Compilation
Master Informatique M1 2006-2007

Examen Partiel

24 octobre 2006

Les deux exercices sont indépendants. L'énoncé est composé de 3 pages. L'examen dure 3 heures. Les notes de cours et de TD manuscrites ainsi que les supports de cours distribués cette année sont les seuls documents autorisés.

1 Analyse Syntaxique (6 pts)

1. Soit la grammaire G_1 définie sur l'ensemble des terminaux $\{\mathtt{0}, \mathtt{1}, \mathtt{,}\}$.

$$\boxed{\begin{array}{l} S ::= T\,\mathtt{,}\,X\,T \\ T ::= \epsilon \\ T ::= X\,T \\ X ::= \mathtt{1} \\ X ::= \mathtt{0} \end{array}}$$

2. (0.5pt) Quel est le langage engendré par cette grammaire?
3. (1.5pt) On ajoute la règle $S' ::= S\sharp$ où $\sharp$ est un marqueur de fin de chaîne. Cette grammaire est-elle LL(1)? Une réponse sans justification ne donne pas de point.
4. (2pt) Cette grammaire est-elle LR(0), SLR(1), LR(1)?
5. (1pt) Définir un automate d'analyse ascendante.
6. (1pt) A l'aide de cet automate, analyser le mot `1,1`$\sharp$ en indiquant l'état courant de l'automate, le contenu de la pile et l'état de la chaîne d'entrée.

2 Analyse Lexicale (5 pts)

1. (1.5pt) Écrire l'expression régulière décrivant les adresses électroniques sous la forme suivante:

   `Nom.Prenom@site.extension`

   `Nom` et `Prenom` désignent des chaînes alphanumériques commençant par une majuscule. `site` désigne une suite de caractères pouvant contenir l'un des caractères spéciaux suivants: `-`, `_`. Enfin, `extension` est une suite de 2 ou 3 caractères minuscules.
2. (2.5pt) Écrire l'expression régulière décrivant les nombres flottants dans un langage de programmation.

   Un flottant est un entier suivi éventuellement d'un point et d'un entier. Ce nombre peut éventuellement être suivi d'un exposant qui s'écrit `e` ou `E` suivi éventuellement d'un signe `+` ou `-` plus d'un entier.

   Ainsi, les nombres suivants sont syntaxiquement corrects: `10`, `10.4`, `10.4e10` `10.4E5`, `10E-45`.
3. (1pt) Construire l'automate d'état fini déterministe permettant de reconnaître les mots sur $\{0, 1\}$ contenant un nombre pair de 0 et un nombre pair de 1.

3 Calcul de tailles pour les entiers (9pts)

On considère un petit langage de programmation dans lequel on peut manipuler des entiers signés 8 bits et 16 bits. L'objectif de cet exercice est de définir un système de types permettant de vérifier que les expressions calculées restent toujours représentables sur 8 ou 16 bits.

La syntaxe concrète de ce langage est définie ci-dessous. e désigne la classe des expressions, c la classe des comparaisons, x la classe des identificateurs et i, celle des constantes entières. On introduit le type t des entiers 8 bits et 16 bits.

$$e \ ::= \ x \mid i \mid \texttt{let}\, x{:}t{=}e\, \texttt{in}\, e \mid \texttt{if}\, c\, \texttt{then}\, e\, \texttt{else}\, e \mid e{+}e \mid (e)$$

$$c \ ::= \ e{\leq}e \mid e{\geq}e$$

$$t \ ::= \ \texttt{int8} \mid \texttt{int16}$$

La construction `let/in` permet d'imposer un type à un identificateur. `let x: int16 = -25 in x + 1`, par exemple, signifie que `x` est représenté par un entier 16 bits (alors qu'un entier 8 bits est suffisant pour représenter -25. Certains programmes pourront donc être mal typés comme par exemple `let x:int8 = 10000 in x + 1`.

On définit les types de données Caml suivants:

```
type typ = Tint8 | Tint16

type exp =
    | Evar of string
    | Eint of int
    | Elet of string * typ * exp * exp
    | Eite of comp * exp * exp
    | Eplus of exp * exp
and comp =
    | Ecomp of rel * exp * exp
and rel = Einf | Esup
```

1. (1pt) Indiquer, parmi les deux programmes suivants, ceux qui sont bien typés. Vous justifierez votre réponse.

```
let x:int8 = -25 in
let y:int16 = 1 in
if x>y then x+65 else x+256
```

```
let x:int8 = -25 in
let y:int16 = 1 in
let z:int8 = if x>y then x + 65 else x + 100 in
z + (let z:int8 = z + z in z + y)
```

