Informatique en MP(2)I

Marc de Falco

Introduction à l'informatique

■ **Note 1 Roadmap :**

- tout reprendre. Je suis peu satisfait du résultat.

■

Dans ce chapitre, on présente de manière très survolée l'informatique. On aura l'occasion de revenir en détail sur certaines notions. L'objectif est avant totu d'avoir une vision claire même si elle est naïve de

- ce qu'est un problème informatique
- ce qu'est un programme
- ce qu'est une machine
- ce que signifie de résoudre informatiquement un problème

Cela permet de répondre à des questions comm :

- qu'est-ce qui permet de dire qu'on programme en OCaml alors qu'on a l'impression de ne faire que des suites de définitions ?
- peut-on se contenter de programmer sans chercher à comprendre ?

Précisons ici qu'on parle d'informatique mais que le terme le plus proche de ce que l'on étudie en anglais est *computer science* par complémentarité avec des notions comme celles de *computer engineering* plus proches de considérations matérielles. Il n'y a pas une *unique* vision de l'informatique mais plutôt une grande richesse de point de vue.

I Données, Problème, Algorithme

I.1 Données

Définition I.1 Une *donnée informatique* est un élément d'information fini qu'on peut stocker ou transmettre.

Selon le niveau d'abstraction auquel on se place, on peut considérer une donnée comme étant une succession de bits, c'est-à-dire de valeur valant 0 ou 1, ou comme une donnée plus structurées, comme un texte, une image, un graphe, ...

En informatique, on adapte souvent son point vue sur les notions de base comme celle-ci. Si on fait de la transmission ou de la compression de données, c'est important d'en avoir une représentation la plus primitive possible comme des mots binaires. Mais si s'intéresse à des notions de chemins dans un graphe, il est plus adéquat de considérer un graphe comme un élément de données même si la question de sa représentation n'est pas immédiate.

I.2 Problèmes

Définition I.2 Un *problème informatique* est un ensemble de questions parametrées par des données (les *entrées*) et dont les réponses peuvent être

- soit Oui ou Non, auquel cas on parle de problème de *décision*
- ou d'autres données dépendant des entrées : les *sorties*, on parle alors de problème de *recherche* ou de *construction*.

Exemple

- Étant donné un entier, déterminer s'il est premier ou non.
- Étant donnés deux entiers, calculer leur PGCD.
- Étant donné un graphe pondéré et deux sommets, déterminer un chemin de longueur minimale permettant d'aller de l'un à l'autre.
- Étant un programme informatique et une entrée, déterminer si le programme va s'arrêter ou non.
- Étant donné une image et une base de donnée de visages, déterminer les personnes présentes sur l'image.

On voit que certains de ces problèmes n'ont pas eu besoin d'attendre l'informatique, comme les problèmes d'arithmétique.

On peut prouver qu'un problème comme le problème de l'arrêt ne pourra jamais être résolu par un ordinateur (on dira qu'il est *indécidable*).

I.3 Algorithme

Définition I.3 Un *algorithme* est un procédé systématique et mécanisable permettant de résoudre un problème informatique.

Ce qu'on signifie ici par le mot *systématique* est que le procédé de calcul est précis et non ambigu et par *mécanisable* qu'il s'agit d'un calcul qui puisse être fait par une *machine* dans un sens à venir.

Une autre manière de voir les problèmes est de dire que ce sont des fonctions mathématiques des entrées vers les sorties. Un algorithme est alors une description de la réalisation d'une telle fonction.

II Modèle de calcul, Machines, Programmes

On donne ici des définitions volontairement naïves et imprécises. Elles pourraient être précisées en fixant un cadre théorique plus conséquent, ce qui est prématuré à ce stade.

Définition II.1 Un modèle de calcul est un cadre permettant de décrire comment calculer la solution de problèmes en fonction des entrées.

Définition II.2 Une *machine* est un modèle de calcul défini par un ensemble de *configurations* et des opérations élémentaires, aussi appelées *instructions*, permettant de passer d'une configuration à une autre.

Ces instructions sont regroupées sous la forme d'une suite finie appelée un *programme*.

Pour éxecuter un programme, il y a dans la configuration un entier qui s'appelle le *pointeur d'instruction* et qui

donne le numéro dans le programme de l'insrtuction courante. Généralement, on part de la première instruction en passant ensuite, sauf contre-ordre, à l'instruction suivante, jusqu'à la fin du programme.

II.1 Exemple : ordinateur

Un ordinateur est naturellement un exemple de machine. Une configuration est la donnée de l'état des registres du processeur, de la mémoire et aussi des périphériques d'entrée-sortie. Un programme est une suite d'instructions exécutées par le processeur.

