INFO0947: Récursivité et Élimination de la Récursivité

SeyedPouria SALEHIKATOZI, s192865

Table des matières

1 Formulation Récursive

Pour résoudre un problème de manière récursive, nous allons d'abord chercher un paramètre récursive et un cas de base (une condition d'arrêt). Ensuite, on exprime le problème de manière récursive et enfin, on écrit le code correspondant.

Avant d'abord le projet, j'utilise le même système de représentation des nombres en décimale qui est fourni dans l'énoncé du projet. $(\text{numero})_{base}$
Qui est, décimal : b = 10 et l'ensemble des symboles est $\{0, 1, \ldots, 9\}$. Par exemple, $(77)_{10}$

1.1 Paramétre récursive :

n , La taille de la chaîne de caractère.

1.2 Cas de base :

n = 0

1.3 Définition du problème de manière récursive :

$$\begin{cases} 0 & \text{si n} = 0 \\ convert(hexa[n-1]) + 16 \times hexa\_dec\_rec(hexa, n-1)) & \text{sinon} \end{cases}$$

2 Spécification

```
/*PréCondition :  hexa initialisé ∧ n ≥ 0
 *
 *PostCondition : hexa = hexa0 ∧ n = n0 ∧
 *               hexa_dec_rec = convert(hexa[n-1]) + 16 × hexa_dec_rec(hexa,n-1)
 */
 unsigned int hexa_dec_rec(char *hexa, int n)
```

3 Construction Récursive

```
 unsigned int hexa_dec_rec(char *hexa, int n){
   assert(hexa != NULL && n >= 0);
   {Pré: hexa initialisé ∧ n ≥ 0}

   if(n == 0){
     {n = n0 ∧ hexa = hexa0}
      return 0;
     {Post ≡ n = n0 ∧ hexa = hexa0 ∧ hexa_dec_rec = (0)10}
     ⇒ {PostCondition}
   }
     {n > 0 ∧ n = n0 ∧ hexa = hexa0}
   return convert(hexa[n-1]) + 16 × (hexa_dec_rec(hexa, n-1));
     {Post ≡n = n0 ∧ hexa = hexa0 ∧ hexa_dec_rec = hexa_dec_rec(hexa, n)}
     ⇒ {PostCondition}
   }//fin hexa_dec_rec()
 }
```

4 Traces d'Exécution

4.1 Traces d'Exécution 27 :

		2	
	7	7	39

4.2 Traces d'Exécution A23 :

			10		
		2	2	162	
	3	3	3	3	2595

4.3 Traces d'Exécution A78E :

				10			
			7	7	167		
		8	8	8	8	2680	
	14	14	14	14	14	14	42894

5 Complexité

$T(n) = b + T(n - 1)$
$= b+[b + T(n-2)]$
$= 2\times b+T(n-2)$
$= 2\times b+[b+T(n-3)]$
$= \dots$
$= i \times b + T(n - i)$
$= \dots = n \times b + T(n - n)$
$= n \times b + T(0)$
$= n \times b + a$

Donc,

La Complexité : O(n)

6 Dérécursification

Comme la fonction hexa_dec_rec(*hexa, n) est une fonction récursive de type non-Terminal, car lors de remonter récursive y a de calculs à faire. Pour faire la décursification nous devons transformer cette fonction à une fonction récursive terminal.
Tout d'abord, nous vérifions associativité et commutativité l'opération lors de la remonter récursive.
Ensuite, nous introduisons une fonction λ qui prend les mêmes paramètres que la fonction hexa_dec_rec(*hexa, n) et un ou plusieurs paramètre(s) accumulateur(s) qui sera à conserver les résultats intermédiaires lors de la descendre récursive après chaque invocation récursive.
Enfin, nous appelons la fonction λ dans le corps de la fonction hexa_dec_rec() et nous initialisons le(s) accumulateur(s).

6.1 Le pseudo-code de la fonction hexa_dec_rec()

```
unsigned int hexa_dec_rec(char *hexa, int n):

   if(n = 0):
      then
      r <- 0;

   else:
      r <- convert(hexa[n-1]) + 16 × (hexa_dec_rec(hexa, n-1));
```

6.2 Etape 1 : Verification l'associativité et la commutativité

convert(hexa[n-1]) + 16 × (hexa_dec_rec(hexa, n-1)) ;
Par définition l'addition est à la fois associative et commutative, donc nous pouvons écrire de cette façon et avoir le même résultat,
16 × (hexa_dec_rec(hexa, n-1)) + convert(hexa[n-1]) ;
Comme nous concluons que l'appele récursive de notre fonction hexa_dec_rec() est commutative et associative alors elle remplit les conditions nécessaires pour que nous puissons faire la transformation d'une fonction non-terminal à une fonction terminal.

6.3 Etape 2 : Fonction λ

La fonction λ() prend les mêmes paramètres que la fonction hexa_dec_rec() plus 2 accumulateurs qui sont power et addition.

```
λ(char *hexa, int n, int power, int addition):

   if(n = 0):
      then
      r <- addition;

   else:
      tmp <- power × convert(hexa[n-1]);
      r <- λ(hexa, n-1, 16 × power, addition + tmp);
```

6.4 Etape 3 : Fonction hexa_dec_rec′(*hexa, n)

```
Fonction hexa_dec_rec‘(*hexa, n):

   r <- λ(hexa, n, 1, 0)
```

Puisque notre fonction $\lambda()$ est une fonction récursive terminal, on peut appliquer l'algorithme vu en cours et éliminer totalement la récursivité.

6.5 Etape 4 : Fonction λ'(char *hexa, int n, int power, int addition)

Elimination de la récursivité dans la fonction $\lambda'()$.

```
λ‘(char *hexa, int n, int power, int addition):
   m <- n
   p <- power
   a <- addition

   until m = 0 do
      a <- a + convert(hexa[m - 1]) × p
      p <- p × 16
      m <- m - 1

   end
   r <- a + p × convert(hexa[m-1])
```

6.6 Etape 5 : Fonction hexa_dec_rec"(char *hexa, int n, int power, int addition)

Construction de la fonction non récursive Fonction hexa_dec_rec"().

```
hexa_dec_rec"(char *hexa, int n, int power, int addition):
   m <- n
   p <- 0
   a <- 1

   until m = 0 do
      a <- a + convert(hexa[m - 1]) × p
      p <- p × 16
      m <- m - 1

   end
   r <- a + p × convert(hexa[m-1])
```