Projet CAML 1ère année 2008-2009
Calcul de type

1 Présentation

Le but de ce projet est développer un algorithme de calcul de type pour MINIML, un sous ensemble du langage CAML.

1.1 Syntaxe de MiniML

La syntaxe de MINIML est définie par la grammaire de la figure 1. Sans nuire à la généralité du calcul, les types de base sont restreints à *int* et *bool*.

Expr	→	*Ident*
	|	*Const*
	|	`(function` *Ident* `->` *Expr*`)`
	|	`(`*Expr*`)`
	|	*Expr Expr*
	|	*Expr Binaire Expr*
	|	*Unaire Expr*
	|	`if` *Expr* `then` *Expr* `else` *Expr*
	|	`let` *Liaison* `in` *Expr*
	|	`letrec` *Liaison* `in` *Expr*
Liaison	→	*Ident* `=` *Expr*
Const	→	*Entier* | *Booleen* | `[]`
Unaire	→	`-` | `not` | `hd` | `tl`
Binaire	→	`+` | `-` | `*` | `/` | `&&` | `||` | `=` | `<>` | `<` | `<=` | `>` | `>=` |::

FIG. 1 – Grammaire de MINIML

Les définitions sont toujours locales (*let in* et *letrec in*) et ne font donc pas appel à d'autres définitions précédentes. Il en résulte que toutes les variables utilisées dans une expression sont soit définies dans un *let* englobant, soit comme paramètre d'une *function* englobante. Par exemple $let\ x = 1\ in\ x + y;;$ ne peut pas être typée car y n'est pas définie.

La deuxième conséquence est que pour obtenir le type d'une fonction f on typera l'expression $let\ f = \ldots\ in\ f;;$

Pour les listes les constructeurs sont les opérateurs cons :: et vide [] (pas de possibilité de $[1; 2; \ldots]$. *hd* et *tl* sont considérés comme des opérateurs unaires.

La principale contrainte syntaxique par rapport à CAML est que les fonctions n'ont pas de notation abrégée $let\ f\ x\ = \ldots$ s'écrit nécessairement $let\ f\ =\ (function\ x -> \ldots);$

Exemples :

```
let id = (function x -> x) in id;;
```

```
let rec append = (function x -> (function y ->
   if x = []
   then y
   else (hd x)::(append (tl x) (y))
   ))
in append
;;
```

Un jeu de test `exemples.miniml` vous sera fourni dans la syntaxe de MiniML.

1.2 Syntaxe des types

La syntaxe des types MiniML est définie par la grammaire de la figure 2.

$$
\begin{array}{rcl}
Type & \rightarrow & \texttt{'}Lettre \mid \texttt{int} \mid \texttt{bool} \\
 & \mid & Type \rightarrow Type \\
 & \mid & (Type) \\
 & \mid & Type\ \texttt{list}
\end{array}
$$

Fig. 2 – Grammaire des types de MiniML

Les types seront fortement parenthésés comme dans l'exemple suivant :

```
(('a list) -> (('a list) -> ('a list)))
```

1.3 Calcul de type

Le principe de l'algorithme de calcul du type d'une expression MiniML utilisé dans ce projet, est de parcourir récursivement cette expression, et de produire par inférence, pour chaque sous-expression un type, et un ensemble d'équations sur les types introduits.

Dans un deuxième temps la résolution de l'ensemble d'équations obtenu pour l'expression complète permet de conclure si celle-ci est typable et de construire son type polymorphe le plus général.

1.3.1 Inférence de type

Les règles d'inférence que vous utiliserez pour produire le type et les équations à résoudre sont décrites à la figure 3.

Dans la suite nous considèrerons que α, β, γ, ...sont des variables de type et que τ, σ, ...sont des types quelconques.

Les jugements de typage sont de la forme :

$$\Gamma \vdash e : \tau,\ eq$$

où
- Γ est un environnement de typage sous la forme d'une liste ordonnée de couples de la forme $\{(x_1 : \alpha_1); \ldots; (x_n : \alpha_n)\}$ qui définissent les types α_i des variables x_i,
- e est l'expression typée,
- τ son type,
- eq est un ensemble d'équations de type $\tau_i = \sigma_i$ à résoudre.

