Institut National
Universitaire
Champollion

Types de données, preuves

Introduction

L3 Info - Algorithmes, Types de données, preuves

Présentation de l'UE
Introduction du cours

Laura Brillon

Algorithmes, types de données, preuves

Présentation du cours

Enseignants

- Laura Brillon (PrAg, INU)

Habituellement accompagnée par Martin Strecker
(Maître de Conférences, UPS)

↪ **Changements importants dans l'organisation de l'UE**

UE coupée en deux

- 6 semaines : TDP
- 6 semaines : Initiation à l'IA avec M. Montaut et Panzoli

MCC pour la partie TDP : Un contrôle écrit

Programme du jour

Menu du jour

- Introduction, le processus de compilation
  Comment ce cours s'inscrit-il dans votre semestre ?
  Comment vous prépare-t-il au Master ?

- Introduction à la théorie des langages de programmation
  Cette UE au sein de l'Informatique. Ce qui va nous occuper.

- Plan du cours et début du Chapitre 1

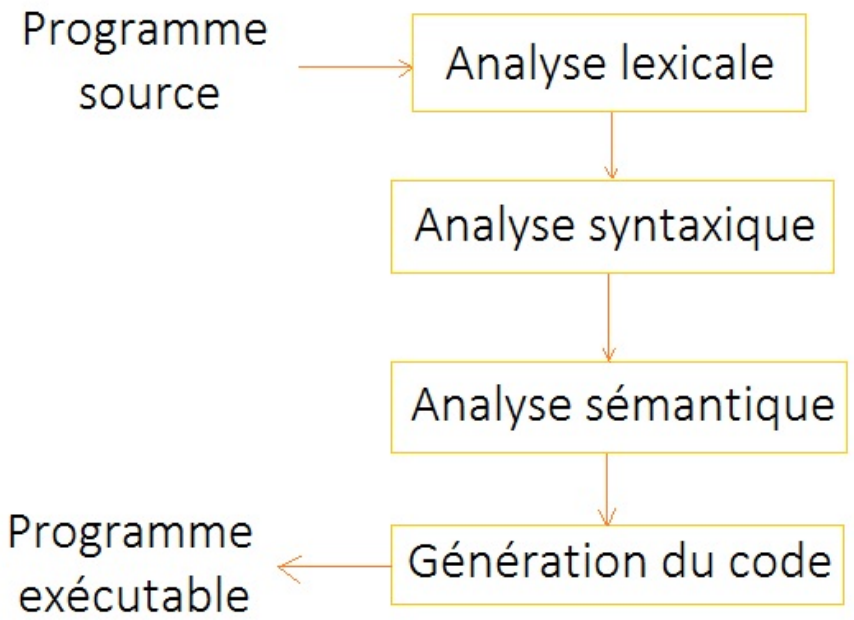
Sommaire

Où en est-on ?

Comment ce cours s'inscrit-il dans votre semestre ?
Comment vous prépare-t-il au Master ?

Le processus de compilation

Les grandes étapes du processus de compilation sont les suivantes :



Pour vous, ce schéma...

1. Je connais déjà
2. On peut faire un rappel ?
3. Jamais vu, kécecé ?

Le processus de compilation

Analyse lexicale

La définition d'un langage de programmation s'appuie sur :

- Un *alphabet* - ensemble des symbôles élémentaires disponibles
- Les *lexèmes* - ensemble de mots
- Un *mot* - groupe de symbôles dont la structure est donnée par une *grammaire*

(cf. Cours de Théorie des Langages)

L'analyseur lexical fait une lecture du programme source et une reconnaissance des lexèmes.

Exemple :

(* fonction $f(x) = x + 1$ *)

```
let f =    function x -> x+1 ;;
```

Analyse syntaxique

L'analyseur syntaxique vérifie que la suite de symbôles issue de l'analyseur lexicale est conforme à la syntaxe.

