1)Préambule (tu peux très bien ne pas lire ça)

salut , tu veux essayer de comprendre la logique , calculabilité et complexité , mais c'est tellement difficile à appréhender que tu ne parvient pas à comprendre , ne t'inquiète pas , t'es pas le seul , moi aussi j'y pige rien pour l'instant , mais ça va pas durer pour nous deux , vu que ce résumé intégrera tous les concepts de la calculabilité , logique et complexité vus en cours , ceci avec une pédagogie solide (du jamais vu je t'assure) , en tout cas , assure toi que tes neurones fonctionnent très bien , mets ton tel en mode ne pas déranger et pose toi confortablement et prépare toi à la tempête qui va ravager tes neurones dans quelques instants .

2)Introduction

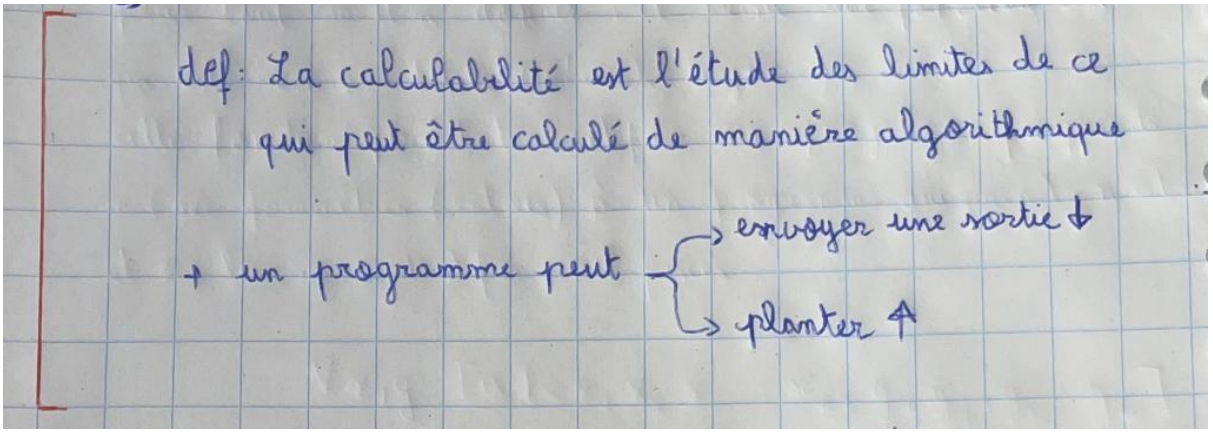
qu'est ce que la calculabilité , en gros "La calculabilité, c'est comme avoir une recette de cuisine. Si tu suis les étapes de la recette et que tu obtiens toujours le même gâteau à la fin, alors la recette 'fonctionne'. En informatique, c'est pareil : si on a une série d'étapes qui nous donne toujours une réponse pour une question, alors on dit que cette question est 'calculable'. pour faire court le but est de savoir si ta recette va cuire (calculable) ou si le poulet ne cuira jamais (jusqu'à l'infini) vu que la plaque n'est pas allumée (non calculable) " .

Plus précisément, une fonction est dite "calculable" si, et seulement si, il existe un algorithme qui, pour toute entrée valide, produit le résultat correct en un nombre fini d'étapes. Cette définition englobe à la fois les fonctions qui peuvent être calculées par des machines modernes et celles qui pourraient être calculées par des machines hypothétiques, comme la machine de Turing, qui sert de modèle standard pour la calculabilité.

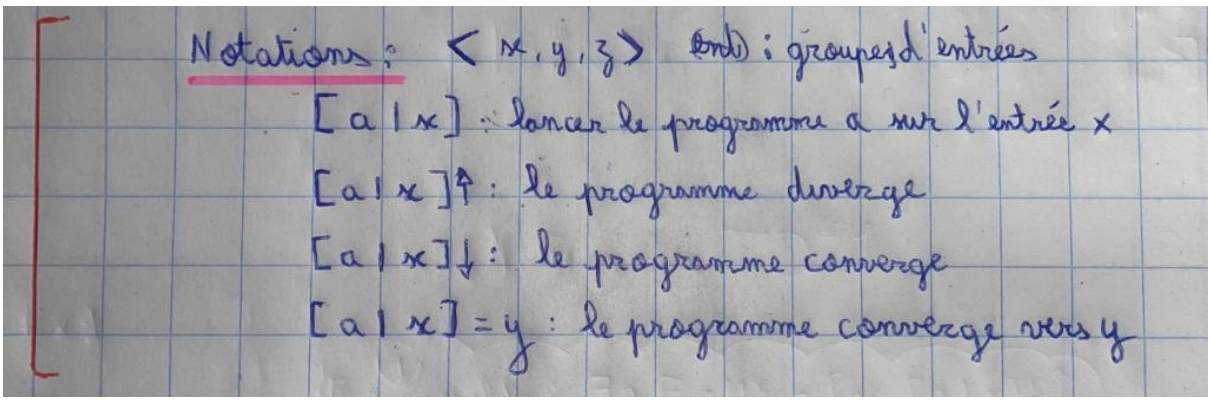
j'ai remarqué qu'on ne parlait jamais de machine de turing dans les cours du coup
machine de turing ==programme

3)Le cours des CM 1 et 2

définition plus courte (comme ça après n tentatives de compréhension on comprends ou on va)



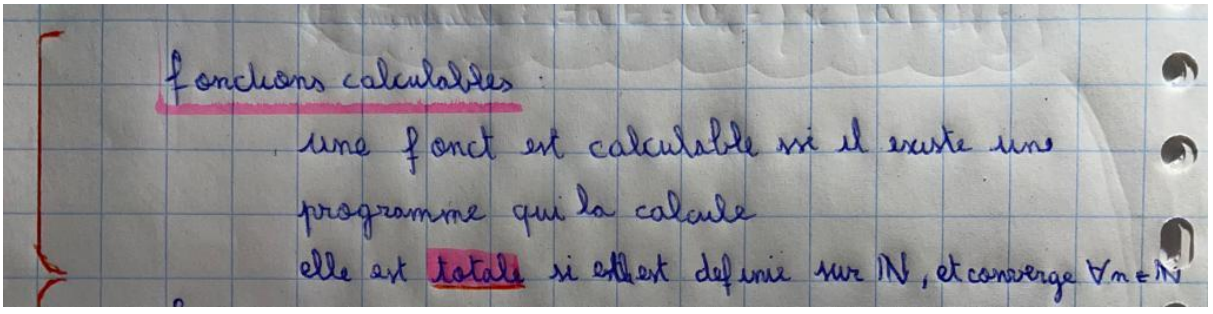
Les notations du cours c'est important du coup voilà
ça diverge =pas de resultat , boucle infinie crash du programme …
ça converge=il y a un resultat , le programme a une sortie …



[a|x] c'est un peu comme a(x) , [a|<x,y,z>] c'est a(x,y,z) du coup !!!

après ça on passera à quelques définitions , dites moi si j'en oublie

Fonction calculable:



Problème de l’arrêt

une fonction F qui retourne 1 si une fonction a est calculable et 0 si elle ne l’est pas n’est pas calculable vu que … si une fonction n’est pas calculable eh ben elle tourne à l’infini dans le temps … c’est ce qu’on appelle le problème de l'arrêt , très intéressant en vrai
voila une petite démo à checker après , sinon bah passe à autre chose je sais c’est un peu compliqué sur papier mais le raisonnement tient si on se concentre:

Problème de l'arrêt :

Le problème de l'arrêt demande s'il est possible de déterminer, pour toute machine de Turing a et pour toute entrée b, si $[a|b]$ converge (c'est-à-dire si la machine a s'arrête lorsqu'elle est exécutée avec b comme entrée).

