3. Calculabilité

3.2.1. Notion de problème

Un **problème de décision** représente une question binaire (de réponse oui/non) et correspond à un ensemble dénombrable E **d'instances** muni d'un sous-ensemble $P \subseteq E$ des **instances positives** c'est-à-dire celles dont la réponse est oui.

Un **problème de décision** correspond à un ensemble dénombrable E et une partition en 2 classes sur E.

Un **problème de décision** correspond à un ensemble dénombrable E muni d'une fonction $f: E \to \{0,1\}$ qui à chaque instance du problème associe sa réponse (oui = 1, non = 0).

3.2.2. Notion de codage

Pour associer un langage à un problème, on utilise un codage.

Un **codage d'instances de E sur un alphabet fini Σ** correspond à une fonction injective $\langle\ \ \rangle: E \to \Sigma^*$, vérifiant quelques propriétés supplémentaires raisonnables : Σ doit avoir au moins 2 caractères pour pouvoir coder un nombre de taille $O(b^n)$ sur $O(n)$ caractères, le codage est supposé suffisamment régulier de sorte que l'on puisse passer d'un élément de $E = \{e_n : n \in \mathbb{N}\}$ au suivant par une fonction successeur calculable $s_\Sigma: \Sigma^* \to \Sigma^*: e \mapsto \begin{cases} \langle e_{n+1} \rangle_\Sigma \text{ si } e = \langle e_n \rangle_\Sigma \text{ et } n < \#E \\ \bot \text{ sinon} \end{cases}$. Cette dernière hypothèse est utile pour montrer que certaines notions de calculabilité sont indépendantes du codage considéré. C'est pourquoi on omet souvent de préciser le codage et son alphabet Σ, et on se contente souvent d'écrire $\langle\ \ \rangle$.

En général on suppose qu'il n'y a qu'un unique codage fixé par le contexte, qu'on qualifie informellement de « naturel » car on cherche à ce que celui-ci soit le plus simple possible.

Le codage d'une instance $x \in E$ est noté $\langle x \rangle_\Sigma \in \Sigma^*$.

Puisque Σ est fini, Σ^* est dénombrable, le codage étant injectif, E doit être dénombrable.

C'est pourquoi on suppose la dénombrabilité de E dans la définition de problème de décision.

On ne peut jamais avoir $E = R$ par exemple car il est alors impossible de définir un codage sur un alphabet fini.

Le langage d'un problème de décision (E, P) est $L_P = \{\langle x \rangle : x \in P\} \subseteq \Sigma^*$ l'ensemble des codages des réponses positives.

Puisque le codage est une injection, on peut identifier une instance de E avec son codage dans Σ^*.

Un **problème de décision** correspond donc à un ensemble $E \subseteq \Sigma^*$ muni d'une partie $L \subseteq E$ des instances positives de E (c'est le langage du problème).

Un **problème de décision** correspond à un ensemble $E \subseteq \Sigma^*$ muni d'une fonction $f: E \to \{0,1\}$.

Codages.

Un mot w sur un alphabet X peut être codé sur un autre alphabet Σ en $\langle w \rangle_\Sigma \in \Sigma^*$

L'ensemble des mots d'un alphabet fini X est dénombrable, donc un mot $w \in X^*$ peut être codé sur $\mathbb{N}$ en un nombre $\langle w \rangle_\mathbb{N} \in \mathbb{N}$.

Un entier $n \in \mathbb{N}$ peut être codé sur un alphabet de codage Σ, en $\langle n \rangle_\Sigma \in \Sigma^*$ par sa représentation par chiffres en base 10 par exemple. on ne peut pas le coder par lui-même car $\mathbb{N}$ serait un alphabet infini.

Exemple : $\langle 153 \rangle =$ "153" où on suppose $\{0,1,2,\dots,9\} \subseteq \Sigma$.

Un tuple $(x_1, \dots, x_k)$ où k est fixé, peut être codé sur un alphabet de codage, en $\langle (x_1, \dots, x_k) \rangle =$ "$\langle x_1 \rangle, \langle x_2 \rangle, \dots, \langle x_k \rangle$" où ','$\in\Sigma$ est un symbole spécial utilisé comme séparateur.

Un ensemble fini $X = \{x_1, \dots, x_p\}$, peut être codé sur un alphabet de codage, en $\langle X \rangle =$ "$\{\langle x_1 \rangle, \langle x_2 \rangle, \dots, \langle x_p \rangle\}$" où '{', '}' $\in\Sigma$ sont 2 symboles spéciaux utilisés comme début et fin d'un ensemble.

Un problème de décision modélise une question mathématique qui porte souvent sur des nombres, des tuple de nombres et/ou des ensembles finis de nombres. Donc plutôt que d'expliciter le codage et l'alphabet de codage, et pour suivre la forme de l'input d'un problème,

On peut représenter un problème comme un k-uplet de données, où chacune de ces données est en

général : soit un entier, soit un tuple, soit un ensemble fini, soit une composition de ces opérations. On note N l'ensemble de ces possibilités, de sorte que l'input d'un problème est un élément de N^k.
Un **problème de décision** peut se voir comme une partie $E \subseteq N^k$ muni d'une sous-partie $P \subseteq E$
Un **problème de décision** peut se voir comme une fonction partielle $f: N^k \to \{0,1\}$
Un problème de décision comme fonction partielle $f: N^k \to \{0,1\}$ permet de représenter une question fermée sur un n-uplet d'entiers x, et sa réponse oui/non donnée par l'image $f(x)$.
Un **problème computationnel/de calcul** peut se voir comme une fonction partielle $f: N^k \to N^l$, donc comme une relation binaire sur $N^k \times N^l$ donc comme un problème de décision particulier sur N^{k+l} avec $(x, y) \in P \Leftrightarrow y = f(x)$.
La notion de problème de décision englobe donc la notion de problème de calcul quelconque. On privilégie la vision problème de décision pour l'étude théorique car plus simple.
Un problème de calcul général comme fonction partielle $f: N^k \to N^l$ permet de représenter une question ouverte sur un n-uplet de données x, et sa réponse donnée par l'image $f(x)$.

