Programmation et algorithmique

IN101
ENSTA Paris - TC 1ère année

François Pessaux

U2IS

2022-2023

prenom.nom@ensta-paristech.fr

Prouver un algorithme (un programme ?)

Vu tout au long du cours...

- Modéliser le problème à résoudre.
- Décomposer en sous-problèmes.
- Élaborer un algorithme indépendant d'un langage de programmation.
- Rédiger dans un langage de programmation.
- Tester l'implantation.

$\Rightarrow$ But : obtenir des programmes qui « *marchent bien* ».

Des programmes qui « *marchent bien* »

- Nécessité de décrire ce qu'est « *bien marcher* ».
- Dépend de l'algorithme / du programme considéré.
- Très souvent description, en langage naturel, approximatif, ambigu.
- Vérification par des tests :
  - Test échoué : bug trouvé.
  - Test passé : pas de bug ... sur ce cas.

  « *Testing shows the presence, not the absence of bugs* » (Dijkstra).
- Besoin de méthodes rigoureuses et mathématiques : formelles.

Propriétés

- « *Prouver un programme* » : aucun sens dans l'absolu.
- Prouver que l'exécution vérifie certaines propriétés.

- Identifier les (des) propriétés pertinentes (spécification).
- Les énoncer dans un langage mathématique.
- Appliquer des outils relevant de preuves mathématiques.
- Démontrer que l'algorithme est correct vis-à-vis de la spécification.

$\Rightarrow$ Confiance accrue en un résultat mathématiquement fondé.

⚠ Algorithme $\approx$ implantation !

Langage formel

- Raisonner mathématiquement ⇒ spécification en langage formel.
- Besoin d'un formalisme logique.
- Dépend de la forme des propriétés à exprimer (et démontrer).
- Dans cette introduction ≈ logique du 1er ordre :

$\wedge$	Conjonction	$\vee$	Disjonction
$\Rightarrow$	Implication	$\Leftrightarrow$	Équivalence
$\neg$	Négation	$\forall$	Quantification universelle
$\exists$	Quantification existentielle		

Rappel : preuve par récurrence

- S'applique à des propriétés sur les entiers naturels.
- Se généralise à toute structure récursive (induction).

  $$\forall P, (P(0) \wedge \forall n \in \mathbb{N}, P(n) \Rightarrow P(n+1)) \Rightarrow \forall n \in \mathbb{N}, P(n)$$

- Prouver $\forall n \in \mathbb{N}, P(n)$ est vraie :
  - prouver que $P(0)$ est vraie
  - prouver que, si l'on suppose $P(n)$ vraie pour un n quelconque, alors on peut prouver que $P(n+1)$ est vraie.

Correction partielle / correction totale

- Correction partielle : propriété est prouvée sous réserve que l'algorithme termine.
- Correction totale : prouver en plus la terminaison de l'algorithme.
- Deux manières de présenter un algorithme :
  - Avec boucles et affectations : style impératif.
  - Avec récursion, sans affectations : style fonctionnel.
- Preuves de terminaison différentes si impératif ou fonctionnel.

Algorithmes fonctionnels

Terminaison de fonction récursive

1. Montrer qu'il existe un (ou des) cas d'arrêt.
2. Montrer que chaque appel récursif est effectué avec des arguments strictement plus petits que ceux de l'appel courant et restants dans le domaine de la fonction.

- Besoin d'un ordre bien fondé $<$ (ordre strict de la relation $\leq$).
- $\Rightarrow \nexists$ suite infinie strictement décroissante.
- Exemples :
  - L'ordre $<$ « habituel » sur les entiers naturels.
  - L'ordre lexicographique sur un produit cartésien $X \times Y$:
    - $(x, y) < (x', y')$ ssi $x <_X x' \vee (x = x' \wedge y <_Y y')$.
  - L'ordre « être préfixe strict » sur les chaînes de caractères.
  - L'ordre lexicographique ~~sur les chaînes.~~
    - "ab" $>$ "aab" $>$ "aaab" ... infinie strictement décroissante.

Multiplication

```
mult (x, y) =
  si y = 0 alors retourner 0
  sinon retourner x + mult (x, (y - 1))
```

- Propriété à prouver $P : \forall x, y \in \mathbb{N}, mult(x, y) = x \times y$.
- Remarque :
  - $mult(x, y)$: domaine informatique.
  - $\times$: domaine mathématique.
  - $\Rightarrow$ lien entre syntaxe et sémantique.
- Équation de la fonction $mult(x,y) = \begin{cases} 0 & \text{si } y = 0 \\ x + mult(x, (y-1)) & \text{si } y > 0 \end{cases}$

Multiplication : terminaison

$$mult(x,y) = \begin{cases} 0 & \text{si } y = 0 \\ x + mult(x,(y-1)) & \text{si } y > 0 \end{cases}$$

- Si $y = 0$ alors *mult* termine trivialement.
- Montrons que $y > 0 \Rightarrow (x,(y-1)) < (x,y)$.
  - Considérons l'ordre lexicographique fondé sur celui des entiers. (1)
  - Arguments récursifs (x et $y-1$) restent dans le domaine de *mult*. (2)
  - Nous avons $x = x$ et $y - 1 < y$. (3)
  - CQFD par (1), (2), (3).