II.2 Exemple : les machines à deux compteurs

Les configurations d'une machine à deux compteurs sont des triplets (A, B, PC) où A et B sont deux compteurs entiers naturels, en général initialisés avec les entrées et dans lesquels on pourra lire les sorties, et PC est le pointeur d'intruction, initialisé à la première instruction.

Un programme est une suite finie et numérotée d'instructions élémentaires parmi les suivantes :

- `INCA` : incrémenter A, c'est-à-dire ajouter 1 à la valeur qu'il contient
- `DECA` : décrementer A, c'est-à-dire soustraire 1 à la valeur qu'il contient
- `IFA i j` sauter à l'instruction numéro i si A est nul, sinon sauter à l'instruction numéro j.
- les instructions `INCB`, `DECB` et `IFB i j` respectives vis-à-vis du compteur B.

Ainsi, on peut considérer le programme suivant

```
1: IFA 6 2
2: DECA
3: INCB
4: INCB
5: IFA 1 1
6: INCA
7: DECA
```

Si on initialise les compteurs (A, B) à $(n, 0)$ et qu'on exécute le programme, on obtient alors $(0, 2n)$ dans les compteurs. Ce programme réalise ainsi un doublement du compteur A et place le résultat dans le compteur B.

Exercice 1 1. Écrire un programme permettant de passer de la configuration (a, b) à $(0, a + b)$.
2. Écrire un programme permettant de passer de la configuration $(n, 0)$ à $(0, 0)$ si n est pair ou $(0, 1)$ sinon.

1.

```
1: IFA 5 2
2: DECA
3: INCB
4: IFA 1 1
5: INCA
6: DECA
```

2.

```
1: IFA 7 2
2: DECA
3: IFA 6 4
4: DECA
5: IFA 1 1
6: INCA
7: INCA
8: DECA
```

II.3 Exemple : le noyau fonctionnel pur d'`OCaml`

On considère ici le noyau `OCaml` avec les entiers, leurs opérations élémentaires, les fonctions, les définitions (récursives ou non) et ce qui correspond à l'évaluation.

Ainsi le terme `OCaml` suivant `fun n -> 2 * n` peut être vu simplement comme prenant en entrée un entier n et renvoyant son double.

Ce qui est important ici, sans qu'on s'attarde sur la définition précise de l'évaluation, c'est de comprendre qu'il s'agit d'un modèle de calcul sans avoir de notion de machine, d'instructions ou de programme.

Bien entendu, quand on interprète ou qu'on compile un programme `OCaml`, cela se passe sur un ordinateur, donc une machine.

On considère que les termes sont les programmes dans un tel langage de programmation. La différence principale entre un programme vu ainsi et un dans le sens usuel, comme un programme Python, c'est que l'on n'a pas conscience de la réalité la machine quand on programme : il n'y a pas de notion d'instructions. Bien sûr, on verra qu'il est possible de retrouver cela dans OCaml et ainsi de se ramener en terrain connu, mais c'est important de comprendre que ce noyau existe indépendamment d'une notion de machine.

Remarque C'est Alonzo Church qui a défini ce noyau, qu'on appelle le λ-calcul, en 1933 alors qu'il souhaitait présenter un cadre pour exprimer les fonctions calculables.

II.4 Exemple : Jeu de la vie

Le *jeu de la vie* défini en 1970 par John Conway dans le but d'être une récréation mathématique s'est révélé être un modèle de calcul très important.

Il s'agit d'un cas particulier d'une plus grande classe de modèles de calcul appelés les **automates cellulaires**.

Une configuration du jeu de la vie est une grille bidimensionnelle de valeur booléenne. On appelle chaque case une *cellule* et on dirait que le booléen permet de déterminer si elle *vivante* ou *morte*.

L'évaluation de ce modèle consiste à définir une nouvelle configuration en appliquant la règle suivante pour chaque cellule :

- si la cellule est vivante et a deux ou trois voisines vivantes, elle reste vivante dans la nouvelle configuration, sinon elle meurt.
- si la cellule est morte, elle devient vivante si et seulement si elle a exactement trois voisines vivantes.

Quand on parle de cellules voisines, on fait référence aux huit voisines directe sur la grille (on compte ainsi les diagonales).

Le point essentiel dans cette procédure permettant de passer d'une configuration à une autre est qu'on ne modifie pas la configuration courante, tout se passe comme si chaque cellule était modifié simultanément.

La configuration initiale est à la fois le programme et ses entrées. Un point critique ici c'est qu'il n'existe pas de notion de fin d'évaluation. En effet, même des configurations très simples n'atteignent pas un point fixe. On devra donc choisir en fonction du problème que l'on cherche à résoudre, comment déterminer la fin du calcul.

