Méthodologie de la programmation

Pablo Rauzy
pr@up8.edu
pablo.rauzy.name/teaching/mdlp

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS
UFR MITSIC / L1 informatique

Séance c
Un aperçu de la calculabilité

Un aperçu de la calculabilité

- La calculabilité est une branche de la logique et de l'informatique théorique.
- C'est une science qui cherche à caractériser ce qu'on peut calculer à l'aide d'un algorithme. Autrement dit, à comprendre les limites de ce que peuvent faire les ordinateurs.
- La formalisation de ces notions a commencé dans les années 1930, en particulier suite à l'introduction du *Entscheidungsproblem* (le problème de la décision) par Hilbert.

- Intuitivement, une fonction f est calculable si étant donné un argument x, il existe une méthode précise qui permet de calculer $f(x)$ en un nombre fini d'étapes.
- Il existe plusieurs formalisations de ce que peut être la méthode de calcul :
  - fonctions récursives,
  - machines de Turing,
  - automates cellulaires,
  - lambda-calcul,
  - circuits booléens,
  - machines à compteurs,
  - etc.
- On peut montrer qu'elles sont équivalentes, c'est à dire qu'elles définissent exactement les mêmes fonctions calculables.

- Intuitivement, une fonction f est calculable si étant donné un argument x, il existe une méthode précise qui permet de calculer $f(x)$ en un nombre fini d'étapes.
- Il existe plusieurs formalisations de ce que peut être la méthode de calcul :
  - fonctions récursives,
  - machines de Turing,
  - automates cellulaires,
  - lambda-calcul,
  - circuits booléens,
  - machines à compteurs,
  - etc.
- On peut montrer qu'elles sont équivalentes, c'est à dire qu'elles définissent exactement les mêmes fonctions calculables.
- Formellement, une fonction calculable est donc une fonction calculable selon l'une de ces définitions.

- Voyons un aperçu de ce à quoi ressemblent quelques uns de ces modèles de calcul.

- Une machine de Turing est une machine abstraite, qui comporte les éléments suivants :
  - Un *ruban infini* divisé en cases consécutives. Chaque case contient un symbole parmi un *alphabet fini* (dont un symbole “blanc”).
  - Une *tête de lecture/écriture*, qui peut lire et écrire un symbole sur le ruban, et se déplacer à droite ou à gauche d’un cran.
  - Un *registre d’état*, qui mémorise l’état courant de la machine de Turing parmi un *ensemble d’états fini* (dont un état spécial de départ).
  - Une *table d’action* qui indique à la machine quel symbole écrire sur le ruban, dans quelle direction déplacer la tête, et quel est le nouvel état, en fonction de l’état courant et du symbole lu sur le ruban (si la combinaison actuelle n’existe pas dans la table, la machine s’arrête).

- Définir une machine de Turing qui ajoute 1 à un nombre en binaire.

Automates cellulaires

- Une automate cellulaire est composé d'une *grille* de cellules qui peuvent être dans un nombre fini d'*états* qui évoluent au cours du temps.
- L'état d'une cellule au temps $t + 1$ est définit en fonction son état et des états des *cellules voisines* au temps t.
- À chaque nouvelle unité de temps, les mêmes *règles de transition* sont appliquées à chacune des cellules pour calculer le nouvel état de grille dans son ensemble.

Le jeu de la vie

- Le jeu de la vie est un exemple très connu d'automate cellulaire en deux dimensions à grille "carré" (voisinage de 8 cellules) et à deux états (vivant/mort).
- Les règles sont :
  - Si la cellule est vivante et entourée par deux ou trois cellules vivantes, elle reste en vie à la génération suivante, sinon elle meurt.
  - Si la cellule est morte et entourée par exactement trois cellules vivantes, elle naît à la génération suivante.
- Ces règles peuvent sembler simples, mais cet automate est en fait *Turing-complet*.

- L'idée de base du λ-calcul est que *tout est fonction*.
- Ces fonctions peuvent contenir des fonctions qui ne sont pas prédéfinie et qui sont alors des variables.
- La seule chose qu'on peut faire avec ces fonctions est de les *appliquer* à des valeurs (qui sont elles-mêmes des fonctions).

