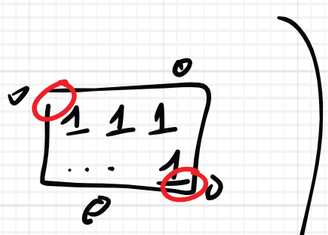
Construire une machine qui connaissant une matrice binaire A (que des 0 et des 1) renvoie la taille du plus grand rectangle de 1 de A.

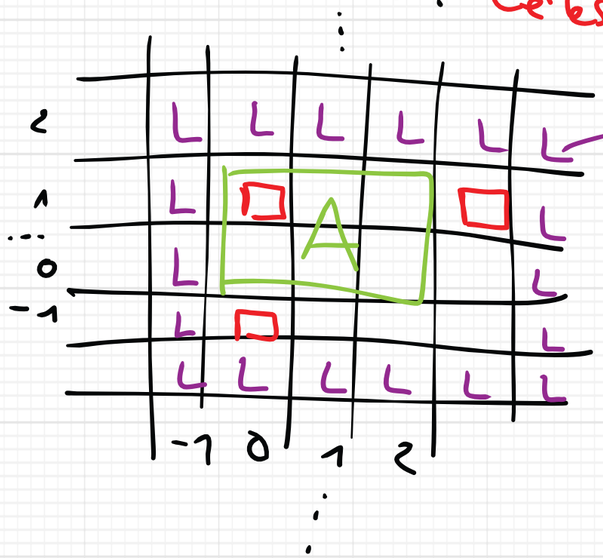


Idée : la machine trouve (de manière non déterministe) les 2 coordonnées rouges (coin HG et BD) et on vérifie que le contenu est constitué uniquement de 1.

On a une machine Π

- une bande 2D
- 3 têtes
- non déterministe
- alphabet $\{0, 1, B, L, S\}$

Bande : entrée : têtes : □



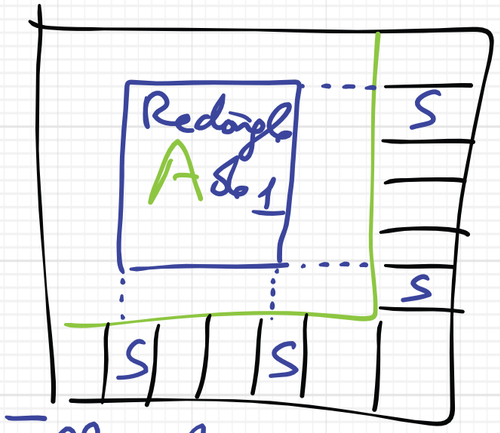
Limite, les têtes ne sortent pas ici

→ Si oui, alors c'est le bon rectangle (la machine ne se trompe jamais)

→ Sinon pas de 1 dans A

Comme expliqué en cours, cette machine calcule mais une machine non déterministe ne devrait pas calculer, juster décider.
Une autre manière de transformer ça en problème de décision serait de dire
"Soit n, m des entiers, existe-t-il un rectangle de taille nxm constitué uniquement de 1?"

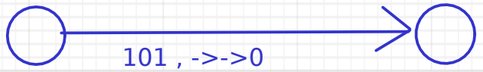
Comment les têtes décident des coins :



Elles placent 2 'S' pour indiquer la première et la dernière ligne / colonne du rectangle.

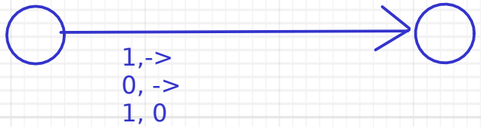
L'idée de la machine est donc la suivante :
- de manière non déterministe, les 2 têtes au bord du rectangle écrivent chacune deux S.
- les quatre S indiquent les 4 coins d'un rectangle dans la matrice
- la 3e tête vérifie que chaque case de ce rectangle est un 1

- Vous pouvez forcer la tête centrale à être toujours sur la même colonne que la tête du bas et sur la même ligne que la tête du haut.
- pour écrire la machine, vous pouvez écrire les transitions ainsi :



Qui se lit, "si la 1ere tête lit un 1, la 2e lit un 0 et la 3e lit un 1 alors la 1ere va à droite, la 2e aussi et la 3e écrit un 0".

ou comme ça



Si vous souhaitez faire des machines de votre côté, vous pouvez aller sur http://turingmachine.io/
L'éditeur est assez simple, par contre vous ne pouvez faire que des machines sur une bande.

