Université Grenoble Alpes
MIAGE L3 – Programmations Déclaratives

Épreuve Terminale
Avril 2018 (première session)

Durée : 90 minutes, 1 document A4 autorisé, Sujet sur 1 recto-verso.

I. Suite auto descriptive de Conway (barème indicatif : 6 points)

Quel nombre prolonge la suite logique suivante 1, 11, 21, 1211 :

[ ] : 1231 ?
[ ] : 3112 ?
[ ] : 111221 ?

Une réponse possible est donnée par la suite de Conway avec la proposition 111221.
La suite de Conway est dite suite « regarde et dis », elle consiste à « regarder » à partir du premier terme 1 (comme sur l'exemple donné) puis à « dire » chaque terme pour fournir le terme suivant. Ainsi après avoir « regarder » 1211, on « dira », en lisant par blocs de chiffres identiques, que dans le terme courant 1211 il y a :

- une fois la valeur 1 puis une fois la valeur 2 puis deux fois la valeur 1,

soit, dit autrement :

- 1 fois 1 puis 1 fois 2 puis 2 fois 1

Soit, en abrégé :

- 111221 (c'est le terme suivant et la solution de la question introductive).

Vient ensuite, avec la même règle le terme 312211.

L'exercice consiste à produire à partir d'une liste de nombres représentant un terme d'une suite de Conway, le terme suivant de la liste de Conway sous forme de liste de nombres. L'application de cet algorithme à partir de la liste 1, 2, 1, 1 donnera donc 1, 1, 1, 2, 2, 1.
Une seconde question consistera à vérifier par programme, sur 1000 termes, que les éléments de la suite de Conway partant de 1 ne comportent que des 1, des 2 et des 3 (ni 0, ni 4, ni 5, …)

Q1. **Terme suivant.** Spécifiez et réalisez un prédicat ProLog qui détermine à partir d'une liste de chiffres (par ex. [1,2,1,1]), le terme suivant de la liste de Conway (dans l'exemple donné : [1,1,1,2,2,1])

Q2. **Seulement 1, 2, 3** ? Spécifiez et réalisez un prédicat ProLog qui produise 1000 termes de la suite de Conway partant de 1 et vérifie qu'ils sont tous écrits avec 1, 2 et 3 seulement.

II. Grammaire pour i18n (barème indicatif : 5 points)

Pour raccourcir l'écriture de grands mots, un jour, un informaticien (?) a imaginer de réduire ces mots à leur initiale + le nombre de lettres jusqu'à la dernière lettre + la dernière lettre (sachant qu'il n'y a pas plus de 99 lettres dans les mots). Ainsi « internationalisation » est devenu i18n, « paramétrisation » est devenu p13n, …
Pour les mots courts, c'est probablement inutile, mais on peut imaginer faire pareil ; exception : pour les mots de 1 et 2 caractères, ils pourront rester comme ils sont.
Le début du texte de cet exercice pourrait donc s'écrire ainsi :
P2r r8r l'é6e de g4s m2s, un j2r, un i11n (?) à i6r de r5e c1s g4s m2s à l2r i6e + le n4e de l4e j3u'à la d6e l4e + la d6e l4e (s5t qu'il n'y a p1s p2s de 99 l5s d2s l1s m2s).
Effectivement, c'est un peu plus court.

En simplifiant (on ne conserve que les chiffres, les lettres, les séparateurs et que l'on considère de manière indifférenciée par un nom symbolique : « chiffre », « lettre » et « séparateur »), une grammaire des textes obtenus peut être donnée par :

S → chiffre S
S → lettre A
S → séparateur S
S → finDeTexte

A → chiffre B séparateur S
A → lettre séparateur S
A → séparateur S
A → finDeTexte

B → lettre
B → chiffre lettre

Au choix, en ProLog ou en Erlang :

Q1. Analyse. Spécifiez et réalisez un programme qui vérifie qu'un texte donné correspond à la grammaire fournie. En entrée, le texte sera fourni sous forme de liste de symboles, par exemple [lettre, chiffre, lettre, séparateur, lettre, chiffre, chiffre, lettre, séparateur, séparateur, lettre, finDeTexte].

Q2. Décompte. Spécifiez et réalisez un programme qui s'appuie sur le programme précédent pour compter le nombre de mots.

