La récursivité (épisode 1)

NFA032

Présentation n°7

Introduction

- Origine
  - Le raisonnement par récurrence
    - Soit $P(k)$ une propriété à démontrer, paramétrée par un entier naturel k
    - Démontrons que $P(0)$ est vraie
    - Démontrons que $P(n)$ est vraie si l'on fait l'hypothèse que $P(n\text{-}1)$ l'est aussi (variante : hypothèse que tous les $P(k)$ sont vraies, quel que soit $k<n$)
    - On en déduit que $P(k)$ est vraie, quel que soit k
- En informatique, le concept se retrouve :
  - au niveau des fonctions
  - au niveau des types

La récursivité en général

- Principe général :
  - La définition d'une entité est récursive si la définition d'au moins l'un de ses composants ou cas s'appuie sur cette même définition
    - Exemple de fonction récursive : *la factorielle d'un naturel n*
      - Définition constructive :
        - » $0! = 1$
        - » $n! = n \times (n-1) \times (n-2) \times \cdots \times 1$ si *n*>0
      - Définition récursive :
        - » $0! = 1$
        - » $n! = n \times (n-1)!$ si *n*>0
    - Exemple de type récursif : *les listes*
      - Une liste d'éléments de type T est :
        - » soit vide
        - » soit non vide, auquel cas elle a :
          - une tête, qui est un élément de type T
          - un reste, qui est une liste d'éléments de type T

La récursivité en général

- Dans cette première partie du cours :
  - on s'intéresse à la *récursivité des fonctions* (méthodes) $\Rightarrow$ *appels récursifs*
  - La récursivité des types (classes) sera l'objet de la prochaine séance
- Un conseil de lecture :
  - *Gödel, Escher et Bach : les Brins d'une Guirlande Éternelle*, de Douglas HOFSTADTER
    - Une vulgarisation peu commune de concepts profonds (récursivité, infini, auto-référence, …) qui fondent l'informatique (mais pas seulement…).

La récursivité dans les méthodes

- Définition
  - Une méthode *m* est récursive si son corps contient :
    - un appel à elle-même $\Rightarrow$ récursivité *directe*
    - un appel à une autre méthode, qui elle-même est récursive $\Rightarrow$ récursivité *indirecte*
      - Si cette autre méthode appelle elle-même *m*, la récursion est dite *mutuelle* ou *croisée*
- Principes de construction
  - Une définition récursive est une définition par cas :
    - Quelques cas particuliers sont solutionnés sans récursion $\Rightarrow$ *cas d'arrêt* ou de *terminaison*
    - Les autres cas impliquent une décomposition en sous-problèmes, dont au moins un est résolu par un appel récursif.

La récursivité dans les méthodes

- Principales difficultés
  - Définir impérativement les cas d'arrêt
  - Montrer que tout traitement d'un autres cas :
    - converge vers au moins un cas d'arrêt
    - en un nombre fini d'appels
  - Montrer que tous les cas sont traités
- Remarque :
  - Les *paramètres de récursion* sont ceux sur lesquels joue la récursivité

Exemple

Calculer la somme de deux naturels *a* et *b* en se limitant aux seules opérations +1 et -1 :

Paramètre de récursion

- Principe général : $\text{somme}(a, b) = a\overbrace{+1 + \cdots + 1}^{b \text{ fois}}$
- cas d'arrêt (*b*=0) : $\text{somme}(a, 0) = a$
- cas général (*b*>0) :
  - $\text{somme}(a, b) = a\overbrace{+1 \cdots + 1}^{b \text{ fois}} = \underbrace{\left(a\overbrace{+1 \cdots + 1}^{b-1 \text{ fois}}\right)}_{\text{somme}(a,b-1)} + 1 = \text{somme}(a, b-1) + 1$
  - autre possibilité :

    $\text{somme}(a, b) = \underbrace{(a+1)\overbrace{+1 \cdots + 1}^{b-1 \text{ fois}}}_{\text{somme}(a+1,b-1)} = \text{somme}(a+1, b-1)$
  - Attention : toute propriété mathématiquement fondée n'est pas nécessairement algorithmiquement intéressante (problèmes de convergence) :

$$\text{somme}(a, b) = \text{somme}(a+1, b-1) = \text{somme}(a-1, b+1) = \text{somme}(a-1, b) + 1$$