3.2.3. Machines de Turing

Intuitivement : Une machine de Turing est constituée d'un ruban infini d'un côté, d'une tête de lecture, et d'un ensemble d'états, dont un état initial et un ensemble fini d'état accepteurs. On considère aussi un symbole spécial blanc #. Initialement, on est dans l'état initial, la tête de lecture sur la première case et un mot d'entrée est inscrit sur le ruban, suivi d'une infinité de #. À chaque étape, la machine lit le symbole sous la tête, remplace ce symbole, change d'état et déplace la tête à gauche ou à droite ou pas.
Une **machine de Turing (non déterministe)** correspond à un tuple $(Q, \Sigma, \Gamma, E, q_0, F, \#)$ avec Q ensemble fini **d'états**, $\Sigma \subseteq \Gamma \setminus \{\#\}$ **alphabet fini d'entrée**, Γ **alphabet fini de bande** contenant Σ et #, $E \subseteq Q \times \Gamma \times Q \times \Gamma \times \{\Leftarrow, \Rightarrow, \Downarrow\}$ un **ensemble fini de transitions** de la forme (p, a, q, b, x) notées $\boldsymbol{p, a \to q, b, x}$
$q_0 \in Q$ est **l'état initial**, $F \subseteq Q$ est **l'ensemble fini des états finaux**, $\# \in \Gamma$ est **le symbole blanc**.
Une autre façon de modéliser les transitions est de remplacer E par une **fonction de transition** $\boldsymbol{\delta}: Q \times \Gamma \to P(Q \times \Gamma \times \{\Leftarrow, \Rightarrow, \Downarrow\}): (p, a) \mapsto \{(q, b, x) \mid p, a \to q, b, x \in E\}$.
Une **machine de Turing est déterministe** ssi $\delta: Q \times \Gamma \to Q \times \Gamma \times \{\Leftarrow, \Rightarrow, \Downarrow\}$ et est une fonction partielle.
Une machine de Turing définit un graphe fini (multiple) dont les sommets sont les états, et chaque transition $p, a \to q, b, x$ représente un arc (p, q) étiqueté par le triplet (a, b, x)
a représente la **lettre lue**, b représente la **lettre écrite**, x représente le **déplacement** gauche ou droite.
Un **contenu de bande d'une machine de Turing** correspond à une suite de lettres de Γ indicée par N, et vérifiant l'invariant de bande : la bande ne contient que des # à partir d'un certain rang, càd la bande ne contient qu'un nombre fini de symboles non #.
Une **configuration d'une machine de Turing** correspond à un (q, B, i), avec $q \in Q$ l'état courant, B un contenu de bande, $i \in N$ la position de la tête sur la bande (son index).
Autrement dit une **configuration d'une machine de Turing** correspond à un (q, u, v) avec q l'état courant, $u \in \Gamma^*$ les lettres strictement à gauche de la tête jusqu'au début de la bande, $v \in \Gamma^*$ les lettres sur ou à droite de la tête jusqu'au dernier caractère non # inclus. (v est bien défini d'après l'invariant).
Un **calcul élémentaire = étape de calcul** correspond à un couple de configurations C, C' telles que :
- soit $C = (q, uc, av), C' = (q', u, cbv)$, et $q, a \to q', b, \Leftarrow \in E$
- soit $C = (q, u, av), C' = (q', ub, v)$, et $q, a \to q', b, \Rightarrow \in E$
- soit $C = (q, u, av), C' = (q', u, bv)$, et $q, a \to q', b, \Downarrow \in E$

On note: $\boldsymbol{C \vdash_M C'}$
L'invariant de bande est préservé à chaque étape de calcul.
Une **configuration est initiale** ssi $C = (q_0, \varepsilon, w)$ et $w \in \Sigma^*$ **mot d'entrée** fini donc suivi par infinité de #.
Une **configuration est finale** ssi $C = (q, u, v)$ avec $q \in F$
Un **calcul** correspond à une suite finie ou non d'étapes de calculs $C_1, C_2, \ldots$ telles que $C_1 \vdash C_2 \vdash \cdots$
Une **configuration d'arrêt/bloquante** est une configuration C telle qu'il n'existe aucune configuration C'

telle que $C \vdash C'$, on ne peut pas s'en échapper.
En général on suppose wlog (ne change pas le langage accepté), quitte à enlever les transitions sortant de configuration finales, que toutes les configurations finales sont des configurations d'arrêt.
Une **configuration de plantage/rejet** est une configuration d'arrêt qui n'est pas une configuration finale.
Une configuration d'arrêt est donc soit une configuration finale, soit une configuration de plantage.
Une **exécution** est un calcul qui part d'une configuration initiale, et qui ne peut être prolongé, donc qui est soit fini dans une configuration d'arrêt, soit infini. Intuitivement cela signifie laisser tourner la MT donnée sur une entrée donnée pour toujours.
Une **exécution est acceptante** ssi elle est finie et sa configuration d'arrivée est finale.
Une **exécution est plantée/rejetante** ssi elle est finie et sa configuration d'arrivée n'est pas finale.
Une **exécution s'arrête/se bloque** ssi elle est finie.
Une exécution finie est donc soit acceptante, soit plantée.
Une exécution peut être infinie
Donc on a 3 types d'exécutions possibles : acceptation, plantage, ou boucle infinie.
Dans une machine de Turing non déterministe, plusieurs exécutions parallèles peuvent avoir lieu, dans le cas d'une machine déterministe, une seule exécution est possible pour une configuration initiale donnée, càd pour un mot d'entrée donné.
Une **machine de Turing M accepte un mot $w \in \Sigma^*$** s'il existe au moins une exécution acceptante depuis ce mot d'entrée càd ssi $\exists q \in F \ (q_0, w, \varepsilon) \vdash_M^* (q, u, v)$
Une **machine de Turing M s'arrête sur un mot $w \in \Sigma^*$** ssi toutes ses exécutions depuis ce mot s'arrêtent.
Une **machine de Turing M s'arrête toujours/est sans calcul infini** ssi elle s'arrête sur tous les mots, donc toutes les exécutions possibles s'arrêtent.
Une machine de Turing M calcule un mot $w \in \Sigma^*$ à partir d'un mot $v \in \Sigma^*$ s'il existe une exécution acceptante dont la bande de la configuration finale est le mot w, et la configuration initiale est (q_0, ε, v).
Pour simplifier, on supposera wlog la configuration finale avec la tête sur la case 0 càd que $\exists q \in F \ (q_0, \varepsilon, v) \vdash_M^* (q, \varepsilon, w)$ de sorte à pouvoir aisément composer des machines (normalisées).
Le langage d'une machine de Turing M, est l'ensemble $L(M)$ des mots de Σ^* qu'elle accepte.
Deux machines de Turing sont **équivalentes (pour l'acceptation)** ssi elles ont le même langage.
S'il existe des calculs acceptants et non acceptants pour une même entrée, on considère quand même le mot comme accepté, donc la notion d'acceptation est dissymétrique. Cela pose problème pour la complémentation, comme pour les autres automates il faut se ramener à des machines déterministes, ou alors utiliser les machines alternantes qui sont plus générales.
Deux machines de Turing sont **complètement équivalentes** ssi elles sont équivalentes pour l'acceptation et s'arrêtent sur les mêmes entrées.