III. Crible d'Eratosthène réduit (barème indicatif : 9 points)

Le crible d'Eratosthène est une méthode pour trouver les nombres premiers, il consiste à enlever de la liste des entiers les nombres multiples de 2, puis de 3, puis de 5, etc. on trouve ainsi de nouveaux nombres premiers parmi les nombres qui restent.

Q1. Sans nombre pair. Spécifiez et réalisez une fonction Erlang qui enlève d'une liste d'entiers tous les nombres multiples de 2. (Indication : pour savoir si N est multiple de 2, le test à faire est N rem 2 == 0)

Le crible d'Eratosthène (suite) : en partant de la liste 2, 3, 4, 5, 6, 7, 8, 9, … si on enlève les nombres pairs, on obtient une seconde liste 3, 5, 7, 9, … qui commence par 3 qui est un nombre premier. Le crible d'Eratosthène consiste à continuer en enlevant les nombres multiples de 3 de cette seconde liste, on obtient alors une troisième liste 5, 7, 11, 13, … qui commence par 5 qui est un nombre premier. Le crible d'Eratosthène consiste à continuer en enlevant les nombres multiples de 5 de cette troisième liste, et ainsi de suite. Le crible trouve ainsi, 2 puis 3, puis 5, puis 7, puis 11, puis 13 comme nombres premiers, jusqu'à ne plus avoir d'entiers dans la liste courante.

Q2. Crible. Spécifiez et réalisez une fonction Erlang qui trouve les nombres premiers de la liste 2, 3, 4, …, N par suppression successive des multiples de 2, puis 3, puis 5, et ainsi de suite selon l'algorithme du crible d'Eratosthène suggéré précédemment puis rend la liste de nombres premiers trouvés.
Vous pourrez supposer qu'une fonction liste2EtSuivants(N) est fournie qui produit le liste initiale 2, 3, 4, … N. (au besoin vous adapterez cette fonction à votre programme).
Exemple de fonction liste2EtSuivants :

```
liste2EtSuivants(N) →
  listeXEtSuivants(2,N).

listeXetSuivants(N,N) →
      [N];
listeXEtSuivants(X,N) →
      [X|listeXEtSuivants(X+1,N)].
```

Q3. Prétraitement parallèle. On souhaite améliorer le temps de calcul du crible d'Eratosthène. Pour cela, dans un premier temps, on souhaite réaliser en parallèle un prétraitement qui consiste à supprimer les multiples de 2, 3 et 5 de la liste initiale. [+ les multiples de 7, voire plus, si cela semble intéressant]
Il s'agit donc de paralléliser sur la liste initiale 2, 3, 4, … N la suppression des multiples des petits nombres premiers.

Deux idées sont explorées (en supposant ici que l'on ne regarde que les multiples de 2, 3 et 5) :

- Première idée, un processus par nombre premier : la suppression des multiples de 2 sera effectuée par un premier processus, la suppression des multiples de 3 sera effectuée par un second processus, la suppression des multiples de 5 sera effectuée par un troisième processus.
- Seconde idée, découper la liste initiale en sous-listes avec un processus par sous-liste : la liste initiale 2, 3, 4, … N est découpée en plusieurs sous-listes, la première sous-liste est traitée par une premier processus (qui lui enlève les multiples de 2, 3 et 5), la seconde sous-liste est traitée par un second processus (qui lui enlève les multiples de 2, 3 et 5), et ainsi de suite.
  Attention, il ne faudrait pas que le découpage soit aussi couteux que le crible ou revienne à faire le crible : découper en 2 sous-listes : nombres pairs, nombres impairs est inadapté. Par contre, couper en 2 sous-listes 2, 3, 4, … N/2 d'un côté puis N/2+1, N/2+2, N/2+3, … N d'un autre côté est envisageable.

Quelle idée vous semble la plus adaptée ? Pourquoi ? Que proposez-vous d'implémenter ? Spécifiez et réalisez les fonctions nécessaires, en particulier bien détailler (et si nécessaire ajouter un schéma) la création des processus avec la définition des paramètres initiaux, et la communication entre processus.

