Qu'est-ce que la calculabilité?

– Syndrome du "mouvement perpétuel".

– Les limites intrinsèques de l'informatique.

– Quels problèmes sont solubles par un programme exécuté sur un ordinateur.

La notion de problème

Caractéristiques de la notion de problèmes :

- générique,
- les instances du problème ont une réponses,
- le problème est indépendant de la manière de le résoudre.

Exemples de problèmes :

- Trier un tableau ;
- Déterminer si un programme termine quelque soit les données ;
- Déterminer si un polynôme à coefficients entiers a des racines entières.

La notion de programme

procédure effective: solution à un problème qui à la forme d'un programme pouvant être exécuté sur un ordinateur.

Solution au problème de l'arrêt : déterminer si le programme n'a pas de boucles ou de séquences d'appels récursifs infinies.

```
function mystere (n: entier) : entier
begin
   if (n = 1)
   then 1
   else if (pair n)
        then mystere(n/2)
        else mystere(3*n+1)
end;
```

Programme = automate

Représentation des problèmes

Nécessité d'une fonction d'encodage.

def.: Un alphabet est un ensemble fini de symboles.

def.: Un mot est une séquence finie de symboles (ε est le mot vide).

Soit un problème encodé sur un alphabet. L'ensemble des mots est partitionné en deux :

– instances positives : la réponse est oui pour ce mot,

– instances négatives : ce mot n'encode pas un problème ou la réponse est non pour ce mot.

Représentation et langages

Un problème est caractérisé par l'ensemble des mots qui sont ses instances positives.

def.: Un langage est un ensemble de mots définis sur le même alphabet.

Équivalence entre résoudre un problème et reconnaître le langage des encodages des instances positives du problème.

Th.: L'ensemble des sous-ensembles d'un ensemble dénombrable n'est pas dénombrable.

Cor.: L'ensemble des langages n'est pas dénombrable.

Langage régulier

L'ensemble R des langages réguliers (ou rationnels) sur un alphabet Σ est le plus petit ensemble satisfaisant :
- $\emptyset \in R$, $\{\varepsilon\} \in R$
- $\{a\} \in R, \forall a \in \Sigma$
- si $A, B \in R$ alors $A \cup B, A \cdot B, A^* \in R$.

Exemple : Les langages $\{a\}^* \cdot (\{a\} \cup \{b\}^*)(\{b\}^*)^*$ et $\{a\}^* \cdot \{b\}^*$ sont des langages réguliers (égaux !).

Les expressions régulières pour un alphabet Σ sont les expressions formées par les règles suivantes :
- $\emptyset$, ε et tous les éléments de Σ sont des expressions régulières.
- si α et β sont des expressions régulières alors $(\alpha\beta)$, $(\alpha + \beta)$, et $(\alpha)^*$ sont des expressions régulières.

Exemple : Les expressions $a^*(a+b^*)(b^*)^*$ et a^*b^* sont des expressions régulières.

Langage régulier (suite)

Le langage $L(\xi)$ représenté par l'expression régulière ξ est défini par :
- $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$,
- $L(a) = \{a\}, \forall a \in \Sigma$,
- $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$,
- $L(\alpha\beta) = L(\alpha) \cdot L(\beta)$,
- $L(\alpha^*) = L(\alpha)^*$.

Th.: Un langage est régulier si et seulement si il est dénoté par une expression régulière.

Exemple : L'expression régulière $a^*(a+b^*)(b^*)^*$ dénote le langage régulier $\{a\}^* \cdot (\{a\} \cup \{b\}^*)(\{b\}^*)^*$.

Th.: L'ensemble des langages réguliers est dénombrable

Modélisation d'un ordinateur

Un ordinateur c'est :

- un processeur + des registres,

- une mémoire pour le programme,

- une mémoire pour les données.

Cycle : exécuter une instruction du programme sur l'ensemble registres + mémoire.

Etat : registres + mémoire.

Cycle : fonction de transition sur les états.

Le couple fonction de transition + état initial détermine l'exécution du programme.

Modélisation d'un programme

Modélisation d'un programme sur un ordinateur :

- un ensemble fini d'états,

- une fonction de transition,

- un état initial.

Modélisation de l'exécution d'un programme : séquence d'états obtenus par applications successives de la fonction de transition a l'état initial.

Séquence infinie : préfixe + (séquence fini)*.

Modélisation d'un programme (suite)

Problème : nombre fini d'instances du problème (états initiaux) vs nombre infini d'instances d'un problème (mots d'un langage).

Solutions:

- nombre d'états infini ou,

- introduire les instances du problème caractère par caractère via un ruban et une tête de lecture (choix provisoire).

Cycle :

1. lecture d'un caractère,
2. application de la fonction de transition,
3. déplacement de la tête de lecture.

Automate fini déterministe

Description informelle d'un AFD
- une tête de lecture qui indique le caractère suivant à lire sur le ruban d'entrée où est placé l'instance du problème ;
- un ensemble d'états finis (état initial et états accepteurs) ;
- une fonction de transition qui indique pour chaque état et chaque symbole lu l'état suivant.

Exécution informelle d'un AFD:
- Départ : état initial et tête de lecture sur le premier caractère du mot d'entrée ;
- A chaque étape : lecture du caractère, application de la fonction de transition, déplacement de la tête de lecture ;
- Arrêt : Le mot est accepté si l'automate est dans un état accepteur.

Description formelle d'un AFD

Soit $M = (Q, \Sigma, \delta, s, F)$
- Q est un ensemble fini d'états,
- Σ est un alphabet,
- $\delta : Q \times \Sigma \to Q$ la fonction de transition,
- $s \in Q$ l'état initial,
- $F \subseteq Q$ l'ensemble des états accepteurs.

M est un automate fini déterministe.

Le couple $(q, w) \in Q \times \Sigma^*$ est une configuration.