II.5 Exemple : FRACTRAN

FRACTRAN est un modèle de calcul également inventé par John Conway en 1987.

Un programme FRACTRAN est une suite finie de fractions d'entiers naturels comme :

$$P = \left(\frac{17}{91}, \frac{78}{85}, \frac{19}{51}, \frac{23}{38}, \frac{29}{33}, \frac{77}{29}, \frac{95}{23}, \frac{77}{19}, \frac{1}{17}, \frac{11}{13}, \frac{13}{11}, \frac{15}{2}, \frac{1}{7}, \frac{55}{1}\right)$$

Une entrée est un entier et pour *exécuter* un tel programme, on considère un entier courant n, initialisé à la valeur de l'entrée, et on parcourt les fractions de gauche à droite jusqu'à trouve une fraction $\frac{p}{q}$ tel que $n\frac{p}{q}$ soit un entier. Dans ce cas, c'est la nouvelle valeur de n et on recommence le processus. Sinon, le programme termine.

Exemple $(\frac{3}{10}, \frac{4}{3})$ sur l'entrée 14 va s'arrêter tout de suite, sur l'entrée 15, on va avoir 20, 6 puis 8 et s'arrêter.

Exercice 2 Exécuter P sur l'entrée 2 suffisament longtemps pour noter les quatres premières puissances de 2 prises par l'entier courant. Attention, cela nécessite sûrement de programmer l'évaluation.

■ **Preuve**

On remarque que les puissances de 2 qui apparaissent sont dans l'ordre : $2^2, 2^3, 2^5, 2^7, 2^{11}, 2^{13}, \ldots$. Ainsi ce programme énumère les nombres premiers.

■

Comment fonctionne FRACTRAN ? En fait, chaque fraction va être de la forme

$$\frac{p_1^{a_1} \dots p_k^{a_k}}{q_1^{b_1} \dots q_l^{b_l}}$$

où les p_i et les q_j sont des nombres premiers distincts.

Pour qu'on sélectionne cette fraction, il faut que n soit divisible par le dénominateur, donc si on écrit n en produit de facteurs premiers, il faut que les puissances correspondant à chaque q_i soit $\geq b_i$. On va alors les diminuer de b_i puis augmenter les puissances correspondant à chaque p_i de a_i.

Ainsi, si on considère qu'un nombre n s'écrit $n = \prod_{i \in \mathbb{N}} p_i^{a_i}$, avec les a_i presque tous nuls et $p_0 < p_1 < p_2 < \dots$ une énumération des nombres premiers, on voit qu'un nombre n'est alors qu'une configuration d'une infinité de compteurs indexés par des entiers et qui sont presque tous à 0. Une fraction est alors une règle de la forme :

$$C[i_1] \geq b_1, \dots, C[i_k] \geq b_k \to C[i_1] - b_1, \dots, C[i_k] - b_k, C[j_1] + a_1, \dots, C[j_l] + a_l$$

où on a nommé $C[0], C[1], \dots$ les compteurs.

FRACTRAN est donc une sorte de machine à compteurs dont les instructions sont exécutées comme dans un filtrage `OCaml` en cherchant de gauche à droite la première règle qui s'applique. Le point le plus délicat c'est que la condition permettant d'appliquer une règle est destructrice : on divise, et on ne peut pas utiliser le même compteur pour la condition et pour le résultat, car sinon on simplifierait la fraction.

Remarque **Comment fonctionne le programme P ?**

Pour cela, on va réecrire P en suivant la syntaxe précédente, comme on a forcément des décrements de compteurs correspondant à la condition d'application, il est inutile de les répeter et on écrira juste

```
2:-2 3:-2 |> 1:+2 0:+1
```

pour la fraction $\frac{18}{1225} = \frac{2.3^2}{5^2.7^2}$ car $p_0 = 2, p_1 = 3, p_2 = 5$ et $p_3 = 7$. Par soucis de lisibilité, on peut de plus nommer les compteurs par des lettres en commençant à a et donc écrire

```
c-2 d-2 |> b+2 a+1
```

Comme les régles de P ne font intervenir que des $+1$ et des -1, on pourra encore abréger en écrivant `a-` plutôt que `a-1` et `b+` plutôt que `b+1`.

