Multiplication Russe

ADOLPHE Benjamin-BERJOLA Matthias

1er mai 2020

CHAPITRE 1

Introduction

Dans le cadre de l'UE calculabilité et complexité nous avons dû réaliser plusieurs tâches sur un algorithme comprenant un contrat pré-conditions/post-conditions clair, et incluant au moins une boucle tant-que.Nous avons choisis de travailler sur l'algorithme représentant la multiplication russe et d'effectuer les tâches suivantes sur ce dernier :

- Écrire son code Dafny
- Montrer la correction totale :
  - Montrer la correction partielle : les invariants seront présentées et prouvés, ainsi que les post-conditions.
  - Prouver sa terminaison : fonctions de rang et éventuels invariants seront présentées et prouvés
- Déterminer sa complexité en temps qui dans le pire des cas sera justifiée et possiblement validée expérimentalement

  Nous allons donc exposé nos travaux dans ce rapport en prenant soin de suivre l'ordre décrit ci-dessus

CHAPITRE 2

Code en Dafny et exemple d'utilisation

2.1 Code Dafny

```
method mulitplicationRusse(x:nat,y:nat) returns (m:nat)
ensures m==x*y{
    var a := x;
    var b := y;
    var r := 0;
    while(a>0)
        invariant a>=0
        invariant r+a*b == x*y
        decreases a
    {
        if(a%2 == 0){
            b:=2*b;
            a:=a/2;
        }else{
            r:=r+b;
            a:=a-1;
        }
    }
    m:=r;
}
```

```
Entrées : x et y, deux entiers naturels
Sorties : un entier
a : entier ← x
b : entier ← y
r : entier ← 0
tant que a > 0 faire
    si a% 2 =0 alors
        b ← 2 * b
        a ← a//2
    sinon
        r ← r + b
        a ← a - 1
    fin
fin
retourner r
```

2.2 Code en Python et exemple d'utilisation

```
def MultiRusse(x:int,y:int):
    a=x
    b=y
    r=0
    while(a>0):
        if(a%2 == 0):
            b=2*b
            a=a//2
        else:
            r=r+b
            a=a-1
    return r
print("MultiRusse(2,4)")
print(MultiRusse(2,4))
print("MultiRusse(3,5)")
print(MultiRusse(3,5))
print("MultiRusse(2,2)")
print(MultiRusse(2,2))
```

```
MultiRusse(2,4)
8
MultiRusse(3,5)
15
MultiRusse(2,2)
4
```

CHAPITRE 3

Correction totale

3.1 Correction partielle

Méthode de détermination de la correction partielle

On commence par déterminer l'invariant de boucle en posant les cas de base et de récursivité.

Invariant en (*) $a \geq 0$:

- Cas Inductif : x est affecté à a or x est un entier naturel ainsi par typage $a \geq 0$.
- Cas Récursif : nous supposons que l'invariant $a \geq 0$ est vrai, montrons alors $a' \geq 0$:
  - 1er cas : est pair. Nous savons que $\forall(a, a', b, b', r, r') \in \mathbb{N}^6 \begin{bmatrix} (a \geq 0 \wedge a > 0 \wedge \\ a' = (\frac{a}{2}) \wedge a\%2 = 0 \wedge \\ b' = 2 * b \wedge r' = r) \end{bmatrix}$.
    Ainsi nous avons $a \geq 0 \Leftrightarrow \frac{a}{2} \geq \frac{0}{2} \Leftrightarrow a' \geq 0$.
  - 2ème cas : a est impair. Nous savons que $\forall(a, a', b, b', r, r') \in \mathbb{N}^6$ $\begin{bmatrix} (a \geq 0 \wedge a > 0 \wedge \\ r' = r + b \wedge b' = b \wedge \\ a' = a + 1 \wedge a\%2 = 1 \end{bmatrix}$. Ainsi nous avons $a \geq 0$ d'après le test de boucle or $a \geq 0 \Leftrightarrow a - 1 \geq 0 - 1 \Leftrightarrow a' \geq 0$.

Conslusion

Dans les deux cas, nous avons bien montré que $a' \geq 0$. $a \geq 0$ est donc bien un invariant de boucle en $(*)$.

Invariant $(*) r + a * b = x * y$:

- Cas Inductif : 0 est affecté à r, x est affecté à a et y est affecté à b ainsi $r + a * b = 0 + a * b = x * y$. On a donc bien $r + a * b = x * y$.
- Cas Récursif : nous supposons que l'invariant $r + a * b = x * y$ est vrai, montrons alors que $r' + a' * b' = x' * y'$ est vrai :