Formellement, nous voulons savoir s'il existe une machine de Turing H telle que :

$$\begin{cases} "vrai" & \text{si } [a|b] \text{ converge} \\ "faux" & \text{sinon} \end{cases}$$

Preuve que le problème de l'arrêt est indécidable :

Supposons, par l'absurde, qu'une telle machine H existe.

Construisons une nouvelle machine de Turing C qui se comporte comme suit pour une entrée x :

1. Si $H(< x, x >)$ renvoie "vrai", alors C entre dans une boucle infinie (ne s'arrête jamais).
2. Si $H(< x, x >)$ renvoie "faux", alors C s'arrête immédiatement.

Maintenant, posons-nous la question : C s'arrête-t-il lorsqu'il est exécuté avec son propre indice comme entrée, c'est-à-dire $[C|C]$?

- Si C s'arrête avec son propre indice comme entrée, alors selon la définition de H, $H(< C, C >)$ devrait renvoyer "vrai". Mais selon la définition de C, cela signifie que C entre dans une boucle infinie.
- Si C ne s'arrête pas avec son propre indice comme entrée, alors $H(< C, C >)$ devrait renvoyer "faux". Mais cela signifie que C s'arrête immédiatement.

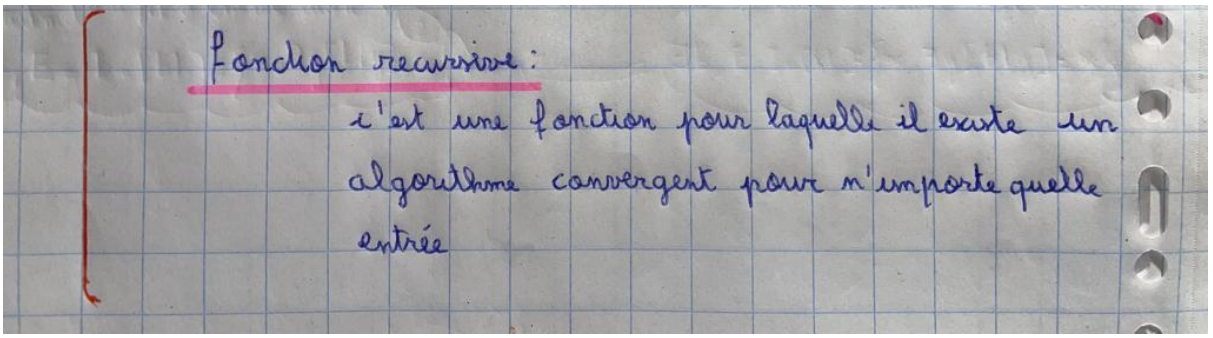
Nous avons donc une contradiction dans les deux cas. Cela signifie que notre hypothèse initiale selon laquelle une telle machine H existe est fausse. Par conséquent, le problème de l'arrêt est indécidable.

Cette preuve montre que le problème de l'arrêt est fondamentalement indécidable : il n'existe aucun algorithme général qui peut déterminer si une machine de Turing donnée s'arrête ou non pour une entrée donnée.

Bref ce qu’il y a à retenir c’est qu’on ne peut pas créer de programme qui analyse la calculabilité d’une fonction

Maintenant un peu de sérieux on passe à la définition d'une

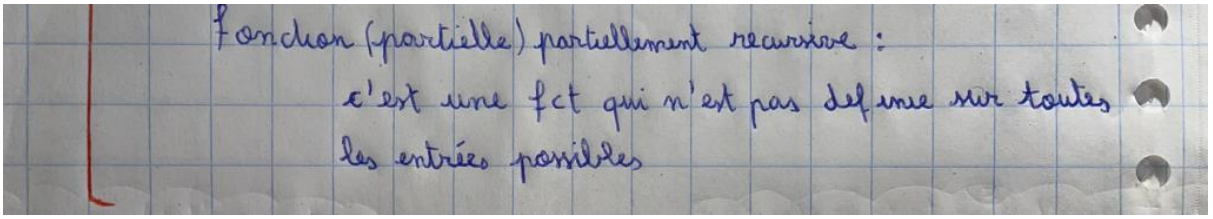
Fonction récursive :



2. **Exemple de fonction récursive en informatique théorique :**
   La fonction d'addition est récursive car elle peut être calculée par une machine de Turing qui s'arrête toujours, quelle que soit l'entrée.

Fonction partiellement récursive

parce que oui si une fonction n'est convergente que pour un groupe d'élément on dit qu'elle est partiellement récursive



Pour donner un exemple simple :
Considérons une fonction qui prend en entrée un nombre entier positif et renvoie "1" si ce nombre est premier et ne renvoie rien (c'est-à-dire qu'elle ne s'arrête pas) si le nombre n'est pas premier. Cette fonction est une fonction partielle, car elle ne donne pas de sortie pour tous les nombres. Si nous avons une machine de Turing qui peut déterminer si un nombre est premier, alors cette fonction est également partiellement récursive.

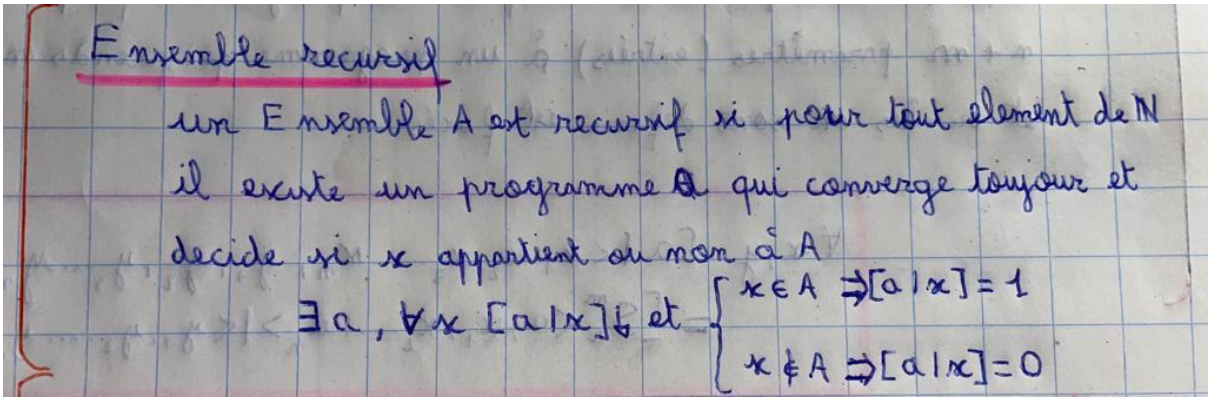
si t'as la mémoire courte une machine de turing c'est un programme tout court te casse pas la tête :3

maintenant un peu de focus sur les

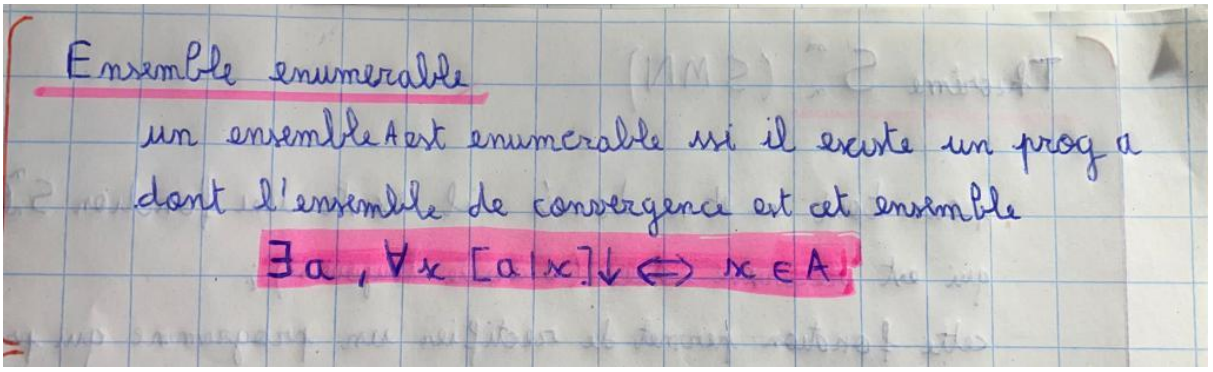
Ensembles

dans le cours il y a deux propriétés qui ont pour but d'induire à un Théorème
il faut pas oublier que les TDs tournent TOUS autours des ensembles enumerables , c'est important bg consacre-y du temps !!!!

