Validation d'algorithmes
L3 MIAGE, UGA
2020/2021

Raphaël Charrondière
raphael.charrondiere@inria.fr
Semaine 2 - échéance S4

Correction TD 1: Preuves

Où l'on commence à manipuler les notions de variant, d'invariant, d'ordre bien fondé

Exercice 1 Ordre bien fondée

1 Rappelez la définition d'ordre, ordre total, ordre strict et ordre bien fondé.

`Correction :`

- Un ordre est une relation entre deux éléments d'un ensemble. Cette relation doit respecter trois propriétés pour être un ordre :
  - réflexive $\forall x \in E,\ x \leqslant x$
  - antisymétrique $\forall x, y \in E,\ (x \leqslant y \wedge y \leqslant x) \implies x = y$
  - transitive $\forall x, y, z \in E,\ (x \leqslant y \wedge y \leqslant z) \implies x \leqslant z$
- Un ordre est total si tout élément x et y «on peut les comparer» $x \leqslant y \vee y \leqslant x$
- Un ordre total est une relation entre deux éléments d'un ensemble. Cette relation doit respecter deux propriétés pour être un ordre total :
  - asymétrique $\forall x, y \in E,\ (x < y \vee x = y) \implies \neg(y < x)$
  - transitive $\forall x, y, z \in E,\ (x < y \wedge y < z) \implies x < z$
- *Note : la différence entre les deux ordres est juste l'ajout de la propriété $x \leqslant x$ ou son "retrait" $\neg(x < x)$*
- Un ordre est bien fondé s'il n'existe pas de suite infinie strictement décroissante (Un ordre bien fondé est toujours stricte).

2 Prouvez que l'ordre usuel $<$ sur $\mathbb{N}$ est bien fondé

`Correction :` L'ordre usuel $<$ sur $\mathbb{N}$ est strict, vérifions qu'il est bien fondé.

Prenons une suite quelconque strictement décroissante sur $\mathbb{N}$: $(x_i)_i$ avec $x_{i+1} < x_i$. On montre qu'il n'y a pas plus de $x_0 + 1$ termes.

On peut aisément montrer que $x_i \leqslant x_0 - i$ par récurrence. $x_0 \leqslant x_0 - 0$ et si $x_i \leqslant x_0 - i$ alors $x_{i+1} < x_i$ d'où $x_{i+1} \leqslant x_i - 1$ et par hypothèse de récurrence $x_i - 1 \leqslant x_0 - i - 1$ donc $x_{i+1} \leqslant x_0 - (i + 1)$.

Ainsi on a prouvé $x_{x_0+1} \leqslant x_0 - (x_0 + 1) = -1$, donc $x_{x_0+1} < 0$ n'existe pas car n'est pas dans $\mathbb{N}$, la suite termine donc.

Note : Cette démonstration est complexe, pour faire plus simple vous pouvez dire que x_0 termes inférieurs strictement au premier.

Note 2 : Cela est très similaire a ce qu'on avait dit dans le cours pour la relation d'inclusions $\subsetneq$ sur les ensembles finis

3 Soit l'ordre $<_2$ sur $\mathbb{N}$ définit par $x <_2 y \Leftrightarrow \exists t \in \mathbb{N}^*\ y = x + 2t$. Prouvez que $<_2$ est un ordre strict, bien fondé mais qu'il n'est pas total.

`Correction :` On peut facilement dire que si $x <_2 y$ alors $x < y$, donc l'asymétrie est vraie, pour la transitivité c'est un peu plus complexe : soit x, y, z avec $x <_2 y$ et $y <_2 z$ par définition il existe $t \in \mathbb{N}^*$ tel que $y = x + 2t$ et il existe $l \in \mathbb{N}^*$ tel que $z = y + 2l$, on a alors $z = x + 2(t + l)$, comme $t + l \in \mathbb{N}^*$ $x <_2 z$. On a ainsi montré que $<_2$ est un ordre strict.

Il est bien fondé car $<$ l'est aussi : une suite décroissante pour $<_2$ l'est aussi pour $<$.

Par contre l'ordre n'est pas total car on n'a ni $3 < 4$, ni $4 < 3$.

4 Soit la relation $|$ (divise) sur $\mathbb{N}$ définie par $x|y \Leftrightarrow \exists t \in \mathbb{N},\ t > 1,\ y = xt$. Prouvez que $|$ est une relation d'ordre strict bien fondée mais pas totale. *Si la notation $|$ vous gêne, utilisez $|_<$*

`Correction :` $|$ définie ici (attention ce n'est pas la déf. usuelle car où on aurait $t \geqslant 1$) est asymétrique $x|x$ est faux, et si $x|y$ alors $\exists k > 1,\ kx = y$ d'où $y = (1/k)x$ $1/k$ n'est pas un entier naturel donc on n'a pas $y|x$. La transitivité est simple si $x|y$ $y|z$ alors $\exists k_1, k_2 \in \mathbb{N} k_1 > 1, k_2 > 2,\ y = k_1 x,\ z = k_2 y$, ce qui donne $z = k_1 k_2 x$ comme $k_1 k_2 \in \mathbb{N}$ et $k_1 k_2 > 1$ on a $x|z$.

