03-Algorithme

PEPS

Comment passe-t-on du programme à l'algorithme?

- la thèse de Church-Turing
  - la machine de Turing
  - les fonctions récursives
  - le lambda calcul
- le modèle des Evolving Algebra
  - la définition axiomatique
  - les machines à états abstraits
  - la complétude algorithmique
- modèles d'algorithmes
  - grandes classes d'algorithmes
  - les algorithmes parallèles
  - les algorithmes distribués

La thèse de Church-Turing

T1. *Ce qui est effectivement calculable l'est par une machine de Turing*

T2. *Ce qui peut être calculé par une machine peut l'être par une machine de Turing.*

T3. *Tout processus qui admet une description mathématique peut être simulé par une machien de Turing.*

La machine de Turing

Un ruban T : $Z \rightarrow \Gamma$. Γ est un alphabet ({'0', '1', '#'} par exemple)

Une tête de lecture i (à valeur dans Z) et deux opérations sur Z (suc et pre, successeur et précédesseur). *Remarque* les élément de Z ne sont pas nommables.

Un état q (à valeur dans un ensemble fini Q : {q1, q2, qS}

Un programme dont les instructions sont de la forme:

```
si T(i) = '0' et q = q2 alors T(i) <- '#', i <- pre(i), q <- qS
```

On exécute le programme **tant que** cela est possible

Exemple : https://interstices.info/jcms/nn_72391/comment-fonctionne-une-machine-de-turing

Les systèmes d'équation d'Herbrand-Gödel

Des fonctions de bases : $0[\vec{X}] = 0$, $Succ[X] = X + 1$, $\pi_i[\vec{X}] = X_i$ les projections

Des schémas de construction:

1. la composition : si $h : N^k \to N$ et $g_i : N^p \to N$ (pour $i \in 1, \ldots, k$) sont des fonctions récursves alors $f(\vec{x}) = h(g_1(\vec{x}), \ldots, g_k(\vec{x})) : N^p \to N$ est une fonction récursive.
2. le schéma de récurrence : si $h : N^{k+2} \to N$ et $g : N^k \to N$ sont définis par des équations récursives alors $f : N^{k+1} \to N$ définie par $f(0, \vec{x}) = g(\vec{x})$ et $f(n+1, \vec{x}) = h(n, f(n, \vec{x}), \vec{x})$

Si on oreinte les équations on obtient un système de réécriture qui correspond à l'exécution d'un calcul.

Théorème: le modèle des équations de HG est Turing complet.

La classe des fonctions récursives

la minimisation : si $f : N^{k+1} \to N$ alors $g : N^k \to N$ définie par $g(\vec{x}) =$ le plus petit y tel que $f(\vec{x}, y) = 0$ est une fonction

Théorème: la classe des fonctions récursives correspond à la classe des fonctions calculables par une machine de Turing.

Le lambda calcul

x, y, ... sont des termes

si t et u sont des termes alors l'application $t\ \ u$ est un terme (u est appliqué à t)

si t est un terme, alors $\lambda x.\, t$ est un terme

La règle de réduction : $\lambda x.\, t\ \ u \Rightarrow t[x \leftarrow u]$ (u remplace x dans t, pour x libre dans t)

Théorème: Le λ-calcul est Turing complet

Les lacunes algorithmiques

Les simulations entre modèles ne sont pas pas-à-pas, elles sont souvent polynomiales.