Ensemble recursif



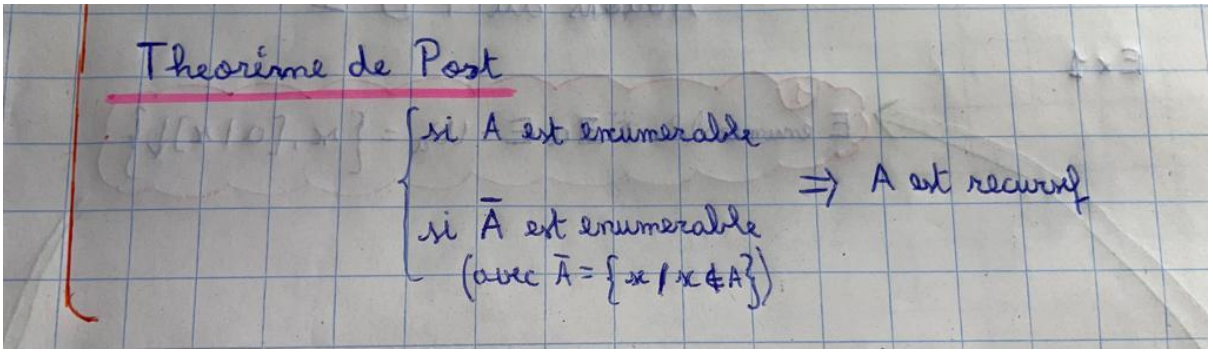
Ensemble énumerable



Propriété :1 ere liaison entre les deux

regarde bien …ça se voit que A récursif ⇒ A enumerable

Théorème de Post



pas trop compliqué le théorème … A bar c’est $\mathbb{N}$-A
petit exemple pour s’assurer

Ensemble A : Considérons l'ensemble A des nombres pairs dans $\mathbb{N}$.
C'est-à-dire $A = \{0, 2, 4, 6, 8, \dots\}$.

Complémentaire de A : Le complémentaire de A est l'ensemble des nombres impairs dans $\mathbb{N}$.
C'est-à-dire $\mathbb{N} \setminus A = \{1, 3, 5, 7, 9, \dots\}$.

1. **Énumérabilité de A :**
   Nous pouvons énumérer tous les nombres pairs en commençant par 0 et en ajoutant 2 à chaque étape. Donc, A est énumérable.
2. **Énumérabilité du complémentaire de A :**
   De même, nous pouvons énumérer tous les nombres impairs en commençant par 1 et en ajoutant 2 à chaque étape. Donc, le complémentaire de A est également énumérable.

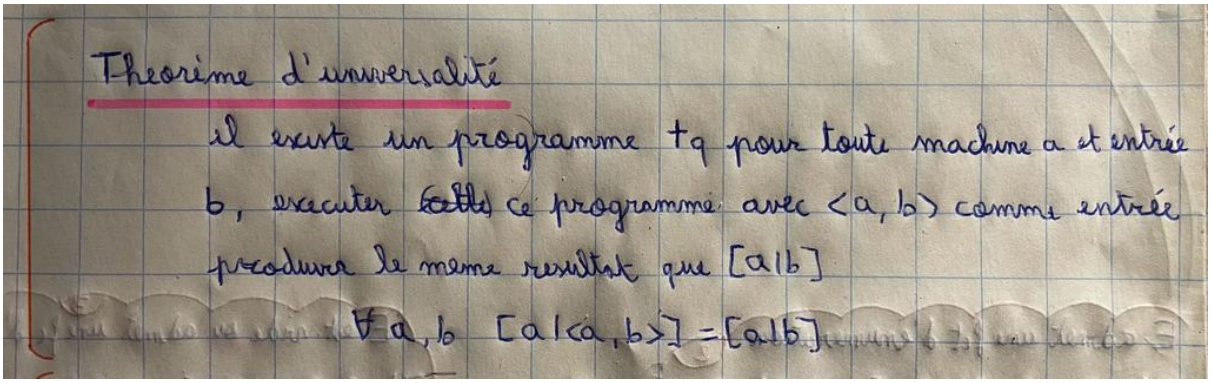
Selon le théorème de Post, puisque A et son complémentaire sont tous deux énumérables, A est récursif.

Application pratique :
Cela signifie que nous pouvons décider en un temps fini si un nombre donné est pair ou non. Bien sûr, dans cet exemple, c'est évident car nous pouvons simplement vérifier si un nombre est divisible par 2. Mais cela illustre le concept : puisque nous pouvons énumérer les nombres pairs et impairs, nous pouvons également décider rapidement si un nombre est pair ou impair.

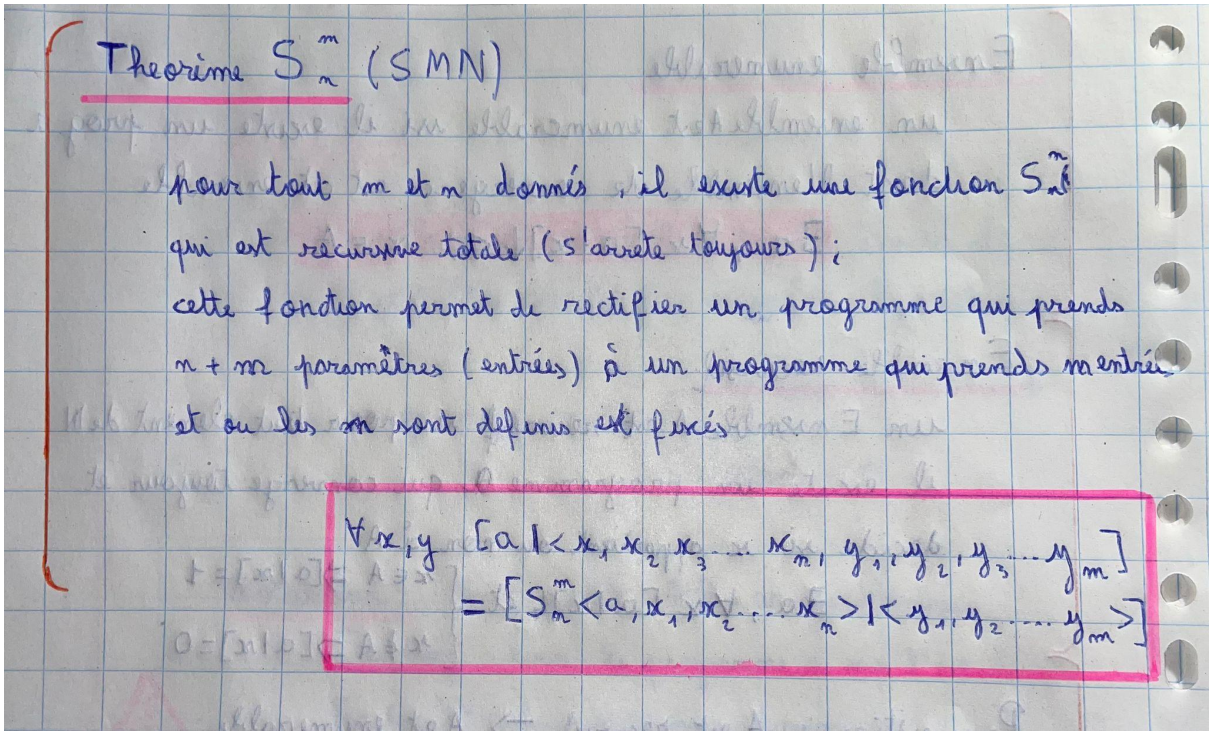
fini le cours sur les ensembles énumérables à toi les notions vues à la fin du CM2 , très importantes et compréhensibles :

Théorème d’universalité :

là en fait on suppose qu’il existe un programme capable d'exécuter TOUS les autres programmes :
en tout cas c’est ce que j’ai compris:



Théorème SMN (le plus facile)



Théorème de rectification de modèle de calcul $S(m, n)$:

Le théorème énonce que pour tout m et n donnés, il existe une fonction $S(m, n)$ qui est récursive totale. Cette fonction permet de "rectifier" ou "adapter" une machine de Turing a qui prend $m + n$ entrées pour qu'elle fonctionne comme une machine qui prend seulement n entrées, tout en "fixant" les m premières entrées à des valeurs spécifiques.