3.2.4. Graphe des configurations

Le **graphe des configurations d'une machine de Turing M** est le graphe de la relation $\vdash_M$ de calcul élémentaire sur les configurations. Un chemin dans ce graphe est donc un calcul de la machine M.
L'acceptation d'une entrée par la machine se ramène donc à un problème d'accessibilité dans le graphe des configurations. Le graphe des configurations est souvent infini, par contre on s'intéresse souvent à une partie finie de ce graphe.
Pour une machine qui s'arrête toujours, l'ensemble des configurations accessibles d'une configuration initiale donnée est fini. La recherche d'une configuration acceptante peut alors se faire par parcours du sous-graphe des configurations accessibles depuis cette configuration initiale.
Si on dispose d'une borne sur le temps ou l'espace utilisé par la machine sur une entrée de taille n, on a alors une borne sur la taille des configurations qui détermine un sous-graphe fini du graphe des configurations.
La déterminisation d'une machine de Turing se fait par BFS de l'arbre des calculs. Cet arbre est en fait un

dépliage du graphe des configurations. La machine déterministe équivalente à une machine M effectue en fait une BFS du graphe des configurations de M.

3.2.5. Normalisation

Pour une machine de Turing M, il existe une machine de Turing M' équivalente, qui se bloque ssi M se bloque, et telle que M' à 2 états q_+, q_- vérifiant $F' = \{q_+\}$, M' se bloque toujours et seulement sur les états q_+, q_-. C'est **la machine de Turing normalisée de M**. Ses propriétés sont très proches de M.
Le principe de la preuve est d'ajouter des transitions pour éviter les blocages.
q_+ symbolise un unique état final acceptant, et q^- symbolise un unique état final de plantage/rejet.
L'intérêt est de pouvoir composer des MT facilement.
Normaliser une machine de Turing déterministe, donne une machine de Turing déterministe.
Chaque calcul dans M est prolongé dans sa normalisation par au plus un calcul élémentaire.
Normaliser une machine de Turing sans calcul infini, donne une machine sans calcul infini.

3.2.6. Variantes

Une **machine de Turing à bande bi-infinie** est un objet mathématique formellement identique à celui d'une machine de Turing, seul le modèle (bande, configuration, calcul) change : on suppose que la bande est infinie des deux côtés donc indicée par $\mathbb{Z}$, avec l'invariant : Il n'y a que des # sur la bande dans les deux directions à partir d'un certain rang.
Une configuration est de la forme (q, u, v) avec q l'état courant, $u \in \Gamma^*$ les lettres strictement à gauche de la tête jusqu'au premier caractère non # inclus, $v \in \Gamma^*$ les lettres sur ou à droite de la tête jusqu'au dernier caractère non # inclus. La définition de calcul élémentaire s'adapte et maintient l'invariant de bande.
Equivalence. Toute machine de Turing est équivalente à une machine de Turing à bande bi-infinie.
Pour passer d'un ruban bi-infini à un ruban infini d'un seul côté, on pose $\Gamma' = \Gamma \times (\Gamma \cup \{\$\}), \Sigma' = \Sigma \times \{\#\}$, $\#' = (\#, \#)$, $Q' = Q \times \{+, -\}, F' = F \times \{+, -\}$, $\$ \notin \Gamma$. L'idée est de dupliquer les états suivant le côté de la bande bi-infini ou l'on est, et d'écrire deux informations sur la même case (a_{-i} et a_i). On replie la bande inferieure $i = -1, -2, \ldots$ en dessous de la bande supérieure $i = 1,2, \ldots$. On écrit \$ en bas pour $i = 0$. Comme \$ est nouveau aucune transition ne le lit, donc bloque la machine, donc permet de détecter la case 0.
Une **machine de Turing à k bandes** dispose de k bandes chacune lue par une tête de lecture indépendante.
Une transition est alors un élément de $Q \times \Gamma^k \times Q \times \Gamma^k \times \{\Leftarrow, \Rightarrow, \Downarrow\}^k$, ou $\Downarrow$ signifie laisser immobile une tête de lecture. Une étape correspond donc à un traitement de chacune des k têtes.
Plusieurs bandes permettent parfois une plus grande souplesse pour certain programmes, par exemple pour construction d'une machine universelle. On a imposé des alphabets identiques pour chaque bande, peu d'intérêt de les distinguer. Pour une bande, pas d'utilité d'avoir l'option de ne pas bouger la tête. Pour plusieurs bandes, il est parfois commode de pouvoir ne pas bouger une ou plusieurs têtes.
Une **configuration d'une machine de Turing à k bandes** correspond à $(q, B_1, \ldots, B_k, i_1, \ldots, i_k)$, l'état courant, les contenus des k bandes et les positions des k têtes respectives.
Equivalence. Toute machine de Turing à k bandes M est équivalente à une machine de Turing M' à une bande. De plus si M est déterministe alors M' aussi. De plus si M s'arrête toujours alors M' aussi.
NTM <=> DTM. Toute machine de Turing (non déterministe) M est équivalente à une machine de Turing déterministe M'. De plus si M s'arrête toujours alors M' aussi.
L'idée est de faire essayer à M' toutes les branches possibles du calcul de M depuis un mot d'entrée w avec une BFS dans le graphe des configurations. Si M' rencontre un état acceptant au cours de l'exploration, elle accepte l'entrée, sinon la simulation ne se termine pas.
M' est construite en utilisant 3 bandes, la première contient w et ne change pas, la deuxième contient une copie de la bande de M au cours du calcul le long d'une branche, la troisième contient un mot sur l'alphabet $\{1, \ldots, k\}$ (où k = nombre de transitions de M), qui détermine éventuellement un calcul de M.

