

TD5 – Prolog

Hugo Castaneda, Rémy Chaput, Nathalie Guin, Marie Lefevre

PARTIE 1 – CUT

Question 1 : Écrire un prédicat `element`, de deux manières, avec et sans le cut (!), permettant de savoir si X est un élément de la liste L. Expliquer la différence et tracer l'arbre de résolution correspondant.

===== Indices de correction =====

Sans le cut :

```
element(X, [ X | _ ]).
element(X, [ _ | Q ]) :- element(X, Q).
```

```
?- element(c, [a, b, c, d]).
true ;
false.
```

```
?- element(a, [a, b, c, d, a]).
true ;
true ;
false.
```

```
?- element(e, [a, b, c, d]).
false.
```

```
?- element(X, [a, b, c, d]).
X = a ;
X = b ;
X = c ;
X = d ;
false.
```

=> marche pour le test et la génération

Avec le cut :

```
element2(X, [ X | _ ]) :- !.
element2(X, [ _ | Q ]) :- element2(X, Q).
```

```
?- element2(c, [a, b, c, d]).
true.
```

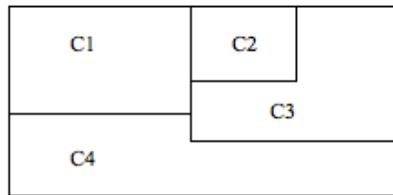
```
?- element2(e, [a, b, c, d]).
false.
```

```
?- element2(X, [a, b, c, d]).
X = a.
```

=> marche pour le test mais pas pour la génération

PARTIE 2 – RESOLUTION DE PROBLEME

On se propose de définir un prédicat permettant de colorier la carte suivante :



Les règles sont les suivantes :

- On dispose de trois couleurs qui sont : vert, jaune et rouge ;
- Deux zones contiguës doivent avoir des couleurs différentes.

Question 2 : Écrivez un prédicat `coloriage(C1, C2, C3, C4)` qui comportera deux parties. La première partie génère toutes les valeurs possibles de C1, C2, C3 et C4. La seconde vérifie si les colorations obtenues sont conformes à la carte par l'utilisation du prédicat $X \neq Y$ sur les couleurs des zones contiguës.

Question 3 : Reprenez ce prédicat, et modifiez le programme en déplaçant les tests de différence de couleurs le plus tôt possible dans l'écriture du prédicat, c'est-à-dire en vérifiant les différences de couleurs dès que celles-ci sont instanciées. Quelle en est la conséquence ?

Indices de correction

```
couleur(vert).  
couleur(jaune).  
couleur(rouge).
```

```
coloriage(C1, C2, C3, C4) :-  
    couleur(C1), couleur(C2), couleur(C3), couleur(C4),  
    C1 \== C2, C1 \== C3, C1 \== C4,  
    C2 \== C3, C3 \== C4.
```

Ici, on génère tout d'abord les couleurs, et on effectue les tests de conformité ensuite.

```
coloriage(C1, C2, C3, C4) :-  
    couleur(C1), couleur(C2), C1 \== C2,  
    couleur(C3), C1 \== C3, C2 \== C3,  
    couleur(C4), C1 \== C4, C3 \== C4.
```

Ici, on effectue les tests de conformité à la volée, ce qui réduit grandement la taille de l'arbre de recherche.

PARTIE 3 – CASSE-TETE

Rejouons avec les **cryptarithmes** vus dans le TD2. Prenons l'exemple suivant :

$$\begin{array}{r}
 \text{H U I T} \\
 + \text{H U I T} \\
 \hline
 = \text{S E I Z E}
 \end{array}$$

Pour rappel, chaque lettre représente un seul chiffre, un chiffre n'est utilisé que pour une seule lettre et le chiffre le plus significatif est différent de zéro. Ce casse-tête a deux solutions : $8253 + 8253 = 16506$ et $9254 + 9254 = 18708$.

Question 4 : Écrivez le programme Prolog permettant de générer et de tester toutes les solutions.

===== Indices de correction =====

```

/* PREMIERE VERSION : on affecte toutes les variables puis on teste */

/* affectation de chacune des variables de X avec une des valeurs de Domaine
(que l'on filtre au fur et à mesure) puis vérification des contraintes */
hhs :-
    X = [H,U,I,T,S,E,Z],
    Domaine = [0,1,2,3,4,5,6,7,8,9],
    affectation(X, Domaine),
    H > 0, S > 0,
    2 * (1000*H + 100*U + 10*I + T) ==
        10000*S + 1000*E + 100*I + 10*Z + E,
    write(X).

/* deux façon de faire le choix pour faire varier les possibilités */
choix(X, [X|R], R).
choix(X, [Y|Xs], [Y|Ys]) :- choix(X, Xs, Ys).

affectation([], _).
affectation([V|Vs], Dom) :- choix(V, Dom, NewDom), affectation(Vs, NewDom).

/* DEUXIEME VERSION PLUS EFFICACE : on teste au fur et à mesure des affectations et
on choisit les variables selon une heuristique : les unités, puis les dizaines... */

valeur(X) :- member(X, [0,1,2,3,4,5,6,7,8,9]).

hhs2([H,U,I,T,S,E,Z]) :-
    valeur(H), H \= 0,
    valeur(S), S \= 0, S \= H,
    valeur(T), not(member(T, [H,S])),
    valeur(E), not(member(E, [T,H,S])), R1 is ((2*T - E)/10),
    valeur(I), not(member(I, [E,T,H,S])),
    valeur(Z), not(member(Z, [I,E,T,H,S])), R2 is ((2*I + R1 - Z)/10),
    valeur(U), not(member(U, [Z,I,E,T,H,S])), R3 is ((2*U + R2 - I)/10),
    2*H + R3 == E + 10*S.

```


Question 6 : Donner les clauses qui décrivent ce catalogue.

Question 7 : Définir le prédicat `voyage(V,D,H,C)` qui s'interprète par : le voyage dans la ville V pendant D semaines avec l'hébergement H coûte C euros.

Question 8 : Définir le prédicat `voyage_eco(V,D,H,C,Cmax)` qui exprime en plus que le coût est inférieur à Cmax euros.

Indices de correction

```
// catalogue(Ville, Trajet, Confort, TarifLogement)
catalogue(rome, 40, hotel, 240).
catalogue(rome, 40, camping, 170).
catalogue(rome, 40, habitant, 80).
catalogue(londres, 20, hotel, 200).
catalogue(londres, 20, camping, 180).
catalogue(londres, 20, habitant, 76).
catalogue(tunis, 40, hotel, 300).
catalogue(tunis, 25, camping, 160).
catalogue(tunis, 25, habitant, 84).
```

```
voyage(Ville, Duree, TypeHebergement, Cout) :-
    catalogue(Ville, PrixTransport, TypeHebergement, CoutH),
    Cout is Duree*CoutH + PrixTransport.
```

Attention Duree doit toujours être fourni pour que cela fonctionne... Par exemple :

```
?- voyage(tunis,2,H,C).
?- voyage(V,2,H,C).
```

```
voyage_eco(Ville, Duree, TypeHebergement, Cout, CoutMax) :-
    catalogue(Ville, PrixTransport, TypeHebergement, CoutH),
    Cout is Duree*CoutH + PrixTransport, Cout=<CoutMax.
```
