NFP120 - Examen 1ère session 2011

1 Prolog

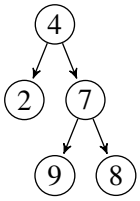
Rappels

- *Les entiers* dans cet examen sont les entiers de Prolog (pas les entiers de Peano) et il ne sera pas pénalisant d'oublier d'utiliser le prédicat **`is`**.
- *Les arbres binaires* sont représentés comme en cours, c'est-à-dire comme suit :
  - le terme `feuille(N)` représente l'arbre à un seul noeud contenant l'entier `N` ;
  - le terme `t(N,X,Y)` représente l'arbre ayant l'entier `N` à la racine et les arbres `X` et `Y` comme sous-arbres.

Par exemple le terme :

```
t(4
  ,feuille(2)
  ,t(7
     ,feuille(9)
     ,feuille(8)))
```

représente l'arbre suivant :



Ex. 1 — (0,5pt) Écrire le prédicat `est_un(X)` qui est vrai si `X` est égal à 1.

Ex. 2 — (1pt) Écrire le prédicat `est_pair(X)` qui est vrai si `X` est un entier pair (positif ou négatif). Un demi point supplémentaire sera accordé si le prédicat ne boucle pas indéfiniment sur les entiers impairs.

Solution (Ex. 2) —

```
est_pair(0).
est_pair(X):-X>0,Y is X-2,est_pair(Y).
est_pair(X):-X<0,Y is X+2,est_pair(Y).
```

Ex. 3 — (1pt) Écrire le prédicat `supl(L,X)` qui est vrai si `X` est plus grand ou égal à tous les éléments de la liste d'entiers `L`.

Solution (Ex. 3) —

```
supl([],X).
supl([X|L],Y):-Y>=X,supl(L,Y).
```

Ex. 4 — (2pt) Écrire le prédicat `aumoins(X,N,L)` qui est vrai si l'entier `X` apparaît *au moins* `N` fois dans la liste d'entiers `L`.

Solution (Ex. 4) —

```
aumoins(_,N,_):-N<=0.
aumoins(X,N,[X|L]):-M is N-1,aumoins(X,M,L).
aumoins(X,N,[Y|L]):-aumoins(X,N,L).
```

Ex. 5 — (1,5pt) Écrire le prédicat `rempli(X,A)` qui est vrai si toutes les valeurs (aux feuilles et aux noeuds) de l'arbre binaire `A` sont égales à `N`.

Solution (Ex. 5) —

```
rempli(N,feuille(N)).
rempli(N,t(N,FG,FD)):-rempli(N,FG),rempli(N,FD).
```

Ex. 6 — (1,5pt) Écrire le prédicat `toutchemin(N,A)` qui est vrai si toutes les branches de l'arbre binaire `A` contiennent au moins une occurrence (dans une feuille ou dans un noeud) de l'entier `N`. Une branche est un chemin de la racine jusqu'à une feuille. Dans les exemples ci-dessous `toutchemin(3,t1)` est vrai mais `toutchemin(3,t2)` n'est pas vrai car il existe une branche (la branche (4,2,8)) qui ne contient pas la valeur 3 :

Solution (Ex. 6) —

```
toutchemin(N,feuille(N)).
toutchemin(N,t(N,_,_)).
toutchemin(N,t(M,FG,FD)):-toutchemin(N,FG),toutchemin(N,FD).
```

Ex. 7 — (1pt) Soit le programme `appartient` suivant :

```
appartient(X,[X|_]).                    /* Règle 1 */
appartient(X,[Y|L]):-appartient(X,L).   /* Règle 2 */
```

Donnez l'arbre de résolution de Prolog sur la requête suivante (vous vous arrêterez au premier succès) :