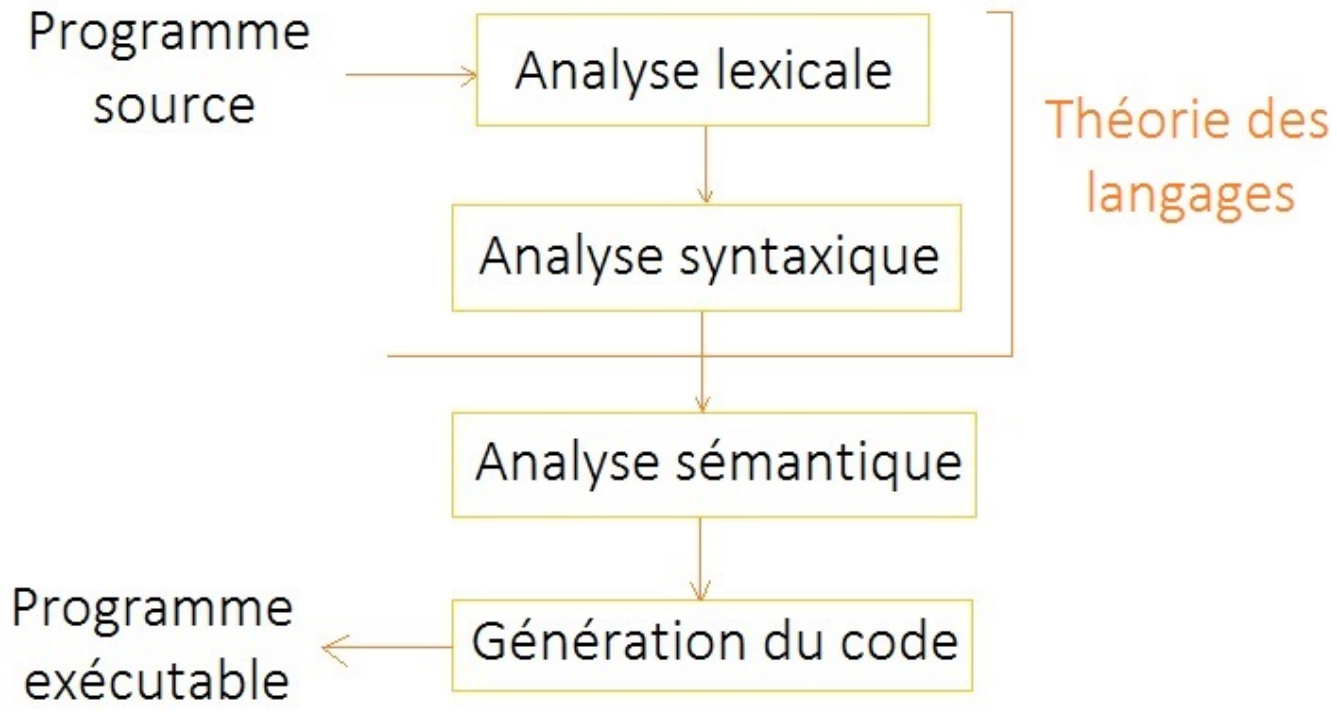
Exemple :

```
let let f = function x -> x + 1 ;;
Error: Syntax error
```

L'analyseur associe au programme un *arbre syntaxique* : les feuilles sont les symbôles, les noeuds intermédiaires sont les objets grammaticaux (définition, identificateur, affectation, ...).

Introduction

Théorie des langages vs. Types de données, preuves



Analyse sémantique

L'analyseur sémantique associe un sens aux différentes phrases du programme source.

- Reconnaître les objets manipulés et analyser leurs propriétés : quel est le type de l'objet ? quelle est sa taille ?...
- Contrôler que l'utilisation de ces objets se fait de manière cohérente : erreurs de typage, déclarations multiples, absentes ou inutiles...

Dans ce cours :

Partie "Types de données" de l'analyseur sémantique.
Vérification de types d'un programme fonctionnel,
Inférence de types.

Mais, pourquoi typer ?

Typage - pourquoi ?

Quelques éléments de réponses...

- **Documentation** - pour le(s) programmeur(s), mais aussi pour le compilateur (optimisation).
- **Prévention des fautes** - volontaires ou non
  "Well-typed programs cannot go wrong" R. Milner

Un type sert à

- classifier les données
- circonscrire la valeur des expressions
- spécifier le comportement d'un programme

Introduction

Génération du code

Génération du code = produit dans un fichier objet le code machine équivalent au code du langage de haut niveau

---

Preuves

Preuves de terminaison de fonctions récursives.

Preuves par récurrences sur des types récursifs.

Réécriture ?

Sommaire

Où en est-on ?