La configuration (q', w') est dérivable en une étape de la configuration (q, w) par l'AFD M (noté $(q, w) \vdash_M (q', w')$) si, par définition, $w = \sigma w', \sigma \in \Sigma$ et $q' = \delta(q, \sigma)$.

Description formelle d'un AFD (suite)

La configuration (q', w') est dérivable de la configuration (q, w) par l'AFD M (noté $(q, w) \vdash_M^* (q', w')$) si, par définition, il existe $k \geq 0$ et des configurations $(q_i, w_i), 0 \leq i \leq k$ telles que
- $(q, w) = (q_0, w_0)$,
- $(q', w') = (q_k, w_k)$,
- $\forall i, 0 \leq i \leq k, (q_i, w_i) \vdash_M (q_{i+1}, w_{i+1})$.

L'exécution d'un automate M sur un mot w est la suite des configurations $(s, w) \vdash_M (q_1, w_1) \vdash_M \ldots \vdash_M (q_n, \varepsilon)$.

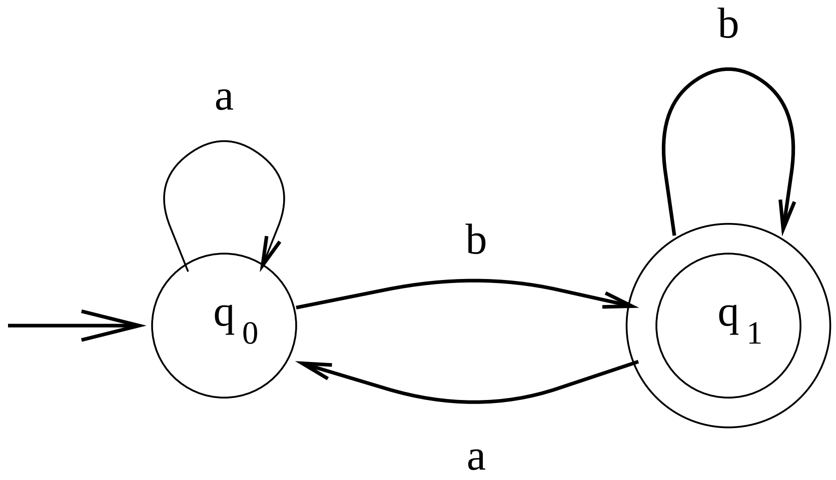
Un mot est accepté par un AFD M si $(s, w) \vdash_M^* (q, \varepsilon)$ avec $q \in F$.

Le langage accepté par un AFD M est l'ensemble $L(M)$ des mots acceptés par M.

Exemples d'AFD

exe.: Soit l'AFD M suivant : $Q = \{q_0, q_1\}$, $\Sigma = \{a, b\}$, $s = q_0$, $F = \{q_1\}$, δ définie par :

$$\begin{array}{l|llll} q & q_0 & q_0 & q_1 & q_1 \\ \sigma & a & b & a & b \\ \delta(q,\sigma) & q_0 & q_1 & q_0 & q_1 \end{array}$$



$L(M) = \{w \mid w$ se termine par $b\}$.

Exemple : Construire un AFD M' sur l'alphabet $\{a, b\}$ qui accepte le langage
$L(M') = \{w \mid w$ ne contient pas consécutivement 2 $a\}$.

Automate fini non déterministe

Description informelle d'un AFND
Les AFND sont des AFD où l'on permet plusieurs transitions correspondant à la même lettre dans chaque état.

Les ε-AFND sont des AFND où l'on permet des transitions sur ε (i.e. sans avancer la tête de lecture).

(On permet parfois des transitions sur des mots de longueur supérieur à 1.)

Description formelle d'un AFND

Soit $M = (Q, \Sigma, \Delta, s, F)$
- Q est un ensemble fini d'états,
- Σ est un alphabet,
- $\Delta \subset (Q \times \Sigma \times Q)$ la relation de transition,
- $s \in Q$ l'état initial,
- $F \subseteq Q$ l'ensemble des états accepteurs.

M est un automate fini non déterministe.

Les autres définitions sont étendues de façon immédiate.

Exemple : Construire un AFND M sur l'alphabet $\{a, b\}$
qui accepte le langage
$L(M) = \{w \mid w \text{ se termine par } ab\}$.

Équivalence d'automates

Deux automates M et M' sont équivalents si ils acceptent le Mme langage (i.e. $L(M) = L(M')$).

Th.: Pour tout AFND il existe un AFD équivalent.

Const.: Soit un AFND $M = (Q, \Sigma, \Delta, s, F)$, nous construisons un AFD $M' = (Q', \Sigma', \delta', s', F')$ équivalent à M de la manière suivante
- $Q' = 2^Q$, $\Sigma' = \Sigma$, $s' = \{s\}$,
- $\delta'(\overline{q}, a) = \bigcup\{p \in Q \mid \exists q \in \overline{q}, (q, a, p) \in \Delta\}$,
- $F' = \{\overline{q} \in Q' \mid \overline{q} \cap F \neq \emptyset\}$.

Exemple : Construire un AFD équivalent à l'AFND suivant :

$$\begin{aligned} M = (& \{q_0, q_1\}, \{0, 1\}, \{(q_0, 0, q_0), (q_0, 0, q_1), \\ & (q_0, 1, q_1), (q_1, 1, q_0), (q_1, 1, q_1)\}, q_0, \{q_1\}). \end{aligned}$$

Th.: Pour tout ε-AFND il existe un AFND équivalent.

Grammaire

Soit $G = (V, \Sigma, R, S)$ tel que
- V est un alphabet,
- $\Sigma \subseteq V$ est l'ensemble des symboles terminaux ($V - \Sigma$ est l'ensemble des symboles non terminaux),
- $R \subseteq (V^+ \times V^*)$ est un ensemble fini de règles ((u, v) est noté $u \rightarrow v$),
- $S \in V - \Sigma$ le symbole de départ.

G est une grammaire.