Lemme de König. Tout arbre infini dont tout nœud est de degré fini admet une branche infinie.
Codage d'une machine de Turing sur $\mathbb{N}$. Pour une machine de Turing $M = (Q, X, \Gamma, E, q_0, F, \#)$,
$\langle Q \rangle = n_Q \in \mathbb{N}$ (n_Q représente le nombre d'états) $Q = \{1, \dots, n_Q\}$
$\langle \Gamma \rangle = n_\Gamma \in \mathbb{N}$ (n_Γ représente le nombre de symboles) $\Gamma = \{1, \dots, n_\Gamma\}$
$\langle X \rangle \subseteq \{1, \dots, n_\Gamma\}$ est une partie finie de $\mathbb{N}$
$\langle q_0 \rangle \in \{1, \dots, n_Q\}$ est un élément de $\mathbb{N}$.
$\langle F \rangle \subseteq \{1, \dots, n_Q\}$ est une partie finie de $\mathbb{N}$
On code $\{\Leftarrow, \Rightarrow, \Downarrow\}$ par $\{1,2,3\}$ dans $\mathbb{N}$.
Une transition est codée par $(p, a, q, b, x) \in Q \times \Gamma \times Q \times \Gamma \times \{\Leftarrow, \Rightarrow, \Downarrow\}$
E est donc une partie finie de $\mathbb{N}^5$
Finalement le codage noté $\langle M \rangle$ d'une machine de Turing M, se voit comme une partie finie de $\mathbb{N}^{10}$.
Après codage, $\langle M \rangle$ est encore une machine de Turing qui est totalement équivalente à M.
A priori la définition d'une MT suggère trop de possibilités pour les compter. On identifie deux machines de Turing ssi elles sont totalement équivalentes. Un représentant canonique de cette classe, est donc son codage. Par conséquent il n'y a pas plus de machine de Turing que le nombre de codage de MT, or il y a une infinité dénombrable de parties finies de $\mathbb{N}^k$.
Il y a donc une infinité dénombrable de machine de Turing distinctes.
Finalement on peut coder sur un alphabet de codage Σ, une MT d'alphabet X, en un mot $\langle M \rangle_\Sigma \in \Sigma^*$.
Machine de Turing Universelle
Le **langage sur Σ d'acceptation des machines sur X** est le langage
$\boldsymbol{L_\epsilon^{\Sigma,X}} = \{\langle M, w \rangle_\Sigma : M \text{ machine de Turing sur X}, w \in X^*, M \text{ accepte } w\} \subseteq \Sigma^*$
En général on ne distingue pas l'alphabet de la machine de celui sur lequel on la code ($X = \Sigma$) et on en fait même pas mention. $L_\epsilon = \{\langle M, w \rangle : M\ MT, M \text{ accepte } w\}$
Le choix du codage est général sans importance car on peut passer d'un codage à un autre par MT.
MTU. Une **machine de Turing codée sur Σ universelle sur X**, est une machine M_U sur Σ dont le langage est le langage d'acceptation: $L_{M_U} = L_\epsilon^{\Sigma,X}$, autrement dit M_U accepte $\langle M, w \rangle$ ssi M accepte w.
On peut toujours construire une telle machine.
On construit M_U avec 2 bandes : La 1e reçoit l'entrée $\langle M, w \rangle$, la 2e est la configuration courante initialement $\langle q_0, \varepsilon, w \rangle$. Pour chaque étape, M_U recherche dans $\langle M \rangle$ une transition applicable puis l'applique pour obtenir la configuration suivante sur la 2e bande. M_U accepte si la configuration est finale pour M. Une exécution acceptante dans M_U correspond donc à une exécution acceptante dans la machine M considérée. M_U accepte donc les mêmes mots. $L_{M_U} = L_\epsilon$.
Attention : Une machine universelle M_U ne peut pas toujours s'arrêter. Il suffit de considérer le programme Q: $\langle M \rangle \mapsto$ **if** M_U accepte $\langle M, \langle M \rangle \rangle$ **then** rejeter **else** accepter
appliqué à lui même $\langle Q \rangle$, M_U ne peut pas s'arrêter sur $\langle Q, \langle Q \rangle \rangle$
Le langage d'arrêt est $H = \{\langle M, w \rangle_\Sigma : M\ MT, M \text{ s'arrête sur l'entrée } w\} \subseteq \Sigma^*$
Problème de l'arrêt. Aucune machine qui s'arrête toujours n'accepte le langage d'arrêt. (argument diag)
3.3. Langages récursivement énumérables
On suppose généralement qu'on se place sur un alphabet fixé Σ fini avec au moins 2 symboles.
Un **langage est récursivement énumérable = semi-récursif = semi-décidable = $\boldsymbol{RE}$** (sur Σ) ssi c'est le langage accepté par une machine de Turing (sur Σ).
Un **problème est récursivement énumérable** ssi son langage l'est, càd si ses instances positives peuvent être énumérées.
Un langage/problème est $\boldsymbol{coRE}$ si son complémentaire est RE.
Un **enumérateur** $(M, \Sigma, \$)$ est une machine de Turing déterministe M multi-bandes qui écrit sur une bande de sortie des mots d'un alphabet Σ séparés par un symbole $\$ \notin \Sigma$, telle que la tête de sortie ne se déplace jamais à gauche, M ne prend pas d'entrée (alphabet d'entrée = vide), la bande de sortie n'est

initialement que des blancs. M étant déterministe et sans entrée, il n'y a qu'une unique exécution possible finie ou non.
Un **mot $w \in \Sigma^*$ est énuméré par un enumérateur** ssi il est écrit sur la bande de sortie entre deux séparateurs \$ « après » (lors de) exécution de l'enumérateur.
Un **langage est énuméré par un enumérateur** ssi tous ses mots le sont.
Un langage $L \subseteq \Sigma^*$ est RE ssi il est énuméré par un enumérateur $(M, \Sigma, \$)$
Enumérateur de L à partir d'une machine de Turing M déterministe acceptant L :
1. **for all** $k \geq 0$ **do**
2. **for all** w tel que $|w| \leq k$ **do**
3. **if** M accepte w en au plus k étapes **then**
4. écrire w suivi de \$ sur la bande de sortie

Machine de Turing déterministe acceptant L à partir d'un enumérateur E de L : **input** w
1. **loop**
2. **repeat**
3. exécuter l'enumérateur E
4. **until** l'enumérateur écrit \$
5. **if** w est égal au mot écrit **then**
6. accepter

Un langage fixé est dénombrable. En particulier Σ^* est dénombrable.
L'ensemble de tous les langages n'est pas dénombrable. $P(N)$ pas dénombrable.
L'ensemble des machines de Turing et donc des langages RE est dénombrable.
Il existe des langages non RE. (un exemple explicite s'obtient par argument diagonal).
L'intersection finie de langages RE est un langage RE.
L'union finie de langages RE est un langage RE.
Exercice : TODO

3.4. Langages décidables

Un **langage $L \subseteq \Sigma^*$ est décidable = récursif = R** ssi c'est le langage accepté par une machine de Turing qui s'arrête toujours. On dit que la **machine M décide le langage L**, c'est plus fort que juste accepter L car pour accepter, rien ne garantit que M s'arrête en dehors de L.
Un langage décidable est en particulier récursivement énumérable. $R \subset RE$
L'intersection finie de langages R est un langage R.
L'union finie de langages R est un langage R.
Le complémentaire dans Σ^* d'un langage $L \subseteq \Sigma^*$ récursif, est un langage récursif. $coR = R$.
$\boldsymbol{RE \cap coRE = R}$**.** Un langage RE dont le complémentaire est RE, s'avère être récursif (et son complémentaire aussi).
Le langage d'acceptation L_ϵ est RE mais n'est pas R, donc n'est pas coRE.
Le langage de l'arrêt H n'est pas R.
On peut énumérer toutes les machines de Turing possibles avec les mots qu'elles acceptent.
Une **fonction $f: \Sigma^* \to \Gamma^*$ est calculable** ssi il existe une machine de Turing qui pour tout entrée $w \in \Sigma^*$ calcule $f(w)$ en temps fini.
La composée de fonctions calculables est calculable.
Une **réduction d'un problème A de langage $L_A \subseteq \Sigma_A^*$, à un problème B de langage $L_B \subseteq \Sigma_B^*$** notée $\boldsymbol{A \leq_m B}$ **(A se reduit à B)** correspond à une fonction $f: \Sigma_A^* \to \Sigma_B^*$ calculable telle que $w \in L_A \Leftrightarrow f(w) \in L_B$ càd que c'est une transformation algorithmique d'une instance de A en une instance de B telle que répondre à B signifie répondre à A.
$\leq_m$ est réflexive et transitive
Si $A \leq_m B$ et B décidable, alors A décidable. Très utile pour prouver la décidabilité.