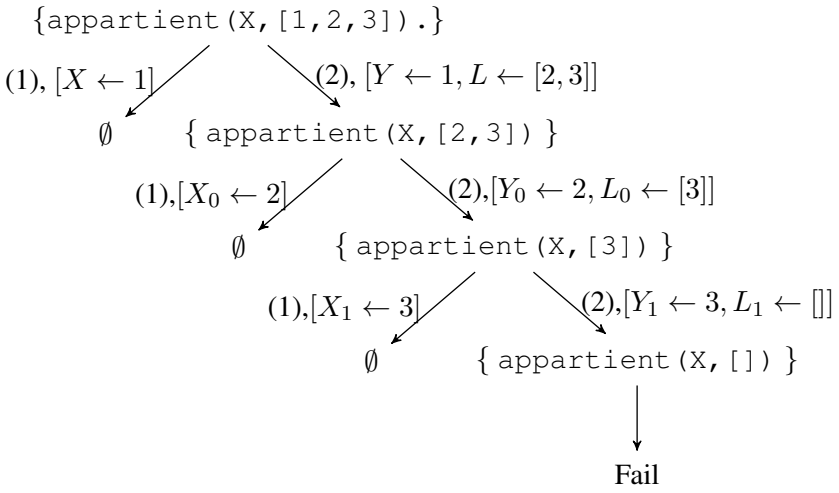
```
?-appartient(2,[1,2,3]).
```

Ex. 8 — (2pt) Sur le même programme `appartient` que la question précédente donnez l'arbre *complet* de résolution de Prolog sur la requête suivante (ne pas s'arrêter après le premier succès, continuer jusqu'à épuisement des règles) :

```
?-appartient(X,[1,2]).
```

Signalez les noeuds aboutissant à des échecs et à des solutions et vous donnez la solution dans ce dernier cas.

Solution (Ex. 8) — pour la requête suivante (un élément de plus) `?-appartient(X,[1,2])` :



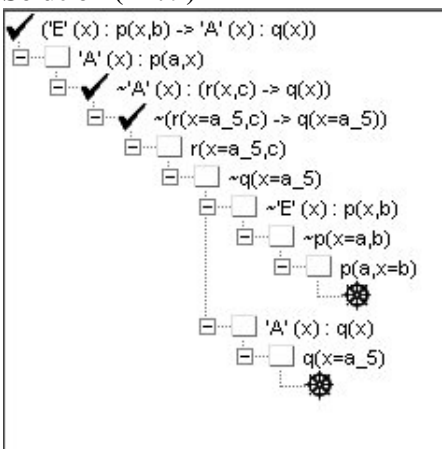
2 Tableaux sémantiques

Vous trouverez un résumé des règles de résolution des tableaux dans l'annexe A.

Ex. 9 — (2pt) Démontrez à l'aide de la méthode des tableaux la propriété suivante :

$$\exists x, p(x,b) \rightarrow \forall x : q(x)\ ,\ \forall x : p(a,x)\ \vdash\ \forall x : (r(x,c) \rightarrow q(x))$$

Solution (Ex. 9) —



3 Logique de Hoare

3.1 Invariant 1

Soit la boucle B suivante :

```
while i < max do
  if t[i] <= u[i]
    then v[i] := u[i]
    else v[i] := t[i];
  i := i + 1
done
```

Ex. 10 — (0,5pt) Donnez un variant de cette boucle.

Solution (Ex. 10) — $max - i$

Ex. 11 — (2pt) Donnez un invariant de cette boucle permettant de prouver le triplet de Hoare suivant (dans lequel n_0 et a_0 sont des constantes) :

$$\{\texttt{i} = i_0 \wedge i >= 0\}\ B\ \{\forall a, i_0 \leq a < max \rightarrow ((v[a] = t[a]) \vee (v[a] = u[a]))\}.$$

On supposera que les bornes des tableaux sont suffisantes.

Solution (Ex. 11) — $\forall j, i_0 \leq j < \texttt{i} \rightarrow (\texttt{v}[i] = \texttt{t}[i] \vee \texttt{v}[i] = \texttt{u}[i])$

3.2 Invariant 2

Soit la boucle C suivante, qui correspond à la boucle interne du tri à bulle : un seul parcours du tableau de gauche à droite, à chaque case `t[i]` permutation avec la case précédente `t[i-1]` si celle-ci contient un entier plus petit que `t[i]` (notez qu'il faut commencer à la case `t[1]` et non à la case `t[0]`).

```
while(i<limite) do
  if t[i] < t[i-1] then
    begin
      tmp := t[i];
      t[i] := t[i-1];
      t[i-1] := tmp;
    end
  i := i + 1
done
```

Ex. 12 — (0,5pt) Donnez un variant de cette boucle.