Exemple : $V = \{ \begin{array}{l} Phrase, Sujet, Verbe, Pronom, \\ il, elle, dort, \acute{e}coute\}, \end{array}$

$\Sigma = \{il, elle, dort, \acute{e}coute\}$,

$$R = \{ \begin{array}{ll} Phrase \rightarrow Sujet\ Verbe & Sujet \rightarrow Pronom \\ Pronom \rightarrow il & Pronom \rightarrow elle \\ Verbe \rightarrow dort & Verbe \rightarrow \acute{e}coute\}, \end{array}$$

$S = Phrase$

$G_f = (V, \Sigma, R, S)$ est une grammaire.

Grammaires (suite)

Soit une grammaire $G = (V, \Sigma, R, S)$ et $u \in V^+, v \in V^*$. La grammaire G permet de dériver v de u en une étape ($u \Rightarrow v$) si
- $u = xu'y$,
- $v = xv'y$ et
- $u' \to v'$

Exemple : $Sujet\ Verbe\ Pronom \ \Rightarrow\ Sujet\ dort\ Pronom$ en effet en posant $x = Sujet$, $y = Pronom$, $u' = Verbe$ et $v' = dort$, nous avons $Verbe \to dort \in R$.

Soit une grammaire $G = (V, \Sigma, R, S)$ et $u \in V^+, v \in V^*$. La grammaire G permet de dériver v de u en plusieurs étapes ($u \Rightarrow^* v$) si $\exists k \geq 0$ et $v_0, \ldots, v_k \in V^+$ tels que
- $u = v_0$,
- $v = v_k$ et
- $v_i \Rightarrow v_{i+1}, 0 \leq i < k$.

Grammaire (suite)

Exemple : $Pronom\ dort\ elle$ est dérivable de $Sujet\ Verbe\ Sujet$ par la grammaire G_f

$$(Sujet\,Verbe\,Sujet \Rightarrow^* Pronom\ dort\ elle)$$

	règles
$Sujet\,Verbe\,Sujet$	$Sujet \rightarrow Pronom$
$Sujet\,Verbe\,Pronom$	$Verbe \rightarrow dort$
$Sujet\ dort\ Pronom$	$Sujet \rightarrow Pronom$
$Pronom\ dort\ Pronom$	$Pronom \rightarrow elle$
$Pronom\ dort\ elle$	

Un mot $v \in \Sigma^*$ est généré par une grammaire G si $S \Rightarrow^* v$.

Exemple : Le mot $elle\ dort$ est généré par la grammaire G_f :

	règles
$Phrase$	$Phrase \rightarrow Sujet\,Verbe$
$Sujet\,Verbe$	$Verbe \rightarrow dort$
$Sujet\ dort$	$Sujet \rightarrow elle$
$elle\ dort$	

Grammaire (suite)

Le langage généré par une grammaire G
$L(G) = \{v \in \Sigma^* \text{ tel que } S \Rightarrow^* v\}$.

Exemple : $L(G_f) = \{il\ dort, il\ écoute, elle\ dort, elle\ écoute\}$.

Exemple : Soit la grammaire

$$
\begin{aligned}
G = \ & (\{S, a, b, A, B\}, \{a, b\}, \\
& \{S \to \varepsilon, S \to aB, \\
& S \to bA, A \to aS, \\
& A \to bAA, B \to bS, \\
& B \to aBB\}, S).
\end{aligned}
$$

Démontrez que $L(G)$ est l'ensemble des mots comportant le meme nombre de lettres a et de lettres b.

Typologie des grammaires

4 types de grammaires :
- type 0 : grammaires générales,
- type 1 : grammaires sensibles au contexte,
- type 2 : grammaires hors contexte,
- type 3 : grammaires régulières.

Th.: type 3 $\subset$ type 2 $\subset$ type 1 $\subset$ type 0.

Une grammaire est régulière si toute production est de la forme $A \rightarrow wB$ ou $A \rightarrow w$ avec $A, B \in V - \Sigma$ et $w \in \Sigma^*$.

Caractérisation des langages réguliers

Th.: Un langage est régulier ssi
il est accepté par un automate fini ssi
il est généré par une grammaire régulière.

Expression régulière : descriptif,
Automate fini : analytique,
Grammaire régulière : génératif.

Langage régulier $\Rightarrow$ automate fini

Const.: Construisons par induction un AFND M_ξ à partir d'une expression régulière ξ ($M_i = (Q_i, \Sigma_i, \Delta_i, s_i, F_i)$ automate pour l'expression
régulière α_i) :

- si $\xi = \emptyset$ alors $M_\xi = (\{q\}, \emptyset, \emptyset, q, \emptyset)$
- si $\xi = \varepsilon$ alors $M_\xi = (\{q\}, \emptyset, \emptyset, q, \{q\})$
- si $\xi = \sigma \in \Sigma$ alors

$M_\xi = (\{q_0, q_1\}, \{\sigma\}, \{(q_0, \sigma, q_1)\}, q_0, \{q_1\})$

- si $\xi = \alpha_1\alpha_2$ alors

$$\begin{aligned} M_\xi = (\ & Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \\ & \Delta_1 \cup \Delta_2 \cup \{(q, \varepsilon, s_2)/q \in F_1\}, s_1, F_2) \end{aligned}$$

- si $\xi = \alpha_1^*$ alors

$$\begin{aligned} M_\xi = (\ & Q_1 \cup \{s\}, \Sigma_1, \Delta_1 \cup \{(q, \varepsilon, s_1)/q \in F_1\}, \\ & s, F_2 \cup \{s\}) \end{aligned}$$

- si $\xi = \alpha_1 + \alpha_2$ alors

$$\begin{aligned} M_\xi = (\ & Q_1 \cup Q_2 \cup \{s\}, \Sigma_1 \cup \Sigma_2, \\ & \Delta_1 \cup \Delta_2 \cup \{(s, \varepsilon, s_1), (s, \varepsilon, s_2)\}, s, F_1 \cup F_2) \end{aligned}$$

