Date limite : ???

Projet Logique, Déduction, et Programmation
Master 1 Informatique 2005-2006

Typage des expressions d'un langage fonctionnel

Description du projet

Il s'agit de programmer un environnement interactif dont le but est de calculer le type des expressions, rentrées par l'utilisateur, d'un langage fonctionnel minimal.
Ce langage minimal, défini comme il suit, pourra être étendu plus tard.

Types

Variables de types : `'a1`, ..., `'an`, ...
Types élémentaires : `int`, `bool`,...
Constructeurs de types : si `t1` et `t2` sont des types, alors

`t1 * t2`, `t1 -> t2`, `t1 list`,

sont des types.

Expressions

La grammaire des expressions est la suivante :

$$
\begin{aligned}
IDENT = {} & [a-z][a-zA-Z0-9\_]* \\
INTCONST = {} & [0-9]+ \\
BOOLCONST = {} & true \mid false \\
EXPR = {} & IDENT \mid INTCONST \mid BOOLCONST \mid \\
& EXPR + EXPR \mid EXPR - EXPR \mid EXPR * EXPR \mid \\
& if\ EXPR\ then\ EXPR\ else\ EXPR \mid EXPR = EXPR \mid \\
& EXPR\ \&\&\ EXPR \mid EXPR \mid\mid EXPR \mid not\ EXPR \mid \\
& (EXPR, EXPR) \mid fst\ EXPR \mid snd\ EXPR \mid \\
& EXPR\ EXPR \mid fun\ IDENT\ ->\ EXPR \mid \\
& [] \mid EXPR :: EXPR \mid \\
& letrec\ IDENT\ :=\ EXPR\ in\ EXPR
\end{aligned}
$$

Évidemment, une expression pourrait ne pas être bien typée (exemple : `true + false`). Il s'agit donc d'expliciter des règles pour pouvoir attribuer un type à une expression. Par exemple, une règle de formation a la forme suivante :

$$\frac{\texttt{e1 : t1},\ \texttt{e2 : t2},\ \ \ldots\ ,\ \texttt{en : tn}}{\texttt{e : t}}$$

Elle peut être lue comme il suit : si pour $i = 1, \ldots, n$ `ei` est un expression bien formée ayant type `ti`, alors `e` est une expression bien formée ayant type `t`.
Par exemple :

$$\frac{e1 : t1,\quad e2 : t2}{(e1, e2) : t1 * t2} \qquad \frac{e : t1 * t2}{fst\ e : t1} \qquad \frac{e : t1 * t2}{snd\ e : t2}$$

$$\frac{e1 : t1 -> t2,\quad e2 : t1}{e1\ \ e2 : t2} \qquad \frac{x : t1 \quad e : t2}{fun\ x\ ->\ e : t1 -> t2}$$

Comme exercice préliminaire, on décrira un telle règle pour chaque opérateur du langage.

Description détaillée

Analyse syntaxique. Il s'agit de transformer des chaînes de caractères en expressions représentés par des arbres syntaxiques.

On a deux choix : on pourra utiliser
- le pré-processeur `camlp4` et les modules `Stream` et `GenLex`, ou
- les outils `ocamllex` et `ocamlyacc`.

Il sera responsabilité du groupe de se renseigner sur ces modules/outils.

Génération et solution de contraintes. Un arbre syntaxique sera examiné et une suite de contraintes portant sur les types de chaque sous-expression sera engendrée. On cherchera une solution à cette suite de contraintes.

Il sera responsabilité du groupe de comprendre comme utiliser les concepts présentés dans le cours pour résoudre un ensemble de contraintes de types.

Modalités de travail.

On travaille impérativement par groupe (no. plus grand ou égal à 2, et plus petit ou égal à 3). Inclure les noms des participants au groupe dans le fichier `README` et dans chaque fichier source, ainsi qu'une description de la contribution de chaque participant à la réalisation du projet.
La structuration du projet sera documenté dans le fichier `README` et sera mise en ouvre à l'aide de fichiers indépendants (modules, possiblement avec interface fichier `.mli`).

Modalités de soumission.

Préparer un répertoire contenant le code source, le fichier `README`, le fichier `Makefile` (possiblement avec le fichier `ocamlmakefile`), et rien d'autre. Prendre contacte avec le responsable de votre groupe de TP (plus de détailles à venir). Date limite de soumission : ???.

Modalités d'évaluation

Dans l'évaluation de votre projet importance particulière sera donnée aux critères suivants :

Documentation. Votre projet contiendra un fichier `README` contenant :
- La description des programmes adressé à un utilisateur non-expert.
- Une description concise des programmes (adressés aux évaluateurs) décrivant les choix d'implémentation, les structures de données utilisés, les résultats accomplis par rapport à chaque tache décrite ci-dessus.

Qualité du Code. Le code source sera bien commenté (commenter d'abord les fichiers interface `.mli`), les noms des variables, des fonctions, et des types seront bien choisis.

Organisation des programmes en unités logiques. Chaque unité logique sera contenue dans un fichier séparé `nom.ml` (avec `nom` est bien choisi) ayant un fichier en interface `nom.mli` par défaut. La liste des unités logiques (donc la liste du code source) apparaîtra dans le fichier `README`, la compilation sera gérée par la commande `make` à l'aide d'un fichier `Makefile` possiblement incluant le fichier `ocamlmakefile`.

Fonctionnement de votre code. Le code source sera compilé au moment de l'évaluation. Votre présence n'est pas nécessaire par l'évaluation (toute explication éventuelle doit être contenue dans le fichier `README`). Le fonctionnement de votre code sera testé par rapport aux résultats déclarés accomplis dans le fichier `README`.

Remarque important

Il est impératif de soumettre le projet avant la date limite. Tout projet rendu après cette date de sera pas considéré. Il ne sera pas possible rendre le projet pendant ou après la session d'hiver. De même, il ne sera pas possible rendre une version améliore du projet pendant ou après la session d'hiver.