Pour le typage d'une expression MiniML complète, on part d'un environnement vide (absence de définitions globales), et on obtient le type de e et un ensemble d'équations à résoudre :

$$(\emptyset \vdash e : \tau,\ eq).$$

Si cette déduction est possible c'est que l'expression ne contient pas de variables libres (non définies dans un *let* ou comme paramètre d'une fonction englobants).

Dans le cas d'une sous-expression e, Γ contient les types des variables libres de e (donc définies dans un *let*, *let rec* ou comme variable d'une function englobants).

Une fonction *newvar*() vous permettra d'engendrer des variables fraîches. Il y a une règle d'inférence pour chaque construction du langage MiniML :

1. Les deux règles $Cstint$ et $Cstbool$ précisent qu'un entier (resp. un booléen) est de type *int* (resp. *Bool*), que l'environnement peut être quelconque et qu'aucune équation n'est introduite.
2. La règle $Cstnil$ précise que la liste vide est de type $List(\alpha)$ où α est une variable fraîche, que l'environnement peut être quelconque et qu'aucune équation n'est introduite.
3. Les règles Var_1 et Var_2 précisent le typage d'une variable x. Si elle est présente dans l'environnement Γ de typage alors son type est le premier à gauche rencontré dans l'environnement et aucune équation n'est produite.

   Si elle n'est pas dans Γ l'expression n'est pas typable. Ceci correspond au fait qu'il n'y a pas de définitions globales. Par exemple $(function\ x \rightarrow\ x + y);;$ ne peut pas être typée et le typage doit alors produire une exception $Undef\ y$.
4. La règle $Unaire_{Int}$ précise que si le type de e est $\tau 1$ dans Γ, alors $-e$ est de type Int dans Γ, et qu'il faut ajouter une équation $\tau = Int$. Vous en déduirez la règle pour la négation sur le type $Bool$.
5. La règle $Binaire_{Bool}$ précise qui si, dans Γ, le type de e_1 est τ_1 et le type de e_2 est τ_2 alors le type de e_1 *op* e_2 est de type $Bool$ (pour tout opérateur binaire sur $Bool$ et le système d'équation est la réunion de eq_1 et eq_2 auquel on ajoute $\tau_1 = Bool$ et $\tau_2 = Bool$. Vous déduirez de cette règle celles concernant les opérateurs binaires sur Int et l'opérateur ::.
6. La règle $Tail$ précise que si e est de type τ dans Γ, alors *tl* e est de même type dans Γ, et qu'il faut ajouter une équation $\tau = List(\alpha)$, avec α. Vous en déduirez la règle pour l'opérateur *hd*.
7. La règle $Fonc$ indique que pour typer $(function\ x \rightarrow\ e)$, il faut ajouter en tête à Γ une liaison $x : \alpha$, où α est une variable fraîche, et typer e dans cet environnement.
8. La règle App précise que si dans Γ on peut typer e_1 et e_2 par τ_1 et tau_2, alors τ_1 doit être de la forme $\tau_2 \rightarrow \alpha$. Le type de e_1 e_2 est alors α et on fait l'union des équations produites.
9. La règle $IfThenElse$ fixe comme contrainte que la condition soit de type `Bool` et que les deux branches soient de même type. Il faut également faire l'union des systèmes d'équations.
10. Pour un $let\ x = e\ in\ e'$ on type e' dans un environnement où x est du type de e et on fait l'union des équations.
11. Dans le cas d'un $let\ rec\ x = e\ in\ e'$ on type e et e' dans le même environnement qui considère que le type de x est α, une variable non déjà utilisée. On fait l'union des équations et on ajoute une équation entre α et le type de e.

$$Cstint\ \frac{}{\Gamma \vdash c : Int,\ \emptyset}\ c \in \texttt{Entier} \qquad Cstbool\ \frac{}{\Gamma \vdash c : Bool,\ \emptyset}\ c \in \texttt{Booleen}$$

$$Cstnil\ \frac{}{\Gamma \vdash [] : List(\alpha),\ \emptyset}\ \alpha\ \texttt{variable fraîche}$$

$$Var_1\ \frac{}{(x : \alpha) :: \Gamma \vdash x : \alpha,\ \emptyset} \qquad Var_2\ \frac{\Gamma \vdash x : \alpha,\ \emptyset}{(y : \beta) :: \Gamma \vdash x : \alpha,\ \emptyset}$$