Syntaxe

- La syntaxe du λ-calcul est minimaliste.
- Si E est une expression du λ-calcul (un λ-terme) alors E est soit :
  - une *variable* : x, y, ... sont des λ-termes ;
  - une *application* : uv est un λ-terme si u et v sont des λ-termes ;
  - une *abstraction* : $\lambda x.v$ est un λ-terme si x est une variable et v est un λ-terme.
- L'application uv peut être vue comme l'application d'une fonction u à la valeur v.
- L'abstraction $\lambda x.v$ peut être vue comme la fonction qui à x associe v (où v contient généralement des occurrences de x).

Variables libres et variables liées

- Comme avec les quantificateurs classiques $\exists$ et $\forall$, les variables sont *liées* par le λ.
- Les variables qui ne sont pas liées sont dites *libres*.
- Tout comme $\forall x, P(x)$ est équivalent à $\forall y, P(y)$, les deux termes $\lambda x.P$ et $\lambda y.P[x/y]$ sont équivalent.
- Exemples :
  - Dans $\lambda x.xy$, x est liée et y est libre, du coup il est équivalent à $\lambda z.zy$ mais pas à $\lambda x.xz$.
  - Le terme $\lambda b.\lambda n.banane$ est équivalent à $\lambda p.\lambda t.patate$.
- L'opération de renommage des variables, qui est parfois utile pour clarifier un terme, s'appelle la *α-conversion*.

Réduction

- Le calcul se fait par *réduction*, défini comme une réécriture :
  - $(\lambda x.v)y \mapsto v[x/y]$.
- Exemples :
  - $(\lambda x.xy)a \mapsto ay$.
  - $(\lambda b.\lambda n.banane)p \mapsto \lambda n.panane$.
  - $(\lambda n.panane)t \mapsto patate$.
- On appelle cette opération la *β-réduction*.

- Un circuit booléen est un *graphe orienté acyclique* fini et connexe dont les feuilles sont les entrées et l'unique racine est la sortie.
- Les sommets sont des portes logiques (par exemple **et**, **ou**, et **non**).

Machines à compteurs

- Une machine à compteurs est composée de (au moins) deux *compteurs* (entier naturel non borné) et d'un *programme*.
- Le programme est une suite d'instructions de la forme :
  - incrémente c_1,
  - incrémente c_2,
  - décrémente c_1,
  - décrémente c_2,
  - si $c_1 = 0$, aller à l'instruction i_1, sinon à l'instruction i_2,
  - si $c_2 = 0$, aller à l'instruction i_1, sinon à l'instruction i_2,

  où c_1 et c_2 sont les deux compteurs, et i_1 et i_2 sont les numéros d'instructions dans le programme.

- La thèse de Church (ou de Church-Turing), existe sous deux formes : physique et psychologique.
- Dans sa forme physique, elle affirme que la notion physique de calculabilité, définie comme étant tout traitement systématisable par un processus physique ou mécanique, peut être exprimé par un ensemble de règles de calcul, défini de plusieurs façons qu'on peut montrer équivalentes.
- Dans sa forme psychologique, elle affirme que la notion intuitive de calculabilité, correspond à ces mêmes règles formelles.

Formulation de la thèse

- On considère qu'une méthode de calcul doit :
  - consister en un ensemble fini d'instructions simples et précises qui sont décrites avec un nombre limité de symboles ;
  - toujours produire le résultat en un nombre fini d'étapes ;
  - pouvoir en pratique être suivie par un humain avec papier/crayon ;
  - ne pas demander plus d'intelligence que ce qui est utile pour comprendre et exécuter les instructions.

Calcul quantique

- En 1982, Feynman a pu montrer que les modèles de calculs classiques sont capables de calculer l'évolution de processus quantiques, mais de manière si inefficace que c'est inapplicable en pratique.
- Des modèles de calcul quantiques ont été proposés par David Deutsch en 1985, et montrent que les ordinateurs quantiques ne sont a priori pas plus puissants que les ordinateurs classiques en terme de calculabilité.
- Cependant, on a pu montrer qu'ils sont capables de générer des nombres purement aléatoires, mais ce n'est pas considéré comme un calcul.

