Compilation
Maîtrise d'Informatique 2000-2001

Examen Partiel
26 octobre 2000

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

1 Analyse descendante et ascendante (4 pts)

On se donne la grammaire suivante avec comme ensemble de symboles terminaux {`(`, `a`, `;`, `)`}, comme ensemble de symboles non-terminaux $\{S, L, A\}$ et S le symbole de départ:

```
S  := (L
S  := a
L  := SA
A  := ;SA
A  := )
```

1. Donner les étapes de l'analyse descendante du mot `(a;(a;a))`.
2. Donner les étapes de l'analyse ascendante du mot `(a;(a;a))`.
3. En déduire les dérivations gauche et droite de ce mot.

2 Reconnaissance d'une chaîne de caractères par une expression régulière (9 pts)

Dans cet exercice, on se propose d'écrire une fonction de test d'appartenance d'un mot au langage $L(r)$ défini par une expression régulière r. Les expressions régulières seront représentées par des objets du type Caml suivant:

```
type  expreg =
      | Vide                              (* Langage vide ∅ *)
      | Epsilon                           (* Mot vide ϵ *)
      | Caractere  of char                (* Caractère c *)
      | Union      of expreg * expreg     (* r1|r2 *)
      | Produit    of expreg * expreg     (* r1r2 *)
      | Etoile     of expreg              (* r* *)
```

1. Définir une fonction `contient_epsilon` : `expreg` → `bool` qui détermine si le mot vide ϵ appartient au langage défini par une expression régulière donnée.
2. On définit le *résidu* d'une expression régulière r par rapport à un caractère c de la manière suivante :
$$Res(r, c) = \{\ w \mid cw \in L(r)\ \}$$
   (a) Calculer $Res(r, c)$ dans le cas où $r = (a|b)^*a$ et $c \in \{a, b\}$.
   (b) Le langage $Res(r, c)$ peut lui-même être décrit par une expression régulière. Écrire une fonction `residu` : `expreg` → `char` → `expreg` calculant à partir de r et de c une expression régulière r' telle que $L(r') = Res(r, c)$.
   (c) Calculer `residu` r c dans le cas où $r = (a|b)^*a$ et $c \in \{a, b\}$.

3. En déduire une fonction `reconnait : expreg` $\rightarrow$ `char list` $\rightarrow$ `bool` qui détermine si une liste de caractères appartient au langage défini par une expression régulière donnée.
4. Appliquer cette fonction à la reconnaissance du mot aba par l'expression régulière $r = (a|b) * a$

3 Analyse lexicale (7 pts)

On se propose de documenter les programmes écrits dans le langage SCALPA. Pour cela on souhaite créer un index de tous les noms de fonctions, de variables et de paramètres apparaissant dans le programme. L'index doit permettre de trouver pour chaque identificateur du programme, les lignes du programme où il est déclaré (comme nom de fonction, de variable ou comme paramètre de procédure) et les lignes du programme où il est utilisé. On ne cherche pas à distinguer plusieurs déclarations du même identificateur.

Ainsi pour le programme :

```
program test
var n : int
function fib(i:int,j:int):int
begin if n=0 then return i
      else if n=1 then return j
      else begin n:=n-1; fib(j,i+j) end
end;
function fact(i:int):int
var j:int
begin if n=0 then i
      else j:=n*i; n:=n-1; fact(j)
end;
begin n:=10; n:=fib(0,1); n:=fact(1) end
```

On souhaite obtenir les tables de déclarations et d'utilisations suivantes (éventuellement dans un ordre différent) :

Déclarations	
n	2
fib	3
i	3, 8
j	3, 9
$fact$	8

Utilisations	
n	4, 5, 6, 10, 11, 13
fib	6, 13
i	4, 6, 10, 11
j	5, 6, 11
$fact$	11, 13

On suppose que l'on dispose d'un type de données (impératif) `table` qui nous permet de stocker les informations contenues dans les index. Les fonctions de base sur ces tables sont :

`init`	: `unit` $\rightarrow$ `table`	`init` *()* crée une nouvelle table.
`ajoute`	: `table` $\rightarrow$ `string` $\rightarrow$ `int` $\rightarrow$ `unit`	`ajoute` *tbl nom n* ajoute la ligne numéro *n* pour l'entrée *nom* dans la table *tbl*.
`imprime`	: `table` $\rightarrow$ `string` $\rightarrow$ `unit`	`imprime` *tbl titre* imprime la table *tbl* avec le titre *titre*.

Syntaxe de SCALPA Nous donnons ci-dessous la grammaire complète du langage SCALPA. Dans cette grammaire, les mots-clés et les terminaux apparaissent en fonte droite. Le terminal `cte` représente les constantes entières et booléennes (`true` et `false`); le terminal `opb` (resp. `opu`) représente les opérateurs binaires (resp. unaires) du langage; le terminal `ident` représente les identificateurs qui sont formés d'un caractère alphabétique suivi d'une suite de caractères alphanumériques ou d'une apostrophe ou du caractère de soulignement. Les commentaires débutent par les deux caractères `(*` et se terminent par les deux caractères `*)`, ils peuvent être imbriqués.

```
program      := program ident vardecllist fundecllist instr
vardecllist  := ε | varsdecl | varsdecl ; vardecllist
varsdecl     := var identlist : typename
identlist    := ident | ident , identlist
typename     := atomictype | arraytype
identlist    := ident | ident , identlist
atomictype   := unit | bool | int
arraytype    := array [ rangelist ] of atomictype
rangelist    := int .. int | int .. int , rangelist
fundecllist  := ε | fundecl ; fundecllist
fundecl      := function ident ( arglist ) : atomictype vardecllist instr
arglist      := ε | arg | arg , arglist
arg          := ident : typename | ref ident : typename
instr        := if expr then instr | if expr then instr else instr
              | while expr do instr | lvalue := expr | return expr | return
              | ident ( exprlist ) | begin sequence end | begin end
sequence     := instr ; sequence | instr ; | instr
lvalue       := ident | ident [ exprlist ]
exprlist     := expr | expr , exprlist
expr         := cte | ( expr ) | expr opb expr | opu expr
              | ident ( exprlist ) | ident [ exprlist ] | ident
```

Questions

1. Donner un programme `ocamllex` qui définit une fonction `index` qui étant donné un fichier contenant un programme SCALPA correct, construit et imprime l'index correspondant.
   Indications : On remarquera que pour résoudre ce problème, il n'est pas nécessaire de connaître la grammaire complète de SCALPA, seule la forme syntaxique des déclarations et la liste des mots-clé est importante.
   La notation portera sur l'explication de la démarche et la spécification des différentes composantes plus que sur la précision syntaxique du code ocamllex.
2. Peut-on modifier le programme ocamllex pour construire trois tables différentes, l'une pour les déclarations de fonctions, l'autre pour les déclarations de variables et la troisième pour les déclarations de paramètres ? Peut-on faire la même chose pour les tables d'utilisation ? Comment ?
3. Peut-on modifier le programme ocamllex pour associer chaque utilisation d'un identificateur à la déclaration correspondante ? Pourquoi ?