$\square$

Multiplication : correction (1)

$$mult(x,y) = \begin{cases} 0 & \text{si } y = 0 \\ x + mult(x,(y-1)) & \text{si } y > 0 \end{cases}$$

$$P : \forall x, y \in \mathbb{N}, mult(x,y) = x \times y$$

- Fonction récursive $\Rightarrow$ preuve par récurrence.
- Deux cas possibles $\Rightarrow$ preuve par cas.
- Cas $y = 0$ (1)
  Prouvons que $mult(x,0) = x \times 0$.
  - Par définition de *mult*, on a $mult(x,0) = 0$ (2)
  - Nous savons que $\forall n \in \mathbb{N}, 0 = n \times 0$ donc $0 = x \times 0$. (3)
  - CQFD par (2), (3). ☑

Multiplication : correction (2)

$$mult(x,y) = \begin{cases} 0 & \text{si } y = 0 \\ x + mult(x,(y-1)) & \text{si } y > 0 \end{cases}$$

$$P : \forall x, y \in \mathbb{N}, mult(x,y) = x \times y$$

- Cas $y > 0$ (1)
  Prouvons que $mult(x,y) = x \times y$.
  - Par définition de *mult*, on a $mult(x,y) = x + mult(x,(y-1))$ (2)
    Donc nous devons prouver $x + mult(x,(y-1)) = x \times y$ (3)
  - Par (1), $y - 1 < y$, donc hypothèse de récurrence applicable. (4)
  - Par hypothèse de récurrence, nous avons $\forall i < y, mult(x,i) = x \times i$. (5)
  - Par (4), (5) nous avons $mult(x,(y-1)) = x \times (y-1)$. (6)
  - Par (3), (6) nous devons prouver $x + x \times (y-1) = x \times y$.
  - Donc que $x \times (1 + y - 1) = x \times y$. ☑

□

Algorithmes impératifs

Correction d'algorithmes impératifs

- Soit P la propriété de correction, c la condition de boucle.
- Correction partielle :
  - Exhiber une propriété I (l'invariant) tel que :
    - ★ I vraie avant le premier passage dans la boucle,
    - ★ (I vraie en début d'itération $\wedge c$) $\Rightarrow$ I vraie en fin d'itération,
    - ★ $(\neg c \wedge I) \Rightarrow P$.
  - $\Rightarrow$ Corollaire : I vraie en sortie de boucle.
- Terminaison :
  - Exhiber une fonction V des variables (le variant) telle que :
    - ★ ≥ 0 tant que l'on rentre dans la boucle,
    - ★ qui décroît strictement à chaque itération.
  - Terminaison car $\nexists$ suites infinies décroissantes dans $\mathbb{N}$.

Appartenance à un tableau

```
mem (x, t, size) =
  tr := faux
  i := 0
  tant que i < size et non tr
    si t[i] = x alors tr := vrai
    i := i + 1
  retourner tr
```

- Notation : $t[0; i[\; \equiv \; \{t[j]\}$ avec $0 \leq j < i$.
- Prouver la terminaison.
- Propriété de correction à prouver :
  $$P : \forall x, t, \forall size \in \mathbb{N}, mem(x, t, size) \Leftrightarrow x \in t[0; size[$$
  Donc en plus court, $P : \forall x, t, \forall size \in \mathbb{N}, tr \Leftrightarrow x \in t[0; size[$
- Propriété formulée approximativement : pas de domaine pour x ni t.

Appartenance à un tableau : terminaison

```
mem (x, t, size) =
  tr := faux
  i := 0
  tant que i < size et non tr
    si t[i] = x alors tr := vrai
    i := i + 1
  retourner tr
```

- Variant :
  - fonction des variables ≥ 0 tant que l'on reste dans la boucle,
  - décroît strictement à chaque itération.

- Boucle tourne tant que $i < size$ ... tant que $0 < size - i$.
- $\Rightarrow$ $size - i$ bien ≥ 0 à chaque entrée de boucle.
- $size - i$ décroît strictement car $i' = i + 1$ et $size$ constant.

- $\Rightarrow$ $size - i$ est un variant.

Appartenance à un tableau : correction (1)

```
mem (x, t, size) =
  tr := faux
  i := 0
  tant que i < size et non tr
    si t[i] = x alors tr := vrai
    i := i + 1
  retourner tr
```

$$P : \forall x, t, \forall size \in \mathbb{N}, tr \Leftrightarrow x \in t[0; size[$$

- Intuition : *tr* vrai ssi x a été trouvé dans $t[0; i[$ pour un i courant.
- $\Rightarrow$ Invariant $I : tr \Leftrightarrow (x \in t[0; i[\wedge i \leq size)$.
- Remarque : I renforcée (nécessaire pour terminer la preuve).
- Remarque : propriété I « plus locale » que P.