Ainsi, P devient la suite de règles suivantes (numérotées pour y faire référence ensuite).

```
 0: d-f- |> g+
 1: c-g- |> a+b+f+
 2: b-g- |> h+
 3: a-h- |> i+
 4: b-e- |> j+
 5: j-   |> d+e+
 6: i-   |> c+h+
 7: h-   |> d+e+
 8: g-   |>
 9: f-   |> e+
10: e-   |> f+
11: a-   |> b+c+
12: d-   |>
13:      |> c+e+
```

On commence avec la configuration $n = 2 = 2^1$ donc uniquement 1 dans le compteur `a` et 0 dans les autres. On notera

$$a : 1$$

cette configuration.

On remarque donc que si on a la configuration

$$a : n - 1$$

on va appliquer la règle 11 jusqu'à obtenir la configuration

$$b : n - 1, c : n - 1$$

Ensuite, on applique la règle 13 puis successivement les règles 4 et 5 jusqu'à être dans la configuration

$$c : n, d : n - 1, e : 1$$

La règle 10 permet de passer à

$$c : n, d : n - 1, f : 1$$

On va effectuer une succession de règle afin de déterminer si $n - 1$ divise n. En fait, cette sucession de règle va déterminer si d divise c. Imaginons donc qu'on a la configuration

$$c : n, d : m, f : 1$$

on applique successivement les règles 0 et 1 pour obtenir la configuration

$$a : m, b : m, c : n - m, f : 1$$

La règle 9 s'applique alors pour passer à

$$a : m, b : m, c : n - m, e : 1$$

et on effectue alors la succession des règles 4 et 5 qu'on a déjà vu pour passer à

$$a : m, c : n - m, d : m, e : 1$$

Comme il n'y a plus rien dans **b** c'est la règle 10 qui s'applique et on repasse à

$$a : m, c : n - m, d : m, f : 1$$

On repasse alors dans les règles 0 et 1 jusqu'à obtenir

$$a : 2m, b : m, c : n - 2m, f : 1$$

et ainsi de suite. Pour l'arret, deux cas peuvent alors se produire :

- Soit on a vidé **c** en effectuant la soustraction de m, ce qui correspond au cas où m ne divise pas n, et alors la règle 1 ne pourra plus s'appliquer car elle nécessite $c \geq 1$. Si $n = qm + r$ avec $0 < r < n$, on aboutit alors avec une dernière application de 0 en

  $$a : n, b : r, d : m - r - 1, g : 1$$

  Comme la règle 1 ne s'applique plus, on passe alors à la règle 2 vers

  $$a : n, b : r, d : m - r - 1, h : 1$$

  on enchaine alors les règles 3 et 6 jusqu'à obtenir la configuration

  $$b : r - 1, c : n, d : m - r - 1, h : 1$$

On ne peut alors qu'appliquer la règle 7 qui permet de passer à

$$b : r - 1, c : n, d : m - r, e : 1$$

puis l'alternance 4 et 5 transfère b dans d vers

$$c : n, d : m - 1, e : 1$$

Dans ce cas, si on a $m - 1 > 0$ et on va alors repasser dans la règle 10 pour tester la divisibilité de n par $m - 1$. Mais ce qui est subtil ici, c'est que le cas $m - 1 = 0$ a déjà été pris en compte car il correspond à $m = 1$, c'est-à-dire à une divisibilité qui aboutit forcément et qui est donc traité par le cas suivant.

- Soit m divise n, donc $n = km$ et on va aboutir forcément par soustraction après une règle 0 à une configuration de la forme

$$a : n, d : m - 1, g : 1$$

Donc, contrairement au cas précédent, la règle 2 ne s'applique pas car $b = 0$. On passe donc à la règle 8 qui produit

$$a : n, d : m - 1$$

C'est ici qu'on peut produire un nombre premier, car si $m - 1 = 0$, cela veut dire que la seule divisiblité s'est produite pour 1, donc que n est premier. Or, on a alors uniquement n dans le compteur a, et ainsi ça correspond au 2^n.
On poursuit alors par une étape de nettoyage pour passer à l'entier $n + 1$ et continuer. Pour cela, on applique la règle 11 comme précédemment. Le seul changement c'est qu'avant de passer à la règle 13, on va appliquer la règle 12 pour vider le compteur d.

Ainsi, ce programme pourrait être traduit par le code Python suivant :

```python
a = 1
while True:
  a = a+1
  for d in reversed(range(1,a)):
    b = a
    reste = False
    while b != 0:
      for i in range(d):
        if b == 0:
          reste = True
          break
        b = b-1
    if not reste:
      break
  if d == 1:
    print(a,' est premier')
```

Python

II.6 Modèle Turing complets

Tous les exemples présentés ci-dessus sont équivalents : ils permettent de calculer les mêmes fonctions mathématiques pourvu qu'on précise la manière dont on donne l'entrée et comment on lit la sortie.