Explication :

1. **Fonction récursive totale** : Une fonction est dite récursive totale si elle est calculable (il existe une machine de Turing qui la calcule) et si elle donne un résultat pour toutes les entrées possibles (elle s'arrête toujours).
2. **Rectification** : L'idée est de transformer une machine a qui nécessite $m + n$ entrées en une nouvelle machine qui n'a besoin que de n entrées, tout en préservant le comportement de a pour les m premières entrées spécifiées.
3. **Formulation** :
   $\forall x, y \quad [a| < x_1, x_2, x_3, ..., x_n, y_1, y_2, ..., y_m >] = [S(m, n) < a, x_1, ..., x_n > | < y_1, y_2, ..., y_m >]$
   Ce que cela signifie, c'est que si vous exécutez la machine a avec les entrées $x_1, x_2, ..., x_n$ suivies de $y_1, y_2, ..., y_m$, cela équivaut à exécuter une autre machine (déterminée par la fonction $S(m, n)$) avec juste les entrées $y_1, y_2, ..., y_m$, tout en "fixant" les entrées $x_1, x_2, ..., x_n$ à leurs valeurs spécifiées.

Un exemple très simple de l'SMN page suivante me remercie pas ^^

Exemple SMN

Exemple : Machine à café

Imaginons que nous ayons une machine à café complexe (notre machine de Turing a) qui nécessite deux types d'entrées pour préparer une tasse de café :

1. **Paramètres de configuration** (les x dans notre notation) : Ce sont des réglages que vous définissez une fois, comme la force du café (léger, moyen, fort) et la taille de la tasse (petite, moyenne, grande).
2. **Données d'entrée** (les y dans notre notation) : Ce sont des choix que vous faites à chaque fois que vous voulez une tasse de café, comme le type de café (espresso, latte, cappuccino) et si vous voulez du sucre ou non.

Maintenant, disons que vous avez un réglage préféré : vous aimez toujours votre café moyen en force et dans une grande tasse. Plutôt que de régler ces paramètres à chaque fois, vous souhaitez avoir une machine simplifiée qui "se souvient" de ces préférences et ne vous demande que le type de café et le sucre.

C'est là qu'intervient la fonction $S(m, n)$. Elle prend votre machine à café complexe et vos préférences (moyen et grande tasse) et vous donne une nouvelle machine (disons une version "personnalisée" de la machine originale). Cette nouvelle machine ne vous demande que le type de café et le sucre, mais elle prépare le café exactement comme l'aurait fait la machine originale avec vos préférences.

En termes formels :
$[a| < \text{moyen, grande, espresso, sucre} >] = [S(2,2) < a, \text{moyen, grande} > | < \text{espresso, sucre} >]$

Cela signifie que préparer un espresso sucré avec la machine originale (en spécifiant moyen, grande, espresso, sucre) est équivalent à préparer un espresso sucré avec la machine personnalisée (où moyen et grande sont "pré-configurés").

C'est une simplification de l'idée derrière le théorème $S(m, n)$, mais j'espère que cela aide à illustrer le concept de manière intuitive.

c’est bon !!! si t’as compris c’est que tu comprends au moins ou va le cours !!! après les Tds c’est une autre histoire on verra ça de prêt …

Logique - Calculabilité - Complexité

Université de Montpellier

TD calculabilité n°1 - 2023

Exercice 1 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 : $\exists b, E = \text{Im}[b|\cdot] = \{x,\ \exists y\ [b|y] = x\}$
3. E est vide ou admet une ***fonction d'énumération totale*** calculable : $\exists b, E = \text{Im}[b|\cdot] = \{x,\ \exists y\ [b|y] = x\}$

1. Montrez que 1 $\implies$ 3.
2. Montrez que 3 $\implies$ 2.
3. Montrez que 2 $\implies$ 1.

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.

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.

Logique - Calculabilité - Complexité

Université de Montpellier

TD calculabilité n°2 - 2023

Exercice 1 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.

Exercice 2 Enumération des fonctions totales

Nous allons montrer dans cet exercice qu'il n'est pas possible d'avoir un système de programmation "raisonnable" où les programmes s'arrêtent toujours. Supposons que ce système existe et notons $[x|y]'$ le résultat du calcul du x-ième programme sur l'entrée y.

1. On suppose que dans un de nos programmes on peut en appeler un autre, et que la fonction successeur soit calculable. Montrez que $g : x \mapsto [x|x]' + 1$ est calculable dans ce système – on notera n son numéro : $[n|\cdot]' = g(\cdot)$.
2. Que vaut $g(n)$? En déduire qu'un tel système n'existe pas.

Exercice 3 Ensembles énumérables - clôture

Nous prouvons dans cet exercice que la classe des ensembles énumérables et bien close par union, intersection et produit cartésien. Soient A et B deux ensembles énumérables.

1. Montrez que $A \cup B$ est énumérable.
2. Montrez que $A \cap B$ est énumérable.
3. Montrez que $A \times B = \{< x, y > \ x \in A \wedge y \in B\}$ est énumérable.

TD n°2 - Ensembles énumérables : suite

1 Exercice I : Ensembles énumérables - mieux comprendre

Même exercice que Exercice 2 du TD1

2 Exercice 2 Enumération des fonctions totales

Nous allons montrer dans cet exercice qu'il n'est pas possible d'avoir un système de programmation "raisonnable" où les programmes s'arrêtent toujours. Supposons que ce système existe et notons $[x|y]'$ le résultat du calcul du x-ième programme sur l'entrée y.

1. On suppose que dans un de nos programmes on peut en appeler un autre, et que la fonction successeur soit calculable. Montrez que $g : x \mapsto [x|x]' + 1$ est calculable dans ce système – on notera n son numéro : $[n|\cdot] = g(\cdot)$.
2. Que vaut g(n) ? En déduire qu'un tel système n'existe pas.

2.1 Question n°1 : MQ g est calculable

> On sait que [x|y]' est calculable et que la fonction successeur est aussi calculable.

> On a donc un programme $n : x \mapsto [x|x]' + 1$ tel que $n : calculable + calculable$ alors il calcule bien g donc g est calculable dans ce système.

2.2 Question n°2 : MQ contradiction

> Par l'appel de n sur n on a $[n|n] = g(n)$.

> Mais par la définition de g sur l'entrée n on a $g(n) = [n|n]' + 1$ alors on a une contradiction.

On en déduit alors que le système de programmation "raisonnable" de l'énoncé n'existe pas.

3 Exercice 3 : Ensembles énumérables - clôture

Nous prouvons dans cet exercice que la classe des ensembles énumérables et bien close par union, intersection et produit cartésien. Soient A et B deux ensembles énumérables.

1. Montrez que A $\cup$ B est énumérable.
2. Montrez que A $\cap$ B est énumérable.
3. Montrez que A $\times$ B $= < x, y > x \in A \wedge y \in B$ est énumérable.