Exemple sur des programmes qui vérifient si un mot est un palindrome

```
def palindromeLINEAIRE(mot):
    i = 0
    j = len(mot)-1 #on peut le calculer!
    while (i < j):
        if (mot[i]==mot[j]):
            i = i + 1
            j = j - 1
        else:
            break
    if (j <= i):
        print ("%s est un palindrome"%mot)
    else:
        print("%s n'est pas un palindrome"%mot)
```

```
def palindromeQUADRATIQUE(mot):
    for l in mot:
        k = k + 1
    k = k - 1
    etat = 1
    while (etat == 1):
        if (j >= k):
            etat = 2
        elif (mot[i]=='a'):
            j = j + 1
            while (i < k):
                i = i + 1
            if (mot[i]=='a'):
                k = k - 1
                while (i > j):
                    i = i - 1
            else:
                etat = 0
        elif (mot[i]=='b'):
            j = j + 1
            while (i < k):
                i = i + 1
            if (mot[i]=='b'):
                k = k - 1
                while (i > j):
                    i = i - 1
            else:
                etat = 0


  if (etat != 0):
        print '-%(mot)s- est un palindrome' % {"mot" : mot}
    else:
        print("\"%s\" n'est pas un palindrome"%mot)
```

Théorème (Colson, 91): le calcul du minimum de 2 entiers m et n est programmable en temps le minimum

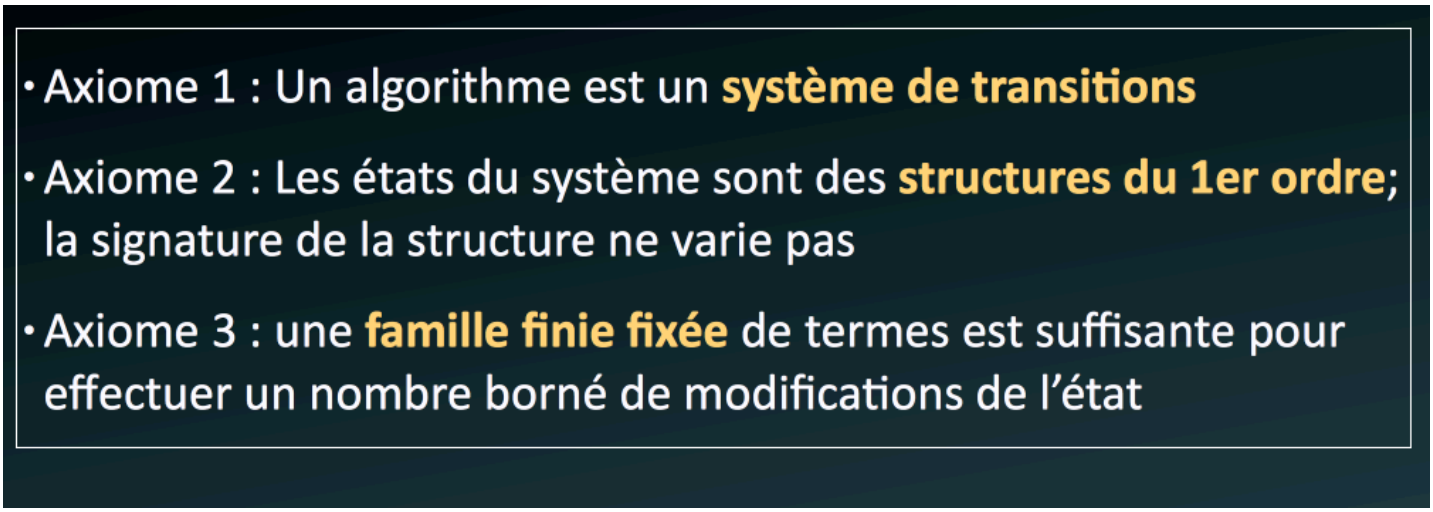
des 2 entrées sur une machine de Turing avec 2 bandes MAIS il n'est pas programmable en temps moins que $O(n)$ ou $O(m)$ dans le langage de la récursion primitive.