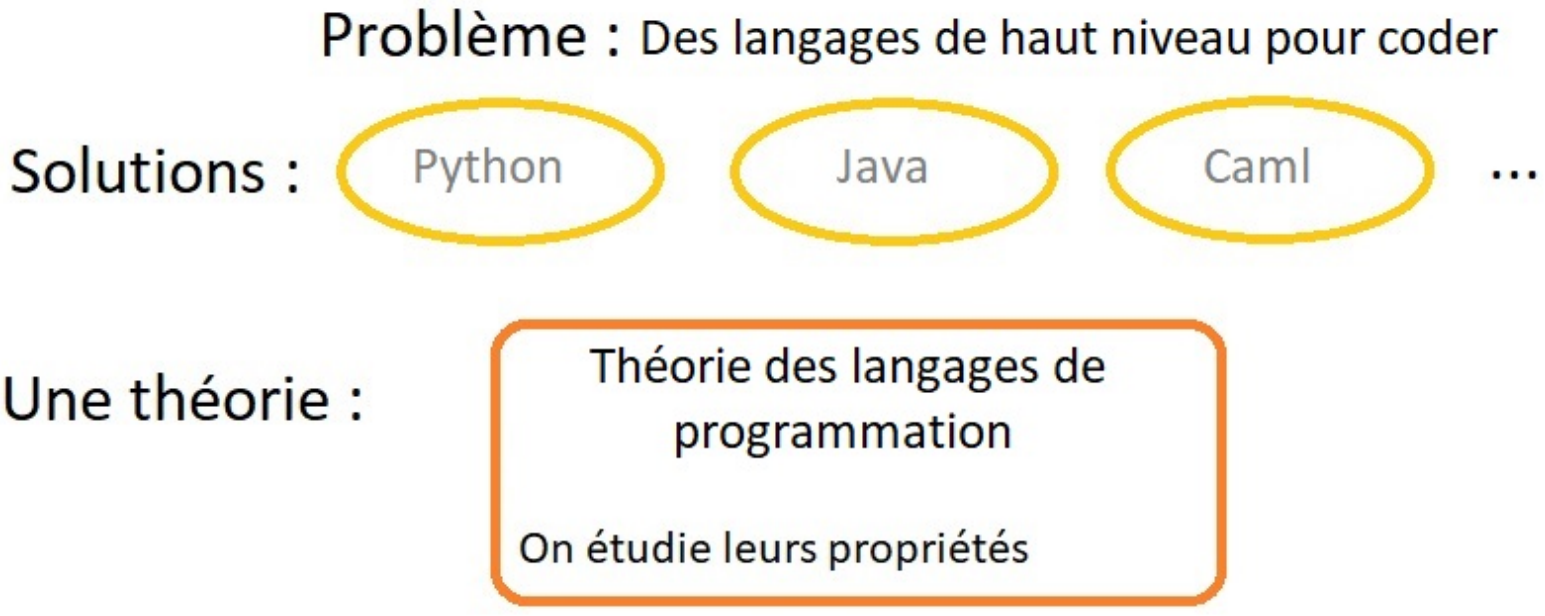
La Théorie des Langages de Programmation, kécecé ?

Introduction à la TLP

Une analogie un peu grossière...

Introduction à la TLP

Une analogie un peu grossière...



Dans ce cours, on va s'intéresser en particulier aux Types de Données.

Introduction à la TLP

Typage unique/multiple

Typage unique : Une expression a un seul type.
Typage multiple : Une expression peut avoir plusieurs types (possiblement hiérarchisés).

Question : En Caml, le typage est

1. unique, et j'en suis sûr(e)
2. unique, mais j'en doute
3. multiple, et j'en suis sûr(e)
4. multiple, mais j'en doute
5. je ne sais pas

Introduction à la TLP

Typage unique/multiple

Typage unique : Une expression a un seul type.
Typage multiple : Une expression peut avoir plusieurs types (possiblement hiérarchisés).

Caml : Typage unique

```
#2 + 3 ;;
- : int = 5
#2.0 +. 3.5 ;;
- : float = 5.5
#2 + 3.5 ;;
Entrée interactive:
>2 + 3.5 ;;
>    ^^^
Cette expression est de type float,
mais est utilisée avec le type int.
#2 + int_of_float 3.5 ;;
- : int = 5
```

? : Typage multiple

Typage unique/multiple

Typage unique : Une expression a un seul type.
Typage multiple : Une expression peut avoir plusieurs types (possiblement hiérarchisés).

Caml : Typage unique

```
#2 + 3 ;;
- : int = 5
#2.0 +. 3.5 ;;
- : float = 5.5
#2 + 3.5 ;;
Entrée interactive:
>2 + 3.5 ;;
>    ^^^
Cette expression est de type float,
mais est utilisée avec le type int.
#2 + int_of_float 3.5 ;;
- : int = 5
```

C, Java : Typage multiple

Introduction à la TLP

Typage dynamique/statique

Typage dynamique : Le type d'une expression est connu au moment de l'exécution.
Typage statique : Le type est connu au moment de la compilation.

