Compléments de Programmation

Benoit Donnet

Année Académique 2023 - 2024

Agenda

Agenda

- Chapitre 9: Elimination de la Récursivité
  - Principe
  - Récursivité Terminale
  - Récursivité Non Terminale
  - Algorithme Générique

Agenda

- Chapitre 9: Elimination de la Récursivité
  - Principe
    - ✓ Idée
    - ✓ Exemple
    - ✓ Transformation
  - Récursivité Terminale
  - Récursivité Non Terminale
  - Algorithme Générique

Principe

- Rappel: que se passe-t-il lors d'un appel de fonction?
  1. création d'un contexte sur la pile
  2. copie des valeurs des paramètres sur la pile
  3. exécution de la fonction
  4. récupération de la valeur de retour
     - ✓ éventuellement, récupération des paramètres
  5. destruction du contexte
- Une fonction récursive obéit à un tel schéma

Principe (2)

- Une fonction récursive peut être moins efficace qu'une solution itérative
  - dans le temps
    - ✓ l'appel de fonction a un prix
  - dans l'espace
    - ✓ il faut pouvoir stocker le contexte

Principe (3)

- Pourquoi éliminer la récursivité?
  - efficacité
  - meilleure compréhension de la récursivité

Exemple

- Le plus grand commun diviseur de 2 naturels
- Spécification

```
/*
 * @pre: a, b ≥ 0
 * @post: (a mod r = 0), (b mod r = 0),
 *        ¬(∃ s, (s > r) ∧ (a mod s = 0) ∧ (b mod s = 0))
 */
pgcd(int a, int b) = (int r)
```

- Définition récursive

```
pgcd(int a, int b):
  if(b=0)
    then
      r ← a;
    else
      r ← pgcd(b, a mod b);
```

Exemple (2)

- Calcul de pgcd(420, 75)

pgc(420, 75) = pgcd(75, 45)

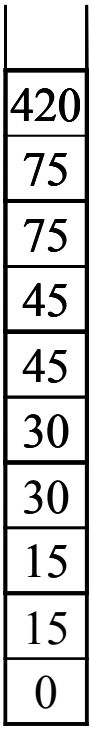
pgc(75, 45) = pgcd(45,30)

pgc(45, 30) = pgcd(30, 15)

pgc(30, 15) = pgcd(15, 0)

pgcd(15, 0) = **15**

Pile



Exemple (3)

- Une fois le cas de base atteint
  - récupération des paramètres
    - ✓ remontée récursive
  - ici, il n'y a pas d'autres calculs à effectuer
    - ✓ le résultat final est connu dès le cas de base atteint
    - ✓ récursivité terminale

Transformation

- Toutes les fonctions récursives peuvent être transformées en une forme non-récursive
- Différentes méthodes
  - récursivité terminale
    - méthode de transformation "simple"
  - récursivité non terminale
    - 2 méthodes
      - transformation de non terminale vers terminale
      - algorithme générique utilisant une pile
- Les compilateurs utilisent ces techniques pour optimiser le code
  - cfr. INFO0085

Agenda

- Chapitre 9: Elimination de la Récursivité
  - Principe
  - Récursivité Terminale
    - Technique
    - Exemples
  - Récursivité Non Terminale
  - Algorithme Générique

Technique

- Algorithme récursif générique

```
f(x):
  if(cond(x))
    then
      r ← g(x);
    else
      T(x);
      r ← f(x_int);
```

cas de base
traitement quelconque
appel récursif

- Elimination de la récursivité

```
  f'(x):
  u ← x;
  until cond(u) do
    T(u);
    u ← h(u);
  end
  r ← g(u);
```

utilisation d'une variable temporaire
tant que la condition de base n'est pas atteinte
traitement quelconque sur u
"avancer" u vers le cas de base
permet d'obtenir le résultat final

Exemples

- Type abstrait permettant de manipuler une chaîne de caractères
  - String

```
Type:
  String
Utilise:
  Char, Boolean
Opérations:
  empty_string: → String
  cons: Char × String → String
  car: String → Char
  cdr: String → String
  is_empty: String → Boolean
Préconditions:
  ∀ s ∈ String
  car(s) is defined iff is_empty(s) = False
```

Exemples (2)

- Obtenir le dernier caractère d'une chaîne
  - `last: String → Char`

```
last(String s):
  if(is_empty(cdr(s))
    then
      r ← car(s);
    else
      r ← last(cdr(s));
```