On a deux définitions de l'**énumérabilité** :

- $\underline{x \in A}$: x appartient à A si le programme a, sur l'entrée x, converge.
- $\underline{a(x) \in A}$: a(x) appartient à A si le programme a converge sur l'entrée x. On a alors x qui est l'indice d'un élément qui est dans A ou pas.

Ainsi pour chaque question on va utiliser une des deux versions de la définition.

3.1 Question n°1 : A ∪ B

> On utilise la définition 2, donc on a un programme a qui est l'énumération de l'ensemble A et un programme b qui est l'énumération de l'ensemble B.

> On cherche alors le programme c qui affiche à la fois les éléments de A et à la fois les éléments de B. On va alors afficher une fois sur deux, un élément de A puis un élément de B.

On va donc trouver la fonction d'énumération c de $A \cup B$:

```
c : si x = 2*p alors return [a|p]
    sinon
        si x = 2*p + 1 alors return [b|p]
```

On a donc le programme c qui est l'énumération de $A \cup B$.

3.2 Question n°2 : A ∩ B

> On utilise la définition 1, donc on a un programme a tel que, $[a|x] \searrow$ (converge) si et seulement si $x \in A$ et un programme b tel que, $[b|x] \searrow$ (converge) si et seulement si $x \in B$.

> On cherche alors le programme c qui énumère les éléments de A qui sont aussi dans B. On va alors regarder si a converge, puis voire si b converge.

On va donc trouver la fonction d'énumération c de $A \cap B$:

```
c : [a|x];[b|x];return 1;
```

On a donc le programme c qui est l'énumération de $A \cap B$.

3.3 Question n°3 : A × B

> On utilise la définition 1, donc on a un programme a tel que son domaine soit A et un programme b tel que son domaine soit B.

> On cherche alors le programme c où son domaine est $A \times B$. On va alors regarder si a converge sur l'entrée x, puis voire si b converge sur l'entrée y.

On va donc trouver la fonction d'énumération c de $A \times B$:

On a n = <p, q>, p = Pi1(n) et q = Pi2(n) :

```
c : [a|Pi1(n)];[b|Pi2(n)];return 1;
```

On a donc le programme c qui est l'énumération de $A \times B$.

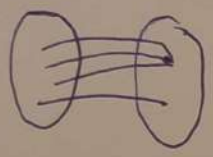
CM3 - Les Réductions

Révisions

- Modèle de calcul
- Universalité: $[u|\langle a,b\rangle] = [a|b]$
- step $\langle a,x,t\rangle$
- SNM

① Réduction

- Réduction many-one (réduction classic)
- Réduction $\neq$ Turing (oracles)



one-one-injectif

many-one

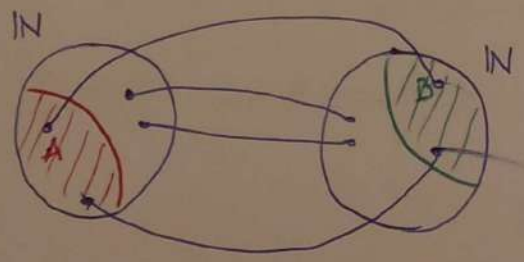
Def.

A, B - ensembles d'entiers

$A \leq B$ = la réduction qui signifie A n'est pas plus difficile que B.
le problème A peut se réduire à problème B

$A \leq B$ si il existe f calculable totale.

$\forall x \quad x \in A \Leftrightarrow f(x) \in B$



Example:

$a: \langle x,y\rangle \longmapsto$ if $[x|x]\downarrow$ then return 0
sinon $\perp$

$f : x \to S^1_1(\langle 0, x \rangle)$

$\mathbb{K} < \mathbb{K}_0$

$\mathbb{K} = \{ x, [x, |x|]\downarrow \}$

$\mathbb{K}_0 = \{ x, [x | 0]\downarrow \}$

$f(x) \in \mathbb{K}_0$ si $x \in \mathbb{K}$

$f(x) \notin \mathbb{K}_0$ si $x \notin \mathbb{K}$

$x \in \mathbb{K} \Leftrightarrow f(x) \in \mathbb{K}_0$

$\mathbb{K} < \mathbb{K}_0$

- On a pas une programme pour $\mathbb{K}_0$, car on a un problème d'arrêt qui n'est pas décidable.

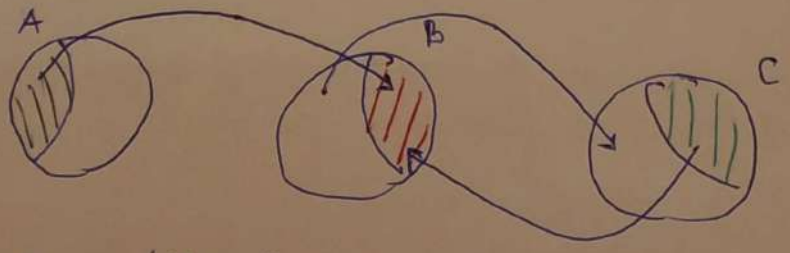
Prop. $\leq$ est un préordre. Pour être un préordre

- faut vérifier la transitivité $A \leq B$ et $B \leq C$ alors $A \leq C$
- et la ...

Preuve: • $A \leq A$ réflexivité

On choisit $f = Id$

• transitivité



$g \circ f(x) = g(f(x))$

$g \circ f$ est récursive totale et converge pour $A \leq C$

Propriété:

Soit A récursif non-trivial ($\neq \emptyset$

Si $B \leq A$ alors B est récursif

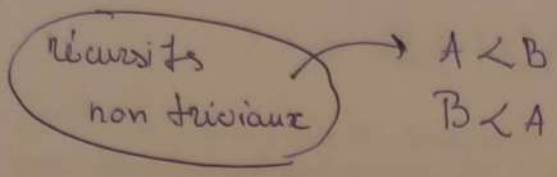
Preuve: Il existe f récursive totale

$\forall x \quad x \in B \Leftrightarrow f(x) \in A$

b: $x \longmapsto$ if $f(x) \in A$ then return oui
else return non

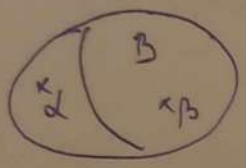
(le programme b décide B)

Récapitulatif



Propriété: Si B est non trivial et A récursif $A \prec B$.

Preuve



$\beta \in B$
$\alpha \notin B$

$x \in A \Leftrightarrow f(x) \in B \longrightarrow$ on sait que A est récursif donc on peut tester

"$x \in A$?" calculable totale

$x \longmapsto$ if $x \in A$ then return β
else return α

Propriété:

Si $A \prec B$ et B est énumerable alors A est énumerable.

Preuve: $x \in A \Leftrightarrow f(x) \in B$

b tq $[b|x]\downarrow$ ssi $x \in B$

$a : x \longmapsto$ return $[b|f(x)]$

$x \longmapsto$ if $[b|f(x)]\downarrow$ $\overset{(\Leftrightarrow)}{}$ then return 0;

Propriété: Si $A \prec K$ alors A est énumerable

Preuve: $x \in A \iff \gamma(x) \in \mathbb{K}$

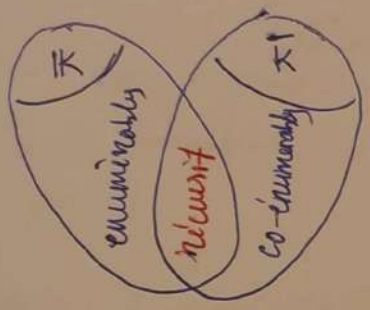
$a : x \longmapsto$ if $[\gamma(x) | \gamma(x)]\downarrow$ then return 0.

