Informatique ? OCaml ?

Alain Camanes

alain.camanes@free.fr

Stanislas

Option Informatique
2021-2022

« L'informatique n'est pas plus la science des ordinateurs que l'astronomie n'est celle des télescopes. »

Edsger W. Dijkstra (Rotterdam 1930 - Nuemen 2002)
Prix Turing 1972.

Plan

Bref historique (I)

↪ Vers -300. Euclide : calcul du plus grand commun diviseur de deux entiers.

↪ Vers 800. Al Khwarizmi écrit un traité d'algèbre.

↪ 1645. Blaise Pascal présente la première machine à additionner, la pascaline.

↪ 1834. Charles Babbage propose les plans d'une machine à calculer programmable.

↪ 1843. Ada Lovelace propose un programme de calcul des nombres de Bernoulli avec la machine analytique.

Bref historique (II)

↪ 1900. David Hilbert et ses 23 problèmes ouverts pour le XX-ème siècle.
Existe-t-il une méthode permettant de déterminer si une équation diophantienne admet des solutions.

↪ 1922. David Hilbert.
Existe-t-il une procédure générale permettant, en un nombre fini d'étapes, de dire si un énoncé mathématique est vrai ?

↪ 1936. Alan Turing introduit la notion de machine de Turing, formalisant la notion d'algorithme.

↪ 1945. John Von Neumann décrit le fonctionnement d'un ordinateur. L'ENIAC a été mis en service en 1946, l'EDVAC en 1949, l'ACE en 1950,...

Récursif & Impératif

↪ David Hilbert (1928). Problème de la décision.
Peut-on définir un processus qui, en un nombre fini d'étapes, permet de déterminer si une assertion de la logique du premier ordre est vraie ?

↪ Alan Turing (1936).
Non. Machine de Turing : automate lisant et écrivant sur un ruban.

↪ Alonzo Church (1936).
Non. λ-calcul & Fonctions anonymes.

Récursif & Impératif

↪ David Hilbert (1928). Problème de la décision.
Peut-on définir un processus qui, en un nombre fini d'étapes, permet de déterminer si une assertion de la logique du premier ordre est vraie ?

↪ Alan Turing (1936).
Non. Machine de Turing : automate lisant et écrivant sur un ruban.
→ Programmation impérative.

↪ Alonzo Church (1936).
Non. λ-calcul & Fonctions anonymes.
→ Programmation fonctionnelle.

Plan

OCaml

↪ Categorical Abstract Machine Language.
Programmation *fonctionnelle*.
Machine abstraite utilisant la théorie des catégories.
Famille de langage proposant un certain *système de types*.

↪ Développé par Inria depuis 1985.

↪ Implémentation actuelle : OCaml (1996).
Créé notamment par X. Leroy.
Ajout d'une couche de programmation *orientée objet*.

↪ Utilisations.
Facebook, Bloomberg, Dassault Systemes, Jane Street (trading),...

↪ A inspiré.
`F#` (Microsoft Research), `Rust` (Mozilla Research),...

Installation

↪ Site d'OCaml.
http://ocaml.org/docs/install.html

↪ Essai en ligne.
https://try.ocamlpro.com/

↪ Ubuntu : Emacs + Tuareg.
https://doc.ubuntu-fr.org/ocaml

Première description

↪ Multiparadigme.

Programmations : *fonctionnelle*, impérative, orientée objet.

↪ Boucle d'interprétation.

- Analyse de la syntaxe.
- Inférence des types.
- Évaluation de l'expression.

↪ Compilation pour obtenir un fichier exécutable.

```
ocamlopt -o <name>.out <name>.ml
```

Syntaxe et Sémantique

↪ Syntaxe. Règles grammaticales du langage.

- `2 + 3`
- `true && (false || true)`
- `f 3 + 4`

↪ Sémantiques. Dénotationnelle et Opérationnelle.

- `2 + 3`
  Addition entre entiers ; S'évalue `5`.
- `true && (false || true)`
  Opérations sur les booléens ; S'évalue `true`.
- `float_of_int 3 +. 4.5`
  Conversion et opération sur les flottants ; S'évalue en `7.5`.

↪ Interpréter les programmes en termes de...