Le modèle des Evolving-Algebra

Définition axiomatique

Un algorithme séquentiel est défini par

1. [**Séquentialité**] une fonction de transition
2. [**Etat**] un état qui est une structure logique du premier ordre
3. [**Finitude**] Modification/Lecture finies bornées

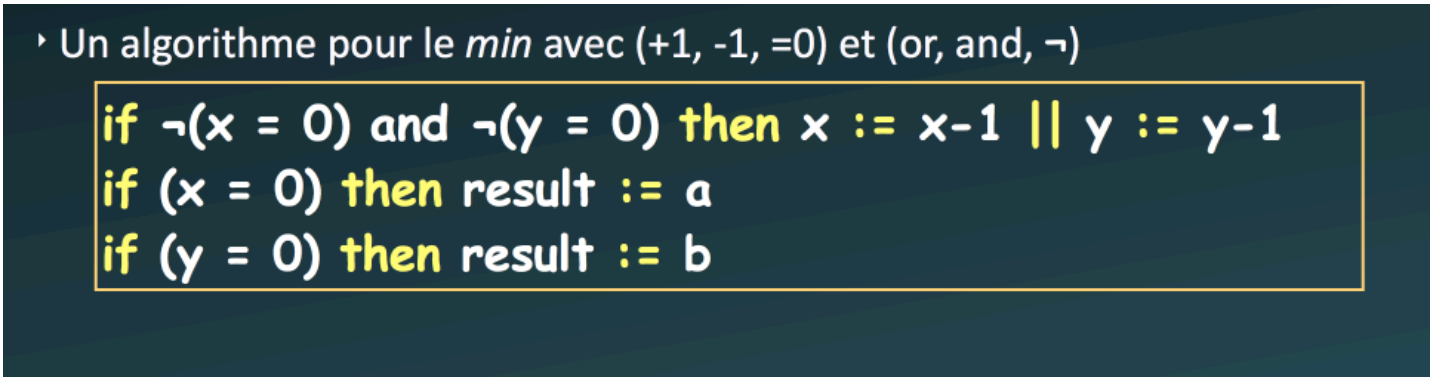


	how data is structured	how data evolves
Turing mach. (1936) How Gandy sees it:	infinite tape ≡ growing array	test & set program
Kolmogorov (1953) Schönhage (1970)	dynamic graph undirected, bounded degree directed, bounded fan-out	test & set program
Cook & Reckow (1973) RAM	Indirect access	test & set program
Gurevich (1984) Abstract State Mach. (aka Evolving Algebras)	Algebra on a multisort domain (= finite tuple of partial functions)	test & set program

ultimate extension

Les machines à états abstraits

1. une algèbre (et sa sémantique)
2. un programme contenant les instructions suivantes
   - `if C then f(t1, ..., tN) := t0` (conditionnelle et mise à jour)
   - `I1 || ... || Ik` (simultanéité)



Complétude algorithmique

Soit C une classe d'algorithme, on dira qu'un modèle M est C-algorithmiquement complet si tout algorithme de C est k-simulable dans M. [k-simulable ≡ une étape dans C se fait en au plus k étapes dans M]

Autres modèles d'algorithmes

Classes de résolutions de problèmes

- méthode par retour arrière [backtracking] : à certain moment dans le calcul on revient en arrière à un endroit "marqué" (mémorisé)
- méthode diviser pour régner : on découpe le problème en sous-problèmes sur lequel s'applique aussi la méthode et on reconstruit le résultat en "fusionnant"
- méthode dynamique : on construit la solution en augmentant le domaine sur lequel on agit
- méthode exhaustive : on regarde tous les cas possibles
- méthode aléatoire : on fait un choix "aléatoire" au cours de la méthode

Algorithmes parallèles

Plusieurs endroits pour faire le calcul: les calculs sont synchronisés [les différentes machines "s'attendent" régulièrement]. Il y a le même problème sur chacune des machines [ce qui ne veut pas dire qu'ils exécutent la même instruction au même moment]

- Automates Cellulaires
- Bulk Synchronous Parallel (BSP)
- Parallel RAM
- ...

Algorithmes distribués

Des programmes différents, le temps n'est pas le même pour chacune des machines. La communication se fait par échange de messages.