Université de Marne-la-Vallée. Mars 1999
Examen de Logique. Licence et IUP d'informatique
Durée 2 heures. Tous les documents sont autorisés.

Tous les exercices sont indépendants.

Exercice 1.— Indiquer si les formules suivantes sont des tautologies, des antilogies ou ni l'un ni l'autre.

a) $((p \leftrightarrow q) \leftrightarrow p) \leftrightarrow (\neg q)$

b) $((p \wedge q) \rightarrow (r \wedge s)) \rightarrow (p \rightarrow r)$

Exercice 2.— Soit $\mathcal{A}$ un ensemble quelconque de propositions. Soient A, B deux propositions. Montrer que si $\mathcal{A} \cup \{A\} \models B$ alors $\mathcal{A} \cup \{\neg B\} \models \neg A$. On fera une preuve uniquement sémantique en utilisant les définitions et les interprétations.

Exercice 3.— Si A, B, C désignent des propositions, montrer que :

a) $\vdash ((A \vee C) \wedge ((B \vee \neg C)) \rightarrow (A \vee B)$

b) $\vdash A \rightarrow ((A \wedge \neg B) \vee (A \wedge B))$

c) $\vdash (A \rightarrow B) \rightarrow ((A \rightarrow B) \rightarrow B \rightarrow C) \rightarrow A \rightarrow C$

Le système de déduction à utiliser est le système de Hilbert présenté en cours. On pourra utiliser le fait que $A \vee B$ est une abréviation de $\neg A \rightarrow B$. On pourra utiliser tous les résultats du cours (notamment le théorème de déduction), à l'exclusion toutefois du théorème de complétude.
Définir une fonction Caml de nom `toto` du type :

```
('a ->'b) -> (('a ->'b) ->'b ->'c) -> 'a ->'c
```

Exercice 4.— On note $\uparrow$ l'opérateur binaire tel que $p \uparrow q$ a même valeur sémantique que $\neg(p \wedge q)$. Sachant que $\{\neg, \vee, \wedge\}$ est sémantiquement complet, montrer que $\{\uparrow\}$ est sémantiquement complet.

Exercice 5.— On définit en Caml le type suivant `'a arbre` qui permet de manipuler des arbres binaires dont les étiquettes des noeuds sont formés d'éléments de type `'a`.

```
type 'a arbre =
    Vide
  | Noeud of 'a * 'a arbre * 'a arbre ;;
```

a) Ecrire une fonction `eclate` qui, étant donné un arbre t dont les étiquettes des noeuds sont des couples, construit un couple d'arbres (t_1, t_2) tels que t_1 (resp. t_2) est obtenu à partir de t en masquant, sur chaque noeud, les deuxièmes (resp. premières) composantes des couples. Le type de `eclate` sera :

```
('a * 'b) arbre -> 'a arbre * 'b arbre
```

b) Ecrire une fonction `renverse` qui prend an argument un arbre dont les étiquettes sont des couples et renvoie l'arbre obtenu à partir de celui-ci en renversant tous les couples. Le type de `renverse` sera :

```
('a * 'b) arbre -> ('b * 'a) arbre
```

c) Ecrire une fonctionnelle `map_tree` de type

```
('a -> 'b) -> ('a arbre) -> ('b arbre)
```

qui prend en argument une fonction f de type `'a -> 'b` et un arbre t de type `'a arbre` et renvoie un arbre de type `'b arbre` obtenu en appliquant f sur chaque étiquette d'un noeud de t.
Réécrire `renverse` en utilisant `map_tree`.

Exercice 6.— Effectuer une résolution restreinte et ordonnée de l'ensemble de clauses :

– $\Delta = \{pqs, \bar{p}qs, \bar{q}s, \bar{q}\bar{r}, pr\bar{s}, \bar{p}r\bar{s}\}$

On prendra comme ordre sur les variables : $p < q < r < s$. On rappelle la notation pour les clauses : $\bar{p}qs$ désigne la clause $\neg p \vee q \vee s$.
Que peut-on dire de Δ ?

Exercice 7.— QUESTION SUBSIDIAIRE (ne traitez cette question que s'il vous reste du temps)
Montrer que le système de connecteurs $\{\perp, \leftrightarrow\}$ n'est pas fonctionnellement complet.