La Programmation Fonctionnelle

Jean-Luc Falcone

23 Février 2023

WTFIFP ?

Comment améliorer ce code ?

```
def apply( x: X, y: Y ): Z = {
  val a = f(x,y)
  val b = g(x,a)
  val c = f(x,y)
  h(a,c)
}
```

Mauvaise Définition

La programmation fonctionnelle consiste à programmer avec des fonctions !

Concept 1: Transparence référentielle

Définition

Une expression est référentiellement transparente si on peut substituer chacune de ses occurence avec le résultat de son évaluation sans changer le fonctionement d'un programme.

Exemples (C/java/...)

```
//Referentiellement transparente
PI / 2;
sqrt( sin(x)*sin(x) + cos(x)*cos(x) );
j + 1;

//Referentiellement opaque
i = 3;
j + (++i);
new int[12];
```

Expression vs. instruction (C/Java/...)

Les expressions sont évaluées, alors que les instructions sont exécutées:

Expressions

Peut-être référentiellement transparentes

```
i + j;
sqrt(x);
i > 0 ? i : -i;
```

Instructions

Jamais référentiellement transparentes

```
i += j;
a[i] = sqrt(x);
sqrt(x)
```

Expression vs. instuction: `if ... else ...`

En python

```
if a > 0:
    b = a
else:
    b = -1
```

En Scala

```
val b = if(  a > 0 )
           a
        else
           -a
```

Immutabilité

La transparence référentielle exclut les mutations:

- Pas de variables
- Structures immutables uniquement
- On ne modifie pas une structure, on retourne une copie modifiée

Effet de bord

La transparence référentielle exclut des effets de bords:

- Pas de mutation
- Pas d'I/O
- Pas d'update de DB
- ...

Concept 2: Fonctions pures

Définition

Une fonction pure est une fonction référentiellement transparente par rapport à ses arguments.
Elle retourne toujours le même résultat pour de mêmes arguments.

Exemples (python)

```python
#Fonction pure
def isEmpty( lst ):
    return len(lst) == 0

#Fonction impure
emptyNum=0
def countIfEmpty( lst ):
    if isEmpty(lst):
        emptyNum += 1
    return emptyNum
```

Fonctions Pures ??? (exemple en java)

```
double randomNoise( double x ) {
  return x + Math.random()/100;
}

LocalDate current() {
  return LocalDate.now();
}
```

Fonctions Pures ??? (encore en java)

```
double randomNoise( double x, long seed ) {
  Random rand = new Random(seed);
  return x + rand.nextDouble()/100;
}

LocalDate current(int year, int month, int day) {
  return LocalDate.of(year,month,day);
}
```

Fonctions Pures ??? (toujours en java)

```
int sum( int[] is ) {
  int sum = 0
  for( int i: is ) {
    sum += i;
  }
  return sum;
}
```

Programmation fonctionnelle

Définition

Style de programmation basé sur l'utilisation d'expression réf. transparentes et de fonctions pures.

Avantages

- Facilite l'analyse du code
- On peut raisonner avec le code
- Composabilité
- Toujours *thread-safe*
- Certaines optimisation deviennent évidentes

Désavantages

- Pas d'IO (effets de bord)
- Peut être plus lent (p.e. copie conservative)
- Nécéssite des structures de données appropriées
- Les algorithmes sont souvent présentés de manière procédurale.
- Le hardware a un fonctionnement impératif.

- Implique un changement d'habitude (apprentissage)

Analyse (1)

```
//f, g, h sont pures
def apply( x: X, y: Y ): Z = {
  val a = f(x,y)
  val b = g(x,a)
  val c = f(x,y)
  h(a,c)
}
```

Analyse (1)

```
//f, g, h sont pures
def apply( x: X, y: Y ): Z = {
  val a = f(x,y)
  val b = g(x,a)
  val c = f(x,y)
  h(a,c)
}
```

Conclusion: `a==c`, `b` est inutile

Optimisation

```
//Avant
def apply( x: X, y: Y ): Z = {
  val a = f(x,y)
  val b = g(x,a)
  val c = f(x,y)
  h(a,c)
}

//Après
def apply( x: X, y: Y ): Z = {
  val a = f(x,y)
  h(a,a)
}
```

Analyse (2) (Exemple en C)

```c
int f( int a, int b ) {
  return a + b;
}

int i = 0
f( ++i, --i ); //Valeur de retour ?
f( i++, i-- ); //Et là ?
```

Modèle d'évaluation par substitution (1)

```
def f( a: Int, b: Int ) = a + b

val i = 0
f( i+1, i-1 ) //Valeur de retour ?
```

Modèle d'évaluation par substitution (2)

```
def f( a: Int, b: Int ) = a + b
val i = 0

f( i+1, i-1 )
f( 0+1, 0-1 ) // Subst i
f( 1, 0-1)    // Eval
f( 1, -1)     // Eval
1 + (-1)      // Subst f
0             // Eval
```

Modèle d'évaluation par substitution (3)

```
def f( a: Int, b: Int ) = a + b
val i = 0

f( i+1, i-1 )
(i+1) + (i-1) // Subs f
(0+1) + (0-1) // Subst i
(0+1) + (-1)  // Eval
1 + (-1)      // Eval
0             // Eval
```

Modèle d'évaluation par substitution (3)

```
def f( a: Int, b: Int ) = a + b
val i = 0

f( i+1, i-1 )
(i+1) + (i-1) // Subs f
(0+1) + (0-1) // Subst i
(0+1) + (-1)  // Eval
1 + (-1)      // Eval
0             // Eval
```

Conclusion: L'odrre n'a pas d'ipormtnace !

Modèle d'évaluation par substitution (4)

```
def fac( n: Long ) = if(n== 0 ) 1
else n * fac(n-1)

fac(3)
if(3==0) 1 else 3*fac(3-1)                    //Subst fac
if(3==0) 1 else 3*{
  if(3-1 == 0) 1 else (3-1)*fac((3-1)-1)
}                                             //Subst fac
if(3==0) 1 else 3*{
  if(2 == 0) 1 else (3-1)*fac((3-1)-1) //Eval
}
if(3==0) 1 else 3*(3-1)*fac((3-1)-1)    //Eval
if(3==0) 1 else 3*2*fac((3-1)-1)        //Eval
if(3==0) 1 else 3*(2)*fac(1)            //Eval
...
```

Preuve de code

Prouvons que `sign(x)*abs(x) == x` en substituant:

```
def abs( x: Double ): Double =
  if( x < 0 ) -x else x

def sign( x: Double ): Double =
  if( x < 0 ) -1
  else if( x > 0 ) 1
  else 0
```

Autres concepts clés

- Typage fort et statique (pas tous les langages FP)
- Fonctions totales
- Structures persistentes
- Code == données