Question : En Caml, le typage est

1. dynamique, et j'en suis sûr(e)
2. dynamique, mais j'en doute
3. statique, et j'en suis sûr(e)
4. statique, mais j'en doute
5. je ne sais pas

Introduction à la TLP

Typage dynamique/statique

Typage dynamique : Le type d'une expression est connu au moment de l'exécution.
Typage statique : Le type est connu au moment de la compilation.

Question :
En Caml, le typage est

1. dynamique, j'en suis sûr(e)
2. dynamique, j'en doute
3. statique, j'en suis sûr(e)
4. statique, j'en doute
5. je ne sais pas

Un indice :

```
#3/0 ;;
Exception non rattrapée: Division_by_zero
#(3/0) + [2] ;;
Entrée interactive:
>(3/0) + [2] ;;
>        ^^^
Cette expression est de type int list,
mais est utilisée avec le type int.
```

Introduction à la TLP

Typage dynamique/statique

Typage dynamique : Le type d'une expression est connu au moment de l'exécution.
Typage statique : Le type est connu au moment de la compilation.

Caml : Typage statique
Les erreurs de type sont détectées avant le début de l'évaluation.

```
#3/0 ;;
Exception non rattrapée: Division_by_zero
#(3/0) + [2] ;;
Entrée interactive:
>(3/0) + [2] ;;
>        ^^^
Cette expression est de type int list,
mais est utilisée avec le type int.
```

Typage dynamique/statique

Typage dynamique : Le type d'une expression est connu au moment de l'exécution.
Typage statique : Le type est connu au moment de la compilation.

Lisp : Typage dynamique
Les erreurs de type sont détectées pendant l'exécution.

```
(defun foo (a)
  (cond ((<= a 3) (+ a 1))
        (t (+  (car a) 1))))
```

- Appel `(foo 3)` donne 4
- Appel `(foo 4)` : erreur
- Appel `(foo (2,3))` donne 3

Introduction à la TLP

Typage faible/fort

Typage fort : Une discipline stricte est imposée.
Typage faible : Des manquements à la discipline sont tolérés.

Remarque : Attention à cette "définition" !

C : Typage faible

Java : Typage fort

Caml : Typage fort, mais ...

```
 public Liste  TriTopo(){
int[] D = this.degE() ;
Liste S = new Liste();
Liste Tri = new Liste();
int x,y ;
for (int i = 1; i<=n; i++){
```

```
let rec tranche = fun
|(1,1,a::l)->[a]
|(1,m,a::l)->a::tranche(1,m-1,l)
|(n,m,a::l)->tranche(n-1,m-1,l)
|_-> failwith "la liste est trop courte";;
tranche : int * int * 'a list -> 'a list = <fun>
```

Introduction à la TLP

Typage faible/fort

Typage fort : Une discipline stricte est imposée.
Typage faible : Des manquements à la discipline sont tolérés.

Remarque : Attention à cette "définition" !

C : Typage faible

Java : Typage fort

Caml : Typage fort, mais ... pas de déclarations explicites !

↬ *inférence de types*

```
 public Liste  TriTopo(){
int[] D = this.degE() ;
Liste S = new Liste();
Liste Tri = new Liste();
int x,y ;
for (int i = 1; i<=n; i++){
```

```
let rec tranche = fun
|(1,1,a::l)->[a]
|(1,m,a::l)->a::tranche(1,m-1,l)
|(n,m,a::l)->tranche(n-1,m-1,l)
|_-> failwith "la liste est trop courte";;
tranche : int * int * 'a list -> 'a list = <fun>
```

Sommaire

Où en est-on ?

Vérification de types

Pour déterminer si un programme est bien typé, on peut

Vérifier si les types du programme sont cohérents.
Il faut disposer au préalable d'annotations de type

Caml permet la vérification de programmes annotés :

```
let f = fun (x : int) -> x + 2 ;;
f : int -> int = <fun>

let g = fun (x : bool) -> x + 2 ;;
Cette expression est de type bool,
mais est utilisée avec le type int.
```

Inférence de types

Pour déterminer si un programme est bien typé, on peut aussi

Inférer des annotations cohérentes.
Pourvu qu'elles existent !

Caml permet l'inférence de types :

```
let f = fun x -> x + 2 ;;
f : int -> int = <fun>

let f = fun (x,y) -> x < y ;;
f : 'a * 'a -> bool = <fun>
```

L'objectif du cours est de comprendre ces deux mécanismes

Plan du cours

Programme