- fonction mathématique : sémantique dénotationnelle.
- suite d'états de la machine : sémantique opérationnelle.

Typage statique et fort

↪ Typage d'une variable.
Associer à une *variable symbolique* un *type* de donnée.

↪ Typage statique. ✓
Détecter les erreurs *avant* l'évaluation.
Optimiser l'utilisation de la *mémoire*.

↪ Typage dynamique. ×
Détecte le type *à la volée*.
Modification plus facile du type.
Erreurs détectées lors des *tests*.

↪ Typage fort.
Conversions *explicites*.

```
# float_of_int 3 +. 2.5;;
- : float = 5.5
```

Plus de types ! (I)

↪ Inférence de types.

Deviner le type le plus général possible d'après les opérateurs.

Polymorphisme. Sera instancié au moment de l'évaluation.

```
# let f x y = 3.14 *. x +. 1.5;;
val f : float -> 'a -> float = <fun>
# let min x y = if x < y then x else y;;
val min : 'a -> 'a -> 'a = <fun>
# let h = min 2;;
val h : int -> int = <fun>
# h 3;;
- : int = 2
```

↪ Type abstrait. Ensemble de données et des opérations possibles.

Plus de types ! (II)

↪ Filtrage par motifs. *Pattern-matching.*

Clarté et *légèreté* du code

Vérification de l'*exhaustivité* et proposition de motifs manquants

```
# let n = 1;;
val n : int = 3
# match n with
    | 0 -> 1
    | 1 -> 12
    | _ -> 3;;
- : int = 12
```

↪ Définition de nouveaux types.

Plan

↪ Commentaires. On place les commentaires entre (* et *).

```
(* Ceci est un commentaire *)
```

↪ Résultat. Pour terminer une requête Caml, on utilise ;;.

```
# 3 + 5;;
- : int = 8

# print_string "hello";;
hello- : unit = ()

# "hello" = "toto";;
- : bool = false
```

Types élémentaires

↪ `unit` - Un seul élément : `()`.

↪ `bool` - Deux éléments : `true`, `false`.

↪ `int` - Signés sur 31/63 bits selon architecture.
1 bit pour la gestion de la mémoire.

↪ `float` - Double précision.

↪ `char` - Caractère sur 8 bits (simple quotes).

↪ `string` - Chaînes de caractères (double quotes).

↪ ATTENTION. Ces types sont disjoints.
Pas de conversion automatique, Pas de surcharge d'opérateurs.

Type `bool`

↪ Valeurs. `true` et `false`.

↪ Opérateurs (par ordre de priorité). `not`, `&&` (et), `||` (ou).
Évaluation paresseuse.

↪ Comparateurs polymorphes. `=`, `<>`, `<`, `>`, `<=` et `>=`.
`?'a -> 'a -> bool` ?

↪ ATTENTION. `and` et `==` ont une autre signification ! !

Type `int`

↪ Valeurs. Entiers de $[\![-2^{62}, 2^{62}-1]\!]$ en architecture 64 bits.
↪ Calculs modulo 2^{63}.

```
# max_int + 1;;
- : int = -4611686018427387904
```

↪ Opérations.

- l'addition `+`, la soustraction `-`,
- la multiplication `*`,
- le quotient de la division euclidienne `/`, le reste de la division euclidienne `mod`,
- la valeur absolue `abs`.

Pas de fonction puissance.

Type `float`

↪ Valeurs. Norme IEEE754.

↪ Opérations.

- l'addition `+.`, la soustraction `-.`,
- la multiplication `*.`, la division `/.`,
- la puissance `** : float -> float -> float`,
- la partie entière inférieure `floor : float -> float` et la partie entière supérieure `ceil`,
- `sqrt`, `exp`, `log` (ln), `log10`, `sin`, `cos`, `tan`, `acos`, `asin`, `atan`, `abs_float`.

```
# 2 + 2.1;;
Characters 4-7:
  2 + 2.1;;
      ^^^
Error: This expression has type float
         but an expression was expected
         of type int
```

Conversion

↪ Les types `int` et `float` sont *disjoints* et *incompatibles*.

`float_of_int` ou `float` : `int -> float`

`int_of_float` : `float -> int`

```
# int_of_float (1. /. 3.);;
- : int = 0
```