Programmation fonctionnelle

L2 Info et Math-info, 2018–19

Marc Zeitoun

24 septembre 2018

Contrôle continu

Test 1 cette semaine

- ~ 25mn,
- Sur feuille, en cours intégré.
- Programme : ce qui a été vu jusqu'à présent.

Plan

Récursivité (rappels)

Notion d'efficacité

Types « somme », types récursifs

Récursivité, récapitulatif

Pour définir une fonction récursive,

- on doit avoir un ordre $\leqslant$ sur les entrées du problème.
- Doit être **bien fondé** : **pas de** suite infinie strictement décroissante.

Récursivité, récapitulatif

Pour définir une fonction récursive,

- on doit avoir un ordre $\leqslant$ sur les entrées du problème.
- Doit être **bien fondé** : **pas de** suite infinie strictement décroissante.

Sous ces hypothèses, pour écrire un algorithme, on peut

- décrire son comportement sur les éléments minimaux pour $\leqslant$,
- spécifier son comportement sur les autres entrées en fonction de son comportement sur des entrées **strictement plus petites**.

Exemples d'ordres bien fondés

- Entiers naturels $\{0, 1, 2, \ldots\}$, ordre $<$.
  - on définit la fonction sur 0,
  - on définit récursivement la valeur en $n > 0$ à partir de valeurs $< n$.
  - Exemples : factorielle, suite de Fibonacci, etc.

Exemples d'ordres bien fondés

- Entiers naturels $\{0, 1, 2, \ldots\}$, ordre $<$.
  - on définit la fonction sur 0,
  - on définit récursivement la valeur en $n > 0$ à partir de valeurs $< n$.
  - Exemples : factorielle, suite de Fibonacci, etc.

- Les couples (x, y) d'entiers naturels, ordre lexicographique.

$$(x, y) < (x', y') \quad \textbf{ssi} \quad \textbf{soit } x < x', \textbf{ soit } x = x' \text{ et } y < y'$$

Exemples d'ordres bien fondés

- Entiers naturels $\{0\,,1\,,2\,,\ldots\}$, ordre $<$.
  - on définit la fonction sur 0,
  - on définit récursivement la valeur en $n > 0$ à partir de valeurs $< n$.
  - Exemples : factorielle, suite de Fibonacci, etc.

- Les couples $(x\,,y)$ d'entiers naturels, ordre lexicographique.

$$(x, y) < (x', y') \quad \textbf{ssi} \quad \textbf{soit } x < x', \textbf{ soit } x = x' \text{ et } y < y'$$

  - on définit ce que vaut la fonction sur $(0\,,0)$
  - on définit récursivement la valeur en $(x\,,y) > (0\,,0)$ à partir de valeurs inférieures à $(x\,,y)$.

Exemple : marches d'escalier

Supposément posé par les recruteurs chez Amazon.

- `https://youtu.be/5o-kdjv7FD0`
- Une personne monte un escalier à n marches.
- Elle monte à chaque pas soit une, soit deux marches.
- De combien de façons peut-elle monter les escaliers ?

Plan

Récursivité (rappels)

Notion d'efficacité

Types « somme », types récursifs

Rappels : exponentielle et logarithme

- Fonctions **exponentielles** courantes : de la forme c^n avec $c > 1$.
- **Exemples** typiques : 2^n, 3^n.

Rappels : exponentielle et logarithme

- Fonctions **exponentielles** courantes : de la forme c^n avec $c > 1$.
- **Exemples** typiques : 2^n, 3^n.

- **Logarithme** à base $c > 1$ = fonction réciproque de $n \longmapsto c^n$.
- **Exemples** :
  - logarithme à base 10 $\simeq$

Rappels : exponentielle et logarithme

- Fonctions **exponentielles** courantes : de la forme c^n avec $c > 1$.
- **Exemples** typiques : 2^n, 3^n.

- **Logarithme** à base $c > 1$ = fonction réciproque de $n \longmapsto c^n$.
- **Exemples** :
  - logarithme à base 10 $\simeq$ nombre de divisions par 10 pour atteindre $\leqslant 1$.

Rappels : exponentielle et logarithme

- Fonctions **exponentielles** courantes : de la forme c^n avec $c > 1$.
- **Exemples** typiques : 2^n, 3^n.

- **Logarithme** à base $c > 1$ = fonction réciproque de $n \longmapsto c^n$.
- **Exemples** :
  - logarithme à base 10 $\simeq$ nombre de divisions par 10 pour atteindre $\leqslant 1$.
  - logarithme à base 10 $\simeq$ nombre de chiffres en base 10.

Rappels : exponentielle et logarithme

- Fonctions **exponentielles** courantes : de la forme c^n avec $c > 1$.
- **Exemples** typiques : 2^n, 3^n.

