PFON - Janvier 2019 - PROMO 2021

**I Introduction et Définitions**

Programmation Fonctionnelle = paradigme de programmation de type déclaratif

**Paradigme de programmation** (Définition)

* Approche de la prog info et des solutions aux pb et leur formulation ds un langage de programmation.
* Affecte l’expressivité, la manière de penser du langage

Paradigme de programmation de type **déclaratif** (Définition)

* Création des applications sur la base de composants logiciels indépendants du contexte et ne comportant aucun état interne.
* L’appel d’un des composants avec les mêmes arguments produit exactement le même résultat, quelque soit le moment et le contexte de l’appel. (// fonction mathématique)
* Description du **quoi** : le contenue du composant par opposition à la prog impérative où on décrit le comment, **la structure de contrôle** correspondant à la solution.
  + Programmation sans effet de bord
  + L’ensemble des applications est perçu comme un ensemble de fonctions mathématiques : application de fonction =/= évaluation de fonction

2. Origine

* lambda-calcule
* Lisp 1958

**II Langages Fonctionnels : pas d’effet de bord**

Fonction à effet de bord = fonction qui modifie un état en dehors de son environnement local, qui se répercute dans un autre scope de def, autre que par un retour de valeur (ex: fonction qui modifie une variable locale statique, une variable globale).

1. Pas d’effet de bord, comment ?

* Interdiction de toutes opérations d’affectation : pas de variable changeante au cours du temps mais des **constantes**.
* Fonction agissent comme des boîtes noires
* **Transparence Référentielle**: principe selon lequel le résultat du programme ne change pas si on remplace une expression par une expression de valeur égale.

2. Langages Fonctionnels Impurs

* possibilité d’introduire des variables assignables / mutables

**III Modèle mémoire et compilation**

1. Utilisation de la stack (pile)

* Pas de nécessité de stocker des données temporaires dans des tableaux
* Utilisation fréquente de la récursivité => optimisation par le compilateur **Récursion terminale**
  + Accumulation des résultats intermédiaires dans une case mémoire de la stack
  + Celle ci est passé en paramètre de l’appel récursif
  + Evite de stocker les multiples appels de fonctions dûs à la récursivité dans la stack
  + Stock une succession de saut en mem
  + Compilateur optimise ça comme une boucle impérative

2. S’appuie sur le modèle de **Machine à états** (cf machine de Turing et THL)

* nécessite connaissance modèle + état mémoire que le programme modifie
* plusieurs manière de faire
  + **Programmation structurée**: limitation scope
  + **Encapsulation des données**: POO (public, private, protected ...)
* Variables désallouées par compilateur (ou interpreteur) immédiatement en sortie de procédure = **GARBAGE COLLECTOR**

**IV Fonctions**

1. Fonction

* pas nécessairement nommées : **fonction anonymes**  (**lambda functions**)
* utilisation directe / littérale (FIXME)
* calcule d’une valeur de sortie (retour) en fonction de valeurs d’entrée (arguments) (**Fonctionnel pur**)
* **2 archétypes du passage d’argument fonctionnel**
  + **Mapping**: traitement individuel des éléments d’une liste par une fonction
  + **Folding**: combiner les éléments d’une liste par une fonction

2. **Fonction d’ordre supérieur**

* Une fonction est dite d'ordre supérieur lorsqu'elle peut prendre des fonctions comme arguments (<=> callback) ou renvoyer une fonction comme résultat.
* Les fonctions sont des **objets de première classe**: elles sont manipulables aussi simplement que les types de base.

**CCL fonction:** Facilite la description des opérations (le code quoi) ex: la concaténation de listes, l'application d'une fonction à une liste (le parcours de la liste se faisant de façon récursive)

=> Gain d’**abstraction**

**V Expressions**

* **Littérales** : s’évaluent à elles-mêmes
* **Combinées**: application de procédures (opérateurs, fonctions) à des arguments (expressions)
* **Abstraites**: nommage / assignation d’expressions simples, combinées ou fonctionnelles.
  + Nommage (fonctionnel pur)
    - Donner un nom à une valeur
    - Une valeur peut avoir différents noms
    - Peut être **syntaxique** ou **fonctionnel**
  + Assignation (fonctionnel impur)
    - Donner une valeur à un nom
    - Un nom peut donc avoir différentes valeurs

1. Typage

* Type **Static**
* Variables typées
* Vérification type à la compilation
* Contraintes compensées par le **polymorphisme** : définition unique pour tout type
* Type **Dynamique**
* **Valeurs** typées: information contenue dans chaque objet
* Vérification à l’exécution
* Information de typage accessible (typeof)

2. **Evaluation des expressions** : 2 types

* **Stricte**: Les arguments / expressions sont évalués d’abord (Lisp)
* **Lazy**: Les arguments / expressions ne sont évalués que quand ils sont utilisés (Haskell)
  + possibilité de manipuler des listes infinies
  + nécessairement fonctionnel pur: on ne peut pas s’appuyer sur l’ordre d’évaluation