Université Grenoble Alpes
MIAGE L3 – Programmations déclaratives

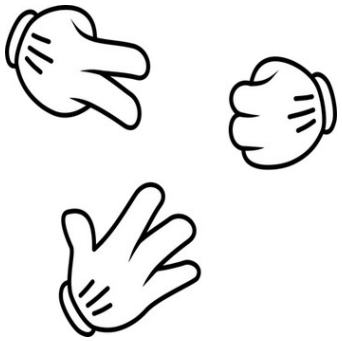
Examen Session 1
Avril 2022

Durée : 90 minutes. 1 document personnel manuscrit A4 autorisé. Sujet sur 1 recto simple.

I. Sur la base du projet Pierre-Papier-Ciseaux – en ProLog (barème indicatif : 5 points)

Rappel du projet, les stratégies sont des prédicats ayant le profil suivant :

```
%% nomStratégie(en entrée : un entier, une liste ; en sortie : un coup)
%%  rem. la liste des coups joués est l'historique d'une partie [joueur1,joueur2]
%%  ex. pour 4 manches : [[pierre,papier],[pierre,pierre],[papier,ciseaux],[papier,pierre]]
%%  pour faciliter l'accès, la dernière manche [pierre,papier] est au début
```



Q1 Stratégie différentDernier. Spécifiez et réalisez une stratégie en ProLog pour le joueur1, pour qu'il rejoue le premier coup différent de son dernier coup joué. Dans l'exemple donné en introduction, le coup joué serait le papier (indiqué en gras dans l'exemple).

Q2*. Stratégie sur l'historique sans doublon. Spécifiez et réalisez une stratégie en ProLog pour le joueur1, pour qu'il rejoue l'un de ses coups en évitant tous ses coups récents (hors le plus ancien). Indication : vous pouvez définir et utiliser un algorithme de suppression des doublons de l'historique qui garde les coups récents. Dans l'exemple donné, le coup joué serait encore le papier.

II. Plus longue sous-liste contiguë croissante - en ProLog ou en Erlang (barème indicatif : 5 points)

Cet exercice cherche la plus longue sous-liste contiguë croissante extraite d'une liste donnée (plus longue sous-liste croissantes d'éléments pris consécutivement dans l'ordre de la liste initiale, sans omettre d'éléments de la liste initiale [entre le premier et le dernier élément de la sous-liste choisie]).

Q1. Plus long préfixe croissant. Spécifiez et réalisez un prédicat ProLog ou une fonction Erlang qui détermine le plus long préfixe croissant d'une liste (i.e. : sous-liste contiguë croissante la plus longue qui commence au début de la liste donnée). Exemple, pour [42,94,96,72,9,6], cela correspond à [42,94,96].

Q2. Liste des sous-listes contiguës croissantes les plus longues. Spécifiez et réalisez un prédicat ProLog ou une fonction Erlang qui sépare une liste en une liste de sous-listes contiguës croissantes les plus longues possibles (dès qu'une sous-liste a atteint sa longueur maximale, commencer une nouvelle sous-liste). Exemple, pour [4,2,9,49,6,7,2,9,6], cela correspond à [[4], [2,9,49], [6,7], [2,9], [6]].

Q3. Plus longue sous-liste contiguë croissante. Spécifier et expliquer (seulement) ce qui reste à faire pour produire la plus longue sous-liste contiguë croissante extraite d'une liste donnée, en utilisant ce qui précède (le code n'est pas demandé). Donner un exemple.

III. Décodage d'un code de César en parallèle - en Erlang (barème indicatif : 10 points)

Cet exercice cherche à retrouver un message codé avec un code de César. Le codage de césar d'un texte avec une clé C numérique comprise entre 1 et 25 consiste à décaler chaque lettre du texte initial de C lettres dans l'alphabet. Ainsi, « BONJOUR » avec une clé C=3 devient « ERQMRXU ». En Erlang, les textes sont des listes de codes ascii. Pour simplifier : on supposera que les textes sont écrits en majuscules, que les espaces sont conservés (code 32 en ascii), et qu'il n'y a pas de ponctuation. Attention, après addition de la clé, quand le code ascii dépasse 90 (code de Z en ascii) enlever 26 pour retrouver un code ascii de lettre.

Q1. Codage de César. Donnez une fonction Erlang qui réalise le codage de César d'un texte donné, pour une clé donnée.

Q2*. Retrouver un mot dans un texte. Pour identifier qu'un message décodé est correct, une méthode consiste à recherche un mot courant dans le message. Donnez une fonction Erlang qui détermine si un mot donné apparaît dans un texte donné. (rappel : mot et texte sont des listes d'entiers)

Q3. Parallélisation.** Mettre en concurrence 25 processus pour tenter de décoder et rechercher un mot donné dans un texte codé donné. Chaque processus doit avoir une clé différente (1, .., 25). Les processus qui trouvent le mot renvoient leur clé, les autres renvoient -1. Un processus supplémentaire reçoit tous les résultats et retourne seulement les clés positives.