Si $A \leq_m B$ et A indécidable, alors B indécidable. Très utile pour prouver l'indécidabilité.
Le langage des machines de Turing de langage non vide $L_\emptyset = \{\langle M\rangle : L(M) \neq \emptyset\}$ est indécidable. (Rice)
Le langage $L_{\neq} = \{\langle M, M'\rangle \mid L(M) \neq L(M')\}$ est indécidable. (Rice)
Une **propriété P est non triviale sur RE** ssi $\exists L, L' \in RE \;\; P(L)$ et non $P(L')$
Rice*. Pour une propriété P non triviale sur RE, le problème de savoir si le langage $L(M)$ d'une machine de Turing M vérifie P est indécidable.
Càd $L_P = \{\langle M\rangle : M \text{ machine de Turing et } P(L(M))\}$ indécidable.
Preuve : On montre $L_\epsilon \leq_m L_P$ avec la réduction $f: \langle M, w\rangle \mapsto \langle M_w\rangle$ avec M_w la machine : $u \mapsto$
$if\ M$ accepte $w\ then$ simuler M_0 sur $u\ else$ rejeter
avec M_0 une machine de Turing fixée telle que $P(L(M_0))$ ($\exists$ car P non triviale).
Si $L \in RE$ alors $L^* \in RE$
Si $L \in R$ alors $L^* \in R$
Si $L \notin R$ alors $K = 0L + 1(\Sigma^* \setminus L)$ n'est ni RE ni $coRE$.
Il existe un langage RE de complémentaire infini, dont l'intersection avec tout langage RE infini est non vide.

Compléments codage.

Un codage d'un ensemble dénombrable $E = \{e_n\}_n$ sur un alphabet à au - 2 lettres Σ, correspond à:
Une fonction $\langle\ \ \rangle_\Sigma : E \to \Sigma^*$ injective,
Une machine de Turing S_Σ qui calcule $s_\Sigma: \Sigma^* \to \Sigma^*: e \mapsto \begin{cases} \langle e_{n+1}\rangle_\Sigma \text{ si } e = \langle e_n\rangle_\Sigma \text{ et } n < \#E \\ \perp \text{ sinon} \end{cases}$
Soit $(\Sigma, \langle\ \ \rangle_\Sigma, S_\Sigma)$ codage de $E = \{e_n\}_n$ et $(\Sigma', \langle\ \ \rangle_{\Sigma'}, S_{\Sigma'})$ codage de $F = \{f_n\}_n$
On définit $\langle \boldsymbol{D}\rangle_\Sigma = \{\langle e\rangle_\Sigma : e \in D\}$ **le codage d'un ensemble D de E**
On définit $\langle \boldsymbol{f}\rangle_\Sigma : \langle D\rangle_\Sigma \to \langle F\rangle_{\Sigma'} : \langle e\rangle_\Sigma \mapsto \langle f(e)\rangle_{\Sigma'}$ **le codage d'une fonction $f: D \subseteq E \to F$.**
On définit $t_{\Sigma\to\Sigma'}: \langle E\rangle_\Sigma \subseteq \Sigma^* \to \langle F\rangle_{\Sigma'} \subseteq {\Sigma'}^* : \langle e\rangle_\Sigma \mapsto \langle e\rangle_{\Sigma'}$ **la fonction de transcodage de Σ à Σ'.**
La fonction de transcodage est bijective et vérifie $\langle\ \ \rangle_{\Sigma'} = t_{\Sigma\to\Sigma'} \circ \langle\ \ \rangle_\Sigma$
La fonction de transcodage est MT calculable. **Preuve.** Elle est calculée par la MT suivante:
input $a \in \Sigma^*$ **:**
if (simuler S_Σ sur a) $= \perp$ **then return** $\perp$ //vérifie si a est bien dans $\langle E\rangle_\Sigma$ sinon renvoie $\perp$.
$x \coloneqq \langle e_0\rangle_\Sigma$, $y \coloneqq \langle f_0\rangle_{\Sigma'}$
while $x \neq a$:
    simuler S_Σ sur x
    simuler $S_{\Sigma'}$ sur y
return y
$\langle f\rangle_\Sigma$ est MT semi-calculable ssi $\langle f\rangle_{\Sigma'}$ l'est. Car $\langle f\rangle_{\Sigma'} = t_{\Sigma\to\Sigma'} \circ \langle f\rangle_\Sigma \circ t_{\Sigma'\to\Sigma}$ est MT semi-calculable comme composée de fonctions MT semi-calculables.
$\langle f\rangle_\Sigma$ est MT calculable ssi $\langle f\rangle_{\Sigma'}$ l'est. Car $\langle f\rangle_{\Sigma'} = t_{\Sigma\to\Sigma'} \circ \langle f\rangle_\Sigma \circ t_{\Sigma'\to\Sigma}$ est MT calculable comme composée de fonctions MT calculables.
En conséquence, semi calculabilité / calculabilité sont indépendantes du codage choisi moyennant quelques hypothèses raisonnables.

Compléments Fonction/Problème calculable/semi-calculable.