Il est bien fondé car $<$ l'est aussi : une suite décroissante pour $|$ l'est aussi pour $<$.

$|$ n'est pas totale car on n'a ni $3|4$ ni $4|3$.

Raphaël Charrondière
raphael.charrondiere@inria.fr

Exercice 2 Dictionnaires et ordre

D'ordinaire un dictionnaire donne des définitions aux mots, ce qui nous intéresse, c'est que les mots sont classés par ordre lexicographique (On va supposer qu'il s'agisse d'un dictionnaire standard, un petit malin aura toujours le plaisir de classer les mots autrement). Pour nous informaticien, un mot est juste un séquence de caractère, et «zabtrd» est un mot au même titre que «orange»

1 Montrez que l'ordre lexicographique n'est pas bien fondé. *Indice : Il suffit de montrer qu'il existe une séquence décroissante infinie de mots, la taille des mots de cette liste va certainement croitre avec la suite*

`Correction :` En fait on peut donner une exemple de suite infinie décroissante pour montrer que l'ordre lexicographique n'est pas bien fondé, x_i est le mot composé de i fois la lettre a, suivi d'un b. Le début de la suite est $b > ab > aab > aaab$.

2 La relation «est préfixe de» est aussi un ordre, est-il total ? bien fondé ?

`Correction :` Notons «est préfixe de» par $\sqsubseteq$.

$\sqsubseteq$ est un ordre : Un mot est préfixe de lui-même (réflexivité). Si $x \sqsubseteq y \wedge y \sqsubseteq x$ alors x a une longueur inférieur à y et inversement, donc les deux mots ont la même longueur, et comme x est préfixe de y $x = y$ (antisymétrie). Si $x \sqsubseteq y$ et $y \sqsubseteq z$ alors il existe deux sous mots m, n tel que y s'écrit xm et z s'écrit yn, donc z s'écrit xmn, x est bien un préfixe de z (transitivité)

$\sqsubseteq$ est bien fondé : Ça n'a de sens que si l'on considère préfixe au sens strict $x \sqsubset y$ x est préfixe de y est $x \neq y$. Si l'on considère une suite de mot décroissante au sens de $\sqsubset$, alors la longueur des mots entre deux termes décroit strictement. Donc on se ramène à l'ordre usuel $<$ sur $\mathbb{N}$.

$\sqsubseteq$ n'est pas totale car a n'est pas préfixe de b et b n'est pas préfixe de a.

3 On pourrait remplacer les lettres par des entiers, cela ne changerait pas grand chose, si ce n'est qu'il y a une infinité d'entier. Reprendre les questions précédentes.

`Correction :` En fait si on remplace juste la lettre a par 1 et b par 2 dans les exemples cela ne change rien dans la rédaction des réponses.

4 Fixons maintenant une limite de taille pour les mots, est-ce que les ordres précédents sont toujours des ordres ? bien fondés ? *Vous pouvez dire que les mots ont tous la même taille si on les complète avec un nouveau symbole ou -1*

`Correction :` Le seul endroit où la longueur des mots intervient est que la relation $<$ (ordre lexicographique strict) n'est pas bien fondée. Sur un alphabet : avec le petit indice, si l'alphabet compte n lettres, que l'on borne la taille des mots par k, il y a moins que $(n+1)^k$ mots possibles, donc toute suite strictement décroissante ne pourra pas avoir plus de $(n+1)^k$ termes.

En utilisant un nombre infini de symboles, c.-à-d. des entiers, c'est plus compliqué à montrer. Pour les mots de taille 1 cela revient à montrer que $<$ est une relation d'ordre stricte, ce que l'on sait déjà. On va prouver par récurrence que la relation est bien fondée. On suppose que tout les mots ont une taille fixe comme suggéré dans l'énoncé.

