λ
Programmation Fonctionnelle: L'ordre supérieur.

Yann Régis-Gianas
(IRIF, Univ. Paris Diderot, Inria) – yrg@irif.fr

2019-09-27

Rappel du cours précédent

- On retrouve en OCaml les types du premier ordre habituels : entiers, booléens, flottants, caractères, chaînes de caractères, n-uplets, enregistrements, énumérations et tableaux.
- Deux nouveautés découvertes la dernière fois :
  1. les types algébriques ;
  2. l'analyse par motifs.

Plan

Le noyau fonctionnel d'OCaml

Comment calculer des fonctions?

À quoi servent les fonctions de première classe?

La programmation d'ordre supérieur

Ordre supérieur

Un programme est d'**ordre supérieur** s'il calcule avec des valeurs qui contiennent du code exécutable.

La POO est d'ordre supérieur

Un objet est formé d'un état interne et d'un ensemble de méthodes qui font évoluer cet état.

La programmation fonctionnelle est d'ordre supérieur

Une fonction pure, c'est un code exécutable sans état.

Pourquoi utiliser la programmation d'ordre supérieur?

Programmation d'ordre supérieur = modularité + expressivité

Qu'est-ce que la modularité?

Modularité

Un composant logiciel est **modulaire** s'il est:

1. cohérent : on peut résumer ce qu'il fait en une phrase simple.
2. faiblement couplé : il a un nombre limité de dépendances.

Une architecture logiciel est modulaire si on peut substituer n'importe lequel de ses composants par un composant équivalent sans que cela n'impacte le bon fonctionnement du logiciel.

Qu'est-ce que la modularité?

Modularité

Un composant logiciel est **modulaire** s'il est:

1. cohérent : on peut résumer ce qu'il fait en une phrase simple.
2. faiblement couplé : il a un nombre limité de dépendances.

Une architecture logiciel est modulaire si on peut substituer n'importe lequel de ses composants par un composant équivalent sans que cela n'impacte le bon fonctionnement du logiciel.

Propriété de la programmation d'ordre supérieur

Le fait de pouvoir paramétrer un objet par d'autres objets, de pouvoir paramétrer une fonction par d'autres fonctions, permet de réduire le couplage.

Qu'est-ce que la modularité?

Modularité

Un composant logiciel est **modulaire** s'il est:

1. cohérent : on peut résumer ce qu'il fait en une phrase simple.
2. faiblement couplé : il a un nombre limité de dépendances.

Une architecture logiciel est modulaire si on peut substituer n'importe lequel de ses composants par un composant équivalent sans que cela n'impacte le bon fonctionnement du logiciel.

Propriété de la programmation d'ordre supérieur

Le fait de pouvoir paramétrer un objet par d'autres objets, de pouvoir paramétrer une fonction par d'autres fonctions, permet de réduire le couplage.

À propos de la cohésion

La cohésion d'un composant est haute si ce composant est petit et général.

Le style fonctionnel de programmation

Le style fonctionnel de la programmation encourage

- l'écriture de fonctions courtes, générales et cohérentes ;
- construites par applications de fonctions générales et d'ordre supérieur.

Le style fonctionnel de la programmation s'appuie sur le **principe de compositionalité**. Cela signifie que l'on doit pouvoir donner un sens à un fragment de programme uniquement à partir du sens donné à ses composants. En d'autres termes, la programmation fonctionnelle favorise la **localité du raisonnement**, nécessaire à toute forme de modularité.

Qu'est-ce que l'expressivité d'un langage?

> Programming languages appear to be in trouble. Each successive language incorporates, with a little cleaning up, all the features of its predecessors plus a few more. [...] Each new language claims new and fashionable features... but the plain fact is that few languages make programming sufficiently cheaper or more reliable to justify the cost of producing and learning to use them.
> – Backus (Turing Award, 1977)

Dans l'article "The next 700 programming languages", Peter Landin montre que **la plupart des mécanismes calculatoires utilisées par les différents styles de programmation** peuvent se simuler efficacement à l'aide d'un langage fonctionnel minimal.

Expressivité pratique

On dit qu'un langage $\mathcal{L}$ peut exprimer une construction C d'un autre langage $\mathcal{L}'$ s'il existe une **transformation locale** permettant de traduire C en termes de constructions de $\mathcal{L}$ en préservant le sens de C.

Cette notion est distincte de l'expressivité calculatoire au sens de Turing.

Quel est ce langage fonctionnel minimal dont parle Landin?

Il est constitué des expressions construites par seulement 3 constructions :

- La variable `x`, `y`, ...
- La fonction "`fun x -> a`" où `a` est une expression.
- L'application "`a b`" où `a` et `b` sont des expressions.