- **Logarithme** à base $c > 1$ = fonction réciproque de $n \longmapsto c^n$.
- **Exemples** :
  - logarithme à base 10 $\simeq$ nombre de divisions par 10 pour atteindre $\leqslant 1$.
  - logarithme à base 10 $\simeq$ nombre de chiffres en base 10.
  - $\log_2(n) = \log_2(10) \times \log_{10}(n) \simeq 3\ .32 \log_{10}(n)$

Notion de complexité

Pour mesurer l'efficacité en temps d'un algorithme, on évalue le nombre d'opérations élémentaires

- sur une entrée **de taille n**, en **fonction de n**,
- dans **le pire** cas.

Le temps de calcul doit être fini !

Efficacité

Temps de calcul, sur un ordinateur 1GHz, pour

- Des complexités en n, $n\log_{10}(n)$, n^2, 2^n,
- et des tailles de problèmes de $n = 10$, 100, 1000, 10^6, 10^9.

	n	$n\log_2(n)$	n^2	2^n
$n = 10$	✓	✓	✓	✓
$n = 100$	✓	✓	✓	$> 10^{13}$ ans ✗
$n = 1000$	✓	✓	✓	$> 10^{280}$ ans ✗
$n = 10^6$	✓	✓	16mn	✗
$n = 10^9$	$1s$	$9s$	> 31 ans ✗	✗

✓ = moins de 10^{-2}s

Âge estimé de l'univers : $\simeq 14\ .10^9$ ans

Ordre de grandeur, notation $O()$

- $f = O(g)$: pour n assez grand, f est majorée par $c\,.g$ pour $c > 0$.

$$f = O(g) \iff \exists K \in \mathbb{N}, \exists c > 0, \forall n\ n > K \Rightarrow f(n) \leqslant cg(n).$$

Exemples

- $42n^7 + 2017n^4 + 1111n^3 + 2 = O(n^7)$
- $42n^7 + 2017n^4 + 1111n^3 + 2 = O(n^8)$
- $n^{1000} = O(1\,.1^n)$
- $n\log(n) = O(n^2)$

Récursivité : efficacité

- Exemple : calcul de 2^x pour $x \geqslant 0$.

```
let rec pow2 = fun x ->
  if x <= 0
  then 1
  else pow2 (x-1) + pow2 (x-1)
```

- Quelle est la complexité de ce calcul de 2^x ?
- Comptons le nombre $A(x)$ d'opérations '+' effectuées par `pow2(x)`

Récursivité : efficacité

- Exemple : calcul de 2^x pour $x \geqslant 0$.

```
let rec pow2 = fun x ->
  if x <= 0
  then 1
  else pow2 (x-1) + pow2 (x-1)
```

- Quelle est la complexité de ce calcul de 2^x ?
- Comptons le nombre $A(x)$ d'opérations '+' effectuées par `pow2(x)`
- $A(0) = 0$, et $A(x) = 2 \cdot A(x-1) + 1$.

Récursivité : efficacité

- Exemple : calcul de 2^x pour $x \geqslant 0$.

```
let rec pow2 = fun x ->
  if x <= 0
  then 1
  else pow2 (x-1) + pow2 (x-1)
```

- Quelle est la complexité de ce calcul de 2^x ?
- Comptons le nombre $A(x)$ d'opérations '+' effectuées par `pow2(x)`
- $A(0) = 0$, et $A(x) = 2 \cdot A(x-1) + 1$.
- Donc $A(x) = 2^x - 1$ [Ce n'est pas étonnant : pourquoi ?].

Récursivité : efficacité

```
let rec pow2bis = fun x ->
    if x <= 0
    then 1
    else 2 * pow2bis (x-1)
```

- Quelle est la complexité de ce calcul de 2^x ?
- Comptons le nombre $M(x)$ d'opérations '$*$' effectuées.
- $M(0) = 0$, et $M(x) = M(x-1) + 1$.
- Donc $M(x) = x$.

On gagne une exponentielle par rapport au cas précédent, grâce à 1 appel récursif au lieu de 2.

Récursivité : efficacité

- On peut encore améliorer cette complexité.
- On utilise le fait que $2^x = (2^{x/2})^2 \cdot 2^{x\,\%\,2}$.

```
let rec pow2ter = fun x ->
    if (x <= 0)
    then 1
    else let y = pow2ter(x/2) in
    if (x mod 2 = 0)
    then y * y
    else y * y * 2
```

- La complexité $B(x)$, en nombre de '$*$' est telle que $B(0) = 0$ et $B(x) \leqslant B(x/2) + 2$.
- Il y a donc $O(\log(x))$ multiplications.