Soit un problème de calcul vu comme une fonction partielle $f: D \subseteq N^k \to N^l$ ou comme le problème de décision sur N^{k+l} de langage $L_P = \{\langle x, f(x)\rangle : x \in D\}$

f est $\boldsymbol{RE}$**/semi-calculable/semi-récursive** ssi il existe une machine de Turing qui calcule pour toute entrée ayant une solution $\langle x \in D \rangle$ la solution $\langle f(x) \rangle$.
Une machine de Turing M accepte le langage d'un problème ssi sa fonction f est semi-calculable.
MT calculatrice M' à partir d'une MT acceptrice M : **input** $\langle x \in N^k \rangle$
for all $k \geq 0$ **do** **for all** $\langle w \rangle$ tel que $|w| \leq k$ **do**
if M accepte $\langle x, w \rangle$ en au plus k étapes **then** écrire en sortie w et stop
Acceptrice à partir d'une MT calculatrice : **input** $\langle x \in N^k, w \rangle$, calculer $\langle f(x) \rangle$ accepter ssi $\langle w \rangle = \langle f(x) \rangle$
On pourra abusivement considérer $f \in RE$ / $P \in RE$ / $L \in RE$
f est $\boldsymbol{R}$**/ calculable/récursive** ssi il existe une machine de Turing qui s'arrête toujours, qui calcule pour toute entrée $\langle x \in N^k \rangle$ la solution $\langle f(x) \rangle$ si $x \in D$, ou alors s'arrête sans output si $x \notin D$.
Une fonction totale semi-calculable est calculable, puisqu'alors la machine de Turing qui la calcule s'arrête pour tout input possible.
A proprement parler, une fonction calculable n'est pas forcement totale, mais on peut l'étendre en une fonction totale (définie sur N^k), toujours calculable, en posant par exemple $f(x) = \perp$ pour $x \notin D$. Avec $\perp$ un symbole non utilisé.
Une machine de Turing M décide le langage d'un problème ssi la fonction du problème f est calculable.
MT calculatrice M' à partir d'une MT acceptrice M : **input** $\langle x \in N^k \rangle$
for all $k \geq 0$ **do** **for all** $\langle w \rangle$ tel que $|w| \leq k$ **do**
if M rejette x en au plus k étapes **then** output $\perp$ et stop
if M accepte $\langle x, w \rangle$ en au plus k étapes **then** output w et stop
Acceptrice à partir d'une MT calculatrice : **input** $\langle x \in N^k, w \rangle$, calculer $\langle f(x) \rangle$ accepter ssi $\langle w \rangle = \langle f(x) \rangle$
On pourra abusivement considérer $f \in R$ / $P \in R$ / $L \in R$
Ensemble RE/R d'entiers.
Un ensemble $A \subseteq N$ est **RE/semi-récursif/semi-calculable** ssi son indicatrice l'est.
Un ensemble $A \subseteq N$ est **R/récursif/calculable** ssi son indicatrice l'est.

3.5. Problème de correspondance de Post

3.5.1. Présentation

Le **problème de correspondance de Post PCP sur un alphabet fini Σ a au - 2 symboles** est défini par:
Une **instance de PCP** est constituée de 2 listes de même longueur $u_1, \dots, u_m, v_1, \dots, v_m$, de mots sur Σ.
On peut l'écrire comme liste de paires $\big((u_1, v_1), \dots, (u_m, v_m)\big)$ que l'on visualise comme des dominos.
Une **solution d'une instance de PCP** est une suite de $n \geq 1$ indices $(i_1, \dots, i_n) \in \{1, \dots, m\}^n$ telle que les concaténations $u_{i_1} \dots u_{i_n} = v_{i_1} \dots v_{i_n}$ sont égales.
Le problème est de savoir si au moins une solution existe.
Autrement dit une **instance de PCP est positive** si elle admet au moins une solution de PCP.
PCP peut être formulé en termes de morphismes. Pour une instance de PCP $\big((u_1, v_1), \dots, (u_m, v_m)\big)$ sur Σ, un alphabet avec autant d'éléments (m) $A = \{a_1, \dots, a_m\}$ disjoint de Σ, et deux morphismes de mots $\mu, \nu : A^* \to \Sigma^*$ définis par $\mu(a_i) = u_i, \nu(b_i) = v_i$, une solution de l'instance de PCP correspond à un mot $w \in A^*$ pour lequel les morphismes coïncident $\mu(w) = \nu(w)$.

3.5.2. Indécidabilité*

PCPM. Le **PCP modifié sur un alphabet fini Σ a au - 2 symboles** est défini (pour démontrer indécid.) par:
Une **instance de PCPM** est constituée de 2 listes de même longueur $u_1, \dots, u_m, v_1, \dots, v_m$, de mots sur Σ.

Une **solution d'une instance de PCPM** est une suite de $n \geq 1$ indices $(i_1, \dots, i_n) \in \{1, \dots, m\}^n$ telle que $i_1 = 1$ et $u_1 u_{i_1} \dots u_{i_n} = v_1 v_{i_2} \dots v_{i_n}$. Une **instance de PCPM est positive** si elle a au moins une solution.

Post 1946. PCP et PCPM sont indécidables.

Indécidabilité de PCP permet de montrer (par réduction à PCP) l'indécidabilité de questions très naturelles sur les grammaires algébriques.

3.5.3. Application aux grammaires algébriques*

Pour une instance de PCP $\big((u_1, v_1), \dots, (u_m, v_m)\big)$ sur Σ, un alphabet avec autant d'éléments $A = \{a_1, \dots, a_m\}$ disjoint de Σ, on définit

$\boldsymbol{L_u} = \{u_{i_1} \dots u_{i_n} a_{i_n} a_{i_{n-1}} \dots a_{i_1} : n \in N, (i_1, \dots, i_n) \in \{1, \dots, m\}^n\} \subseteq (A + \Sigma)^*$

$\boldsymbol{L'_u} = \{w a_{i_n} a_{i_{n-1}} \dots a_{i_1} : n \in N, (i_1, \dots, i_n) \in \{1, \dots, m\}^n, w \in \Sigma^*, w \neq u_{i_1} \dots u_{i_n}\} \subseteq (A + \Sigma)^*$

L_u est algébrique car engendré par S via la grammaire algébrique: $S \to \sum_{i=1}^{n} u_i S a_i + \varepsilon$

L'_u est algébrique car engendré par S via la grammaire algébrique:

$$\begin{cases} S \to \sum_{i=1}^{m} u_i S a_i + \sum_{\substack{i=1..m \\ |u|=|u_i| \\ u \neq u_i}} u R a_i + \sum_{\substack{i=1..m \\ |u|<|u_i|}} u T a_i + \sum_{\substack{i=1..m \\ b \in \Sigma}} u_i b V a_i \\ R \to \sum_{i=1}^{m} R a_i + \sum_{b \in \Sigma} b R + \varepsilon, T \to \sum_{i=1}^{m} T a_i + \varepsilon, V \to \sum_{b \in \Sigma} b V + \varepsilon \end{cases}$$

On a $(A + \Sigma)^* \setminus L_u = L'_u \cup ((A + \Sigma)^* \setminus \Sigma^* A^*)$

Pour 2 grammaires algébriques $(G, S), (G', S)$, est-ce que leurs langages sont <u>disjoints</u> $L_G(S) \cap L_{G'}(S') = \emptyset$? est indécidable.

Pour 2 grammaires algébriques $(G, S), (G', S)$, est-ce que leurs langages sont égaux $L_G(S) = L_{G'}(S')$? est indécidable.