Si $[a|x]\downarrow$ alors $\gamma(x) \in \mathbb{K}$

Si $[a|x]\uparrow$ alors $\gamma(x) \notin \mathbb{K}$

Vocabulaire : Un ensemble est co-truc, si son complementaire est truc.

décidable = recursif



Théorème de Post

Théorème de Rice (généralisation du problème de l'arrêt)

Soit $\mathcal{C}$ une propriété sur les fonctions $\mathbb{N} \to \mathbb{N}$ (partielles)

Soit $P_{\mathcal{C}} = \{ x, [x|0] \in \mathcal{C} \}$

$P_{\mathcal{C}}$ - l'ensemble des programmes qui calculent les fonctions qui ont la propriété $\mathcal{C}$.

Alors $P_{\mathcal{C}}$ est soit trivial - (Soit vrai pour tout, soit jamais vrai)
soit indecidable.

Exemple ($\mathbb{K}_0$)

$\mathbb{K}_0 = \{ x, [x|0]\downarrow \}$

$\mathcal{C}$: "est defini en 0"

$\mathcal{C}$: "$\gamma(0)$ est defini"

$P_{\mathcal{C}} = \{ x, [x|0] \in \mathcal{C} \} = \{ x, [x|0]\downarrow \}$ $\longrightarrow$ $\mathbb{K}_0$ non trivial donc décidable.

Preuve :

Où est $\perp$ fonction définie nulle part ?

Si $\perp \in \mathcal{C}$ on travaille sur $\overline{\mathcal{C}}$

$$P_{\overline{\mathcal{C}}} = \overline{P_{\mathcal{C}}}$$

Supposons que $P_{\mathcal{C}}$ n'est pas trivial (donc non-vide)

$$\exists c \in P_{\mathcal{C}}$$

Soit le propriété suivant :

$a : \langle x, y \rangle \longmapsto$ if $[x|x]\downarrow$ then $[c|y]$

else $\perp$

$f : x \longmapsto S'_1 \langle a, x \rangle$

On remarque que $[f(x), y] =$ $[c|y]$ si $x \in K$; $\perp$ sinon

$P_{\mathcal{C}}$ est indécidable car s'il était décidable on pourrait décider K.

* La preuve donne : $K \prec P_{\mathcal{C}}$

Renforcement

Soit $T_{\mathcal{C}}$ est trivial

$K \prec T_{\mathcal{C}}$ ou $K \prec \overline{P_{\mathcal{C}}}$

TD n°3 - Réductions et Theorème de Rice

1 Exercice 1 : Réductions

Soit $A = \{x,\ \forall y\ [x|y]\downarrow\}$.

1. En utilisant avec soin le théorème de Rice, montrez que A n'est pas récursif.
2. Montrez que $\mathbb{K} \prec$ A.
3. Montrez que $\mathbb{K} \prec \bar{A}$ (où $\bar{A}$ est le complémentaire de A).
4. Montrez que ni A ni $\bar{A}$ ne sont énumérables.

1.1 Question n°1 : A non récursif par le théorème de Rice

> On a A l'ensemble des (codes de) programmes qui convergent pour chaque entrée y.

> On utilise donc le théorème de Rice, tel que $\mathcal{C} = \forall y\ [x|y]\downarrow$ soit une propriété de la fonction $c : z \mapsto [x|z]$, tel que $dom(c) = \mathbb{N}$.

> D'après le théorème de Rice, on cherche à montrer que A est non trivial. Alors, on doit trouver un programme qui est dans A ($A \neq \emptyset$) et un programme qui n'est pas dans A ($A \neq \mathbb{N}$) :

1. Soit Id un programme tel que : $Id : x \mapsto x$ et $Id \in A$.
2. Soit bot un programme tel que : $bot : x \mapsto \perp$ et $bot \notin A$

> On doit maintenant prouver que ces deux programmes respectent la propriété $\mathcal{C}$ ou non :

1. Id converge sur toutes entrées alors $dom(Id) = \mathbb{N}$, on en déduit que $Id \in A$ donc $A \neq \emptyset$.
2. bot diverge sur toutes entrées alors $dom(bot) = \emptyset$ soit $dom(bot) \neq \mathbb{N}$, on en déduit que $bot \notin A$ donc $A \neq \mathbb{N}$.

> On vient de prouver que A n'est pas trivial alors par le theorème de Rice, A n'est pas récursif.

1.2 Question n°2 : $\mathbb{K} \prec$ A

> On doit trouver une fonction de réduction calculable totale telle que $x \in \mathbb{K} \Rightarrow f(x) \in A$.

> Soit le programme suivant :

$$a < x, z >: si\ [x|x]\downarrow\ alors\ return\ z$$

(z doit être indépendant de l'ensemble $\mathbb{K}$ car $dom(c) = \mathbb{N}$ avec c vu à la question 1)

La fonction de réduction est donc :

$$f : x \mapsto S_1^1 < a, x >: z \mapsto si\ [x|x]\downarrow\ alors\ return\ z$$

> f est calculable totale car on associe à f le code d'un programme pour toute entrée. Cependant, cela ne veut pas dire que ce dernier est totale.

> F est bien la fonction de réduction car :

1. $x \in \mathbb{K} \Rightarrow [x|x]\downarrow \Rightarrow [f|x] = z \Rightarrow f(x) \in A$
2. $x \notin \mathbb{K} \Rightarrow [x|x]\uparrow \Rightarrow [f|x] = \perp \Rightarrow f(x) \notin A$

> On vient donc de montrer que $\mathbb{K} \prec$ A

1.3 Question n°3 : $\mathbb{K} \prec \bar{A}$

> On doit trouver une fonction de réduction calculable totale tel que $x \in \mathbb{K} \Rightarrow f(x) \in \bar{A}$ (avec $\bar{A} = x, x$ programme qui converge des fois.

> Soit le programme suivant :

$$b < x, z >: si\ step < x, x, z >= 0\ alors\ return\ z$$

(step = 0, on teste si le programme diverge)

La fonction de réduction est donc :

$$f : x \mapsto S_1^1 < b, x >: z \mapsto si\ step < x, x, z >= 0\ alors\ return\ 0\ sinon\ \perp$$

> f est calculable totale car on associe à f le code d'un programme pour toute entrée. Cependant, cela ne veut pas dire que ce dernier est total.

> F est bien la fonction de réduction car :

1. $x \in \mathbb{K} \Rightarrow [x|x]\downarrow \ \Rightarrow [f|x] = \perp \Rightarrow f(x) \notin A \Rightarrow f(x) \in \bar{A}$
2. $x \notin \mathbb{K} \Rightarrow [x|x]\uparrow \ \Rightarrow [f|x] = 0 \Rightarrow f(x) \in A \Rightarrow f(x) \notin \bar{A}$

> On vient donc de montrer que $\mathbb{K} \prec \bar{A}$.

1.4 Question n°4 : A et $\bar{A}$ ne sont pas énumérables.

> On sait que $K \prec A$, donc $\bar{K} \prec \bar{A}$ et on sait que $K \prec \bar{A}$, donc $\bar{K} \prec A$.

> On sait également que K est énumérable alors $\bar{K}$ n'est pas énumérable.

> Alors, $\bar{A}$ n'est pas énumérable et A n'est pas non plus énumérable.

2 Exercice 2 : classique

ATTENTION : Cet exercice n'a pas été fait en TD ainsi rien n'est sûr sauf la méthode.

Soit $B = \{x, [x|0]\downarrow et\ [x|1]\uparrow\}$.

1. En utilisant avec soin le théorème de Rice, montrez que B n'est pas récursif.
2. B et son complémentaire $\bar{B}$ sont-ils énumérables ?

2.1 Question n°1 : B non récursif par théorème de Rice

> On a B l'ensemble des (codes de) programmes qui convergent pour l'entrée 0 et diverge pour l'entrée 1.