$(f_1() = 1) \in PR$

~~$\sigma(0)$~~ $\sigma \circ 0$

(comme expliqué en cours, la notation de gauche est légale, juste moins claire à mon sens que celle de droite qui évite de confondre entier et fonction)

$(f_2(x) = 1) \in PR$

Créons $(f(x) = 0)$: $f = \sigma(0, P_{(3,3)})$

$f_2 = \sigma \circ f$

$0, \sigma, P_{(i,p)}$

\+ comp $\circ$

récp ↺

Exemple

$f_2(2) = \sigma \circ f(2) = \sigma(f(2))$

$= 1$

$f(2) = \sigma(0, P_{33})(2)$

$f(z, n)$ → vide

$= P_{33}(\ , 1, f(1))$

$= f(1)$

$= P_{33}(\ , 0, f(0))$

$= f(0)$

$= 0() = 0$

Slide 8 : argument diagonal : $PR \neq F$

Remarque : Cette démonstration est exactement la même qui prouve que toutes les fonctions ne sont pas calculables/récursives.
La seule différence est que la fonction g qui est construite ici est calculable.

- 1 : on peut lister les fonctions de PR dans l'ordre lexicographique
- 2 : $L = f_1, f_2, f_3 \dots f_n \dots \in F_1$ (si $f_i \in F_R$, $f_i(n) = f_i(n\,n\,\dots\,n)$)

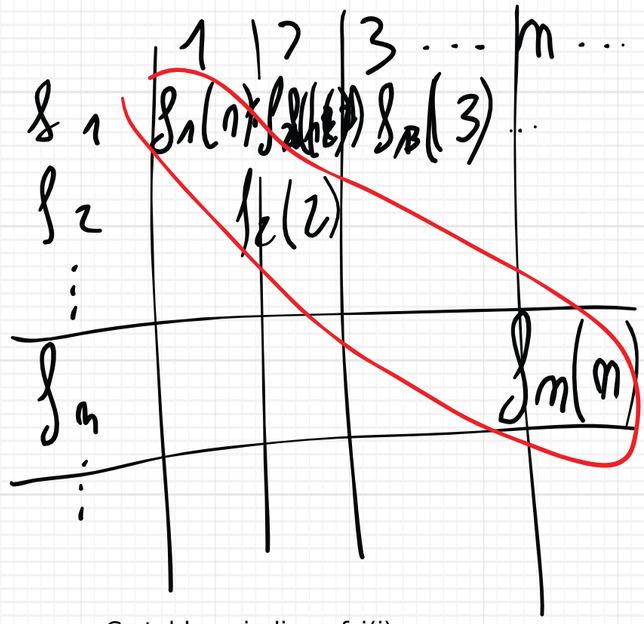
simplification ↑ justification ↑

3 : $h : n \longrightarrow f_n(n)$

4 : $g : n \longrightarrow h(n)+1$

$g \notin PR \Longrightarrow^{si}$ $g \in PR$ $\exists k$ / $g = f_k$

$\Longrightarrow g(k) = f_k(k) = h(k) = g(k)-1$!