Appartenance à un tableau : correction (1)

```
mem (x, t, size) =
  tr := faux
  i := 0
  tant que i < size et non tr
    si t[i] = x alors tr := vrai
    i := i + 1
  retourner tr
```

$$I : tr \Leftrightarrow (x \in t[0; i[\wedge i \leq size)$$

- Prouvons I avant le premier passage dans la boucle.
  - Prouvons $tr \Rightarrow (x \in t[0; 0[\wedge 0 \leq size)$.
    - Par définition de *mem*, on a $tr = faux$.
    - Donc faux impliquant vrai, CQFD. ☑
  - Prouvons $(x \in t[0; 0[\wedge 0 \leq size) \Rightarrow tr$.
    - x ne peut pas appartenir au tableau vide.
    - Donc faux impliquant vrai, CQFD. ☑

Appartenance à un tableau : correction (2)

```
mem (x, t, size) =
  tr := faux
  i := 0
  tant que i < size et non tr
    si t[i] = x alors tr := vrai
    i := i + 1
  retourner tr
```

$$I : tr \Leftrightarrow (x \in t[0; i[\wedge i \leq size)$$

- Prouvons I à la fin d'une itération sous l'hypothèse que I était vraie en début.
  $tr' = vrai$ si $t[i] = x$ sinon tr (1)
  $i' = i + 1$ (2)
  Par hypothèse de récurrence, $tr \Leftrightarrow (x \in t[0; i[\wedge i \leq size)$ (3)
  Prouvons que $tr' \Leftrightarrow (x \in t[0; i'[\wedge i' \leq size)$.
  - Par (2), il faut prouver que $tr' \Leftrightarrow (x \in t[0; i+1[\wedge i+1 \leq size)$.
  - Donc que $tr' \Leftrightarrow (x \in t[0; i] \wedge i < size)$.
  - Par condition de la boucle, on a $i < size$. (4)
  - Par (4) il reste à prouver $tr' \Leftrightarrow x \in t[0; i]$.
  - Examinons le cas $t[i] = x$.
    - ★ Dans ce cas, par (1) on a $tr' = vrai$ **et** $x \in t[0; i]$ (6)
    - ★ CQFD par (6) (puisque $(A \wedge B) \Rightarrow (A \Leftrightarrow B)$).

Appartenance à un tableau : correction (3)

```
mem (x, t, size) =
  tr := faux
  i := 0
  tant que i < size et non tr
    si t[i] = x alors tr := vrai
    i := i + 1
  retourner tr
```

$$I : tr \Leftrightarrow (x \in t[0; i[\wedge i \leq size)$$

- Prouvons I à la fin d'une itération …
  $tr' = vrai$ si $t[i] = x$ sinon tr (1)
  $i' = i + 1$ (2)
  Par hypothèse de récurrence, $tr \Leftrightarrow (x \in t[0; i[\wedge i \leq size)$ (3)
  Prouvons que $tr' \Leftrightarrow (x \in t[0; i'[\wedge i' \leq size)$.
  - …
  - Par (4) il reste à prouver $tr' \Leftrightarrow x \in t[0; i]$.
  - Examinons le cas $t[i] \neq x$.
    - Par (1), on a $tr' = tr$. (5)
    - Par (3), (5), on a $tr' \Leftrightarrow x \in t[0; i[$.
    - Puisque $t[i] \neq x$ on a $tr' \Leftrightarrow x \in t[0; i]$. ☑ (raccourci « un peu » rapide)

□

Appartenance à un tableau : correction (fin)

- Invariant prouvé : $I : tr \Leftrightarrow (x \in t[0; i[\wedge i \leq size)$.
- Reste à prouver : $P : \forall x, t, \forall size \in \mathbb{N}, tr \Leftrightarrow x \in t[0; size[$.

- À la fin de la boucle : $tr = vrai \vee i \geq size$. (1)
- Deux cas pour tr :
  - $tr = faux$ (2)
    - ★ Par (1), on a $i \geq size$. (3)
    - ★ i incrémenté à chaque tour $\Rightarrow i = size$.
    - ★ Par (3) et I, on a $x \notin t[0; size[$.
  - $tr = vrai$ (4)
    - ★ Par (4) et I, on a $x \in t[0; i[\wedge i \leq size$, donc $x \in t[0; size[$.
- Si pas d'entrée dans la boucle, $size \leq 0$, t vide et $tr = faux$.

□

Les preuves dans la « vraie vie »

- Intérêt : argumentaire de correction solide (bases mathématiques).
- Cadres logiques différents selon la forme des propriétés.
- Domaine de recherche actif (depuis des décennies).
- Automatisation des preuves : impossible dans le cas général.
- Outils : prouveurs automatiques, ateliers d'aide à la preuve.
- Tâche restant (très) souvent compliquée.
- Propriétés non fonctionnelles peuvent être aussi importantes :
  - Coût mémoire.
  - Coût temporel.
  - Consommation énergétique, . . .

Conclusion

- Présentation très informelle.
- Orientée méthodologie de preuve.
- Exemples simples.
- Pas de présentation des concepts théoriques sous-jacents.
- Cadre logique non formalisé ($\approx$ logique du premier ordre).