Automate fini $\Rightarrow$ Grammaire régulière

Const.: Construisons une grammaire régulière $G = (V_G, \Sigma_G, S_G, R_G)$ à partir d'un AFND $M = (Q, \Sigma, \Delta, s, F)$ telle que $L(G) = L(M)$ comme suit :

- $V_G = Q \cup \Sigma$
- $\Sigma_G = \Sigma$
- $S_G = s$
- $R_G = \begin{array}{l} \{A \rightarrow wB / (A, w, B) \in \Delta\} \cup \\ \{A \rightarrow \varepsilon / A \in F\} \end{array}$

Grammaire régulière $\Rightarrow$ Automate fini

Const.: Construisons un AFND
$M = (Q, \Sigma, \Delta, s, F)$ à partir d'une grammaire régulière $G = (V_G, \Sigma_G, S_G, R_G)$ tel que $L(M) = L(G)$ comme suit :

- $Q = (V_G - \Sigma_G) \cup \{f\}$, $(f \notin V_G)$,
- $\Sigma = \Sigma_G$,
- $s = S_G$,
- $F = \{f\}$,
- $\Delta = \begin{array}{l} \{(A, w, B) / A \to wB \in R_G\} \cup \\ \{(A, w, f) / A \to w \in R_G\} \end{array}$

Théorème du gonflement et langages non réguliers

Th.: Soit L un langage régulier infini. Alors $\exists u, v, w \in \Sigma^*, v \neq \varepsilon$ tels que $\forall n \geq 0, uv^n w \in L$.

Pr. Inf. Soit $M = (Q, \Sigma, \delta, s, F)$ avec $\mid Q \mid = m$ et $x \in L(M)$ tel $|x| > m$. Alors $\exists q \in Q, uvw \in \Sigma^*$ et $f \in F$ tels que $x = uvw$ et $(s, x) \vdash_M^* (q, vw) \vdash_M^* (q, w) \vdash_M^* (f, \varepsilon)$ donc $uv^*w \in L(M)$.

Th.: Soit L un langage régulier infini et soit $x \in L$ tel que $|x| \geq |Q|$, Q ensemble des états d'un automate non déterministe acceptant L. Alors $\exists u, v, w \in \Sigma^*, v \neq \varepsilon, | uv| \geq |Q|$ tels que $uvw = x$ et $\forall n \geq 0, uv^n w \in L$.

ex.: Le langage $L = a^{n^2}$ n'est pas régulier.

ex.: Le langage $L = \{a^n \mid n \text{ premier}\}$ n'est pas régulier.

ex.: Le langage $L = a^n b^n$ n'est pas régulier.

Automates à pile

Soit $M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$
- Q est un ensemble fini d'états,
- Σ est un alphabet d'entrée,
- Γ est un alphabet de pile,
- $\Delta \subset ((Q \times \Sigma^* \times \Gamma^*) \times (Q \times \Gamma^*))$ la relation de transition (ensemble fini),
- $Z \in \Gamma$ est le symbole initial de la pile,
- $s \in Q$ l'état initial,
- $F \subseteq Q$ l'ensemble des états accepteurs.

M est un automate à pile.

Exemple : Soit $P_1 = (\{s, p, q\}, \{a, b\}, \{A, Z\}, \Delta, Z, s, \{q\})$
avec Δ :{ $((s, a, \varepsilon), (s, A))$
$((s, \varepsilon, Z), (q, \varepsilon))$
$((s, b, A), (p, \varepsilon))$
$((p, b, A), (p, \varepsilon))$
$((p, \varepsilon, Z), (q, \varepsilon))$}

P_1 est un automate à pile.

Automates à pile (suite)

Le triplet $(q, u, \beta) \in Q \times \Sigma^* \times \Gamma^*$ est une configuration.

La configuration (q', w', α') est dérivable en une étape de la configuration (q, w, α) par l'automate à pile M (noté $(q, w, \alpha) \vdash_M (q', w', \alpha')$) si $w = uw', u \in \Sigma^*$, $\alpha = \beta\delta, \beta, \delta \in \Gamma^*$, $\alpha' = \gamma\delta, \gamma \in \Gamma^*$, et $((q, u, \beta), (q', \gamma)) \in \Delta$.

Exemple : $(p, baba, AAZ) \vdash_{P_1} (p, aba, AZ)$ par :
- $p = q$, $p = q'$, $w = baba$, $w' = aba$, $\alpha = AAZ$, $\alpha' = AZ$,
- $u = b$, $\beta = A$, $\gamma = \varepsilon$, $\delta = AZ$,
- $((p, b, A), (p, \varepsilon)) \in \Delta$,

Une configuration C' est dérivable en plusieurs étapes de la configuration C par l'automate à pile M ($C \vdash_M^* C'$) si $\exists k \geq 0$ et $C_0, \ldots, C_k$ des configurations telles que
- $C = C_0$,
- $C' = C_k$,
- $C_i \vdash_M C_{i+1}, \forall i, 0 \leq i < k$.

Automates à pile (suite)

L'exécution d'un automate à pile M sur un mot w est la suite des configurations $(s, w, Z) \vdash_M (q_1, w_1, \alpha_1) \vdash_M \ldots \vdash_M (q_n, \varepsilon, \gamma)$.

Exemple : $(s, aabb, Z) \vdash^*_{P_1} (q, \varepsilon, Z)$:

configurations	transitions
$(s, aabb, Z)$	$((s, a, \varepsilon), (s, A))$
(s, abb, AZ)	$((s, a, \varepsilon), (s, A))$
(s, bb, AAZ)	$((s, b, A), (p, \varepsilon))$
(p, b, AZ)	$((p, b, A), (p, \varepsilon))$
(p, ε, Z)	$((p, \varepsilon, Z), (q, \varepsilon))$
$(q, \varepsilon, \varepsilon)$	

Un mot est accepté par un automate à pile M si $(s, w, Z) \vdash^*_M (q, \varepsilon, \gamma)$ avec $q \in F$.