Si la relation $<$ est bien fondée pour les mots de taille k (C'est notre hypothèse de récursion). Supposons qu'il existe une suite infinie décroissante quelconque $(m_i)_i$ de mots de taille $k+1$. La première lettre des termes ne peut que décroitre par définition d'ordre lexicographique, vu que sur $\mathbb{N}$ $<$ est bien fondée, il y a un terme à partir duquel la première lettre est toujours la même. Du coup, on supprime les premier termes de la suite pour que la première lettres des mots de la suite soit toujours identique, cette suite est encore infinie. Par définition d'ordre lexicographique, on peut supprimer la première lettre de chaque terme de cette nouvelle suite en gardant l'ordre lexicographique strictement décroissant. Cette nouvelle suite $(n_i)_i$ est alors une suite de mots de taille k strictement décroissante pour l'ordre lexicographique et infinie. Or l'hypothèse de récursion nous indique que l'ordre lexicographique sur les mots de taille k est bien fondée, donc l'existence de $(n_i)_i$ est impossible, c'est donc que notre supposition est fausse : il n'existe pas de suite

Raphaël Charrondière
raphael.charrondiere@inria.fr

infinie décroissante de mot de taille $k+1$ pour la relation lexicographique.

```
   Entrées : a, b ∈ ℕ
   Output : pgcd(a, b)
1  tant que a ≠ b faire
2  |  si a > b alors
3  |  |  a ← a − b;
4  |  sinon
5  |  |  b ← b − a;
6  |  fin
7  fin
8  retourner a
```

Algorithme 1 : L'algorithme d'Euclide

```
   Entrées : a, b ∈ ℕ
   Output : a × b
1  r ← 0;
2  tant que a > 0 faire
3  |  si a = 0 mod 2 alors
4  |  |  a ← a/2;
5  |  sinon
6  |  |  r ← r + b;
7  |  |  a ← (a − 1)/2;
8  |  fin
9  |  b ← 2b;
10 fin
11 retourner r
```

Algorithme 2 : Multiplication rapide

Exercice 3 L'histoire d'Euclide

On a vu en cours qu'un invariant devait être vérifié avant d'exécuter chaque bloc d'instructions (libre à nous de définir ces blocs), et qu'il ne portait pas forcément sur tout le code mais seulement un bloc, pas exemple un invariant pour l'algo 2 ne considérera pas la première ligne.

On rappelle aussi qu'un invariant est de la forme $P(\mathcal{M}_0, \mathcal{M})$ avec $\mathcal{M}_0$ l'état mémoire initial. P doit être vérifié au début de chaque bloc si $P(\mathcal{M}_0, \mathcal{M}_0)$ est vrai, c'est-à-dire si l'invariant est vrai en entrée.

1 L'algorithme d'Euclide permet de calculer le PGCD de deux entiers. On rappelle que le PGCD (plus grand diviseur commun) de deux nombres strictement positifs a et b est le plus grand entier qui divise a et b. La découpe du code en bloc, est ici très facile : on considère le sous-bloc lignes 2 à 6 uniquement. Il faut alors trouver l'invariant qui permet de prouver que le résultat retourné par l'algorithme est bien le PGCD des nombres donnés en entrée.

`Correction :` L'algorithme ne change pas le PGCD, l'invariant est alors simplement $P(\mathcal{M}_0, \mathcal{M}) \equiv \text{PGCD}(\mathcal{M}_0(a), \mathcal{M}_0(b)) = \text{PGCD}(\mathcal{M}(a), \mathcal{M}(b))$.

On doit bien **justifier** : on note alors $\mathcal{M}_0$ l'état mémoire en entrée de fonction, $\mathcal{M}_i$ l'état mémoire en début d'une itération de boucle et $\mathcal{M}_f$ l'état mémoire à la fin de cette itération.

Rappel : on note $a_x = \mathcal{M}_x(a)$ pour simplifier les notations. Ainsi on veut montrer $\text{PGCD}(a_i, b_i) = \text{PGCD}(a_f, b_f)$.

Tout d'abord la condition $a \neq b$ indique qu'on ne considère que le cas $a_i \neq b_i$. On peut se concentrer sur le cas $a_i > b_i$ et par symétrie on aura le cas $a_i < b_i$. On note $P_i = \text{PGCD}(a_i, b_i)$ et $P_f = \text{PGCD}(a_f, b_f)$

- on a $a_f = a_i - b_i$ et $b_f = b_i$
- P_i divise a_i et b_i donc P divise la différence $a_f = a_i - b_i$, donc P_i divise P_f.
- P_f divise a_f et b_f donc P_f divise la somme $a_f + b_f = a_i$ donc P_f divise P_i.
- Ainsi $P_f = P_i$

2 *(Cours)* Un invariant de boucle permet-il de prouver la terminaison d'une boucle ? Si oui, justifier, si non qu'est-ce qui permet de prouver la terminaison d'une boucle ?

`Correction :` Non, un invariant ne permet pas de prouver la terminaison, il faut un variant de boucle.