Par exemple, s'il n'est pas possible de réaliser l'incrémentation en FRACTRAN, il est possible d'écrire un programme permettant de passer de 2^a à 3^{a+1}.

On dit que ces modèles sont **Turing-complet** en référence au modèle de calcul de référence : les machines de Turing.

III Langages, Compilateur, Interprète

III.1 Langages

Un langage de programmation est une manière de représenter textuellement un programme. Cela peut être un programme en lien avec une notion de machine mais aussi une notion plus abstraite comme on va le voir.

Comme on l'a vu, un ordinateur est une machine. Un programme pour cette machine est ce qu'on appelle un binaire et l'ensemble des instructions s'appelle le langage machine. C'est le langage naturel d'un ordinateur et il dépend de son processeur. Ainsi, un binaire pour un téléphone n'est pas directement utilisable sur son ordinateur, car les premiers ont des processeurs utilisant un jeu d'instructions arm et les seconds un jeu d'instructions x86_64.

III.2 Compilateur

Un **compilateur** est un traducteur d'un langage à un autre. On considère le plus souvent des compilateurs vers un langage machine, qu'il soit lié à des processeurs réels ou une machine spécifique appelée une machine virtuelle.

III.3 Interprète

Un interprète est un programme qui va lire un programme écrit dans un langage donné et l'évaluer en interprétant les instructions. Cela diffère d'une machine virtuelle dans le fait que le programme utilise toute la richesse du langage de programmation dans lequel il a été programmé pour son interprète. De plus, le fait de ne pas passer par une machine virtuelle permet de plus facilement relier l'environnement de l'interprète et celui du langage interprété. Un autre avantage discutable et la possibilité dans le langage cible de faire référence à l'interpréte. Cela permet notamment de permettre l'évaluation dynamique de code (`eval` en Python).

Pour autant, les interprètes ont de nombreux défauts qui font qu'on a de plus en plus recours à des compilateurs vers des machines virtuelles. Parmi ceux-ci, citons le fait de devoir relire directement le programme à chaque fois qu'on va l'exécuter, ce qui est particulièrement critique quand on veut faire appels à des fonctions définies ailleurs, ou le fait que l'interprète lui-même est souvent très verbeux.

III.4 Unité de compilation, modules

Un programme est souvent structuré sous la forme de nomnbreux sous-programmes reliés entre eux. C'est une bonne pratique pour regrouper des fonctions selon un même thème et permettre la réutilisation du code sans duplication. On utilise pour cela la notion de modules ou de bibliothèques. Cela correspond au niveau du programme à rajouter du code existant au moment de la compilation ou de l'interprétation. Il est alors possible de réutiliser une partie de la compuilation sur ce code, et ainsi, on utilise en général une version intermédiaire qui est du code precompilé utilisable par un autre programme, soit en copiant le résultat dans le programme ou en faisant réference à des versions définies dans le système (bibliothèque dynamique). Un programme est chargé de faire le lien entre les différentes bibliothèques et le programme, c'est l'éditeur de liens.

IV Paradigmes

Un paradigme de programmation est une manière de concevoir des programmes. On va citer ici trois paradigmes.

IV.1 Impératif structuré

Un programme impératif est un programme construit plus ou moins directement en lien avec une notion de machines comme une suite d'instructions. Afin de permettre de programmer convenablement, un tel langage fourni une notion de structure qui permet de structurer le code. D'une certaine manière, la notion de classe et la programmation objet enrichi ce paradigme en permettant de structurer le code autour d'une notion d'objets.

IV.2 Fonctionnel déclaratif

On a vu qu'en programmation fonctionnelle, tout n'est qu'une expression, un terme, qu'on évalue. Un programme fonctionnel déclaratif est ainsi consistué d'une succession de définitions de valeurs ou de fonctions. C'est ainsi qu'en `OCaml` n'est qu'une suite de `let` ou `let rec`.

Comment calcule un tel programme ? En évaluant chacune des déclarations, certaines vont effectivement produire des effets de bords et permettre la lecture ou l'écriture. L'usage en `OCaml` est ainsi de finir un programme par une déclaration nommée `main` ou `_` qui va se charger d'appeler les déclarations précédentes.

IV.3 Programmation logique

Le paradigme de la programmation logique est sûrement le plus déroutant des trois. Un programme logique est en fait une formule logique dont une solution correspond à son résultat. Sans rentrer dans les détails, on remarquera que cela correspond à ce qu'on fait en SQL pour les bases de données où une requête de recherche est définie sous une forme de formule logique qui décrit les propriétés qui doivent être vérifiés par les éléments qu'on cherche.