Exemple : Le mot $aabb$ est accepté par l'automate à pile P_1.

Automates à pile (suite)

Le langage accepté par un automate à pile M est l'ensemble $L(M)$ des mots acceptés par M.

Exemple : L'automate à pile P_1 accepte le langage $\{a^n b^n / n \geq 0\}$.

Exemple : Décrire un automate à pile qui accepte le langage $\{ww^R\}$.

Grammaires hors-contexte

Une grammaire est une grammaire hors-contexte si toutes les règles sont de la forme $A \to B, A \in V - \Sigma, B \in V^*$.

Exemple : La grammaire $G_1 = (\{S, a, b\}, \{a, b\}, R_1, S)$, $R_1 = \{S \to aSb, S \to \varepsilon\}$, est hors-contexte et génère le langage $\{a^n b^n / n \geq 0\}$.

Exemple : Décrire la grammaire qui génère le langage $\{ww^R\}$.

Caractérisation des langages hors-contexte

Th. Un langage est hors-contexte si et seulement si il est accepté par un automate à pile.

Const.: Grammaire hors-contexte $\Rightarrow$ automate à pile. Soit la grammaire hors contexte $G = (V_G, \Sigma, R_G, S_G)$, nous construisons l'automate à pile $M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$ tel que $L(G) = L(M)$ de la manière suivante :

- $Q = \{s, q, r\}$, $\Gamma = V \cup \{Z\}, Z \notin V$, $F = \{r\}$,
- $\Delta =$ $\{((s, \varepsilon, \varepsilon), (q, S)), ((q, \varepsilon, Z), (r, \varepsilon))\} \cup$
  $\{((q, \varepsilon, A), (q, x))/A \to x \in R\} \cup$
  $\{((q, a, a), (q, \varepsilon))/a \in \Sigma\}$

Théorème du gonflement et langage hors-contexte

Th.: Soit un langage hors-contexte L. Alors il existe une constante K telle que tout mot $w \in L$ satisfaisant $|w| > K$ peut etre écrit $w = uvxyz$ avec v ou $y \neq \varepsilon$ et $uv^nxy^nz \in L, \forall n > 0$.

exe.: Le langage $L = \{a^nb^nc^n\}$ n'est pas hors-contexte.

rem.: L'intersection de deux langages hors-contexte n'est pas nécessairement hors-contexte : $L_1 = \{a^nb^mc^m\}$ et $L_2 = \{a^mb^mc^n\}$ sont hors-contexte mais $L_1 \cap L_2 = \{a^mb^mc^m\}$ n'est pas hors-contexte.

Machines de Turing

Alan Turing (1912-1954).

Description informelle d'une MT
- une tête de lecture qui indique le caractère suivant à lire sur le ruban d'entré où est placé l'instance du problème ;
- un ensemble d'états finis (état initial et états accepteurs) ;
- une fonction de transition qui indique pour chaque état et chaque symbole lu l'état suivant, le symbole écrit et le sens de déplacement de la tête de lecture

Machines de Turing (suite)

Exécution informelle d'une MT

– Départ : état initial et tête de lecture sur le premier caractère du mot d'entrée ;

– A chaque étape :
  1. lecture du caractère sous la tête de lecture ;
  2. remplacement du caractère lu par celui de la fonction de transition ;
  3. déplacement de la tête de lecture selon la fonction de transition ;
  4. changement d'état selon la fonction de transition.

– Arrêt : Le mot est accepté si la MT atteint un état accepteur.

Description formelle d'une MT

Soit $M = (Q, \Gamma, \Sigma, \delta, s, \#, F)$
- Q est un ensemble fini d'états,
- Γ est un alphabet de ruban,
- Σ est un alphabet d'entrée,
- $\delta : ((Q \times \Gamma) \to (Q \times \Gamma^* \times \{g, d\}))$ la fonction de transition,
- $s \in Q$ l'état initial,
- $\# \in \Gamma - \Sigma$ le caractère "blanc",
- $F \subseteq Q$ l'ensemble des états accepteurs.

M est une machine de Turing.

Exemple : Soit $M_1 = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b, X, Y, \#\}, \{a, b\}, \delta, q_0, \#, \{q_4\})$

avec $\delta : \{ ((q_0, a)(q_1, X, d)), ((q_0, Y), (q_3, Y, d)), ((q_1, a), (q_1, a, d)), ((q_1, b), (q_2, Y, g)), ((q_1, Y), (q_1, Y, d)), ((q_2, a), (q_2, a, g)), ((q_2, X), (q_0, X, d)), ((q_2, Y), (q_2, Y, g)), ((q_3, Y), (q_3, Y, d)), ((q_3, \#), (q_4, \#, d))\}$.

M_1 est une machine de Turing.

Description formelle d'une MT (suite)

Le triplet $(q, \alpha, \beta) \in (Q \times \Gamma^* \times (\varepsilon \cup \Gamma^*(\Gamma - \{\#\})))$ est une configuration avec :
- q l'état de la machine,
- α le mot apparaissant avant la tête de lecture,
- β le mot apparaissant entre la position de la tête de lecture (incluse) et le dernier caractère non blanc.

De la configuration (q, α, β), les configurations suivantes sont dérivables en une étape par la machine de Turing M ($\beta = b\beta'$ avec $b = \#$ si $\beta = \varepsilon$)
- si $\delta(q, b) = (q', b', d)$ alors $(q, \alpha, \beta) \vdash_M (q', \alpha b', \beta')$;
- si $\delta(q, b) = (q', b', g)$ et $\alpha = \alpha' a$ alors $(q, \alpha, \beta) \vdash_M (q', \alpha', ab'\beta')$.

Remarque : Lors d'un déplacement à gauche (g) $\alpha \neq \varepsilon$ donc il existe $\alpha' \in \Gamma^*$ et $a \in \Gamma$ tels que $\alpha = \alpha' a$.