Dans cet exemple, il suffit d'ajouter un cas d'arrêt : $\text{somme}(0, b) = b$

Exemple

- Adoptons la définition :

$$\text{somme}(a, b) = a \qquad \textbf{si } b = 0 \quad [1]$$
$$= \text{somme}(a, b-1) + 1 \qquad \textbf{si } b > 0 \quad [2]$$

- Définition en Java :

```
// version récursive non terminale
static int somme_rnt(int a, int b) {
  if (b==0) return a;
  else      return 1+somme_rnt(a, b-1);
}
```

Exemple

- Calculons $\text{somme}(7,4)$:

$$
\begin{aligned}
\text{somme}(7,4) &= \text{somme}(7,3) + 1 && \text{applique [2]} \\
&= (\text{somme}(7,2) + 1) + 1 && \text{applique [2]} \\
&= \big((\text{somme}(7,1) + 1) + 1\big) + 1 && \text{applique [2]} \\
&= \Big(\big((\text{somme}(7,0) + 1) + 1\big) + 1\Big) + 1 && \text{applique [2]} \\
&= \Big(\big((7 + 1) + 1\big) + 1\Big) + 1 && \text{applique [1]} \\
&= \big((8 + 1) + 1\big) + 1 && \text{calcul en attente} \\
&= (9 + 1) + 1 && \text{calcul en attente} \\
&= 10 + 1 && \text{calcul en attente} \\
&= 11 && \text{fin}
\end{aligned}
$$

Récursion et itération

- Une méthode récursive s'appelle elle-même un nombre fini de fois
  - La récursion est une forme d'itération
- Théorèmes importants (calculabilité)
  - Toute définition récursive peut être récrite en une version équivalente itérative
  - Toute définition itérative peut être récrite en une version équivalente récursive
  - Malheureusement, les preuves ne sont pas constructives dans le cas général : il n'existe aucune méthode pour passer d'une définition (récursive ou itérative) à l'autre (itérative ou récursive, respectivement).

Récursion et itération

- Récursion terminale et non terminale
  - Une méthode récursive est dite *terminale* si le résultat de **tout** appel récursif **est** le résultat final de la méthode ; est dite *non terminale* sinon
  - Exemple de définition :
    - récursive non terminale :

$$\begin{aligned} \text{somme}(a,b) &= a && \textbf{si } b = 0 \quad [1] \\ &= \boxed{\text{somme}(a,b-1)} + 1 && \textbf{si } b > 0 \quad [2] \end{aligned}$$

    - récursive terminale :

$$\begin{aligned} \text{somme}(a,b) &= a && \textbf{si } b = 0 \quad [1] \\ &= \boxed{\text{somme}(a+1,b-1)} && \textbf{si } b > 0 \quad [2] \end{aligned}$$

- Il existe un algorithme capable de transformer toute définition récursive terminale en sa version itérative équivalente
  - algorithme exploité par tous les compilateurs d'aujourd'hui

Récursion et itération

- Intérêt de la récursion terminale
  - Pas de calcul en attente
- Adoptons la définition :

$$\text{somme}(a,b) = a \quad \textbf{si } b = 0 \quad [1]$$
$$= \text{somme}(a+1, b-1) \quad \textbf{si } b > 0 \quad [2]$$

- Calculons $\text{somme}(7,4)$:

$\text{somme}(7,4)$	$= \text{somme}(8,3)$	applique [2]
	$= \text{somme}(9,2)$	applique [2]
	$= \text{somme}(10,1)$	applique [2]
	$= \text{somme}(11,0)$	applique [2]
	$= 11$	applique [1]

Exemple : somme en Java

```java
// version récursive non terminale
static int somme_rnt(int a, int b) {
  if (b==0) return a;
  else      return 1+somme_rnt(a, b-1);
}
```

```java
// version récursive terminale
static int somme_rt(int a, int b) {
  if (b==0) return a;
  else      return somme_rt(a+1, b-1);
}
```

```java
// version itérative
static int somme_it(int a, int b) {
  int s = a, y = b;
  while (y>0) { s++; y--; };
  return s;
}
```

