Corrigé de logique formelle et calculabilité

Logique

1.

$$(p \vee p) \rightarrow p \tag{1}$$
$$q \rightarrow (p \vee q) \tag{2}$$
$$(p \vee q) \rightarrow (q \vee p) \tag{3}$$
$$(q \rightarrow r) \rightarrow ((p \vee q) \rightarrow (p \vee r)) \tag{4}$$

Pour montrer qu'un système de preuve est correct, il s'agit de montrer que tous les théorèmes que l'on peut prouver avec sont des tautologies. D'après la définition d'une preuve avec PM, il s'agit donc de montrer que

(a) les axiomes sont des tautologies ;

(b) la règles d'inférence est correcte, i.e. conclut sur une tautologie si les prémisses sont des tautologies.

Ces démonstrations peuvent se faire en utilisant une table de vérité (ou un autre système de preuve que l'on sait correct). Par exemple pour l'axiome 3, on obtient :

p	q	$p \vee q$	$q \vee p$	$(p \vee q) \rightarrow (q \vee p)$
0	0	0	0	1
0	1	1	1	1
1	0	1	1	1
1	1	1	1	1

De même pour les autres axiomes.

Pour la règle d'inférences, on procède semblablement :

A	B	$A \rightarrow B$	$A \wedge (A \rightarrow B)$
0	0	1	0
0	1	1	0
1	0	0	0
1	1	1	1

On a bien B vrai quand A et $A \rightarrow B$ sont vrais.

2.

$$a \rightarrow (b \rightarrow c) \tag{5}$$
$$a \rightarrow (d \rightarrow b) \tag{6}$$
$$a \tag{7}$$
$$d \tag{8}$$

Résolution : les formules 5, 6, 7 et 8 sont des clauses. On considère la négation de la conclusion :

$$\neg c \tag{9}$$

Une solution :

$$\begin{aligned} 5,9 &\rightsquigarrow \neg a \vee \neg b && (10)\\ 10,7 &\rightsquigarrow \neg b && (11)\\ 11,6 &\rightsquigarrow \neg a \vee \neg d && (12)\\ 12,7 &\rightsquigarrow \neg d && (13)\\ 13,8 &\rightsquigarrow \square && (14) \end{aligned}$$

Calcul des séquents

$$\dfrac{\dfrac{\dfrac{\dfrac{}{7, \underline{8} \longrightarrow \underline{d}}}{\underline{d \rightarrow b}, 7, 8 \longrightarrow b}\rightarrow'_G \quad \dfrac{}{\underline{7}, 8 \longrightarrow \underline{a}, b}}{\underline{6}, 7, 8 \longrightarrow b}\rightarrow_G \quad \dfrac{}{\underline{c}, 6, 7, 8 \longrightarrow \underline{c}}}{\underline{b \rightarrow c}, 6, 7, 8 \longrightarrow c}\rightarrow_G \qquad \dfrac{}{6, \underline{7}, 8 \longrightarrow \underline{a}, c}}{\underline{5}, 6, 7, 8 \longrightarrow c}\rightarrow_G$$

3. On peut effectuer une preuve par résolution. On considère la forme clausales des hypothèses :

$$\begin{aligned} p(x) \vee p(i(x,y)) && (15)\\ \neg p(y) \vee p(i(x,y)) && (16)\\ (p(i(x,y)) \wedge p(x)) \rightarrow p(y) && (17) \end{aligned}$$

ainsi que la négation de la conclusion :

$$\neg p(i(a, i(i(a,b),b))) \quad (18)$$

Ce qui donne :

$$\begin{aligned} 18,15 &\rightsquigarrow p(a) && (19)\\ 18,16 &\rightsquigarrow \neg p(i(i(a,b),b)) && (20)\\ 20,16 &\rightsquigarrow \neg p(b) && (21)\\ 20,15 &\rightsquigarrow p(i(a,b)) && (22)\\ 22,17 &\rightsquigarrow \neg p(a) \vee p(b) && (23)\\ 23,19 &\rightsquigarrow p(b) && (24)\\ 24,21 &\rightsquigarrow \square && (25) \end{aligned}$$

Calculabilité

1. En itérant n fois (ce que fait l'entier de Church n) la transformation proposée :

$$\begin{aligned} < FALSE, n > &\rightsquigarrow < TRUE, n >\\ < TRUE, n > &\rightsquigarrow < FALSE, n+1 > \end{aligned}$$

en partant de la paire $< FALSE, \overline{0} >$ on obtient la paire $< FALSE, n/2 >$ si n et pair et $< TRUE, (n-1)/2 >$ si n est impair.
On utilise donc les combinateurs suivants :

$$\begin{aligned} \overline{0} &= \lambda fx.x && (26)\\ \overline{+1} &= \lambda nfx.(n\ f\ (f\ x)) && (27)\\ TRUE &= \lambda xy.x && (28) \end{aligned}$$

$$
\begin{aligned}
FALSE &= \lambda xy.y && (29)\\
IF &= \lambda ite.(i\ t\ e) && (30)\\
< A, B > &= \lambda z.(z\ A\ B) && (31)\\
FST &= \lambda p.(p\ \lambda xy.x) && (32)\\
SND &= \lambda p.(p\ \lambda xy.y) && (33)\\
STEP2 &= \lambda bn.(IF\ b\ < FALSE, (\overline{+1}\ n) > < TRUE, n >) && (34)\\
STEP &= \lambda p.(STEP2\ (FST\ p)\ (SND\ p)) && (35)\\
DIV2 &= \lambda n.(SND\ (n\ STEP\ < FALSE, \overline{0} >)) && (36)
\end{aligned}
$$

2. `let rec goedel = fun n f x -> if n = 0 then x else f n (goedel (n-1) f x);;`

```
let rec fact1 = fun n ->
  if n = 0 then 1 else n * fact1 (n-1);;
let fact2 = fun n ->
  goedel n (fun x y -> x * y) 1;;
let fact3 = fun n ->
  let f = ref 1 in
  for i = 2 to n do f := !f * i done;
  !f;;
let fact4 = fun n ->
  let f = ref 1 and i = ref 2 in
  while !i <= n do
    f := !f * !i;
    incr i
  done;
  !f;;
```

Ce qui nous importe du point de vue de la calculabilité, c'est le fait que les calculs terminent et que la preuve de terminaison soit aisée.

La fonction idéale de ce point de vue est `fact3` qui utilise une boucle bornée, construction terminante par excellence (attention ce n'est pas le cas d'une boucle `for` de C).

Les deux fonctions `fact1` et `fact2` sont semblables, récursives. La preuve de terminaison pour ces deux versions s'effectue en utilisant le théorème de récurrence classique. La version `fact2` est préférable puisqu'elle utilise le combinateur général `goedel` : il suffit de prouver une fois pour toutes que ce combinateur termine.

La fonction `fact4` est la moins bonne : la preuve de terminaison d'une boucle `while` n'est pas aisée ; il faut avoir recours à une notion d'état (la valeur de `i`) qu'il faut faire évoluer. La terminaison n'est pas « claire » car la valeur de `i` peut être modifiée n'importe où dans le corps du `while` ; dans une fonction récursive seul la valeur dans l'appel récursif (`(n-1)`) importe.