- Il peut être démontré qu'il existe des fonctions non-calculables.
- En effet, il y a un nombre dénombrables d'algorithmes, mais un nombre indénombrable de fonctions de par le *théorème de Cantor*.

Le théorème de Cantor

- Le théorème de Cantor dit que le cardinal d'un ensemble E est toujours strictement inférieur au cardinal de l'ensemble de ses parties $P(E)$.
- On montre cela par une *diagonalisation*.
- L'idée est de montrer qu'il n'existe pas de bijection $f : E \to P(E)$.
- Pour cela, on construit un ensemble $D = \{x \in E | x \notin f(x)\}$ qui n'a pas d'antécédent (c'est à dire qu'il y a pas d'élément $e \in E$ tel que $D = f(e)$.
- En effet, si on suppose le contraire, soit e est dans D et donc n'appartient pas à D par construction ; soit e n'est pas dans D et donc appartient à D par construction.
- Il n'existe donc aucune fonction surjective de E dans $P(E)$.

Le nombre de fonction

- On note B^A l'ensemble des fonctions de A dans B.
- Le cardinal $|B^A|$ de l'ensemble des fonctions de A dans B est égale à $|B|^{|A|}$.
- L'ensemble des parties d'un ensemble A correspond à l'ensemble des fonctions de de A dans $\mathbb{B}$.
- L'ensemble des fonctions est donc bien strictement plus grand que l'ensemble des algorithmes.

Exemples de fonctions non-calculables

- L'exemple le plus courant est celui du *problème de l'arrêt*.
- Un autre exemple plus perturbant est celui du *castor affairé*.
- Un autre exemple plus joli est celui du *nombre Ω de Chaitin*.

Le problème de l'arrêt

- Le problème de l'arrêt est un problème de *décision* qui détermine si un programme s'arrête ou non.
- Cela se prouve par diagonalisation.
- On suppose qu'un programme $halt(p, a)$ existe tel que :
  - si $p(a)$ s'arrête, alors $halt$ accepte (p, a) en temps fini,
  - si $p(a)$ ne s'arrête pas, alors $halt$ refuse (p, a) en temps fini.
- On peut alors construire le programme $diag(x)$:
  - si $halt$ accepte (x, x), faire une boucle infinie,
  - sinon, accepter.

Le problème de l'arrêt

- Le problème de l'arrêt est un problème de *décision* qui détermine si un programme s'arrête ou non.
- Cela se prouve par diagonalisation.
- On suppose qu'un programme $halt(p, a)$ existe tel que :
  - si $p(a)$ s'arrête, alors $halt$ accepte (p, a) en temps fini,
  - si $p(a)$ ne s'arrête pas, alors $halt$ refuse (p, a) en temps fini.
- On peut alors construire le programme $diag(x)$:
  - si $halt$ accepte (x, x), faire une boucle infinie,
  - sinon, accepter.
- On a un paradoxe avec $diag(diag)$.

Le castor affairé

- Le jeu du castor affairé à n états correspond à trouver la machine de Turing à n états qui va écrire le plus de 1 possible sur sa bande avant de s'arrêter.
- Cette fonction croît plus rapidement que n'importe quelle fonction calculable.
- Déterminer le castor affairé les machines de Turing à n états est donc un problème insoluble algorithmiquement.
- En pratique, on ne peut même pas espérer le résoudre pour un nombre n au-delà de 10.
  - Pour l'instant on ne connaît les valeurs exactes que jusqu'à $n < 5$ (1, 6, 21, 107).
  - Pour $n = 5$ le record (de 1989) produit 4098 1 en 47 176 870 étapes.
  - Pour $n = 6$, le record (de 2010) produit plus de 10^{18267} 1 en plus de 10^{36534} étapes.

Le nombre Ω de Chaitin

- Le nombre Ω de Chaitin est tout à fait passionnant.
- Il est défini comme la probabilité qu'un programme choisit aléatoirement s'arrête.
- Il a ceci d'étonnant que bien qu'étant parfaitement défini, il est totalement incalculable car chacun de ses bits dépend du problème de l'arrêt.
- C'est aussi un nombre parfaitement aléatoire (au sens où il est incompressible).