

Spécification et vérification de programmes

Paul Patault

ENS Paris-Saclay

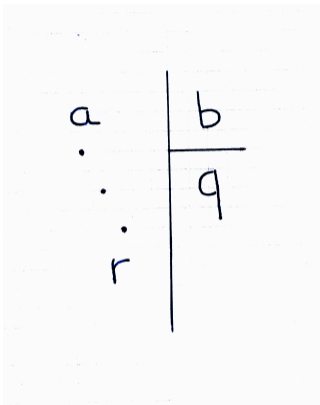


CONGRÈS JUNIOR
PLURIDISCIPLINAIRE



Laboratoire
Méthodes
Formelles

Division euclidienne



```
def division(a, b):  
    """division euclidienne"""  
    q, r = 0, a  
    while b <= r:  
        r = r - b  
        q = q + 1  
    return q, r
```

Comment savoir si ce programme est **correct** ?

Tests

```
def tests():  
    assert division(5, 2) == (2, 1)  
    assert division(12, 6) == (2, 0)  
    assert division(34, 8) == (4, 2)  
    :
```

Spécification formelle

plusieurs choses à définir :

- ◇ ce que le programme calcule
- ◇ les hypothèses implicites

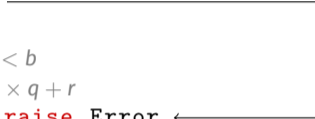
```
def division(a, b):  
    #@ requires  $b > 0$   
    #@ returns  $(q, r)$   
    #@ ensures  $0 \leq r < b$   
    #@ ensures  $a = b \times q + r$   
    ...
```

Vérification durant l'exécution

```
def division(a, b):  
    #@ requires  $b > 0$   
    #@ returns  $(q, r)$   
    #@ ensures  $0 \leq r < b$   
    #@ ensures  $a = b \times q + r$   
  
    q, r = 0, a  
    while b <= r:  
        r = r - b  
        q = q + 1  
  
    return q, r
```

Vérification durant l'exécution

```
def division(a, b):  
    #@ requires b > 0  
    #@ returns (q, r)  
    #@ ensures 0 ≤ r < b  
    #@ ensures a = b × q + r  
    if not (b > 0): raise Error  
    q, r = 0, a  
    while b <= r:  
        r = r - b  
        q = q + 1  
  
    return q, r
```



Vérification durant l'exécution

```
def division(a, b):  
    #@ requires b > 0  
    #@ returns (q, r)  
    #@ ensures 0 ≤ r < b  
    #@ ensures a = b × q + r  
    if not (b > 0): raise Error  
    q, r = 0, a  
    while b <= r:  
        r = r - b  
        q = q + 1  
    if not (0 <= r < b): raise Error  
  
    return q, r
```

Vérification durant l'exécution

```
def division(a, b):  
    #@ requires  $b > 0$   
    #@ returns  $(q, r)$   
    #@ ensures  $0 \leq r < b$   
    #@ ensures  $a = b \times q + r$   
    if not (b > 0): raise Error  
    q, r = 0, a  
    while b <= r:  
        r = r - b  
        q = q + 1  
    if not (0 <= r < b): raise Error  
    if not (a == b * q + r): raise Error  
    return q, r
```


Preuve mathématique

« Pour tous entiers a et b , avec $b > 0$,
si `division(a, b)` renvoie une paire (q, r) ,
alors $a = b \times q + r$ et $0 \leq r < b$. »

Preuve assistée par ordinateur (Why3)

The screenshot displays the Why3 Interactive Proof Session interface. On the left, a project tree shows the structure of the 'division.miw' project, including a 'Division' module and its verification conditions (VCs). The right pane shows the source code for the 'division' module, which defines a function 'division' that performs Euclidean division. The code includes comments, imports, and a while loop with an invariant and a variant.

```
1
2 (** Euclidean division *)
3
4 module Division
5
6   use int.Int
7   use ref.Refint
8
9   let division (a b: int) : int
10    requires { 0 <= a && 0 < b }
11    ensures { exists r. result * b + r = a && 0 <= r < b }
12  =
13    let ref q = 0 in
14    let ref r = a in
15    while r >= b do
16      invariant { q * b + r = a && 0 <= r }
17      variant { r }
18      incr q;
19      r -= b
20    done;
21    q
22
23 end
24
```

At the bottom of the interface, there is a command prompt area with the text '0/0/0 type commands here' and a 'Messages' tab.

Je vous remercie pour votre attention.