> On utilise donc le théorème de Rice, tel que $\mathcal{C} = \{[x|0]\downarrow \&\& [x|1]\uparrow\}$ soit une propriété de la fonction $c : z \mapsto [x|z]$, tel que $0 \in dom(c)$ et $1 \notin dom(c)$.

> D'après le théorème de Rice, on cherche à montrer que B est non trivial. Alors, on doit trouver un programme qui est dans B ($B \neq \emptyset$) et un programme qui n'est pas dans B ($B \neq \mathbb{N}$) :

1. Soit *zero* un programme tel que : $zero : x \mapsto if\ x = 0\ return\ 0$ et $zero \in B$.
2. Soit *un* un programme tel que : $un : x \mapsto return\ 1$ et $un \notin B$

> On doit maintenant prouver que ces deux programmes respectent la propriété $\mathcal{C}$ ou non :

1. *zero* converge sur l'entrée 0 et diverge sur toutes les autres entrées dont 1, alors $0 \in dom(zero)$ et $1 \notin dom(zero)$, on en déduit que $zero \in B$ donc $B \neq \emptyset$.

2. *un* converge sur toutes entrées dont l'entrée 1, alors $dom(un) = \mathbb{N}$ soit $0 \notin dom(un)$, on en déduit que $un \notin B$ donc $B \neq \mathbb{N}$.

> On vient de prouver que B n'est pas trivial alors par le théorème de Rice, B n'est pas récursif.

2.2 Question n°2 : B et $\bar{B}$ non énumérables

> On doit procéder comme l'exercice précédent : 1. Prouver $\mathbb{K} \prec B$, et $\mathbb{K} \prec \bar{B}$.

> $\mathbb{K} \prec B$:

- On doit trouver une fonction de réduction calculable totale telle que $x \in \mathbb{K} \Rightarrow f(x) \in B$.
- Soit le programme suivant :

$$b < x, z >: si\ [x|x] \downarrow\ \&\&\ z = 0\ alors\ return\ z$$

(z doit être indépendant de l'ensemble $\mathbb{K}$ car $dom(c) = \mathbb{N}$ avec c vu à la question 1)

La fonction de réduction est donc :

$$f : x \mapsto S_1^1 < b, x >: z \mapsto si\ [x|x] \downarrow\ \&\&\ z = 0\ alors\ return\ z$$

- f est calculable totale car on associe à f le code d'un programme pour toute entrée. Cependant, cela ne veut pas dire que ce dernier est total.
- F est bien la fonction de réduction car :
  1. $x \in \mathbb{K} \Rightarrow [x|x] \downarrow\ \Rightarrow [f|x] = z = 0 \Rightarrow f(x) \in B$
  2. $x \notin \mathbb{K} \Rightarrow [x|x] \uparrow\ \Rightarrow [f|x] = \perp \Rightarrow f(x) \notin B$
- On vient donc de montrer que $\mathbb{K} \prec$ B

> On fait de même avec $\mathbb{K} \prec \bar{B}$:

- On doit trouver une fonction de réduction calculable totale telle que $x \in \mathbb{K} \Rightarrow f(x) \in \bar{B}$ (avec $\bar{B}$, le programme qui converge sur 1 mais qui diverge sur 0.
- Soit le programme suivant :

$$b < x, y, z >: si\ step < x, x, y >= 0\ \&\&\ z = 0\ alors\ return\ z\ sinon\perp$$

(on a 3 variables car y est le temps et z une variable independante)

La foontion de réduction est donc :

$$f : x \mapsto S_1^2 < b, x >: y, z \mapsto si\ step < x, x, y >= 0\ \&\&\ z = 0\ alors\ return\ z\ sinon\perp$$

- f est calculable totale car on associe à f le code d'un programme pour toute entrée. Cependant, cela ne veut pas dire que ce dernier est total.
- f est bien la fonction de réduction car :
  1. $x \in \mathbb{K} \Rightarrow [x|x] \downarrow\ \Rightarrow [f|x] = \perp \Rightarrow f(x) \in \bar{B} \Rightarrow f(x) \notin B$
  2. $x \notin \mathbb{K} \Rightarrow ([x|x] \uparrow\ \Rightarrow [f|x] = z = 0 \Rightarrow f(x) \notin \bar{B} \Rightarrow f(x) \in B$
- On vient donc de montrer que $\mathbb{K} \prec \bar{B}$

> Maintenant, prouvons que B et $\bar{B}$ ne sont pas énumérables :

- On sait que $K \prec B$, donc $\bar{K} \prec \bar{B}$ et on sait que $K \prec \bar{B}$, donc $\bar{K} \prec B$.
- On sait également que K est énumérable alors $\bar{K}$ n'est pas énumérable.
- Alors, $\bar{B}$ n'est pas énumérable et B n'est pas non plus énumérable.

Cours magistral 4 calculabilité

Inséparabilité

Définition :
Deux ensembles A et B qui sont disjoints sont dits **inséparables** si et seulement si il n'existe pas d'ensemble récursif qui les sépare. Formellement, cela signifie qu'il n'existe pas d'ensemble R qui est récursif tel que A est un sous-ensemble de R et B est un sous-ensemble du complémentaire de R.

Exemple :
Les ensembles K et $\overline{K}$ (le complémentaire de K) sont un exemple classique d'ensembles inséparables.

Proposition :
Il existe deux ensembles énumérables qui sont inséparables. Une remarque importante est que le fait d'être inséparable est équivalent à être récursivement inséparable.

Preuve :
Considérons les ensembles A et B définis comme suit :

- $A = \{x \mid [x|x] = 0\}$
- $B = \{x \mid [x|x] = 1\}$

Il est évident que A et B sont disjoints car un élément ne peut pas à la fois satisfaire $[x|x] = 0$ et $[x|x] = 1$.

Supposons par l'absurde que A et B soient séparables par un ensemble récursif R. Alors, nous pouvons définir une fonction r telle que :

$$\begin{cases} 0 & \text{si } x \in R \\ \text{non défini} & \text{sinon} \end{cases}$$

Si $[r|r] = 1$, alors r appartient à R et donc aussi à B.
Si $[r|r] = 0$, alors r n'appartient pas à R et donc appartient à A.

Cela conduit à une contradiction, car il existe une solution calculable partielle qui ne peut pas être étendue en une fonction totale. En d'autres termes, il y a des programmes où le bug est inévitable.

Points fixes

Théorème :
Soit f une fonction calculable totale. Il existe un entier n tel que le comportement du programme n est identique à celui du programme $f(n)$. En d'autres termes, $[n|.] = [f(n)|.]$.

Ici, f est une fonction qui transforme un programme. C'est ce qu'on appelle une "transformation de programme".

Exemple :
Considérons une transformation simple f qui prend un programme x et le transforme pour qu'il renvoie toujours une valeur constante a. Dans ce cas, un point fixe n pour cette transformation est un programme tel que $[n|.] = [f(n)|.] = [a|.]$. Cela signifie que le programme n a le même comportement que le programme transformé $f(n)$, qui renvoie toujours a.

Exemple d'auto-référence :
Il est possible d'avoir un programme qui écrit son propre code. Prenons par exemple la fonction a définie par :
$a :< x, y > \rightarrow \text{return } x$
Si on applique a avec les arguments x et y, elle renvoie simplement x.

Considérons maintenant une transformation f définie par :
$f : x \rightarrow s(1,1) < a, x >$
Il existe un entier n tel que pour tout y, $[n|y] = [f(n)|y] = n$. Cela signifie que le programme n renvoie son propre code.

Exemple de composition :
Supposons que nous ayons un programme m qui calcule $f(g(x))$. Si nous définissons n comme étant $g(m)$, alors :
$[n|.] = [g(m)|.] = [[n|m]|.] = [f(g(n))|.] = [f(n)|.]$
Cela montre que le programme n a le même comportement que le programme $f(n)$ lorsqu'il est appliqué à m.