3 Prouver alors la terminaison de l'algorithme.

`Correction :` On définit le variant $\phi(\mathcal{M}) = \mathcal{M}(a) + \mathcal{M}(b)$. On doit juste vérifier qu'à chaque itération le variant décroit strictement. On défini alors $\mathcal{M}_i$ l'état mémoire en début d'une itération de boucle et $\mathcal{M}_f$ l'état mémoire à la fin de cette itération.

Raphaël Charrondière
raphael.charrondiere@inria.fr

Comme avant il y a deux cas symétriques, on ne considère que le cas $a_i > b_i$. On a $a_f + b_f = a_i - b_i + b_i = a_i$. Là l'œil affuté doit s'arrêter et remarquer qu'il y a un problème. On a $\phi(\mathcal{M}_i) > \phi(\mathcal{M}_f)$ a condition d'avoir $b_i > 0$. On constate alors que si b (ou a) est nul en entrée l'algorithme est une boucle infinie. *L'aviez-vous remarqué ? Le correcteur vous affirme qu'en écrivant l'algorithme, il n'avait pas fait attention à son erreur, mais il la détecte facilement grâce à cette technique.*

Il ne reste a démontrer que a et b sont toujours strictement positifs. Il faut corriger l'entrée de l'algorithme en $a, b \in \mathbb{N}^*$. Maintenant à chaque itération : en supposant $a_i > 0$, $b_i > 0$

— Si $a_i > b_i$ $b_f = b_i > 0$, alors $a_f = a_i - b_i > 0$.
— Idem si $b_i > a_i$
— On rappelle qu'on n'a pas le cas $a_i = b_i$ car c'est la condition de fin de boucle.

Exercice 4 Un autre exemple simple

On continue avec un exercice très similaire avec la multiplication rapide qui calcule le produit de deux entiers en utilisant que des additions, soustractions, multiplications par 2, divisions par 2.

On pourrait discuter de l'intérêt de cet algorithme, ce n'est pas vraiment le sujet qui nous intéresse, mais en base 2, une multiplication par 2 est l'ajout d'un bit nul à droite (bit de poids faible), et la division entière est le retrait du bit à droite.

(1) Prouver que l'algorithme 2 renvoie la bonne valeur.

`Correction :` On défini l'invariant suivant $P(\mathcal{M}_0, \mathcal{M}) \equiv \mathcal{M}_0(a) \times \mathcal{M}_0(b) = \mathcal{M}(r) + \mathcal{M}(a) \times \mathcal{M}(b)$. On considère le bloc ligne 3 à 9. On doit prouver : (1) si en entrée de bloc, on a l'état mémoire $\mathcal{M}_i$, en fin de bloc $\mathcal{M}_f$, $P(\mathcal{M}_0, \mathcal{M}_i) \implies P(\mathcal{M}_0, \mathcal{M}_f)$, on dispose de l'hypothèse $\mathcal{M}_i(a) > 0$. Il faut aussi prouver que (2) $P(\mathcal{M}_0, \mathcal{M}_0)$ est vrai, où $\mathcal{M}_0$ désigne l'état mémoire juste avant l'entrée dans la boucle.

La preuve de (2) est assez immédiate car $\mathcal{M}_0(r) = 0$.

Prouvons (1) : Par hypothèse $a_0 \times b_0 = r_i + a_i \times b_i$ il y a deux cas

— $a_i \equiv 0 \mod 2$ (a_i est pair) On a $a_f = a_i/2$, $b_f = 2b_i$, $r_f = r_i$. Donc $r_f + a_f \times b_f = r_i + a_i/2 \times 2b_i = r_i + a_i \times b_i$, par hypothèse $r_f + a_f \times b_f = a_0 \times b_0$
— $a_i \equiv 1 \mod 2$ (a_i est impair) On a $a_f = (a_i - 1)/2$, $b_f = 2b_i$, $r_f = r_i + b_i$. Donc $r_f + a_f \times b_f = r_i + b_i + (a_i - 1)/2 \times 2b_i = r_i + a_i \times b_i$, par hypothèse $r_f + a_f \times b_f = a_0 \times b_0$

Notons $\mathcal{M}_e$ l'état mémoire en fin de boucle, nous venons de prouver $P(\mathcal{M}_0, \mathcal{M}_e)$, la sortie de boucle indique $a_e = 0$, c'est-à-dire $r_e = a_0 \times b_0$, ce que conclue la preuve de la correction.

(2) Prouver que l'algorithme 2 termine.

`Correction :` Il suffit de trouver un variant de boucle. Tout simplement $\phi(\mathcal{M}) = \mathcal{M}(a)$. Dans une itération a_i n'est pas nul, $a_f = a_i/2 < a_i$ ou $a_f = (a_i - 1)/2 < a_i$.