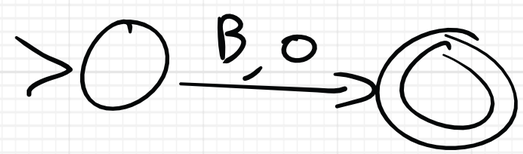
Ce tableau indique f_i(j)
(mais le panneau interactif a fait un peu n'importe quoi et j'ai pas compris tout de suite quand les étudiants me le faisait remarquer clairement)

$R \subseteq$ calculable

L'idée est de montrer que, pour toute fonction f de R, il existe une machine M qui calcule f.

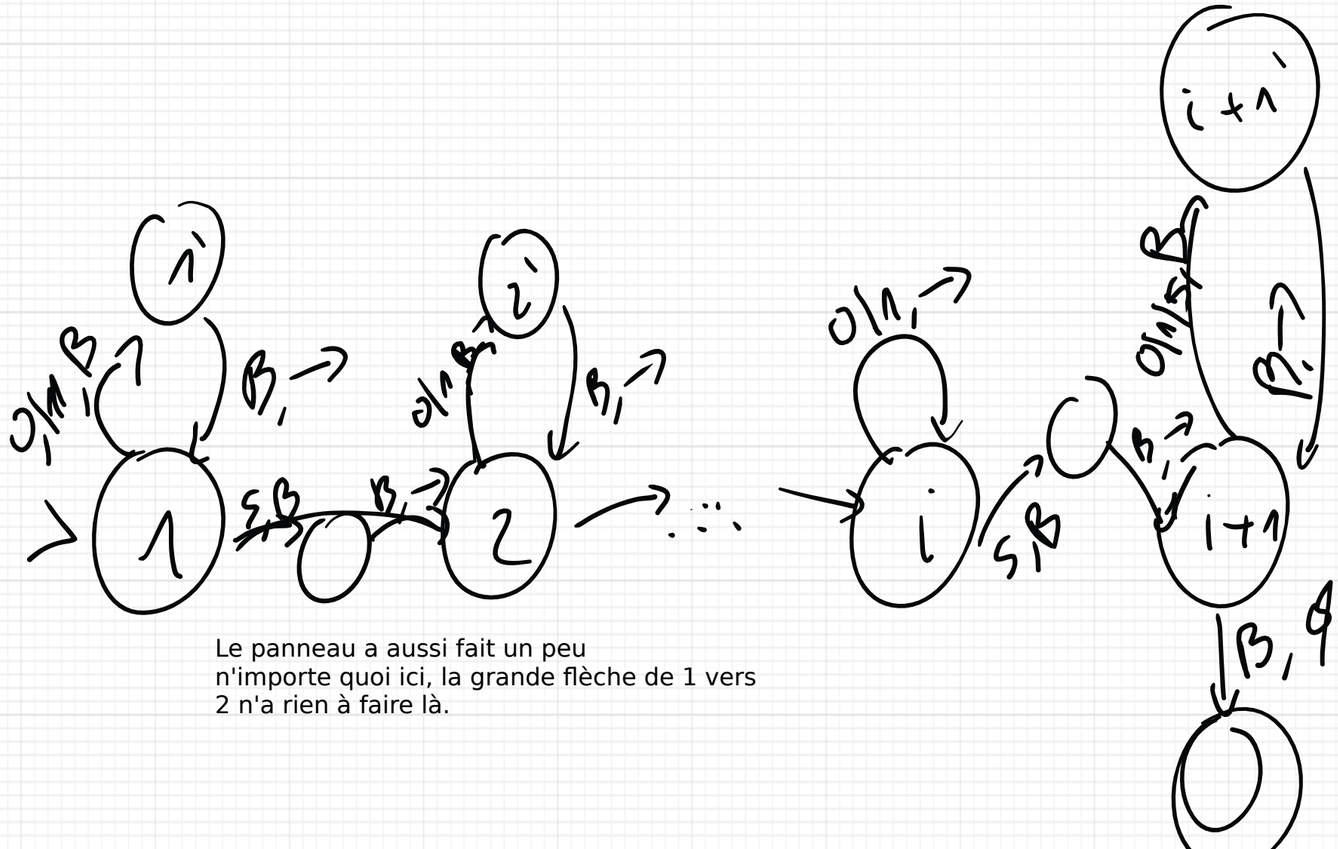
1 : on définit une machine par fonction initiale $(0, \sigma, P_{i,p})$ → On commence donc par construire explicitement une machine pour chaque fonction f initiale.