$$Unaire_{Int}\ \frac{\Gamma \vdash e : \tau,\ eq}{\Gamma \vdash op\ e : Int,\ eq \cup \{Int = \tau\}} \quad Tail\ \frac{\Gamma \vdash e : \tau,\ eq}{\Gamma \vdash tl\ e : \tau,\ eq \cup \{\tau = List(\alpha)\}}\ \alpha\ \texttt{variable fraîche}$$

$$Binaire_{Bool}\ \frac{\Gamma \vdash e_1 : \tau_1,\ eq_1 \quad \Gamma \vdash e_2 : \tau_2,\ eq_2}{\Gamma \vdash e_1\ op\ e_2 : \sigma,\ eq_1 \cup eq_2 \cup \{\tau_1 = Bool, \tau_2 = Bool\}}$$

$$Fonc\ \frac{(x : \alpha) :: \Gamma \vdash e : \tau,\ eq}{\Gamma \vdash (\texttt{function}\ x\ \texttt{->}\ e) : \alpha \to \tau,\ eq}\ \alpha\ \texttt{variable fraîche}$$

$$App\ \frac{\Gamma \vdash e_1 : \tau_1,\ eq_1 \quad \Gamma \vdash e_2 : \tau_2,\ eq_2}{\Gamma \vdash e_1\ e_2 : \alpha,\ \{\tau_1 = \tau_2 \to \alpha\} \cup eq_1 \cup eq_2}\ \alpha\ \texttt{variable fraîche}$$

$$IfThenElse\frac{\Gamma \vdash b : \tau,\ eq \quad \Gamma \vdash e_1 : \tau_1,\ eq_1 \quad \Gamma \vdash e_2 : \tau_2,\ eq_2}{\Gamma \vdash \texttt{if}\ b\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau_1,\ \{\tau = \texttt{Bool}\} \cup \{\tau_1 = \tau_2\} \cup eq \cup eq_1 \cup eq_2}$$

$$Let\ \frac{\Gamma \vdash e : \tau,\ eq \quad \{x : \tau\} :: \Gamma \vdash e' : \tau',\ eq'}{\Gamma \vdash \texttt{let}\ x\ \texttt{=}\ e\ \texttt{in}\ e' : \tau',\ eq \cup eq'}$$

$$Letrec\ \frac{\{x : \alpha\} :: \Gamma \vdash e : \tau,\ eq \quad \{x : \alpha\} :: \Gamma \vdash e' : \tau',\ eq'}{\Gamma \vdash \texttt{let rec}\ x\ \texttt{=}\ e\ \texttt{in}\ e' : \tau',\ \{\alpha = \tau\} \cup eq \cup eq'}\ \alpha\ \texttt{variable fraîche}$$

FIG. 3 – Règles de typage

1.3.2 Résolution du système d'équations

Le résultat d'un jugement de typage d'une expression MiniML complète est de la forme : $(\emptyset \vdash e : \tau,\ eq)$.

Le type τ contient des variables qui sont également présentes dans le système d'équations eq.

Ce système est une liste d'équations qui sont

1. soit de la forme (1) : $\alpha = \beta$, ou $\alpha = \tau$, ou $\tau = \alpha$,
2. soit de la forme (2) : $\tau = \tau'$,

avec α et β variables de types et $\tau,\ \tau' \in \{\texttt{Int}, \texttt{Bool}, \tau_1 \rightarrow \tau_2, \tau_1\ list\}$.

La résolution du système consiste à éliminer une à une les équations de la manière suivante :

- Une équation de la forme (1), est éliminée en remplaçant (substitution) la variable α par sa valeur dans les autres équations du système.
- Pour les équations du type (2) on élimine $Int = Int$ et $Bool = Bool$, on remplace $\tau_1 \rightarrow \tau_2 = \tau_3 \rightarrow \tau_4$ par $\tau_1 = \tau_3$ et $\tau_2 = \tau_4$, et on remplace $\tau_1\ list = \tau_2\ list$ par $\tau_1 = \tau_2$.
- Dans tous les autres cas ($Int = Bool$, $Int = \tau\ list$, $\tau\ list = \tau_1 \rightarrow \tau_2, \ldots$ le système n'a pas de solution.
- Le remplacement des variables dans le système d'équations (substitution) doit être fait simultanément sur le type global de l'expression à typer.