Description formelle d'une MT (suite)

Une configuration C' est dérivable en plusieurs étapes de la configuration C par la machine de Turing M ($C \vdash_M^* C'$) si $\exists k \geq 0$ et $C_0, \ldots, C_k$ des configurations telles que
- $C = C_0$,
- $C' = C_k$,
- $C_i \vdash_M C_{i+1}, \forall i, 0 \leq i < k$.

Exemple : Montrer que la machine de Turing M_1 dérive (en plusieurs étapes) la configuration $(q_4, XXYY\#, \varepsilon)$ de $(q_0, \varepsilon, aabb)$.

Un mot est accepté par une machine de Turing M si $(s, \varepsilon, w) \vdash_M^* (q, \alpha, \beta)$ avec $q \in F$.

Un langage est récursivement énumérable si il est accepté par une machine de Turing.

Exécution d'une MT

L'exécution d'une machine de Turing M sur un mot w est la suite maximale (possiblement infinie) des configurations $(s, \varepsilon, w) \vdash_M (q_1, \alpha_1, \beta_1) \vdash_M \ldots \vdash_M (q_n, \alpha_n, \beta_n) \vdash_M \ldots$.

Cette suite est
- soit finie (se terminant en une configuration dont l'état est accepteur ou à partir de laquelle il n'y a pas de transition possible),
- soit infinie.

Un langage L est décidé par une machine de Turing M si
- M accepte L,
- M n'a pas d'exécution infinie.

Un langage est récursif si il est décidé par une machine de Turing.

Langages acceptés, langages décidés et automates

La notion de langage décidé n'est valable que pour des automates déterministes.

– Automate fini déterministe :
  langage accepté $\equiv$ langage décidé.

– Th.: Étant donné une grammaire hors-contexte G (et donc un automate à pile par le théorème d'équivalence), il existe un algorithme pour déterminer si un mot donné appartient à $L(G)$.

– Automate à pile déterministe:
  langage accepté $\equiv$ langage décidé,
  si il n'y a pas de transition sur ε.

Thèse de Turing-Church

Une procédure effective est une solution à un problème qui a la forme d'un programme pouvant être exécuté sur un ordinateur en un temps fini pour toutes les entrées.

Thèse de Turing-Church : Les langages reconnus par une procédure effective sont exactement ceux décidés par une machine de Turing.

lem.: Tout langage décidé par une machine de Turing est reconnu par une procédure effective.

Conjecture : Pour tout langage reconnu par une procédure effective, il existe une machine de Turing qui le décide.

Extensions des Machines de Turing

- Ruban infini dans les deux sens,
- Rubans multiples,
- Machines à mémoire à accès direct.

Toutes ces machines peuvent être simulées par une machine de Turing.

- Machines de Turing non-déterministes (une relation de transition remplace la fonction de transition).

Th.: Tout langage accepté par une machine de Turing non-déterministe est aussi accepté par une machine de Turing déterministe.

Fonctions calculables par une machine de Turing

Une machine de Turing calcule une fonction $f : \Sigma^* \rightarrow \Sigma^*$ si, pour tout mot d'entrée w, elle s'arrête toujours dans une configuration ou se trouve $f(w)$ sur le ruban.

Thèse de Turing-Church : Les fonctions calculables par une procédure effective sont les fonctions calculables par une machine de Turing.

Exemple : Écrire une machine de Turing pour additionner deux nombres entiers dans une représentation binaire.

Fonctions Récursives

Approche basée sur les fonctions définies sur les nombres naturels pour caractériser celles qui sont calculables par une procédure effective.

Fonctions calculables connues :
- les fonctions de base dont un algorithme est connu pour les calculer sur une représentation ;
- la composition finie de fonctions de base ;
- la récursion (. . . en partie).

Exemple : Exemples de fonctions récursives :
- + et x sur les réels,
- la factorielle,
- la racine carré d'un réel positif par la méthode de Newton

Fonctions récursives primitives

Les fonctions récursives primitives simples sont :
- la fonction $\overline{0}$,
- les fonctions de projection
$\pi_i^k = (\lambda n_1 \ldots n_k \;\; n_i)$,
- la fonction successeur $(\lambda n \;\; (\sigma \;\; n))$.

La composition de la fonction g à l arguments et des fonctions $h_1, \ldots, h_l$ à k arguments est la fonction f à k arguments définie par :
$(\lambda n_1 \ldots n_k \; (g \; (h_1 \; n_1 \ldots n_k) \ldots (h_l \; n_1 \ldots n_k)))$.

La récursion primitive à partir des fonctions g à k arguments et h à $k+2$ arguments est la fonction f à $k+1$ arguments définie par :

$$
\begin{aligned}
(\lambda \quad & n_1 \ldots n_k m \; (if \; (m = (\sigma \; m')) \\
& then \; (h \; (f \; n_1 \ldots n_k \; m') \; n_1 \ldots n_k \; m') \\
& else \; (g \; n_1 \ldots n_k)))
\end{aligned}
$$

Fonctions récursives primitives (suite)

Les fonctions récursives primitives sont définies inductivement par :
- les fonctions récursives primitives simples,
- la composition et la récursion primitive.

Exemple : Les fonctions constantes $\overline{j}$ sont des fonctions primitives récursives :
$$\overline{j} = \underbrace{(\sigma\ (\sigma \ldots (\sigma\ \overline{\mathsf{0}})))}_{\text{j fois}}.$$

Exemple : L'addition, la soustraction et la multiplication sur les naturels sont des fonctions récursives primitives.

Exemple : La factorielle est une fonction récursive primitive.

Exemple : Le produit borné par m d'une fonction $(\lambda n_1 \ldots n_k i\ (g\ n_1 \ldots n_k\ i))$ défini par le produit de g pour les m premières valeurs de i est une fonction récursive primitive.

Les prédicats récursifs primitifs

Un prédicat à k arguments est un sous-ensemble de $\mathbb{N}^k$.