M_0 :



M_σ : Voir le cours précédent

$M_{i,p}$:



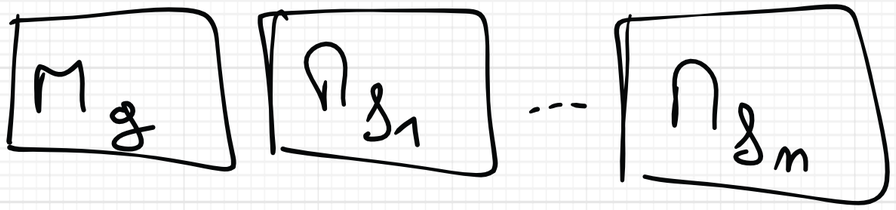
Le panneau a aussi fait un peu n'importe quoi ici, la grande flèche de 1 vers 2 n'a rien à faire là.

$R \subseteq$ calculable

Ensuite, pour chaque fonction qu'on peut construire par composition, récursion primitive et minimisation, on montre qu'il est possible de construire une machine qui calcule cette fonction.

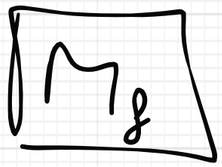
2 : définir $\circ$ et $\circlearrowleft$ et μ

$g \circ (f_1 - f_n)$: "Si je dispose d'une machine qui calcule g, f1, f2, \dots, fn, alors je peux construire une machine qui calcule g rond (f1, f2, ... fn)



: Appliquer M_{f_1} — M_{f_n}
Copier les résultats, Appliquer M_g

$\mu(f)$:

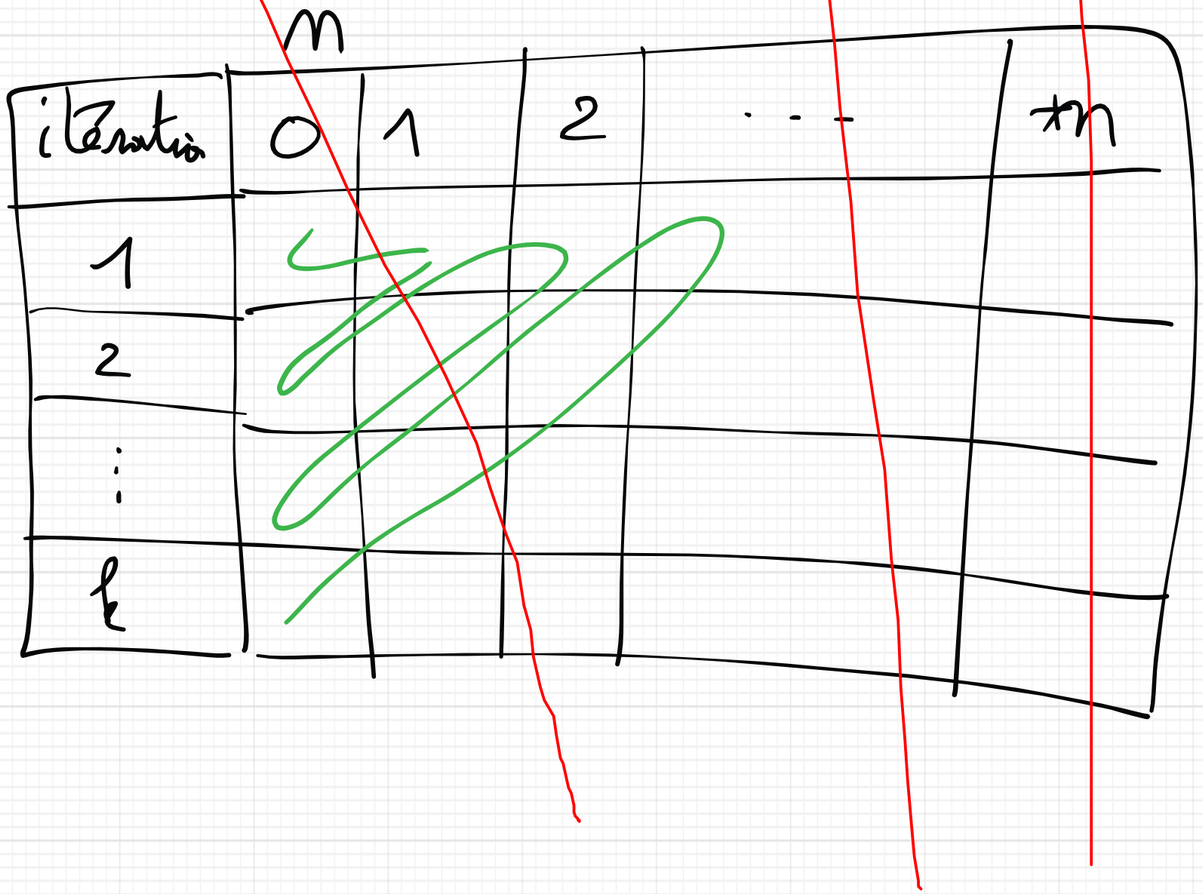


Idée 1 appliquer $f(x\,n)$ (M_f avec x et n en entrée
↳ que $f(x\,n) > 0$ ⚠ si $\forall n\ f(x,n) = 0$
→ pas grave $\mu(f(x))$ non définie
⇒ la machine M_μ ne s'arrête pas

Comment gérer f quand f est partielle (non définie partout) ?

⚠ si $f(x,n)$ non défini, on ne teste pas $f(x,n+1)$ car M_f ne s'arrête pas

Gérer μ avec les fonctions partielles



Je reviens sur ce que j'ai dit. J'ai été un peu trop confiant envers mes sources. (vive internet!)

En réalité, il n'est pas possible de construire une machine qui calcule mu(f) si f est partielle. Pour être exact, disposer d'une telle machine permettrait de décider du problème de l'arrêt.

Le problème est le suivant.

Comme expliqué en cours : une machine de Turing qui calcule une fonction partielle f s'arrête sur l'entrée x si et seulement si f(x) est définie.

Si on définit
mu(f)(x) = Le plus petit n tel que f(x, n) est défini et f(x, n) > 0.

Alors on ne peut pas appliquer l'algo de la slide précédente (calculer f(x,n) pour tout n de 0 à l'infini jusqu'à ce que f(x, n) > 0). Car alors on peut tomber sur un couple (x,n) où f n'est pas défini et la machine ne s'arrête pas.

Mais n'existe-t-il pas un autre algorithme qui calculerait mu(f)(x) ?
Hé bien non car si f est une fonction qui prend en entrée une machine x, une entrée y et un entier n telle que
- f(x, y, 0) = 1 si la machine x s'arrête avec l'entrée y et n'est pas définie sinon
- f(x, y, n) = 1 pour tout autre n, quels que soient x et y
Donc (x, y) s'arrête ssi mu(f)(x, y) = 0
Donc disposer d'une machine qui calcule mu(f) résolverait le problème de l'arrêt.

Un moyen de passer outre ce problème est de définir mu ainsi:
mu(f)(x) = Le plus petit n tel que f(x, n) > 0 et f(x, m) est défini pour tout m < n.
Si ce n n'existe pas mu(f)(x) n'est pas défini et l'algo précédent ne s'arrête pas,
ce qui est cohérent.
(le cours est corrigé sur ce point)

(source : j'ai posé la question sur Computer science stack exchange,
https://cs.stackexchange.com/questions/91608/can-turing-machines-simulate-the-unbounded-minization-operator-applied-to-a-part)

Problème de l'arrêt :

Supposons $\exists A$ machine qui répond
↪ Si l'arrêt est décidable

$A(n, x) = 1$ si $n(x)$ s'arrête
0 sinon

Soit P une machine "paradoxale"

tq $P(n, x)$:

Soit M tournes $A(n, x)$
si A répond 1 → P boucle à l'infini
0 → P s'arrête

n est une machine qui prend 1 machine en entrée.