2 Travail demandé

Ce projet sera réalisé en monôme. À l'issue du projet, vous devrez rendre à l'aide de l'outil Moodle un fichier archive `votrenom.tar` contenant :

- un rapport résumant le travail effectué ;
- l'ensemble des sources Caml des modules à implanter.

Ce travail sera également testé pendant 10 minutes par un enseignant en votre présence.

2.1 Fournitures

Vous pourrez récupérer sur Moodle dans l'archive `miniml_distrib_etud.tar` les modules suivants :

- Le module `Asyn` qui contient le type *expr* qui est la description de la syntaxe abstraite (représentation interne) des programmes en MiniML.
- Le module `Miniml` qui contient la fonction $read\_expr() :\ unit \rightarrow\ expr$ qui lit sur le flot d'entrée (clavier), une expression MiniML se terminant par "`;;`" et renvoie sa représentation interne.
- Le module `Type_ml` qui contient le type *ct* définissant la syntaxe abstraite des types. Dans cette représentation les variables de type sont simplement numérotées sous la forme $Var\ n$. La fonction $string\_of\_type :\ ct \rightarrow\ string$, renvoie une chaîne de caractères correspondant à l'écriture habituelle d'un type en Caml.
- L'interface `typeur.mli` du module `Typeur`, que vous devrez réaliser, et un canevas `typeur.ml` de sa réalisation.
- Un makefile.
- Un fichier d'exemples de fonctions à typer `exemples.miniml`.

Vous devez impérativement ne modifier que le fichier `typeur.ml`, pour que la compilation puis le test soit possible.

Pour tester votre programme `typeur.ml` avec l'interprète, vous devez charger les modules fournis avec les commandes suivantes (le caractère # fait partie de la commande (doit être frappé en plus du # d'invite) :

```
#load "miniml_lexer.cmo";;
#load "miniml_parser.cmo";;
#load "miniml.cmo";;
#load "type_ml.cmo";;
```

Enfin il est conseillé d'expérimenter les fonctions *read_expr* et *string_of_type*, pour vous familiariser avec la représentation interne des fonctions et de types CAML.

2.2 Programmation

Vous devez impérativement suivre les indications contenues dans ce sujet. En particulier les contraintes sur le nom et le type des fonctions nous permettront de tester votre programme sur des jeux de tests prédéfinis.

Vous devez implanter l'algorithme de typage dans un style fonctionnel, i.e. sans références, sans tableaux, sans procédures.

Le choix de l'algorithme n'interviendra pas dans la notation, dès lors que les solutions adoptées sont justifiées et commentées. De plus, ces solutions devront être motivées par une exigence de clarté, d'exhaustivité et de concision en priorité absolue par rapport à un souci quelconque d'optimisation.

Le texte source devra bien sûr être clair, lisible, correctement indenté, intelligemment annoté, en respectant les règles évoquées en TD et en TP.

Les fichiers CAML rendus devront impérativement pouvoir être compilés même si les fonctions demandées ne sont pas totalement développées.

Lorsque votre programme fonctionnera ou plutôt **semblera** fonctionner, il est important que vous le testiez à l'aide de jeux de tests significatifs, qui doivent montrer le bon fonctionnement de votre programme dans tous les cas que vous jugerez utile de distinguer. Ces tests devront bien sûr être joints au rapport.

2.3 Rapport et Tests

Le travail de conception et de programmation effectué devra figurer *in extenso* dans le rapport de projet à rendre. Vous devrez clairement détailler les découpages en fonctions et les algorithmes de votre programme, expliciter et justifier les choix de conception. Le principe est d'expliquer suffisamment clairement les algorithmes, de manière à pouvoir comprendre le programme sans avoir à lire le listing.

Votre programme sera soumis à une batterie de tests semi-automatiques qui permettront de juger sa correctio par rapport à l'objectif du projet. Vous devrez également proposer vous-même à cette occasion des jeux de tests destinés à prouver la correction de votre application. Enfin vous devrez être capables de répondre aux questions de l'enseignant de façon claire et synthétique.

2.4 Les Dates à retenir

Remise du source et du rapport	:	mardi 11 décembre 2008 à 18h
Tests	:	mercredi 12 décembre 2008 après-midi