Algorithmique & programmation

Chapitre 2 : Raisonnement par récurrence
Présentation

Le raisonnement par récurrence

- On souhaite écrire une fonction qui calcule la valeur de n^p (p : entier positif ; n : entier)
- On sait :
  - $\mathbf{n^1 = n}$
    - ➔ **Base**

  si on connaît n^i, on peut calculer n^{i+1}
  - $\mathbf{n^{i+1} = n^i \times n, \quad \forall i \in [1 \ldots p\text{-}1]}$
    - ➔ **Équation de récurrence**

Le raisonnement par récurrence

- L'idée
  - On part de d'hypothèse que l'on a déjà résolu une partie du problème
- La question
  - A-t-on résolu le problème complètement ou non ?
    - Si on a résolu le problème, c'est fini !
    - Si on n'a pas résolu le problème alors on continue !

Calcul de n^p

- Hypothèse
  - $\mathbf{r = n^i, i \leq p}$
- 2 possibilités
  - $\mathbf{i = p} \Rightarrow r = n^p$ et le **calcul est terminé**
  - $\mathbf{i < p}$ alors
    - pour se rapprocher de la solution
      - $r := r \times n \,; \Rightarrow r = n^i \times n = n^{i+1}$
      - si on incrémente i de 1 ($i := i + 1$), on obtient de nouveau $\mathbf{r = n^i, i \leq p}$
      - **il faut recommencer**

Calcul de n^p

- On peut donc écrire l'itération suivante :

$\boxed{\{r = n^i,\ i \leq p\}}$
tantque $i < p$ ***faire***
  $r := r \times n\ ;\ \{r = n^{i+1},\ i < p\}$
  $i := i + 1\ ;\ \boxed{\{r = n^i,\ i \leq p\}}$
finfaire

Invariant de l'itération

- Il faut une **initialisation** (valeurs de i et de r) qui permette de vérifier l'hypothèse (l'invariant)

$r := n\ ;\ i := 1\ ;\ \{r = n^i,\ i \leq p\}$

- **L'hypothèse devient un invariant après l'initialisation**

Calcul de n^p

- On a l'itération suivante :

tantque $i < p$ ***faire***
  …
finfaire

- À la sortie de l'itération on sait :
  - $\neg (i < p) \Leftrightarrow i \geq p$ (condition de sortie du ***tantque***)
  - $r = n^i,\ i \leq p$ (invariant)
- Alors
  - d'une part : $i \geq p$ et $i \leq p \Rightarrow i = p$
  - d'autre part : $i = p$ et $r = n^i \Rightarrow \boxed{r = n^p}$

Résultat attendu

Calcul de n^p

- En résumé

Hypothèse $r = n^i$, $i \leq p$

Algo terminé

- $i = p \Rightarrow r = n^p \Rightarrow$ résultat = r ✳
- $i < p \Rightarrow r := r \times n$; $i := i + 1$; ➡ ***H***

Itération tantque $i < p$ faire …

Initialisation $r := n$; $i := 1$; ➡ ***H***

fonction **puissance** (algorithme)

fonction puissance (d n , p : entier) : entier ;

spécification *{ p > 0 }* ➔ *{ résultat = np }*

r, i : entier ;

debfonc

Initialisation du raisonnement

r := n ; i := 1 ;

$\{ r = n^i , i \leq p \}$

itération du raisonnement

tantque i < p **faire**

r := **r** × n ; $\{ r = n^{i+1}, i < p \}$

i := i + 1 ; $\{ r = n^i, i \leq p \}$

finfaire ;

cas retournant à l'hypothèse du raisonnement

$\{ i \geq p , r = n^i, i \leq p \Rightarrow r = n^p \}$

retour r ;

finfonc ;

produire le résultat

fonction **puissance** (ada)

```
function puissance (n, p : in integer) return integer is
--spécification { p > 0 } ➔ { résultat = np }
   r, i : integer;
begin
  r := n;
  i := 1;
  -- r = n^i , i ≤ p
  while i < p loop
    r := r * n;     -- r = n^(i+1), i < p
    i := i + 1;      -- r = n^i, i ≤ p
  end loop;
  --i ≥ p , r = n^i, i ≤ p ⇒ r = n^p
  return r;
end puissance;
```


Rappel

- L'hypothèse …
  - dit où en est l'algorithme dans le calcul du résultat à atteindre
- Exemples
  - calcul de n^p
    - *« j'ai calculé n^i et $i \leq p$ »*
    - ***Hypothèse*** $r = n^i$, $i \leq p$
  - calcul de la somme des n premiers entiers positifs
    - *« j'ai calculé la somme des i premiers entiers et $i \leq n$ »*
    - ***Hypothèse*** $r = \sum_{j=1}^{i} j$, $i \leq n$

Conclusion

- Le raisonnement par récurrence permet :
  - de construire l'itération
  - de dégager l'invariant
  - de trouver les initialisations nécessaires
  - de prouver que la valeur finale du résultat est bien celle que l'on cherche
    - en combinant la **condition de sortie** de l'itération avec l'**invariant**