Algorithme et Machine de Turing

Algorithme

- Suite finie et organisée d'actions permettant d'aboutir de façon certaine à la solution déterminée d'un problème donné.
- Un algorithme est réalisé dans le but de résoudre un problème
- Un problème est une question générique
  Problème de décision = problème dont la réponse est binaire

Machine de Turing $M = (Q, \Gamma, \Sigma, \delta, q_0, B, F)$

Q ensemble fini d'état
Γ alphabet du ruban
$\Sigma \subseteq \Gamma$ alphabet d'entrée
$q_0 \in Q$ est l'état initial
$B \in \Gamma \setminus \Sigma$ le symbole blanc
$\delta : Q \times \Gamma \to Q \times \Gamma \times \{G, D\}$ fonction de transition
$F \subseteq Q$ ensemble des états finaux

La machine de Turing est composée de :
- mémoire infinie divisée en case = ruban
- tête de lecture qui se déplace sur le ruban
- ensemble fini d'états
- fonction de transition

M déterministe $\iff \forall q \in Q, \forall s \in \Gamma \quad Card(\delta(q,s)) \leqslant 1$

Langage accepté par M = ensemble des mots w_0 qui permettent d'atteindre un état final en n'ayant pas nécessairement lu tout le mot.

Langage décidé ① Langage accepté par M et ② M_n'a pas d'exécution infinie

Thm
- Les langages reconnus par une procédure effective sont ceux décidés par une machine de Turing déterministe
  $\iff$ Un problème de décision admet une solution ssi $\exists$ une machine de Turing qui le décide
- Pour tout langage L accepté par une machine à k rubans M_k, $\exists$ une machine de Turing à 1 ruban qui accepte L
- Pour tout langage L accepté par une machine de Turing Non-Déterministe, $\exists$ une machine de Turing Déterministe qui accepte L.
- $\exists$ des langages qui ne sont pas décidés par une machine de Turing.

Complexité temporelle

- Soit M machine de Turing qui s'arrête $\forall$ mot d'entrée w
  La complexité temporelle de M est le nombre de transitions franchies par M pour un mot d'entrée w
- f et g de $\mathbb{N} \to \mathbb{R}^+$
  $f(n) = O(g(n))$ (ssi) $\exists c$ et n_0 / $\forall n \geqslant n_0$ $f(n) \leqslant c\, g(n)$
  $f(n) = \Omega(g(n))$ (ssi) $\exists c$ et n_0 / $\forall n \geqslant n_0$ $f(n) \geqslant c\, g(n)$
  f et g sont de même ordre $f = \Theta(g)$ et $g = \Theta(f)$ (ssi) $f = O(g)$ et $g = O(f)$
  f est négligeable devant g $f(n) = o(g(n))$ ssi $\frac{f(n)}{g(n)} \xrightarrow[n \to +\infty]{} 0$
- $\forall$ machine de Turing à k rubans M_k, de complexité $t(n)$ (avec $t(n) \geqslant n$), $\exists$ M_1 qui reconnait le même langage que M_k de complexité asymptotique $O(t^2(n))$
- $\forall$ M_N (non déterministe) à 1 ruban de complexité $t(n)$, $\exists$ M_D à 1 ruban qui reconnait le même langage que M_N et dont la complexité est $2^{O(t(n))}$

Complexité Spatiale

Soit M machine de Turing qui s'arrête pour tout mot d'entrée w. La complexité spatiale de M est le nombre de case du ruban utilisées par M pour un mot d'entrée w

Preuve d'algorithme

While-programs

La sémantique opérationnelle de WP est la relation de transition $\langle P_1, \lambda_1 \rangle \rightarrow \langle P_2, \lambda_2 \rangle$

Terminaison

Un WP termine $\iff$ il atteint finalement une configuration dont l'état est le programme vide

Conection Partielle

PreCond = formule que l'on suppose satisfaite par les paramètres du WP
PstCond = formule $\Lambda c \in$ ensemble des contextes pour lesquels PstC est vrai

Technique de preuve

P précond Q PstCond S instructions $\{P\}\ S\ \{Q\}$

$$\frac{\text{prémices}}{\text{conséquences}}$$

Instruction « skip »

$$\frac{}{\{P\}\ \text{skip}\ \{P\}}$$

Instruction « := »

$$\frac{}{\{P[e/x]\}\ x := e\ \{P\}}$$

Instruction « ; »

$$\frac{\{P\}\ S_1\ \{Q\},\ \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1, S_2\ \{R\}}$$

Instruction « if... then... else ... endif »

$$\frac{\{P \wedge B\}\ S_1\ \{R\},\ \{P \wedge \neg B\}\ S_2\ \{R\}}{\{P\}\ \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ endif}\ \{R\}}$$

Instruction « while ... do ... endwhile »

P est l'invariant

$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \text{while } (B) \text{ do } S \text{ endwhile}\ \{P \wedge \neg B\}}$$

```
{PréC}
[0]
{P}
while B do
    [1]
    {P}
    S
    {P}
endwhile
[2]
{PstC}
```

Il faut montrer les obligations de preuve

[0] PréC implique P_i

[1] $P \wedge B$ implique P'

[2] $P \wedge \neg B$ implique PstC

Exemple

PreCond = {vrai} $B = (b' > 0)$

```
{a*b = 0 + a*b} = Pi
b' := b;
{a*b = 0 + a*b'}
m := 0;
{a*b' = m + a*b'}
while (b'>0) do
    {a*b = m + a + a*(b'-1)} = P'
    m := m + a;
    {a*b = m' + a*(b'-1)}
    b' := b' - 1
    {a*b = m + a*b'} = invariant = P
endwhile
PstCond = {m = a*b}
```

[0] $\dfrac{\text{vrai}}{(a*b = 0 + a*b)}$

[1] $\dfrac{(a*b = m + a*b') \wedge (b' > 0)}{(a*b = (m+a) + a*(b'-1))}$

[2] $\dfrac{(a*b = m + a*b') \wedge !(b' > 0)}{(m = a*b)}$

Les obligations de preuve deviennent

[0] $\dfrac{\text{PréC}}{P_i}$ [2] $\dfrac{P \wedge \neg B}{\text{PstC}}$

[1] $\dfrac{P \wedge B \wedge (m = vol)}{P' \wedge (m' < vol)}$

Preuve de Terminaison

Pour montrer la terminaison d'une boucle while, on utilise la méthode des espaces bien fondés. On prend une mesure de tel sorte qu'elle décroit à chaque itération de la boucle.
On pose m = mesure.

avec vol une valeur qcq et m' équivalent de m après la boucle

Dans l'exemple ci dessus mesure = b' et on prend vol = X
comme dans la boucle on fait b' := b' - 1 m' = b' - 1