Questions

- Peut-on faire mieux (asymptotiquement) que la complexité précédente pour calculer 2^x ?
- Comment adapter la technique pour le calcul du $x^{\text{ème}}$ nombre de Fibonacci ? Pour toute récurrence linéaire à 2 termes ?
  - Par exemple, pour Fibonacci, l'algorithme naïf

```
let fib = fun x ->
    if (x <= 1)
    then 1
    else fib (x-1) + fib(x-2)
```

    a une complexité $\approx \phi^x$, soit double exponentielle, où $\phi = \frac{1+\sqrt{5}}{2}$.
  - Peut-on trouver une version en $O(x)$? En $O(\log x)$?
- Donner une version non récursive de la fonction `pow2ter`.

Plan

Récursivité (rappels)

Notion d'efficacité

Types « somme », types récursifs

Types « Somme »

- Définition de types contenant un ensemble fini de valeurs :

```
type figure = Roi | Dame | Cavalier | Valet
```

- Similaire, pour l'instant, aux `enum` de C.

Types « Somme »

- Définition de types contenant un ensemble fini de valeurs :

```
type figure = Roi | Dame | Cavalier | Valet
```

- Similaire, pour l'instant, aux `enum` de C.
- Le nom du type est un identificateur, commençant par une minuscule.
- Les valeurs sont des noms commençant par une majuscule.

Types « Somme »

- On veut parfois faire porter une information supplémentaire aux valeurs.

```
type entier_ou_infini = Entier of int | Infini
```

Types « Somme »

- On veut parfois faire porter une information supplémentaire aux valeurs.

```
type entier_ou_infini = Entier of int | Infini
```

- Des valeurs légales de ce type sont
  - `Entier(8)`
  - `Entier(-42)` ,
  - `Infini` .

Types « Somme »

- On veut parfois faire porter une information supplémentaire aux valeurs.

```
type entier_ou_infini = Entier of int | Infini
```

- Des valeurs légales de ce type sont
  - `Entier(8)`
  - `Entier(-42)` ,
  - `Infini` .

Exercice : définir un type pour représenter des cartes de tarot.

Types Produit Cartésien

- L'opérateur `*` permet de représenter un **produit Cartésien**.
- Le produit Cartésien de A et B est l'ensemble

$$A \times B = \{(a, b) \mid a \in A \text{ et } b \in B\}.$$

Un type pour représenter des listes

- On peut redéfinir un type **liste d'entiers** de la façon suivante.

```
type liste_entiers = ListeVide
                   | Noeud of int * liste_entiers
```

(Définition en fait inutile, les constructeurs de listes sont prédéfinis)

Un type pour représenter des listes

- On peut redéfinir un type **liste d'entiers** de la façon suivante.

```
type liste_entiers = ListeVide
                   | Noeud of int * liste_entiers
```

(Définition en fait inutile, les constructeurs de listes sont prédéfinis)

- Éléments légaux de ce type ?

Un type pour représenter des listes

- On peut redéfinir un type **liste d'entiers** de la façon suivante.

```
type liste_entiers = ListeVide
                   | Noeud of int * liste_entiers
```

(Définition en fait inutile, les constructeurs de listes sont prédéfinis)

- Éléments légaux de ce type ?
- Comment définir un type liste **générique** ?

Un type pour représenter des listes

- On peut redéfinir un type **liste d'entiers** de la façon suivante.

```
type liste_entiers = ListeVide
                   | Noeud of int * liste_entiers
```

(Définition en fait inutile, les constructeurs de listes sont prédéfinis)

- Éléments légaux de ce type ?
- Comment définir un type liste **générique** ?

```
type 'a liste = ListeVide
              | Noeud of 'a * 'a liste
```

La construction match

- Permet d'**inspecter** la forme d'une valeur.
- Calcule une expression.
- **Très utile** pour les types somme.

```
match x with
  | ListeVide   -> .....
  | Noeud(x, l) -> .....
```

La construction match

- Permet d'**inspecter** la forme d'une valeur.
- Calcule une expression.
- **Très utile** pour les types somme.

```
match x with
  | ListeVide    -> .....
  | Noeud(x, l) -> .....
```

- Ce qui est calculé par `match` est une expression des `.....`, celle qui correspond à la valeur de l'expression `x`.

La construction match

- Permet d'**inspecter** la forme d'une valeur.
- Calcule une expression.
- **Très utile** pour les types somme.

```
match x with
  | ListeVide    -> .....
  | Noeud(x, l) -> .....
```

- Ce qui est calculé par `match` est une expression des `.....`, celle qui correspond à la valeur de l'expression `x`.
- **Exemple** Calcul de la longueur d'une liste.