- Elimination de la récursivité

```
last'(String s):
  l ← s;
  until is_empty(cdr(l)) do
    l ← cdr(l);
  end
  r ← car(l);
```


Exemples (3)

- Obtenir le caractère à la position `i` d'une chaîne `s`
  - `get: Integer × String → Char`

```
get(String s, int i):
  if(i=0)
    then
      r ← car(s);
    else
      r ← get(cdr(s), i-1);
```

- Elimination de la récursivité

```
get'(String s, int i):
  l ← s; k ← i;
  until k=0 do
    l ← cdr(l);
    k ← k-1;
  end
  r ← car(l);
```

Exemples (4)

- Déterminer si un caractère appartient à la chaîne
  - `is_member: Char × String → Boolean`

```
is_member(Char c, String s):
  if(is_empty(s))
    then r ← False;
  if(car(s)=c)
    then r ← True;
    else r ← is_member(c, cdr(s));
```

- Elimination de la récursivité

```
is_member'(Char c, String s):
  l ← s;
  until is_empty(l) ∨ car(l)=c do
    l ← cdr(l);
  end
  if(is_empty(l))
    then r ← False
    else r ← True
```


Exemples (5)

- Plus grand commun diviseur

```
pgcd(int a, int b):
  if(b=0)
    then
      r ← a;
    else
      r ← pgcd(b, a mod b);
```

- Elimination de la récursivité

```
pgcd'(int a, int b):
  u ← a;
  v ← b;
  until v=0 do
    temp ← v;
    v ← u mod v;
    u ← temp;
  end
  r ← u;
```

Agenda

- Chapitre 9: Elimination de la Récursivité
  - Principe
  - Récursivité Terminale
  - Récursivité Non Terminale
    - ✓ Technique
    - ✓ Exemples
    - ✓ Boucler la Boucle
  - Algorithme Générique

Technique

- L'approche précédente n'est pas toujours applicable
  - ne concerne que les fonctions récursives terminales
- Pourquoi ça ne fonctionne pas pour de la récursivité simple (par exemple)?
  - remontée récursive
    - ✓ il faut stocker l'opération et les opérandes
    - ✓ il faut effectuer les opérations durant la remontée
    - ✓ fact(3) = 3 × fact(2) = 3 × 2 × fact(1) = 3 × 2 × 1 = 3 × 2 = 6

Technique (2)

- Ce qui est fait n'est plus à faire!
  - les calculs intermédiaires doivent être faits durant la descente récursive au lieu d'attendre la remontée
    - $\text{fact}(3) = 3 \times \text{fact}(2) = 3 \times 2 \times \text{fact}(1) = 6 \times \text{fact}(1) = 6 \times 1 = 6$
- Plus rien n'est à faire durant la remontée
  - récursivité terminale!
- Il faut donc transformer la récursivité non terminale en récursivité terminale!

Technique (3)

- Une seule variable additionnelle est suffisante pour stocker les calculs durant la descente
  - il suffit donc de conserver le résultat des opérations
    - et non l'opération à effectuer
- On ajoute donc un paramètre à la fonction récursive
  1. le prototype change
  2. on définit une fonction "chapeau", λ, avec plus de paramètres
  3. la fonction récursive non terminale se contente d'appeler λ

Technique (4)

- Attention, ça ne fonctionne pas toujours!
- Les opérations ne sont pas effectuées dans le bon ordre
  - l'opérateur doit être
    - ✓ commutatif
      - x α y = y α x
    - ✓ associatif
      - x α (y α z) = (x α y) α z
- Méthode plus générale à suivre

Technique (5)

- Approche étape par étape
  1. vérifier que l'opérateur est commutatif et associatif
  2. créer une fonction λ pour éliminer la récursivité, avec plus de paramètres
     - ✓ copie du/des paramètre(s) de la fonction récursive
     - ✓ ajout d'accumulateur(s) pour les opérations
  3. la fonction récursive appelle λ
     - ✓ passer en paramètre le(s) paramètre(s) récursif(s)
     - ✓ initialiser le(s) accumulateur(s)
  4. corps de la fonction λ
     - ✓ traitement général
       - les opérations intermédiaires sont faites sur le(s) accumulateur(s)
     - ✓ cas de base
       - le résultat est obtenu directement du/des accumulateur(s)

Exemples

- Calculer la longueur d'une chaîne
  - `len: String → Integer`

```
len(String s):
  if(is_empty(s))
    then
      r ← 0;
    else
      r ← 1 + len(cdr(s));
```

