INFO-F408: Computability & complexity

Rémy Detobel

18 Septembre 2017

1 Introduction

C'est donc un cours de math, permettant de classer les problèmes, leur complexité, ... Il s'agit de la suite du cours d'"informatique fondamentale".

1.1 Référence du cours

— Introduction to the Theory of computation 2nd Ed. Michael SIPSER.
— Hilbert (mathématicien), Gödel (il aurait prouvé que complètement transformer les mathématiques en "programme" n'est pas possible), Turing, Kleene, Church, Cook-Levin (NP et ses implications), Karp

1.2 Définition

— **What problems can be solved by a computer ?**
On doit définir plusieurs mots : "problems", "solved" et "computer".
— "Efficace" : quel est réellement sa définition. Que veut-on optimiser : le temps, l'énergie utilisé, la place, ressource naturelle (pour créer la machine par exemple).
— **How much resources are "required" to solve a problem ?**
— Parfois deux problèmes sont très différent mais en résolvant un des deux, on peut très facilement solutionner le problème en changeant peu l'algorithme.
— **Mathematical Treatment**
Il y a une certaine formalisme à avoir pour pouvoir caractériser, classer, ... les problèmes. Il y a différent niveau de langages. Il ne faut pas se satisfaire de comprendre en général mais bien en profondeur.
— **Compilateurs, langages de programmation**
— **Optimisation : problème linéaire (LP), algorithme génétique, problème de satisfaisabilité, plus court chemin**
— **P vs NP (traveling problem (voyageur de commerce))**

1.3 Connexion avec d'autres matières

Biologie, physique (ordinateur quantum (pas abordé dans ce cours)), ...

1.4 Quelques concepts de bases

Sets, logique booléen, séquences, relations, fonctions, graphes, "strings"/chaînes de caractères, preuve par l'absurde ("proof by contradiction").
Référence : chapitre 0 du livre pour se rappeler.

1.5 Problème de décision

Problème où la réponse est "oui" ou "non". Certains peuvent être résolus par des algorithmes et des ordinateurs. Il faut donner des informations qui permettent de décider si la réponse est "oui" ou "non". C'est là qu'est le problème principal des algorithmes. On va donc utiliser seulement des problèmes dont la solution est déjà définie

par l'entrée (e.g. y a-t-il un chemin entre deux points d'un graphe).

— well-defined
— self-contrained

Il y a une différence entre problème et une instance. Le problème est plus général qu'une instance. Une instance est un cas particulier d'un problème, dans lequel on a donc définit les entrées/inputs. Chaque "input" possible est donc une instance...
Problème : la liste des entrée pour lesquelles l'entrée est "oui".

1.5.1 Exemples

Problème : dire si un nombre est pair ou non.
Solution : tous les nombres pairs (donc tous les nombres impairs sont faux).

Problème : un graphe est-il connexe (donc est-il possible d'aller de n'importe quel point dans le graphe vers n'importe quel autre point). **Solution :** tous les graphes connexes (on peut les encoder sous forme de matrice (0 pas de lien, 1 si un lien)).

1.5.2 Input/Entrée

Les entrées/input doivent appartenir à un langages (qui peut être $\{0, 1\}$) : une liste de mots/chaînes de caractères. On considérera aussi que lorsque l'input n'est pas bon, la réponse est "non". Pour le second exemple donc, si on ne reçoit pas un graphe en entrée, on répondra "non". Il faudra donc que l'algorithme vérifie d'abord l'entrée avant d'essayer de résoudre.

1.6 Problème de décision $\neq$ problème d'optimisation

Bisection/dichotomie
Un problème de décision peut aider à savoir s'il est optimisé...
Exemple (problème de décision) : est-ce que le chemin coûte moins que la valeur "d" ? On prend "d" comme le milieu entre 0 et le maximum (nombre de point dans le graphe). **Transformé en problème de décision** : en général c'est possible en reformulant le problème.

2 Automate fini

Cela permet de déjà commencer à répondre à la question "qu'est-ce qu'un ordinateur". 5-uple : $(Q, \Sigma, \delta, q_0, F)$
Référence : chapitre 1.1 du livre

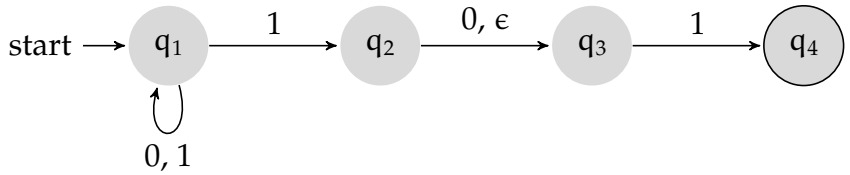
Q = nombre fini d'état (finite set of states).
Σ = alphabet utilisé par le problème (finite set called the alphabet)
$\delta : Q \times \Sigma \to Q$ (produit cartésien (cartesian product)) fonction de transition
$q_0 \in Q$ c'est l'élément de départ (the start state)
$F \subseteq Q$ liste des états finaux (set of accepting states)

2.1 Regular language

Définition : livre point 1.16
A language is called a regular language if some finite automaton recognizes it. (= Regular Exepression (REGEX))
Finite automaton M defines a language $L(M)$.
Exemple : $A = \{w \in \{0,1\} * | w$ contains at least on 1 and an even number of 0 follow the last 1$\}$

3 No determinism / Non déterministe

Référence : chapitre 1.2 du livre
Les automates finis sont déterministes.
ϵ = "empty" symbol



Dans un automate non déterministe le δ ne sera pas définit de la même manière. Il sera écrit :

$$\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$$

$P(Q)$ liste d'états dans Q (set of states in Q)
Les automates "non déterministes" reconnaissent exactement les mêmes langages que les automates "déterministes". Cela dit, les automates non déterministes sont plus simple à écrire et à comprendre au premier abord (la preuve arrivera).
Voir livre 1.39.
Chaque NFA (automate non déterministe) à un équivalent en DFA (automate déterministe).