Est-ce qu'une grammaire algébrique (G, S) engendre tous les mots possibles $L_G(S) = A^*$? est indécidable.

Est-ce qu'une grammaire algébrique (G, S) est ambiguë ? est indécidable.

Est-ce qu'un langage rationnel K est inclus dans un langage algébrique L ? est indécidable.

Est-ce qu'un langage algébrique K est inclus dans un langage rationnel L ? est <u>décidable</u>.

3.6. Théorème de récursion

Un **Quine** M est une MT sur Σ qui n'a pas d'input et qui output son propre code $\langle M \rangle_\Sigma$.

Les Quine existent : On utilise des MT normalisées pour définir la composée MM' de 2 MT facilement.

1. Soit C la MT avec input deux codages de MT $\langle M \rangle, \langle M' \rangle$, qui output le codage $\langle MM' \rangle$
2. Pour un mot w soit P_w la MT sans input, qui output w
3. Soit R_0 la MT avec input un mot w, qui output le codage $\langle P_w \rangle$

Avec 1)2)3) Soit R la MT avec input un mot de la forme $w = \langle M \rangle$ qui output le codage $\langle P_w M \rangle$ de la composition. Alors la machine $Q = P_{\langle R \rangle} R$ est un Quine. $\varepsilon \to^{P_{\langle R \rangle}} \langle R \rangle \to^{R} \langle P_{\langle R \rangle} R \rangle$.

Récursion. Pour une fonction $t: \Sigma^* \times \Sigma^* \to \Sigma^*$ calculable, il existe une machine de Turing M qui calcule la fonction $\Sigma^* \to \Sigma^*: w \mapsto t(w, \langle M \rangle)$

Ce théorème est central en théorie de la calculabilité. Généralisation de l'existence de Quine. Beaucoup de langages de programmation permettent la définition récursive de fonctions. Le théorème de récursion autorise une forme beaucoup plus générale de récursivité en permettant la manipulation même du code de la fonction.

Théorème de point fixe. Pour une fonction f calculable, qui à un codage de MT associe un autre codage de MT, il existe une MT M telle que $f(M)$ est équivalente à M.

Autrement dit toute fonction calculable qui transforme une MT en une MT admet un « point fixe ».
Il suffit de considérer la MT M d'input w, qui calcule $M' \coloneqq f(\langle M \rangle)$ et simule M' sur w. (cette définition est OK d'après th récursion). Le th de récursion permet de reprouver l'indécidabilité de L_ϵ.

3.7. Machines linéairement bornées (MTLB)

3.7.1 Définition

Une **MT est linéairement bornée** ssi elle n'écrit pas en dehors de l'espace utilisé par le mot d'entrée.
Comme on peut toujours supposer qu'une MT n'écrit pas # (en le dupliquant éventuellement en #'),
Une MT est linéairement bornée ssi toute transition qui lit un #, écrit # et va à gauche.
L'espace utilisé par une MTLB peut être artificiellement augmenté en ajoutant de nouveaux symboles à l'alphabet de bande. Cet espace disponible reste proportionnel à la taille de l'entrée car l'alphabet de bande est fini.

3.7.2. Grammaires contextuelles

Une **grammaire formelle $\boldsymbol{G = (A, V, P, S)}$ est contextuelle** ssi toutes ses règles sont de la forme :
1) $S \to \varepsilon$
2) $uTv \to uwv$ avec $T \in V,\ u, v \in (A + V)^*,\ w \in (A + V \setminus \{S\})^+$ mot non vide sans lettre S.
Une **grammaire formelle est croissante** ssi chaque règle $w \to w'$ verifie $|w| \leq |w'|$.
Une grammaire contextuelle sans règle $S \to \varepsilon$ est croissante.
Un langage $L \subseteq A^+$ de mots non vides, est engendré par une grammaire contextuelle ssi il est engendré par une grammaire croissante. (Une grammaire croissante peut être transformée en grammaire contextuelle équivalente).
Un langage est engendré par une grammaire contextuelle ssi il est accepté par une MTLB.

3.7.3. Décidabilité

« Est-ce qu'une MTLB M accepte un mot w ? » est décidable. (nb de configs = $n|Q||\Gamma|^n$ donc graphe fini)
« Est-ce qu'une MTLB M n'accepte aucun mot ? » est indécidable.

3.7.4. Complémentation

Immerman et Szelepcsényi.* Pour une MTLB non-déterministe M, il existe une MTLB non-déterministe M' qui accepte le langage complémentaire. $L(M') = \Sigma^* \setminus L(M)$
Pour $s\colon N \to N$, une **MT est d'espace $\boldsymbol{s}$** ssi pour toute entrée w, toute configuration accessible depuis (q_0, ε, w) est de longueur $\leq s(|w|)$.
Immerman et Szelepcsényi 1987. Pour une MT ND d'espace $s(n) \geq \log n$, il existe une MT ND d'espace Ks qui accepte le langage complémentaire. ($K \in R_+$)

3.8. Décidabilité de théories logiques

On considère la logique du premier ordre ou le modèle est l'ensemble N des entiers.
Une **théorie logique est décidable** ssi pour toute formule close φ, « Est-ce que φ est vrai ? » est décidable.
L'arithmétique de Presburger est la théorie du premier ordre des entiers munis de l'addition mais pas de la multiplication. TODO préciser.
L'arithmétique de Presburger est décidable (1929)*.
Tarski 1936. L'arithmétique de Peano est indécidable.

3.9. Fonctions récursives

Fonctions récursives = modèle alternatif permettant de définir la calculabilité.

3.9.1. Fonctions primitives récursives

L'identité est la fonction $\boldsymbol{id}: N^k \to N^k: (n_1, \dots, n_k) \mapsto (n_1, \dots, n_k)$

Une **fonction constante** est de la forme $\boldsymbol{c}: N^k \to N: (n_1, \dots, n_k) \mapsto c$

Pour $i \in \{1, \dots, k\}$, **la ième projection** est la fonction $\boldsymbol{p_{k,i}}: N^k \to N: (n_1, \dots, n_k) \mapsto n_i$

Pour $r \in N$, **la $r$ieme fonction de duplication** est $d_r: N \to N^r: n \mapsto (n, n, \dots, n)$

La **fonction successeur** est $s: N \to N: n \mapsto n + 1$

Une fonction **primitive récursive <u>basique</u>** est une de ces 5 derniers types.