- Transformation en récursivité terminale

```
len(String s):
  r ← λ(s, 0);
```

```
λ(String s, int acc):
  if(is_empty(s))
    then
      r ← acc;
    else
      r ← λ(cdr(s), acc+1);
```


Exemples (2)

- Calculer la factorielle

```
fact(int n):
  if(n = 0)
    then
      r ← 1;
    else
      r ← n × fact(n-1);
```

- Transformation en récursivité terminale

```
fact(int n):
  r ← λ(n, 1);
```

```
λ(int n, int acc):
  if(n=0)
    then
      r ← acc;
    else
      r ← λ(n-1, acc × n);
```

Boucler la Boucle

- Une fois la forme récursive terminale obtenue, on peut éliminer la récursivité
- Il suffit d'appliquer l'approche vue précédemment
- Le processus de transformation est
  - simple
  - automatique
  - soigné
- ... quand il est applicable

Boucler la Boucle (2)

- Exemple: factorielle
- Version récursive

```
fact(int n):
  if(n = 0)
    then
      r ← 1;
    else
      r ← n × fact(n-1);
```

- Transformation en récursivité terminale

```
fact(int n):
  r ← λ(n, 1);
```

```
λ(int n, int acc):
  if(n=0)
    then
      r ← acc;
    else
      r ← λ(n-1, acc × n);
```

Boucler la Boucle (3)

- Elimination de la récursivité dans la fonction λ

```
λ'(int n, int acc):
  td ← n;
  a ← acc;
  until td = 0 do
    a ← a × td;
    td ← td - 1;
  end
  r ← a;
```

importance de l'ordre!

- Construction de la fonction non récursive `fact''`

```
fact''(int n):
  td ← n;
  a ← 1;
  until td = 0 do
    a ← a × td;
    td ← td - 1;
  end
  r ← a;
```


Agenda

- Chapitre 9: Elimination de la Récursivité
  - Principe
  - Récursivité Terminale
  - Récursivité Non Terminale
  - Algorithme Générique
    - ✓ Technique
    - ✓ Croissance des Nénuphars

Technique

- Les CPUs sont séquentiels
  - mais capables d'exécuter des fonctions récursives
- Il est toujours possible d'exprimer une fonction récursive de façon non récursive
- Principe
  - simuler la pile d'une fonction
  - en utilisant une pile
    - ✓ TAD

Technique (2)

- TAD `Stack` (partiel)

```
Type:
  Stack
Utilise:
  Boolean, Element, Natural
Opérations:
  empty_stack: → Stack
  is_empty: Stack → Boolean
  push: Stack × Element → Stack
  pop: Stack → Stack
  top: Stack → Element
Préconditions: ∀ s ∈ Stack
  ∀ s, ¬is_empty(s), pop(s)
  ∀ s, ¬is_empty(s), top(s)
Axiomes:
  is_empty(empty_stack) = True
  is_empty(push(s, e)) = False
  pop(push(s, e)) = s
  top(push(s, e)) = e
```

Technique (3)

Fonction récursive générique

```
f(int x):
  if(cond(x))
  then
    r ← g(x);
  else
    T(x);
    r ← G(x, f(x_int));
```

appel récursif générique

cas de base

traitement avant appel récursif

Fonction non récursive générique

```
f_derec(int x):
  p ← empty_stack();
  a ← x;
  until cond(a) do
    p = push(p, a);
    a ← h(a);
  end
  r ← g(a);
  until is_empty(p) do
    a ← top(p);
    p ← pop(p);
    T(a);
    r ← G(a, r);
  end
```