Éléments de correction.

Exercice I. Q1. Pour les cas où la liste n'est pas assez longue, ou ne comporte par de coup différent,le coup joué peut être choisi avec une valeur par défaut, ici, ce sera avec papier.

```
différentDernier(N,[],papier).
différentDernier(N,[DernierCoup],papier).
différentDernier(N,[[CJ1,CJ2],[PrecCJ1,PrecCJ2]|L],PrecCJ1) :-
      dif(CJ1,PrecCJ1).
différentDernier(N,[[CJ1,CJ2],[CJ1,PrecCJ2]|L],C) :-
      différentDernier(N,[[CJ1,CJ2]|L],C).
```

Q2.

```
dernierSansDoublon(N,L,C) :-
      extraitJ1(L,LJ1),
      sansDoublon(LJ1,LJ1SansDoublon),
      dernier(LJ1SansDoublon,C).
extraitJ1([],[]).
extraitJ1([[CJ1,CJ2]|L],[CJ1|LJ1]) :-
      extraitJ1(L,LJ1).
```

(à compléter par 2 programmes simples de suppression des doublons et rech. du dernier élément d'une liste)

Exercice II. Q1. Version ProLog, avec croissance stricte.

```
plusLongPrefixeCroissant([],[]).
plusLongPrefixeCroissant([N],[N]).
plusLongPrefixeCroissant([N,M|L],[N|R]) :- {N < M}, plusLongPrefixeCroissant([M|L],R).
plusLongPrefixeCroissant([N,M|L],[N]) :- {N >= M}.
```

Q2.

```
séparation([],[]).
séparation(L,[PlusLongPrefCrois|R]) :-
      plusLongPrefixeCroissantAvecReste(L,PlusLongPrefCrois,Reste),
      dif(PlusLongPrefCrois,[]),
      séparation(Reste,R).
plusLongPrefixeCroissantAvecReste([],[],[]).
plusLongPrefixeCroissantAvecReste([N],[N],[]).
plusLongPrefixeCroissantAvecReste([N,M|L],[N|R],RR) :-
      {N < M}, plusLongPrefixeCroissantAvecReste([M|L],R,RR).
plusLongPrefixeCroissantAvecReste([N,M|L],[N],[M|L]) :- {N >= M}.
```

Q3. Sur la liste de listes produite en Q2, on calcule la longueur des sous-listes (programme classique), et on garde celle des sous-listes qui est de longueur maximale.

Exercice III. Q1.

```
cesar([],Cle) -> [] ;
cesar([32|T],Cle) -> [ 32 |cesar(T,Cle)] ; % pour les espaces
cesar([Car|T],Cle) when Car+Cle > 90 -> [ Car+Cle-26 |cesar(T,Cle)] ;
cesar([Car|T],Cle) -> [ Car+Cle |cesar(T,Cle)].
```

Q2. Pour rechercher un mot, on peut découper le texte selon les espaces pour retrouver les mots. Pour indiquer le résultat, quand le mot n'est pas trouvé le résultat sera -1, sinon ce sera la clé.

```
rechercheMot(Texte,Mot,Clé) -> appartient(Mot,sépareMots(Texte),Clé).
sépareMots([]) -> [[]] ;
sépareMots([32|T]) -> [[]|sépareMots(T)] ;
sépareMots([Car|T]) -> [PremierMot|Mots]=sépareMots(T),[[Car|PremierMot]|Mots].
appartient(M,[],Clé) -> -1 ;
appartient(M,[M|L],Clé) -> Clé ;
appartient(M,[Autre|L],Clé) -> appartient(M,L,Clé).
```

Q3.

```
init(TexteCodé,Mot) ->
      PidC=spawn(prog,consommateur,[25,[]]),
      genereProd(TexteCodé,Mot,25,PidC).
consommateur(0,R) -> R % afficher R aussi ?
consommateur(N,R) -> receive -1 -> consommateur(N-1); Clé -> consommateur(N-1,[Clé|R]).
genereProd(TexteCodé,Mot,0,Pid) -> done ;
genereProd(TexteCodé,Mot,N,Pid) ->
      spawn(prog,producteur,[TexteCodé,Mot,N,Pid]),
      genereProd(TexteCodé,Mot,N-1,Pid)
producteur(TexteCodé,Mot,N,Pid)) -> Pid ! rechercheMot(cesar(TexteCodé,N),Mot,N).
```