Exemples en arithmétique des entiers

- Le produit de deux entiers naturels *a* et *b*
  - Version 1 : ne s'appuie que sur des sommes
    - Principe : $\text{produit}(a,b) = 0\overbrace{+a+\cdots+a}^{b \text{ fois}}$
    - Définition récursive non terminale :

$$\begin{aligned} \text{produit}(a,b) &= 0 && \textbf{si } b = 0 \quad [1] \\ &= \text{a} + \text{produit}(a, b-1) && \textbf{si } b > 0 \quad [2] \end{aligned}$$

  - Version 2 : s'appuie sur la parité de *b*, et autorise multiplication ou division par 2 seulement :

$$\begin{aligned} \text{produit}(a,b) &= 0 && \textbf{si } b = 0 \quad [1] \\ &= \text{produit}\left(2a, \frac{b}{2}\right) && \textbf{si } b > 0 \text{ et pair} \quad [2] \\ &= \text{a} + \text{produit}(a, b-1) && \textbf{si } b \text{ impair} \quad [3] \end{aligned}$$

    ou encore mieux (multiplication des processeurs) :

$$\text{produit}(a,b) = \text{a} + \text{produit}\left(2a, \frac{b-1}{2}\right) \textbf{ si } b \text{ impair} \quad [3]$$

Exemples en arithmétique des entiers

- Version 3 : version terminale de la version 1
  - Définition récursive terminale :

$$\text{produit}(a, b) = \text{produit}(a, b, 0)$$

avec (récursion indirecte) :

$$\text{produit}(a, b, p) = p \qquad \textbf{si } b = 0 \quad [1]$$
$$= \text{produit}(a, b - 1, p + a) \qquad \textbf{si } b > 0 \quad [2]$$

  - Remarques (pas au programme de NFA032) :
    - Transformer une récursion non terminale en une version terminale n'est pas toujours facile
    - Cette transformation conduit souvent à ajouter des paramètres (accumulateurs)
      - ils contiennent dès que possible…
      - …les résultats qui étaient mis en attente dans la version non terminale

Exemples en arithmétique des entiers

- Quotient et reste de la division entière de *a* par *b*>0, en n'exploitant que des additions ou soustractions.
  - En posant *q*=quotient et *r*=reste, on sait que :

$$\begin{array}{r|l} a & b \\ \hline r & q \end{array}$$

    - $a = b\,q + r$
    - $0 \leq r < b$
  - Intuitions à exploiter :
    - *q* indique le nombre de fois où l'on peut retirer la quantité *b* de *a*
    - *r* est ce qu'il reste de *a* une fois retirées les *q* quantités *b*

Exemples en arithmétique des entiers

- Définition récursive du quotient :

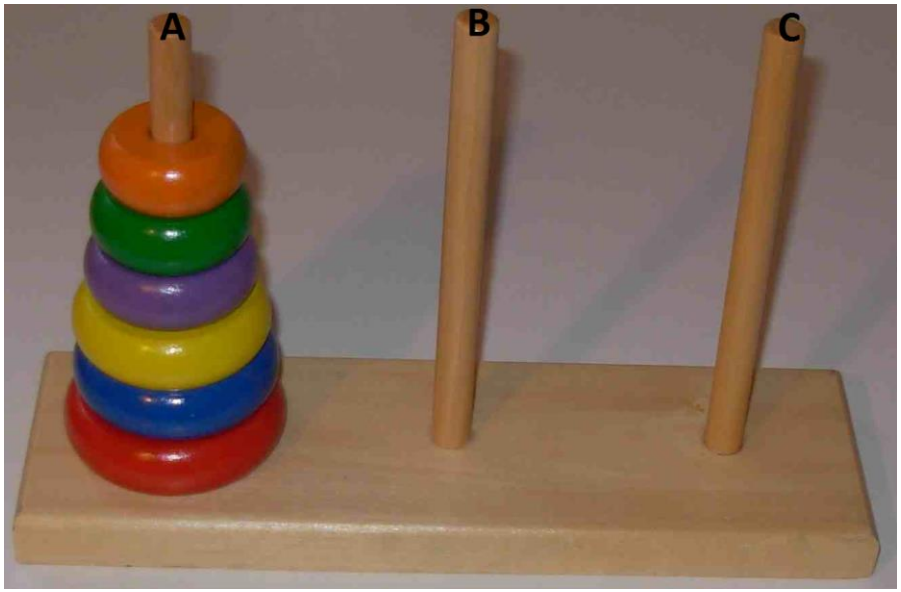
$$\begin{aligned} \text{quotient}(a,b) &= 0 && \textbf{si } b > a \quad [1] \\ &= 1 + \text{quotient}(a-b,b) && \textbf{si } b \leq a \quad [2] \end{aligned}$$

