification (un $\texttt{X}_i$).

Comme dit plus haut, vous pouvez vous fonder sur l'algorithme vu en cours, mais évitez d'engendrer trop de contraintes pour les cas des feuilles (constantes et variables FUN).

À ce stade, vous devriez avoir une fonction qui prend en entrée une expression FUN et renvoie un booléen disant si elle est typable pour le système de types avec lequel on travaille.

2. Dans le cas où l'expression est typable, calculez un type pour l'expression initiale. Vous pouvez instancier les variables d'unification qui sont laissées "libres" après l'unification par un type arbitraire, comme `int`.
3. Affichez un message d'erreur un tant soit peu informatif si l'expression n'est pas typable.
4. Dans le cas où l'expression est typable, renvoyez et affichez une expression dans une variante de FUN où toutes les fonctions sont de la forme `fun` $x : \tau \to e$. Autrement dit, le paramètre de chaque fonction a une annotation de type.
5. Si vous êtes toujours là et que vous ne savez pas quoi faire, implémentez dans un fichier `term_eval.ml` une fonction d'évaluation des termes de FUN correspondant à la sémantique à petits pas, et testez-la sur des exemples.