TD n°1 - Ensembles énumérables

1 Exercice I : Ensembles énumérables

On veut montrer que les propositions suivantes sont équivalentes :

1 - E est *énumérable* (rappel de la définition) : si $\exists a\ E = W_a = \{x,\ [a|x]\downarrow\}$
2 - E admet une *fonction d'énumération* calculable : $\exists b,\ E = Im([b|.]) = \{x,\ \exists y\ [b|y] = x\}$
3 - E est vide ou admet une *fonction d'énumération totale* calculable : $\exists b,\ E = Im([b|.]) = \{x,\ \exists y [b|y] = x\}$

I. Montrez que 1 $\Rightarrow$ 3.
2. Montrez que 3 $\Rightarrow$ 2.
3. Montrez que 2 $\Rightarrow$ I.

- **Im([b|.]) :** L'image de $\mathbb{N}$ sur le programme b, c'est l'équivalent si on remplaçait par x, le point et que $x \in \mathbb{N}$.
- **[b|y] = x :** Cette affectation veut dire qu'il faut que le programme termine et qu'il soit égal à x (on peut avoir un programme qui termine mais qui ne renvoie pas la bonne valeur).
- **Fonction d'énumération :** C'est une fonction qui affiche tous les éléments de l'ensemble

1.1 2. 3 $\Rightarrow$ 2

> Si E est vide, alors je veux trouver une fonction qui ne produise aucun entier soit une fonction qui boucle pour toute entrée.

```
a() : void
    while(1)
```

> E admet une fonction d'énumération totale de E, alors E admet donc une fonction d'énumération.

- **Fonction totale :** Fonction qui répond tout le temps, qui ne boucle jamais

1.2 I. 1 $\Rightarrow$ 3

> Il existe le programme a, qui affiche les éléments de E tel que le programme s'arrête (E est énumérable). On cherche un programme b, qui énumère les éléments de E, à partir du programme a. Pour cela on va utiliser la fonction step du cours (E admet une fonction d'énumération totale).

- **step<a,x,t> :** renvoie 0 tant que le programme n'a pas fini, et vaux [a|x] + 1 quand il trouve la valeur du programme a sur l'entrée x (le + 1 est pour différencier le 0 du 0 de non-retour). C'est une fonction qui permet de lister toutes les valeurs de x où le programme a s'arrête.

> On trouve donc le programme suivant :

```
b() : int
    while (step<a,x,t> = 0){     //parcours graphe representant step
        <x,t> ++;
    return x;
```

1.3 3. 2 ⇒ 1

> Il existe le programme b, qui énumère les éléments de E. (E admet une fonction d'énumération). On cherche un programme a, qui nous dit si x est dans E donc si le programme a, pour l'entrée x, converge (E est énumérable). On va également utiliser la fonction step mais à l'envers.

> On trouve donc le programme suivant :

```
a() : boolean

    //si la focntion step ne renvoie pas la valeur x + 1
    //(l'element x + 1 pour diff rencier avec le 0),
    //alors je continue de chercher

    while (step<b,y,t> != x+1) //si = x+1, alors step c'est arret
                               //b(y)=x qui apartient a E
        <y,t> ++;
    return 1;        //si j'ai la valeur alors je renvoie true
```

2 Exercice 2 : Ensembles énumérables - mieux comprendre

1 - Montrez que si E est un ensemble énumérable infini alors il admet une fonction d'énumération totale bijective.
2 - Soit E un ensemble infini. Montrez que E est récursif si et seulement si il admet une fonction d'énumération croissante.
3 - Soit E un ensemble infini. Montrez que E est récursif si et seulement si il admet une fonction d'énumération strictement croissante.

2.1 Question n°1 :

> E est un *ensemble énumérable infini* signifie qu'il admet un programme a, tel que [a|.]↓.

> On doit alors trouver la fonction d'énumération totale, puis on devra prouver qu'elle est bijective.

> **Programme b**: Je commence par parcourir les valeurs d'un tableau d'éléments, puis quand je vois un élément qui est nouveau je l'ajoute dans l'ensemble sinon je continue.

> Pour prouver que cette fonction est bijective, on doit d'abord rappeler ce que c'est.

- **Fonction bijective :** Une fonction bijective est une fonction qui est à la fois injective et surjective. C'est-à-dire que tout antécédent à une et une seule image et que toutes images a un unique antécédent.
- **Fonction Injective :** Une fonction injective est une fonction qui a pour chaque antécédent une seule image. Ainsi on ne peut pas avoir deux antécédents qui donnent la même image, mais une image peut ne pas avoir d'antécédents : $\forall x \forall y f(x) = f(y) \Rightarrow x = y$.
- **Fonction Surjective :** Une fonction surjective est une fonction où tous les antécédents ont une image. Ainsi il ne peut pas y avoir d'antécédents qui n'ont pas d'image, ou d'image qui n'ont pas d'antécédents. Tout est rempli, on peut alors avoir une image pour deux antécédents.

On remarque donc que la fonction d'énumération de E est injective par construction, chaque élément de E a une image.

De plus, cette fonction est surjective car le programme créé énumère E, donc donne à chaque élément de E un indice, alors tous les éléments de E ont un antécédent unique.

⇒ On en déduit alors que la fonction d'énumération de E est bijective.

2.2 Question n°2 :

> On doit procéder en 2 étapes, prouver d'abord que si E est récursif alors il admet une fonction d'énumération croissante. Puis si E admet une fonction d'énumération croissante alors E est récursif.

> **Sens direct** : E est un *ensemble récursif* signifie qu'il a une fonction caractéristique qui est calculable, ainsi on a un programme a, tel que si x ∈ E alors renvoie vrai et sinon renvoie faux.

On va donc trouver la fonction d'énumération croissante de E : $\begin{cases} g(0) = min(E) \\ g(x+1) = min(y \in E \mid y \geq g(x)) \end{cases}$
Cette fonction est bien croissante car quand on veut g(x+1), on précise que l'élément g(x) est plus petit ou égal à l'élément x qui va être stocké dans g(x+1).

> **Sens indirect :** On part de la fonction d'énumération croissante de E (programme b), et on doit créer un programme qui va nous dire si les éléments sont dans E ou pas (programme a):

```
a() : boolean

    while (step<b,y,t> != x+1) //si = x+1, alors step c'est arret
                               //b(y)=x qui apartient a E
        <y,t> ++;
    return 1;        //si j'ai la valeur alors je renvoie true
```

2.3 Question n°3 :

> **Sens direct** : Pour la question 3 on fait exactement la même chose mais au lieu de demander ≥ on demande > : $\begin{cases} g(0) = min(E) \\ g(x+1) = min(y \in E \mid y > g(x)) \end{cases}$

Cette fonction est bien strictement croissante car quand on veut g(x+1), on précise que l'élément g(x) est plus petit strictement que l'élément x qui va être stocké dans g(x+1).

> **Sens indirect :** On part de la fonction d'énumération strictement croissante de E (programme b), et on doit créer un programme qui va nous dire si les éléments sont dans E ou pas (programme a), c'est la même fonction que pour la fonction d'énumération croissante.