- Définition récursive du reste :

$$\begin{aligned} \text{reste}(a,b) &= a && \textbf{si } b > a \quad [1] \\ &= \text{reste}(a-b,b) && \textbf{si } b \leq a \quad [2] \end{aligned}$$

- Exercices :
  - Définitions terminales ou pas ?
  - Comment transformer les définitions non terminales en terminales ?
  - Définir x^n, où *n* entier naturel et *x* réel
    - Version 1 : exploiter uniquement des produits, des sommes ou des soustractions
    - Version 2 : affiner la version 1 en exploitant la parité de *n* et en s'autorisant la division par 2

Les Tours de Hanoï



- Jeu type *casse-tête* :
  - Objectif : déplacer les *n* disques de la pile A vers la pile C, tous les disques étant placés initialement sur A
  - Règles :
    - Déplacer un disque à la fois, toujours celui en haut de pile
    - Un disque ne peut être placé sur une pile que si elle est vide ou que le disque déjà présent en haut de la pile est plus petit que lui
  - Résultat : il faut procéder à $2^n - 1$ déplacements au minimum…

Exemple plus élaboré : recherche d'un 0 d'une fonction

- Exercice :
  - Soit *f* une fonction réelle continue, [*a*,*b*] un intervalle réel, $\varepsilon \geq 0$ (précision)
  - $\text{zéro}(f, a, b, \varepsilon)$ retourne un ***zéro*** de *f* quand il existe, valeur *z* telle que $a \leq z \leq b$ et $f(z) = 0$ (en fait une version moins stricte : $|f(z)| \leq \varepsilon$) ; il peut ne pas exister, ou en exister plusieurs.
  - Ecrire une version récursive et une version itérative
    - Principes à exploiter :
      - Procéder à une recherche dichotomique : toujours étudier le milieu *m* de [*a*,*b*]
      - Déclencher l'exception `aucunZéroTrouvé` si échec

Exemple : recherche d'un 0

- Fonction $\text{zéro}(f, a, b, \varepsilon)$
  - soit $m = \frac{a+b}{2}$ le milieu de [*a*,*b*]
- Spécification des cas :
  - $\text{zéro}(f, a, b, \varepsilon) = m$ si $|f(m)| \leq \varepsilon$
    - un zéro a été trouvé : *m*
  - $\text{zéro}(f, a, b, \varepsilon) = \text{erreur}$ si $b - a \leq \varepsilon$
    - aucun zéro trouvé jusqu'ici, et l'intervalle est réduit à presqu'un point
  - $\text{zéro}(f, a, b, \varepsilon) = zéro(f, a, m, \varepsilon)$ si $f(a)$ et $f(b)$ ont des signes distincts et $f(b)$ et $f(m)$ ont même signe
    - on est certain qu'il y a un zéro dans l'intervalle [*a*,*m*]
  - $\text{zéro}(f, a, b, \varepsilon) = zéro(f, m, b, \varepsilon)$ si $f(a)$ et $f(b)$ ont des signes distincts et $f(a)$ et $f(m)$ ont même signe
    - on est certain qu'il y a un zéro dans l'intervalle [*m*,*b*]
  - $\text{zéro}(f, a, b, \varepsilon) = z$ si $f(a)$ et $f(b)$ sont de même signe, avec, dans l'ordre :
    1. $z = z_1 = \text{zéro}(f, a, m, \varepsilon)$ si $z_1 \neq \text{erreur}$
    2. $z = z_2 = \text{zéro}(f, m, b, \varepsilon)$ si $z_1 = \text{erreur}$ et $z_2 \neq \text{erreur}$
    3. $z = \text{erreur}$ si $z_1 = z_2 = \text{erreur}$
    - on ne sait rien : essaie d'abord de trouver un zéro dans l'intervalle [*a*,*m*] ; si échec, essaie dans [*m*,*b*] ; sinon conclure en échouant.

Exercices

- Recherche d’une valeur dans un tableau
- Jeu du nombre à deviner
- Tris d’un tableau
  - Tri par recherche du min
  - Tri rapide
- Suites célèbres
  - Suite de Fibonacci
  - Suite de Syracuse