Composition. Pour $f_1: N^k \to N^{k_1}, \dots, f_p: N^k \to N^{k_p}$, et $g: N^{k_1+\dots+k_p} \to N^r$, alors la composée est $\boldsymbol{g(f_1, \dots, f_p)}: N^k \to N^r: (n_1, \dots, n_k) \mapsto g\left(f_1(n_1, \dots, n_k), \dots, f_p(n_1, \dots, n_k)\right)$

Itération. Pour $f: N^k \to N^r$ et $g: N^{k+r+1} \to N^r$ la fonction $h = \boldsymbol{Rec(f, g)}$ est définie récursivement pour tout $n \in N, \overline{m} = (m_1, \dots, m_k) \in N^k$, par $\begin{cases} h(0, \overline{m}) = f(\overline{m}) \\ h(n+1, \overline{m}) = g(n, h(n, \overline{m}), \overline{m}) \end{cases}$

Intuitivement, $h(n, \overline{m})$ représente le résultat d'une boucle for de 0 à $n - 1$ inclus. $\overline{m}$ représente les données initiales. f représente le code initialisateur avant la boucle. g représente le code dans la boucle paramétré par l'indice courant, le résultat de l'étape précédente, et les données initiales.

Une **fonction primitive récursive** est définie inductivement :

1. Une fonction primitive récursive basique est primitive récursive
2. La composition $g(f_1, \dots, f_p)$ de fonctions primitives récursives est primitive récursive.
3. L'itération $Rec(f, g)$ de fonctions primitives récursives est primitive récursive.
4. Il n'y a pas d'autres fonctions primitives récursives.

Autrement dit c'est la plus petite classe contenant les fonctions basiques, close par composition et itération.

Exemples de fonctions primitives récursives:

La somme $+: N^2 \to N$ car $add(0, m) = m,\ add(n+1, m) = s\big(add(n, m)\big).\ add = Rec\big(p_1, s(p_2)\big)$

Le prédécesseur $p: N \to N$ car $p(n) = \max(0, n-1)$ càd $p(0) = 0, p(n+1) = n$

La différence (0 si négative) $sub: N^2 \to N$ car $sub(n, 0) = n, sub(n, m+1) = p\big(sub(n, m)\big)$

Le produit $mul: N^2 \to N$ car $mul(0, m) = 0, mul(n+1, m) = add(mul(n, m), m)$

L'égalité à 0 $eq_0: N \to \{0,1\}$ car $eq_0 = Rec(1,0)$

L'égalité entre 2 entiers $eq: N^2 \to \{0,1\}$ car $eq = eq_0\big(sub(p_1, p_2) + sub(p_2, p_1)\big)$

Le quotient d'une DE : $div: N^2 \to N$ car $div(0, m) = 0, div(n+1, m) = div(n, m) + eq\big(m\big(1 + div(n, m)\big), n+1\big)$. Le reste d'une DE $mod: N^2 \to N$ car $mod(n, m) = n - m \times div(n, m)$

La puissance $pow: N^2 \to N$ ($pow(n, m) = m^n$) car $pow(0, m) = 1, pow(n+1, m) = m \times pow(n, m)$

La racine $root: N^2 \to N$ ou $root(n, m)$ est la racine m-ieme de n cad $\max\{k \in N \mid k^m \leq n\}$ est PR car $root(0, m) = 0, root(n+1, m) = root(n, m) + eq\big(m, pow\big(1 + root(n, m)\big), n+1\big)$

Le logarithme $log: N^2 \to N$ ($\log(n, m)$ logarithme de n en base m) car $\log(1, m) = 0, \log(n+1, m) = \log(n, m) + eq(pow(1 + \log(n, m), m), n+1)$

La fonction qui détermine si un nombre est premier ou non, est primitive récursive.

La fonction $val(n, m)$ donnant la plus grande puissance de m divisant n est primitive récursive.

La fonction d'Ackermann (simple) est mathématiquement bien définie par récurrence par $A: N^2 \to N$

$$\forall k, n \geq 0 \begin{cases} A(0,n) = n+1 \\ A(k+1,0) = A(k,1) \\ A(k+1,n+1) = A\big(k, A(k+1,n)\big) \end{cases}$$

Une **fonction $\boldsymbol{g}$: $\boldsymbol{N} \to \boldsymbol{N}$ majore une fonction $\boldsymbol{f}$: $\boldsymbol{N^k} \to \boldsymbol{N}$** ssi $\forall (n_1, \dots, n_k) \in N^k \; f(n_1, \dots, n_k) \leq g(\max(n_1, \dots, n_k))$

Une fonction primitive récursive $f: N^k \to N$, est toujours majorée par $g: n \mapsto A(p,n)$ pour un certain p fixé.

Corollaire : La fonction d'Ackermann n'est pas primitive récursive.

3.9.2. Fonctions récursives

Pour une fonction $f: N^{k+1} \to N^r$ on définit la fonction partielle de minimisation $\boldsymbol{\mu}(f) = \boldsymbol{Min}(f): D \subseteq N^k \to N$ par $\boldsymbol{\mu}(f)(n_1, \dots, n_k) = \min\{p \in N \mid f(p, n_1, \dots, n_k) = (0,0, \dots, 0) = 0_r\}$.

Elle est définie $(n_1, \dots, n_k) \in D$ ssi le min existe. Intuitivement permet de coder une boucle while avec condition arbitraire.

Une **fonction récursive** est définie inductivement:

1. Une fonction basique est récursive
2. La composition $g(f_1, \dots, f_p)$ de fonctions récursives est récursive.
3. L'itération $Rec(f, g)$ de fonctions récursives est récursive.
4. La minimisation $Min(f)$ d'une fonction récursive est récursive.
5. Il n'y a pas d'autres fonctions récursives.

Autrement dit c'est la plus petite classe contenant les fonctions basiques, close par composition, itération, et minimisation.

Les fonctions récursives peuvent n'être que partielles, la minimisation réduisant leur domaine.

Les fonctions primitives récursives sont totales.

Un argument diagonal donne une fonction non récursive explicite : Soit f_n la suite de toutes les fonctions récursives de $N \to N$. Alors la fonction $g(n) = \begin{cases} f_n(n) + 1 \text{ si } f_n(n) \text{ défini} \\ 0 \text{ sinon} \end{cases}$ n'est pas récursive.

3.9.3. Equivalence avec les machines de Turing

Une fonction $f: N^k \to N^r$ est récursive ssi elle est calculable (par une MT).*

Sens direct : construire des MT pour les fonctions basiques, la composition, l'itération et la minimisation.

Sens indirect : Plus long.

3.9.4. Thèse de Church

Calculabilité par fonctions récursives, par Machine de Turing, par machine à RAM, par λ calcul sont équivalentes.

L'idée que l'on se fait intuitivement du concept de « calculer » est bien formalisée par ces modèles équivalents.

3.10. Compléments

3.10.1. Ecriture des entiers dans une base

3.10.2. Machines de Turing sans écriture sur l'entrée

Le langage accepté par une machine de Turing qui n'écrit jamais sur son entrée est rationnel.