Logique - Calculabilité - Complexité

Université de Montpellier

TD calculabilité n°4 - 2023

Exercice 1 facile

Soit g une fonction calculable.

1. Montrez qu'il existe une fonction calculable *totale* G telle que $[G(n) \mid \cdot] = n + g(.)$
2. Montrez que $\exists n\ [n \mid \cdot] = n + g(.)$.

Exercice 2 tout aussi aisé

1. Montrez qu'il existe une fonction calculable *totale* f telle que $[f(n) \mid \cdot] = [n \mid \cdot] + 1$
2. Quelles fonctions sont calculées par les points fixes de f ?

Exercice 3 entrainement réductions

Lesquels de ces ensembles sont-ils décidables ? Enumérables ? Techniques suggérées : utilisez le théorème de Rice ou une réduction à partir de $\overline{\mathbb{K}}$ vers cet ensemble (lorsque c'est possible), ou encore montrez l'énumérabilité par une des caractérisations de votre choix.

1. $C = \{x,\ [x|0]\downarrow, [x|1]\downarrow,$ et $[x|0] = [x|1]\}$
2. $D = \{x,\ [x|\cdot]$ est totale et injective$\}$
3. $E = \{x,\ \exists y\ [x|y] = 0\}$
4. $B = \{x,\ [x|x] = x$
5. $F = \{x,\ [x|.]$ diverge sur un ensemble infini d'entrées$\}$
6. $G = \{x,\ [x|.]$ converge sur un ensemble infini d'entrées$\}$
7. $H = \{x,\ x$ est un nombre premier$\}$

Exercice 4 opérations ensemblistes

1. Trouvez un exemple d'ensembles S_1 et S_2 non énumérables tels que $S_1 \setminus S_2$ soit énumérable.
2. Trouvez un exemple d'ensembles S_1 et S_2 non énumérables tel que $S_1 \cup S_2$ soit énumérable.

Exercice 5 amusant et à peine plus dur

1. Montrez que dans tout système acceptable de programmation il existe 2 programmes consécutifs qui calculent la même fonction.
2. Proposez un système de programmation dans lequel 3 programmes consécutifs ne calculent jamais la même fonction.

TD4 exo 1

① $G(n) \; S_1^1 a(n)$: $a : \langle n, x \rangle \longmapsto n + g(x)$

$x \longrightarrow n + g(x)$

Tobale car $n \longrightarrow$ "texte" du programme

② $\exists \; n \longrightarrow G(n)$ par un car l'un n'a pas la même ct

$\exists n \forall y \; [n|y] = [G(n)|y]$

TD4 exo 2

$f(n) = S_1^1 a(n) : a\langle n, x \rangle \longrightarrow [n|x]+1$

$x \longrightarrow [n|x]+1$

Kleene $\exists n \; \forall z \quad [f(n)|z] = [n|z]$

$\forall z \quad [n_0|z] + 1 = [n_0|z]$

n_0 ne converge nulle part !

$\forall z \; [n_0|z] \uparrow$

3) $K < \overline{B}$ $\quad \overline{B} = \{x \mid [x|0]\uparrow \text{ ou } [x|1]\downarrow\}$

$c : \langle x, z\rangle \longrightarrow$ si $z \neq 1$ alors 2 sinon $[x|x]$

$S_1^1 c(x) : z \longrightarrow$ si $z \neq 1$ alors 2 sinon $[x|x]$

$x \longrightarrow S_1^1\langle c, x\rangle$ calculable totale texte du programme.

$x \in K$ $\quad f_{(x)}(1)$ cvge $\quad f_{(x)} \notin B \quad (\in \overline{B})$

$f(x) \notin B$ $\quad f_x(1)$ cvge ou $f_x(0)$ diverge

impossible

$f_x(1)$ cvge

$x \notin K \qquad K < \bar{B} \quad \bar{K} < B$

B pas enumerable

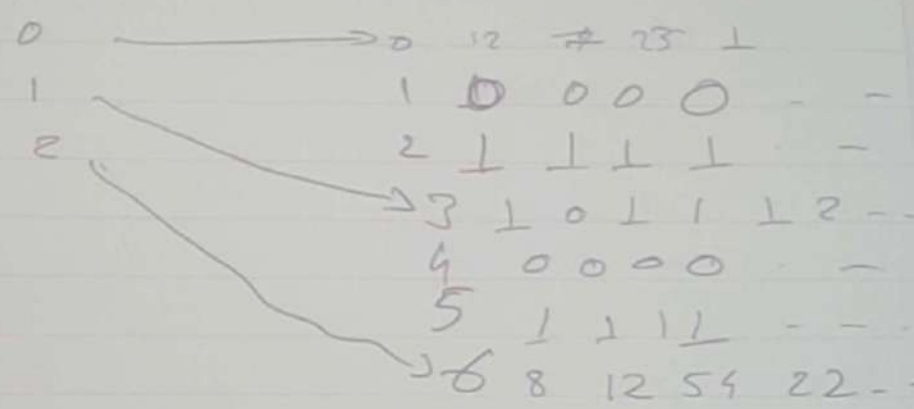
5)

① $\exists x_0 \forall x \; [x_0 | x] = [x_0 + 1 | x]$ $\quad n \longrightarrow n+1$

$a : \langle x, y \rangle \longrightarrow [x+1 | y]$

$x \to S^1_1 x$ $\qquad \exists x_0 \forall y \; [x_0 | y] = [x_0 + 1 | y]$

② récursion



Logique - Calculabilité - Complexité

Université de Montpellier
Partiel de calculabilité - 2022
19 octobre 2022

Durée 1h30
Aucun document n'est autorisé
Pas de calculatrice, téléphone portable, montre programmable,
appel à un ami, consultation de l'avis du public, *etc.*

Justifiez vos réponses avec grand soin!

Dans tout ce qui suit, comme dans le cours, le symbole $\prec$ désigne la *réduction many-one*; les ensembles considérés sont des ensembles d'entiers, qu'ils contiennent des données ou des programmes.

Exercice 1 énumérations

1. Montrez que A est énumérable si et seulement si $A \prec \mathbb{K}$.
2. Soit g une fonction récursive totale croissante (au sens large) et tendant vers l'infini. Montrez que si un ensemble A admet une fonction d'énumération f injective qui vérifie $\forall x\ f(x) \geq g(x)$, alors il est récursif.

Exercice 2 réductions

Soit A l'ensemble des programmes x tels que x s'arrête sur au moins une entrée et, sur deux entrées distinctes où x s'arrête, il ne donne jamais le même résultat.

1. Ecrivez A sous la forme d'une formule du type $A = \{x,\ \dots\}$.
2. En utilisant avec soin le théorème de Rice, montrez que A n'est pas récursif.
3. Montrez que $\mathbb{K} \prec A$.
4. Montrez que $\mathbb{K} \prec \overline{A}$ (où $\overline{A}$ est le complémentaire de A).
5. Montrez que ni A ni $\overline{A}$ ne sont énumérables.

Exercice 3 points fixes

On dit qu'un programme est *minimal* s'il est le plus petit dans l'ensemble de tous les programmes qui calculent la même fonction.
En d'autres termes n est minimal si $\forall m\ [m \mid \cdot] = [n \mid \cdot] \implies m \geq n$; ou encore n est minimal si pour tout (code de) programme m calculant la même fonction que le programme n on a $m \geq n$.
On note M l'ensemble des programmes minimaux et on souhaite montrer que M n'est pas récursif.

1. Expliquez pourquoi on ne peut pas utiliser le théorème de Rice pour traiter le cas de cet ensemble.
2. Montrez que si la propriété d'être minimal était décidable, alors pour chaque entier, on pourrait trouver un entier strictement plus grand que le premier et qui soit minimal.
3. Montrez (on peut utiliser ce qui précède) que la minimalité n'est pas décidable.