Solution (Ex. 12) — `limite - i`

Ex. 13 — (1,5pt) Donnez un triplet de Hoare de la forme `{P}` C `{Q}` exprimant la propriété suivante : à la fin de la boucle le plus grand élément du tableau initial est placé à la dernière case. Il est permis de fixer la valeur initial de `i` à 1.
Remarque : Utilisez la notation $\texttt{t}_0\texttt{[n]}$ pour désigner la $\texttt{n}^{\text{ème}}$ case du tableau `t` avant la boucle.

Solution (Ex. 13) — En ne fixant pas la valeur de `i` à 1 mais à i_0 :

`{i=`i_0 $\wedge$ `t=`t_0`}` `C` `{` $\forall k, i_0 \leq k <$ `limite` $\rightarrow$ `t[`k`]`$\leq$`t`$_0$`[limite-1]` `}`

Ex. 14 — (2,5pt) Donnez un invariant de la boucle suffisant pour en déduire la post-condition `Q` à la sortie de la boucle. Justifiez en expliquant comment déduire, à partir de l'invariant et de la condition de sortie de la boucle, que la post-condition est vérifiée après la boucle.

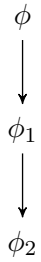
Solution (Ex. 14) — Attention : la case `i` n'est pas encore traitée au début du corps de la boucle. On n'oublie pas la condition sur `i` qui permettra de dire que `i` sera égal à `limite` à la sortie de la boucle.

$$\forall 0 \leq j < \texttt{i}, \texttt{t}[j] \leq t[\texttt{i} - 1] \wedge i \leq \texttt{limite}.$$

Ex. 15 — (1,5pt) Si l'objectif est de démontrer la correction de l'algorithme de tri à bulle (qui nécessite d'imbriquer la boucle C dans une autre boucle), quelle propriété faut-il ajouter à l'invariant de C ? Vous pouvez décrire cette propriété au choix en français où sous la forme d'une propriété logique.

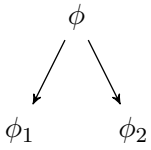
Solution (Ex. 15) — Il faut ajouter la propriété précisant que les éléments du tableau final sont les mêmes que ceux du tableau initial, en même quantité. Bref que le tableau final est une permutation du tableau initial.

A Annexe : Règle de la méthode des tableaux



ϕ	ϕ_1	ϕ_2
$\neg\neg\phi_1$	ϕ_1	
$\phi_1 \wedge \phi_2$	ϕ_1	ϕ_2
$\neg(\phi_1 \vee \phi_2)$	$\neg\phi_1$	$\neg\phi_2$
$\neg(\phi_1 \rightarrow \phi_2)$	ϕ_1	$\neg\phi_2$
$\phi_1 \leftrightarrow \phi_2$	$\phi_1 \rightarrow \phi_2$	$\phi_2 \rightarrow \phi_1$
$\exists x, (\phi_1(x))$	$\phi_1(a_i)$	
$\forall x, (\phi_1(x))$	$\phi_1(c)$	
$\neg\exists x, (\phi_1(x))$	$\neg\phi_1(c)$	
$\neg\forall x, (\phi_1(x))$	$\neg\phi_1(a_i)$	

FIGURE 1 – Les règles de type « et », qui ne créent pas d'embranchement. a_i désigne une nouvelle constante « fraîche » et c désigne une constante existante.



ϕ	ϕ_1	ϕ_2
$\phi_1 \vee \phi_2$	ϕ_1	ϕ_2
$\neg(\phi_1 \wedge \phi_2)$	$\neg\phi_1$	$\neg\phi_2$
$\phi_1 \rightarrow \phi_2$	$\neg\phi_1$	ϕ_2
$\neg(\phi_1 \leftrightarrow \phi_2)$	$\neg(\phi_1 \rightarrow \phi_2)$	$\neg(\phi_2 \rightarrow \phi_1)$

FIGURE 2 – Les règles de type "ou", qui créent un embranchement