La fonction caractéristique d'un prédicat $P \subseteq \mathbb{N}^k$ est la fonction f_P de $\mathbb{N}^k$ dans $\{0,1\}$ définie par :

$$(f_P\, n_1 \ldots n_k) = \begin{array}{l} \overline{0} \text{ si } (n_1, \ldots, n_k) \notin P \\ \overline{1} \text{ sinon} \end{array}$$

Un prédicat est récursif primitif si sa fonction caractéristique est récursive primitive.

Exemple : Le prédicat *nonzero* qui est vrai sauf si son argument est l'entier 0 a pour fonction caractéristique $f_{\overline{\emptyset}}$:

$$\begin{array}{ll} (\lambda m & (if\ (m = (\sigma\, m')) \\ & then\ (\pi^3_3\ (f_{\overline{\emptyset}}\, m')\ m'\ \overline{1}) \\ & else\ \overline{0})) \end{array}$$

Exemple : Écrire les fonctions caractéristiques (récursives primitives) des prédicats *et*, *ou* et *non*.

Les prédicats récursifs primitifs (suite)

La quantification bornée universelle teste si un prédicat est vrai pour toutes les valeurs d'un de ses arguments inférieures à une borne.

prop.: La quantification bornée universelle est un prédicat récursif primitif.

Soit f_P la fonction caractéristique du prédicat P à $k+1$ arguments. Alors la fonction caractéristique (récursive primitive) de la quantification bornée de P pour le $k+1$ argument $\forall_P^{\leq}$ est définie par :

$$\begin{array}{l}
(\lambda n_1 \ldots n_k m\ (if\ (m = (\sigma\ m'))\\
then\ (fois\ \ (\pi_1^{k+2}\ (\forall_P^{\leq}\ n_1 \ldots n_k\ m')\ n_1 \ldots n_k\ m')\\
\qquad\qquad\quad (f_P\ n_1 \ldots n_k\ m'))\\
else\ (\pi_1^{k+2}\ \overline{1}\ n_1 \ldots n_k\ m')))
\end{array}$$

Les prédicats récursifs primitifs (suite)

La quantification bornée existentielle teste si un prédicat est vrai pour des valeurs d'un de ses arguments inférieures à une borne.

prop.: La quantification bornée existentielle est un prédicat récursif primitif.

Pour une borne, la valeur de la fonction de minimisation bornée d'un prédicat (pour un de ses arguments) est le plus petit nombre naturel inférieur à la borne tel que le prédicat soit vrai :

$$\mu_P i \leq m = \begin{cases} \text{le plus petit } i \leq m \text{ tel que} \\ (n_1, \ldots, n_k, i) \in P \\ 0 \text{ si il n'existe pas de tel } i \end{cases}$$

prop.: La minimisation bornée est une fonction récursive primitive.

Les fonctions μrécursives

Th.: Il existe des fonctions calculables qui ne sont pas primitives récursives.

La valeur de la fonction de minimisation non bornée d'un prédicat (pour un de ses arguments) est le plus petit nombre naturel tel que le prédicat soit vrai.

remarque : Dans le cas d'un prédicat toujours faux, la minimisation non bornée n'est pas calculable car ne termine jamais.

Un prédicat P est sûr si pour tout $n_1, \ldots, n_k$, il existe un i tel que $(n_1, \ldots, n_k, i) \in P$.

Les fonctions μrécursives sont celles obtenues par induction sur les fonctions récursives primitives simples, la composition, la récursion primitive et la minimisation non bornée des prédicats sûrs.

Fonctions μrécursives et Machines de Turing

2 approches pour la définition des fonctions calculables par des procédures effectives : les fonctions μrécursives et les fonctions calculables par machine de Turing.

Question : Les fonctions μrécursives sont-elles exactement les fonctions calculables par une machine de Turing ?

Fonctions μrécursives sur les naturels	Machine de Turing sur les caractères

Problème de la représentation effective des nombres par des caractères et inversement : nécessité de procédures effectives de codage.

Nombres et caractères

lem.: Il existe une représentation effective des nombres naturels par des chaînes de caractères.

pr. inf.: Le codage binaire des naturels.

lem.: Il existe une représentation effective des chaînes de caractères par des nombres naturels.

(la Gödelisation) La représentation d'une chaîne de caractère par un nombre de Gödel.

Exemple : Soit $\mathcal{A}$ un alphabet. Nous définissons une bijection gd de $\mathcal{A}$ dans $\{1, \ldots, |\mathcal{A}|\}$. Pour tout mot $w = \sigma_0 \ldots \sigma_l \in \mathcal{A}^*$, le nombre de Gödel de w est défini par :

$$Gd(w) = \Sigma_{i=0}^{l}(1 + |\mathcal{A}|)^{l-i} gd(w_i).$$

Fonctions μrécursives $\equiv$ machines de Turing

Th.: Toute fonction μrécursive est calculable par une machine de Turing.

pr. inf.: En adoptant un codage des entiers narurels, les fonctions primitives récursives simples, leurs compositions, leurs récursions primitives et leurs minimisations non bornées sûres sont calculables par une machine de Turing.

Th.: Toute fonction calculable par une machine de Turing est μrécursive.

pr. inf.: Soient une machine de Turing M sur un alphabet $\mathcal{A}$ calculant une fonction f_M de $\mathcal{A}^*$ dans $\mathcal{A}^*$ et une gödédisation Gd pour $\mathcal{A}^*$. Alors il existe une fonction μrécursive f telle que $(\forall w \in \mathcal{A}^*(f_M(w) = Gd^{-1}(f(Gd(w)))))$.

Fonctions partielles

Une fonction partielle f de $\mathcal{A}^*$ dans $\mathcal{A}^*$ est calculée par une machine de Turing M si,
- pour tout mot d'entrée w tel que f soit définie, M s'arrête toujours dans une configuration ou se trouve $f(w)$ sur le ruban ;
- pour tout mot d'entrée w tel que f ne soit pas définie, M ne s'arrête pas ou s'arrête sur une configuration conventionnelle.