- 1er cas : a est pair. Nous savons que $\forall(a, a', b, b', r, r', x, x', y, y') \in \mathbb{N}^{10}$
$\left[\begin{array}{c} r + a * b = x * y \wedge a > 0 \wedge \\ a\%2 = 0 \wedge a' = \frac{a}{2} \wedge r' = r \wedge \\ b' = 2 * b \wedge x' = x \wedge y' = y \end{array}\right]$.
Ainsi nous avons $r' + a' * b' = r' + \frac{a}{2} * 2 * b = r + a * b$ or d'après invariant $r + a * b = x * y$ et $x * y = x' * y'$. Nous avons donc bien $r' + a' * b' = x' * y'$.
- 2ème cas : a est impair. Nous savons que $\forall(a, a', b, b', r, r', x, x', y, y') \in \mathbb{N}^{10}$
$\left[\begin{array}{c} r + a * b = x * y \wedge a > 0 \wedge \\ a\%2 = 1 \wedge a' = a - 1 \wedge b' = b \wedge \\ r' = r + b \wedge b' = b \wedge x' = x \wedge y' = y \end{array}\right]$.
Ainsi nous avons $r' + a' * b' = r + b + (a - 1) * b = r + a * b + b - b = r + a * b$ or d'après l'invariant nous avons $r + a * b = x * y$et $x * y = x' * y'$. Nous avons donc bien $r' + a' * b' = x' * y'$.

Conclusion

Dans les deux cas, nous avons montré que $r' + a' * b' = x' * y'$. $r + a * b = x * y$ est donc un invariant de boucle en $(*)$.

3.2 Terminaison

Méthode de détermination de la terminaison

On détermine une fonction de rang et on prouve que celle-ci est valide. C'est-à-dire, on vérifie que la fonction de rang $\in \mathbb{R}$ au point T1 et qu'elle décroit lorsque l'exécution passe entre les points T1 et T2.

Fonction de rang à valeur dans $\mathbb{N}$:

— À chaque passage au point T1, montrons que $a \in \mathbb{N}$. D'après l'invariant de boucle $a \geq 0$ nous savons donc que $a \in \mathbb{N}$.

— À chaque fois que l'exécution passe entre le point T1 et le point T2, montrons que a décroit strictement.

- 1er cas : a est pair.Nous savons que $\forall(a, a', b, b', r, r', x, x', y, y') \in \mathbb{N}^{10}$
$\left[\begin{array}{c} r + a * b = x * y \wedge a > 0 \wedge \\ a\%2 = 0 \wedge a' = \frac{a}{2} \wedge r' = r \wedge \\ b' = 2 * b \wedge x' = x \wedge y' = y \end{array}\right]$. Ainsi nous avons $a' = \frac{a}{2}$ donc $a > \frac{a}{2} \Leftrightarrow a > a'$.
Nous avons bien a strictement décroissante.
- 2ème cas : a est impair. Nous savons que $\forall(a, a', b, b', r, r', x, x', y, y') \in \mathbb{N}^{10}$
$\left[\begin{array}{c} r + a * b = x * y \wedge a > 0 \wedge \\ a\%2 = 1 \wedge a' = a - 1 \wedge b' = b \wedge \\ r' = r + b \wedge b' = b \wedge x' = x \wedge y' = y \end{array}\right]$. Ainsi nous avons $a' = a - 1$ donc $a > a - 1 \Leftrightarrow a > a'$. Nous avons bien a strictement décroissante.

Conclusion

Dans les deux cas, nous avons montré que a est strictement décroissante. a est donc une fonction de rang à valeur dans $\mathbb{N}$.

CHAPITRE 4

Complexité

4.1 Détermination de la complexité

Méthode de détermination de la compléxité

Afin de calculer la compléxité de cet Algorithme, nous allons utiliser les propriétés 1,2,3 et 5 du cours de compléxité.

Coup en opération élémentaire

```
def MultiRusse(x:int,y:int):
    a=x  ------------------------> 1
    b=y  ------------------------> 1
    r=0  ------------------------> 1
    while(a>0): ------------------> val init Fonction Rang * T(Corps) = 3x
                ------------------> 1 + max (if,else) = 1 + 2 =3
        if(a%2 == 0):
            b=2*b    ----------> 1
            a=a//2   ----------> 1
        else:
            r=r+b    ----------> 1
            a=a-1    ----------> 1
    return r ---------------------> 1
```

On a donc un cout élémentaire de $1+1+1+1+3*x = 3x+4 = O(x)$. En approfondissant on se rend compte que c'est aussi un $\theta(ln(x))$.
On a donc une compléxité d'ordre de grandeur de $\theta(ln(x))$

4.2 Valeur théorique

n	5	10	20	50	250	1 000	10 000	1 000 000
1	10 ns	10 ns	10 ns	10 ns	10 ns	10 ns	10 ns	10 ns
$\log(n)$	10 ns	10 ns	10 ns	20 ns	30 ns	30 ns	40 ns	60 ns
n	50 ns	100 ns	200 ns	500 ns	2.5 μs	10 μs	100 μs	10 ms
$n\ log(n)$	50 ns	100 ns	200 ns	501 ns	2.5 μs	10 μs	100,5 μs	10 050 μs
n^2	250 ns	1 μs	4 μs	25 μs	625 μs	10 ms	1 s	2.8 heures
n^3	1.25 μs	10 μs	80 ms	1.25 ms	156 ms	10 s	2.7 heures	316 ans
2^n	320 ns	10 μs	10 ms	130 jours	10^{59} ans	...	...	...
$n!$	1.2 μs	36 ms	770 ans	10^{48} ans	...	...	...	...