Nous allons étudier ces constructions très simples aujourd'hui et constater empiriquement leur expressivité. Rendons-nous maintenant sur le sketch:

`https://sketch.sh/s/N4zt2tZ1AX4X8kT2aHWwro/`

Plan

Le noyau fonctionnel d'OCaml

Comment calculer des fonctions?

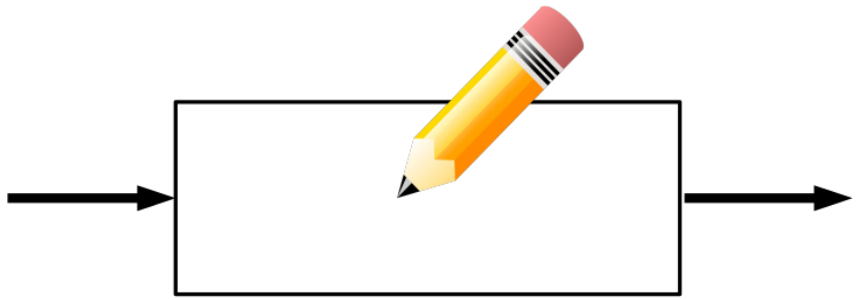
À quoi servent les fonctions de première classe?

Comment construit-on des fonctions en programmation fonctionnelle?

- par une **définition**: c'est-à-dire en donnant leur code (comme vous le faites déjà en programmation procédurale).
- par **composition**: en chaînant les calculs réalisés par d'autres fonctions.
- par **spécialisation**: en fixant l'un des paramètres d'une fonction qui en attend plusieurs.
- par **induction**: en définissant une fonction par cas sur une structure inductive et en prenant le point fixe de cette définition.

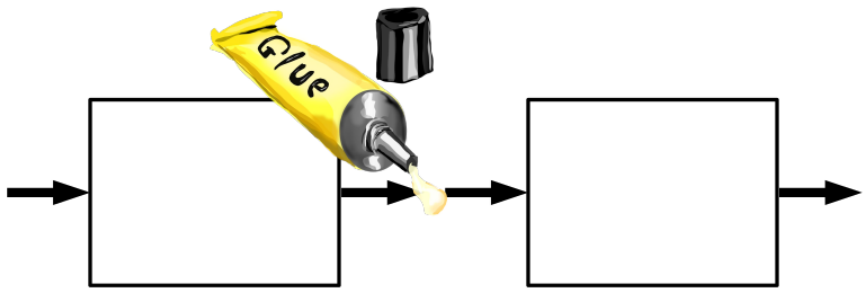
Par définition

Que dois calculer ma fonction?



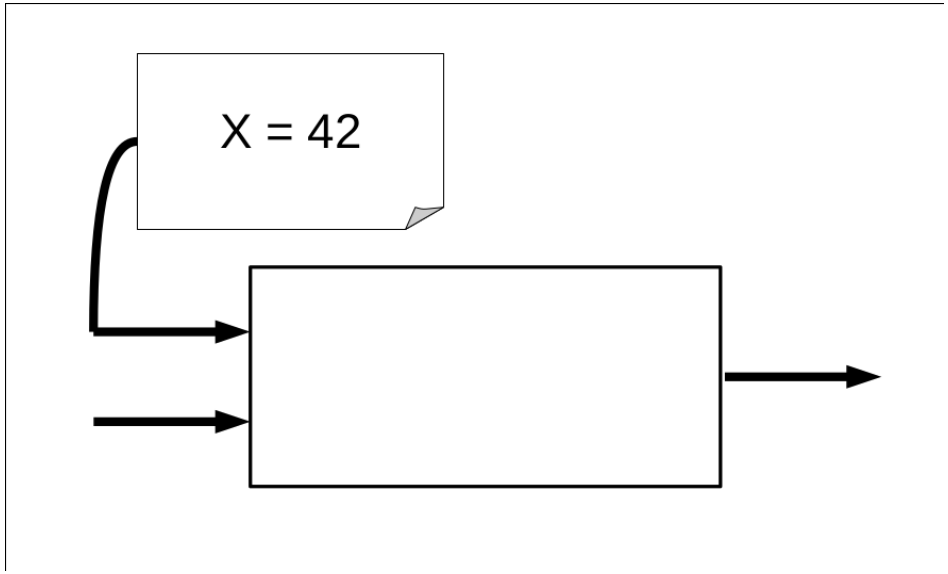
Par composition

Ma fonction peut-elle se décomposer en deux calculs plus simples?



Par spécialisation

Ma fonction est-elle un cas particulier d'une fonction plus générale?



Par induction

La logique de ma fonction est-elle dirigée par une structure inductive?