Une fonction partielle f de $\mathbb{N}$ dans $\mathbb{N}$ est μrécursive si elle est définie par induction sur les fonctions récursives primitives simples, la composition, la récursion primitive et la minimisation non bornée.

Th.: Une fonction partielle est μrécursive si et seulement si elle est calculable par une machine de Turing.

De la décidabilité à l'indécidabilité

La classe de décidabilité $\boldsymbol{R}$ est l'ensemble des langages décidés par une machine de Turing (ou récursifs ou décidables ou calculables ou solubles algorithmiquement).

La classe de décidabilité $\boldsymbol{RE}$ est l'ensemble des langages acceptés par une machine de Turing (ou récursivement énumérables ou semidécidables ou partiellement décidables ou partiellement calculables ou partiellement solubles algorithmiquement).

$$\left.\begin{array}{l}\text{- L'ensemble des machines}\\ \text{de Turing est dénombrable.}\\ \text{- L'ensemble des langages}\\ \text{est non dénombrable.}\end{array}\right\} \quad \begin{array}{l}\text{alors il y a}\\ \text{des langages}\\ \text{indécidables.}\end{array}$$

lem.: $\boldsymbol{R} \subseteq \boldsymbol{RE}$.

Premier langage indécidable

Construisons le tableau A, qui compose toute machine de Turing avec tout mot, défini par :
- $A[M_i, w_j] = \mathit{oui}$ si la machine de Turing M_i accepte le mot w_j.
- $A[M_i, w_j] = \mathit{non}$ si la machine de Turing M_i rejette ou boucle pour le mot w_j.

Th.: Soit le langage L_0 défini par

$$L_0 = \{w / w = w_i \text{ et } A[M_i, w_i] = \mathit{non}\}.$$

Alors $L_0 \notin RE$.

Diagonalisation : Posons que M_k accepte L_0.

$$\begin{array}{ll} & M_k \text{ accepte } w_k \ (w_k \in L_0) \\ \Rightarrow & A[M_k, w_k] = \mathit{oui} \Rightarrow w_k \notin L_0 \\ & M_k \text{ n'accepte pas } w_k \ (w_k \notin L_0) \\ \Rightarrow & A[M_k, w_k] = \mathit{non} \Rightarrow w_k \in L_0 \end{array}$$

Double contradiction donc il n'existe pas de machine de Turing qui accepte L_0 ($L_0 \notin RE$).

Propriétés des classes l'indécidabilité

lem.: $L \in R$ si et seulement si $\overline{L} \in R$.

pr. inf.: Soit M_L une MT qui décide du langage L. Construisons la MT M qui rejette un mot lorsque M_L l'accepte et qui accepte un mot lorsque M_L le rejette (toujours en un temps fini). Alors M décide de $\overline{L}$.

lem.: Si $L \in RE$ et $\overline{L} \in RE$ alors $L \in R$ et $\overline{L} \in R$.

pr. inf.: Soient les MT M_L qui accepte L et $M_{\overline{L}}$ qui accepte $\overline{L}$. Construisons la MT M qui accepte un mot lorsque M_L accepte un mot ou $M_{\overline{L}}$ termine en le rejetant et qui rejette un mot lorsque $M_{\overline{L}}$ accepte un mot ou M_L termine en le rejetant. Alors M décide de L.

Un langage de $RE \cap \overline{R}$

cor.: Soit L un langage. Alors
- soit $L \in R$ et $\overline{L} \in R$
- soit $L \notin RE$ et $\overline{L} \notin RE$
- soit $L \notin RE$ et $\overline{L} \in RE \cap \overline{R}$

lem.: $\overline{L_0} \in RE$.

pr. inf.: Construisons une machine de Turing universelle (qui simule le fonctionnement d'une machine de Turing M donnée sur un mot w donné) qui :
- Énumère les mots pour obtenir w ;
- Énumère les machines de Turing pour obtenir M ;
- Simule M_i sur M pour w.

M accepte w si il est accepté par M_i.

cor.: $\overline{L_0} \in RE \cap \overline{R}$.

Technique de réduction

La technique de réduction établit l'indécidabilité d'un langage L à partie de l'indécidabilité d'un langage L' en exhibant une procédure effective (dite de réduction de L' à L) qui décide de L' si L est décidable.

$$\left.\begin{array}{l} L \in R \Rightarrow L' \in R \\ L' \notin R \end{array}\right\} \Rightarrow L \notin R.$$

Exemple : Soit le langage universelle LU défini par

$$LU = \{< M, w > /M \text{ accepte } w\}.$$

$LU \in RE \cap \overline{R}$ (et $\overline{LU} \notin RE$).

Exemple : Les problèmes de l'arrêt, de l'arrêt dans un langage de programmation, de l'arrêt sur le mot vide, de l'arrêt pour au moins un mot et de l'arrêt pour tout mot sont indécidables.

Exemple : Le problème de déterminer si le langage accepté par une machine de Turing est vide est indécidable.

Propriétés des langages récursivement énumérables

Soit une machine de Turing M. Le langage calculé par M est défini par

$$\{w / \exists u, M \text{ s'arrête sur } u \text{ avec } w \text{ sur le ruban}\}.$$

Th.: Un langage est calculé par une machine de Turing si et seulement si il est accepté par une machine de Turing.

Th.: Un langage est généré par une grammaire si et seulement si il est accepté par une machine de Turing.

Résumé sur la calculabilité

- Les langages réguliers sont générés par les grammaires régulières (T3) et décidés par les automates d'états finis déterministes.

- Les langages hors-contexte sont générés par les grammaires hors-contexte (T2), acceptés par les automates à piles et décidés par les machines de Turing.

- Les langages sensibles au contexte sont générés par les grammaires sensibles au contexte (T1) (acceptés par les automates à borne linéaire) et décidés par les machines de Turing.

- Les langages généraux sont générés par les grammaires non restreintes (T0) et acceptés par les machines de Turing.