$P(P, (M \cdot x)) \rightarrow$ Si $A(P, nx) = 1 \rightarrow P(nx)$ s'arrête

$P(P) \rightarrow$ Si $A(P,P) = 1 \rightarrow P(P)$ s'arrête et $P(P)$ boucle

Def de A

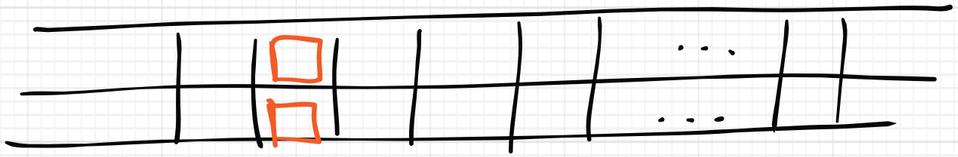
Sinon → $P(P)$ ne s'arrête pas et $P(P)$ s'arrête

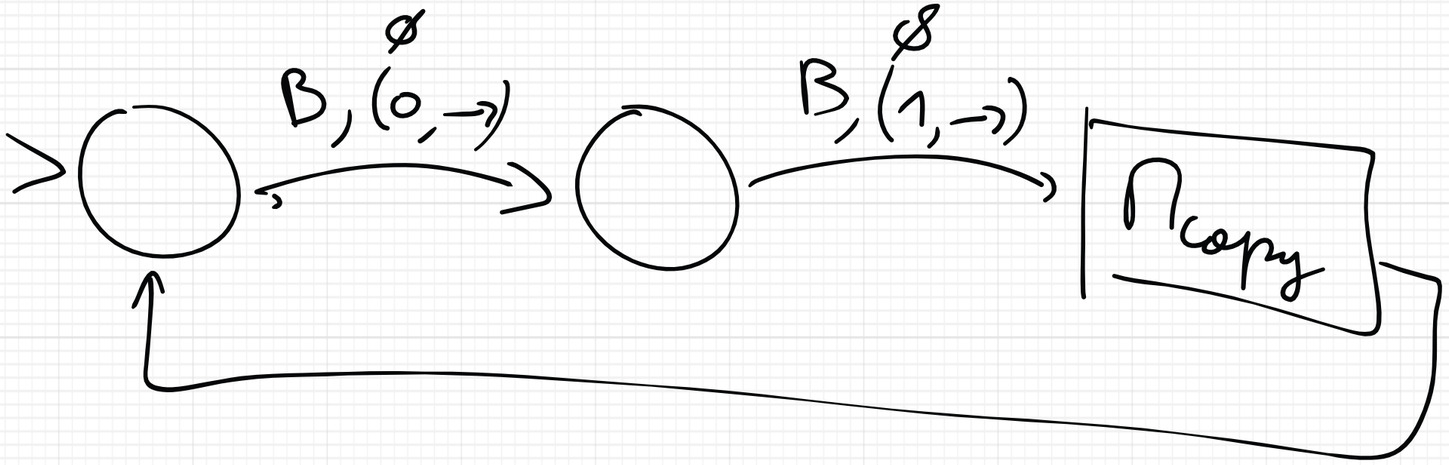
$\{(01)^*\} \in RE$

$=$

$\{(01)^n, n \in \mathbb{N}\}$

On a M_{copy} qui copie le mot en entrée sur sa droite





Cette machine ne s'arrête pas, car elle énumère tous les mots (01)* c'est à dire un ensemble infini.