Eléments de réponse

IV. Suite auto descriptive de Conway (barème indicatif : 6 points)

Q1. **Terme suivant.**

```
% termeSuivantConway(A,B) calcul le terme B qui suit A dans une suite de conway

termeSuivantConway([],[]).
termeSuivantConway([E],[1,E]).
termeSuivantConway([E,E|L],[M+1,E|R]):- % deux éléments identiques de suites, on compte 1 de plus …
        termeSuivantConway([E|L],[M,E|R]).
termeSuivantConway([E,F|L],[1,E,N,F|R]):- % des éléments différents, on commence un nouveau décompte
        dif(E,F),
        next([F|L],[N,F|R]).
```

Q2. **Seulement 1, 2, 3** ?

```
% conway(N,L) pour avoir les N premiers termes de la liste de conway partant de 1, chaque terme mis dans L
conway(1,[[1]]).
conway(N,[F,E|L]):-
  N-1 > 0,
  conway(N-1,[E|L]),
  termeSuivantConway(E,F).

% pour vérifier qu'il n'y a que des 1, 2 et 3, on peut prendre les 2 prédicats suivants

sans4EtSuivants([]).
sans4EtSuivants([E|L]):-
  juste123(E),
  sans4EtSuivants(L).

juste123([]).
juste123([1|L]):-
  juste123(L).
juste123([2|L]):-
  juste123(L).
juste123([3|L]):-
  juste123(L).
```

V. Grammaire pour i18n (barème indicatif : 5 points)

Q1. Analyse.

La traduction en ProLog (et/ou en Erlang) est immédiate.

Q2. Décompte.

Dans la grammaire, il faut identifier les règles qui définissent des mots. Ce sont celles avec des lettres en début de mot :

S → lettre A

et/ou celles en fin de mot :

A → lettre séparateur S
B → lettre
B → chiffre lettre

Il y a moins de règle pour les débuts, on peut donc facilement compter le nombre de mots en comptant les règles de début de mot.
La mise en œuvre suit immédiatement.

VI. Crible d'Eratosthène réduit (barème indicatif : 9 points)

Q1. Sans nombre pair.

```
sansPair([])-> [];
sansPair([E|L]) when E rem 2 == 0 -> % nombre pair, on ne les laisse pas dans le résultat
  sansPair(L);
sansPair([E|L]) -> % nombre impair, on peut le mettre dans le résultat
  [E|sansPair(L)].
```

Q2. Crible.

En généralisant ce qui précède, on peut enlever les multiples d'un entier N donné :

```
sansMultiple([],_N)-> [];
sansMultiple([E|L],N) when E rem N == 0 ->
  sansMultiple(L,N);
sansMultiple([E|L],N) ->
  [E|sansMultiple(L,N)].
```

En utilisant ce qui précède, on combine liste2EtSuivants(N) qui produit [2, 3, … , N] avec un appel au crible définit à la suite qui utilise le premier élément comme nombre premier (et le place en début de résultat) pour obtenir le résultat (appel : crible(liste2EtSuivants(N))) :

```
crible([]) -> [];
crible(Entiers) ->
  [E|L] = Entiers,
  [E|crible(sansMultiple(L,E))].
```

Q3. Prétraitement parallèle.

Faire en parallèle l'élimination des multiples de 2, 3 et 5 sur plusieurs nœuds demande des étapes supplémentaires d'intersection des résultats une fois les résultats rassemblés, le gain en temps est probablement faible.
Par contre, découper la liste initiale [2..N] en 2 sous listes [2..N/2] et [N/2+1, N] permet de faire le traitement en parallèle sur 2 noeuds, avec une concaténation en fin qui peut être effectuée en temps constant (et le découpage en 2 peut être étendu en un découpage en K éléments)

Remarque : le prétraitement proposé, peut s'étendre pour avoir un crible en parallèle, voir le notebook erlang sur github pour une méthode plus générale (élimination de tous les multiples pour n'avoir que les nombres premiers) https://github.com/denisb/notebooks/blob/master/Erlang/Crible%20d'Eratosthene.ipynb

```
init(N) ->
  spawn(bin,pretraitement,[1001,(N-1000) div 2,self()]),
  spawn(bin,pretraitement,[1 + ((N-1000) div 2),N, self()]),
  receive L1 -> R1 = L1 end,
  receive L2 -> R2 = L2 end,
  R1++R2.

pretraitement(Debut,Fin,Id) ->
  Id ! sansMultiple(sansMultiple(sansMultiple(liste(Debut,Fin),2),3),5).
```