2. (0.5pt) On définit un système de types permettant de vérifier que les expressions calculées restent toujours représentable sur 8 ou 16 bits. On introduit pour cela le type $t[i,s]$ où t désigne un type (`int8` ou `int16`), i et s sont deux entiers. Une expression e de type $t[i,s]$ signifie que sa valeur est représentable par un entier de type t et qu'elle est comprise entre les entiers i et s. On notera τ, un type de la forme $t[i,s]$. On définit la relation $\leq$ telle que $t \leq t$ (pour $t \in \{\texttt{int8}, \texttt{int16}\}$) et $\texttt{int8} \leq \texttt{int16}$.
On notera $H \vdash e : \tau$ le jugement de typage qui signifie que l'expression e est de type τ dans l'environnement H. On notera $H(x)$ le type associé à x dans l'environnement H et $H + x : t$ l'environnement H augmenté de la liaison $x : t$. Les règles de typage sont données ci-dessous.

$$\frac{H(x) = \tau}{H \vdash x : \tau} \qquad \frac{\texttt{repr}(i) = t}{H \vdash i : t[i, i]}$$

$$\frac{H \vdash e_1 : t[i_1, s_1] \quad t \leq t_1 \quad H + x : t_1[i_1, s_1] \vdash e_2 : \tau_2}{H \vdash \texttt{let}\, x : t_1 = e_1 \,\texttt{in}\, e_2 : \tau_2}$$

$$\frac{H \vdash c \quad H \vdash e_1 : \tau_1 \quad H \vdash e_2 : \tau_2 \quad \tau = \texttt{sup}(\tau_1, \tau_2)}{H \vdash \texttt{if}\, c \,\texttt{then}\, e_1 \,\texttt{else}\, e_2 : \tau}$$

$$\frac{H \vdash e_1 : \tau_1 \quad H \vdash e_2 : \tau_2 \quad \tau = \texttt{add}(\tau_1, \tau_2)}{H \vdash e_1 + e_2 : \tau}$$

Le jugement de typage $H \vdash c$ signifie que la comparaison c est bien typée. La règle de vérification du type des comparaisons est donnée ci-dessous.

$$\frac{H \vdash e_1 : \tau_1 \quad H \vdash e_2 : \tau_2 \quad \tau = \texttt{sup}(\tau_1, \tau_2)}{H \vdash c}$$

Dans ce système, $\texttt{repr}(i)$ définit le plus petit type nécessaire pour représenter la valeur entière i, s'il existe. $\texttt{sup}(\tau_1, \tau_2)$ retourne un nouveau type τ permettant de représenter à la fois les valeurs de τ_1 et de τ_2. $\texttt{add}(\tau_1, \tau_2)$ retourne un type τ permettant de représenter la somme de deux valeurs de type τ_1 et τ_2.

Écrire une fonction `repr: int -> typ` telle que `repr x` retourne le plus petit type (au sens de $\leq$) permettant de représenter l'entier `x`. On rappelle que les entiers sur 8 bits sont compris entre -128 et 127 et ceux sur 16 bits entre -32768 et 32767. La fonction `repr` lève l'exception `NoRepr` si l'argument `x` n'est pas représentable par un entier 16 bits.

3. (1pt) Le type τ est représenté par le type Caml suivant:

   ```
   type tau = { a_typ: typ; a_inf: int; a_sup: int }
   ```

   Ainsi, une valeur v est de type `tau = { a_typ = t; a_inf = i; a_sup = s }` lorsqu'elle peut être représentée par un entier de taille `t` et qu'elle est comprise entre `i` et `s`.

   Écrire le code de la fonction `sup: tau -> tau -> tau`.
4. (1pt) Écrire le code de la fonction `add: tau -> tau -> tau`.
5. (0.5pt) Proposer un type `env` permettant de représenter l'environnement de typage H utilisé dans les règles d'inférence. Définir les fonctions d'accès et d'ajout nécessaires.
6. (2.5pt) Écrire, en suivant les règles d'inférence, une fonction de typage `type_expr: env -> exp -> tau` qui retourne le type d'une expression et une fonction `type_comp: env -> comp -> bool` qui retourne la valeur vraie si la comparaison est bien typée et la valeur faux sinon.

   Proposer un traitement d'erreur permettant de distinguer les trois cas suivants:
   – Identificateur inconnu
   – Erreur de représentation
   – Conversion impossible
7. (0.5pt) On souhaite maintenant traduire une expression de ce langage en une autre expression dans laquelle les points de conversions (8 bits vers 16 bits) sont indiqués.

   On étend le type `exp` avec un nouveau constructeur permettant d'indiquer les points de conversion (8 bits vers 16 bits) nécessaires.

```
type exp = ...
       | Ecast of exp
```

Indiquer les points de conversion nécessaires sur le programme suivant:

```
let x:int8 = -25 in
let y:int16 = 1 + x in
let z:int8 = if x>y then x + 65 else x + 100 in
z + (y + x)
```

8. (2pt) Écrire, en vous inspirant de la fonction `type_expr`, une fonction
`typer_et_convertir: env -> exp -> tau * exp`
telle que `typer_et_convertir env e` retourne le type de `e` et une nouvelle expression dans laquelle le constructeur `Ecast` a été ajouté pour indiquer les points de conversion nécessaires.