création de la pile pour le contexte

descente récursive

cas de base

remontée récursive

Nénuphar

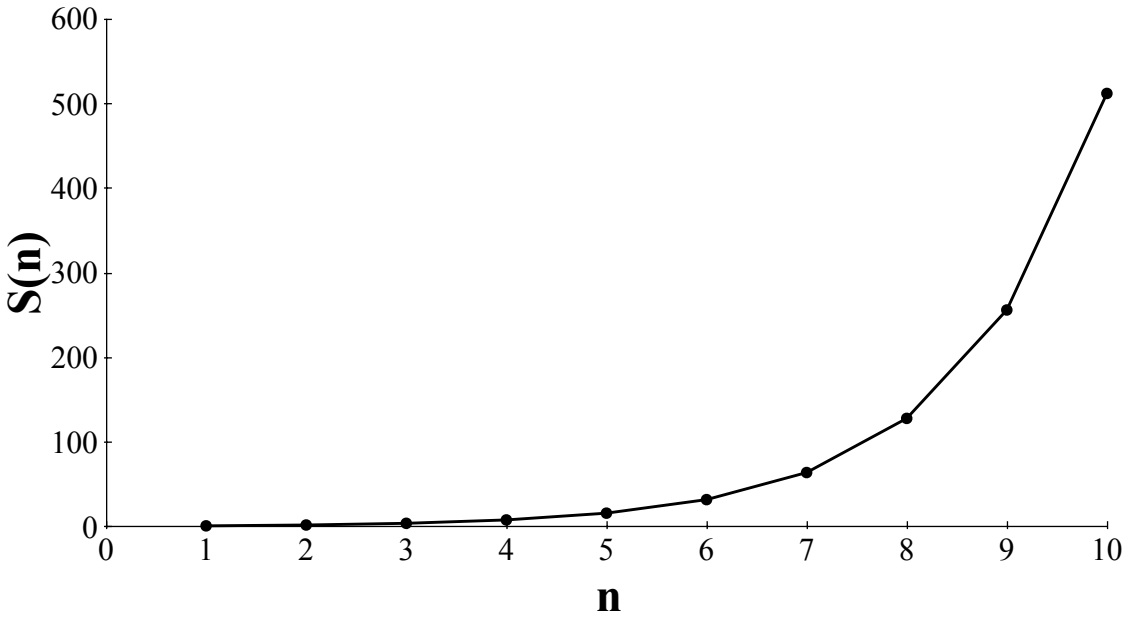
- Soit le problème suivant
  - une colonie de nénuphars double chaque printemps la surface qu'elle occupe
  - mais perd 0,5m^2 chaque automne
  - au début de la 1ère année, sa surface est de 1,5m^2
  - au début de la $n$ème année, sa surface est de $S(n)$m^2
- Questions
  1. définir récursivement $S(n)$
  2. implémenter la fonction récursive $S(n)$
  3. dérécursiver $S(n)$

Nénuphar (2)

1. Définition récursive de *S(n)*
   - *S(1)* = 1,5
   - *S(2)* = 2 × 1,5 - 0,5 = 2,5
     - ✓ 2 × *S(1)* - 0,5
   - *S(3)* = 2 × 2,5 - 0,5 = 4,5
     - ✓ 2 × *S(2)* - 0,5
   - *S(4)* = 2 × 4,5 - 0,5 = 8,5
     - ✓ 2 × *S(3)* - 0,5
   - *S(5)* = 2 × 8,5 - 0,5 = 16,5
     - ✓ 2 × *S(4)* - 0,5
   - ...

Nénuphar (3)

1. Définition récursive de *S(n)* (cont.)



Nénuphar (4)

1. Définition récursive de *S(n)* (cont.)
   - *S(n)* =
     - ✓ 1,5 si n=1
     - ✓ 2 × *S(n-1)* - 0,5 sinon

Nénuphar (5)

2. Implémentation récursive de *S(n)*

```
/*
 * @pre: n ≥ 1
 * @post: nenuphar = S(n)
 */
nenuphar(int n) = (float r)
```

Nénuphar (6)

2. Implémentation récursive de *S(n)* (cont.)

```
nenuphar(int n):
  if(n=1)
  then
    r ← 1.5;
  else
    r ← 2 * nenuphar(n-1) - 0.5;
```


Nénuphar (7)

3. Dérécursiver *S(n)*

```
nenuphar(int n):
  if(n=1)
  then
    r ← 1.5;
  else
    r ← 2 * nenuphar(n-1)-0.5;
```

```
f(int x):
  if(cond(x))
  then
    r ← g(x);
  else
    T(x);
    r ← G(x, f(x_int));
```

```
f_derec(int x):
  p ← empty_stack();
  a ← x;
  until cond(a) do
    p = push(p, a);
    a ← h(a);
  end
  r ← g(a);
  until is_empty(p) do
    a ← top(p);
    p ← pop(p);
    T(a);
    r ← G(a, r);
  end
```

Nénuphar (8)

3. Dérécursiver *S(n)* (cont.)

```
nenuphar_derec(int n):
  p ← empty_stack();
  a ← n;

  until a=1 do
    p ← push(p, a);
    a ← a-1;
  end

  r ← 1.5;

  until is_empty(p) do
    a ← top(p);
    p ← pop(p);
    r ← 2 *r - 0.5;
  end
```