Validation d'algorithmes
L3 MIAGE, UGA
2020/2021

Raphaël Charrondière
raphael.charrondiere@inria.fr
Semaine 4 - échéance S5

Correction TD 2,5: Des exemples de variants et d'invariants

Où il faut maitriser le chapitre 1 et 2.

Exercice 1 Un algorithme, des variants

```
Entrées : a, b ∈ ℕ
1 tant que a > 0 ∧ b > 0 faire
2 |  r ← BooleanAleatoire();
3 |  si r alors
4 |  |  a ← a − 1;
5 |  sinon
6 |  |  b ← b − 1;
7 |  fin
8 fin
9 retourner a + b
```

Algorithme 1 : Course de billes

1 *(Cours)* Comment montre-t-on qu'une boucle termine ?

`Correction :` Il faut utiliser un variant. On se place dans un espace E muni d'une relation d'ordre bien fondée. Puis on définit une fonction $\varphi : \mathbb{M} \to E$ et on montre que si on a l'état mémoire m_0 au début d'une itération de boucle et m_1 à la fin de cette itération, alors $\varphi(m_0) > \varphi(m_1)$

2 Donner un variant de la forme $\varphi : \mathbb{M} \to \mathbb{N}$ pour l'algorithme 1 et montrer que c'est bien un variant.

`Correction :` On propose $\varphi(m) = m(a) + m(b)$. Si on a l'état mémoire m_0 au début d'une itération de boucle et m_1 à la fin de cette itération, alors soit r est vrai et $m_1(a) = m_0(a) - 1$, $m_1(b) = m_0(b)$, soit r est faux et $m_1(a) = m_0(a)$, $m_1(b) = m_0(b) - 1$. Dans tout les cas $\varphi(m_1) = m_1(a) + m_1(b) = m_0(a) + m_0(b) - 1 = \varphi(m_0) - 1$, donc $\varphi(m_1) < \varphi(m_0)$

3 On défini l'ordre $<_\times$ sur les paires d'entiers par $\forall a_1, a_2, b_1, b_2 \in \mathbb{N},\ (a_1, a_2) <_\times (b_1, b_2) \Leftrightarrow a_1 \leqslant b_1 \wedge a_2 \leqslant b_2 \wedge (a_1 \neq b_1 \vee a_2 \neq b_2)$. Montrez que $<_\times$ est un ordre strict bien fondé.

`Correction :`

- $<_\times$ est un ordre strict
  - Soit $a_1, a_2, b_1, b_2 \in \mathbb{N}$ avec $(a_1, a_2) = (b_1, b_2)$ alors on n'a pas $(a_1, a_2) <_\times (b_1, b_2)$ puisqu'on n'a pas $(a_1 \neq b_1 \vee a_2 \neq b_2)$.
  - Soit $a_1, a_2, b_1, b_2 \in \mathbb{N}$ avec $(a_1, a_2) <_\times (b_1, b_2)$. Si on avait $(b_1, b_2) <_\times (a_1, a_2)$, on aurait $a_1 \leqslant b_1$ et $b_1 \leqslant a_1$ donc $a_1 = b_1$, idem $a_2 = b_2$. Ce qui implique $\neg(a_1 \neq b_1 \vee a_2 \neq b_2)$ ce qui par définition de $<_\times$ n'est pas possible, donc $(b_1, b_2) <_\times (a_1, a_2)$ est faux.
  - Soit $a_1, a_2, b_1, b_2, c_1, c_2 \in \mathbb{N}$ avec $(a_1, a_2) <_\times (b_1, b_2)$ et $(b_1, b_2) <_\times (c_1, c_2)$. Par définition on aura $a_1 \leqslant b_1 \leqslant c_1$ donc $a_1 \leqslant c_1$ et de même $a_2 \leqslant c_2$. En supposant $a_1 = c_1$ on aura $b_1 = a_1$, d'où $a_2 < b_2$, on a alors $a_2 < b_2 \leqslant c_2$ ie $a_2 \neq c_2$. Ce qui se résume en $a_1 \neq c_1 \vee a_2 \neq c_2$
- $<_\times$ est bien fondé. Pour toute suite u strictement décroissante au sens de $<_\times$ alors on peut créer un suite v à valeurs dans $\mathbb{N}$ de même taille que u, t.q $v_i = (u_i)_1 + (u_i)_2$ (on somme les termes des paires) v est strictement décroissante au sens usuel sur $\mathbb{N}$, or la décroissance stricte usuelle sur $\mathbb{N}$ est bien fondée, dont v est finie, alors u aussi.Ce qui prouve que $<_\times$ est un ordre bien fondé.

4 On se place maintenant dans l'ensemble des paires d'entiers muni de l'ordre $<_\times$, montrer que $\psi : m \mapsto (m(a), m(b))$ est bien un variant pour l'algorithme 1.

`Correction :` Si on a l'état mémoire m_0 au début d'une itération de boucle et m_1 à la fin de cette itération, alors

- soit r est vrai et $m_1(a) = m_0(a) - 1$, $m_1(b) = m_0(b)$. $(a - 1, b) <_\times (a, b)$
- soit r est faux et $m_1(a) = m_0(a)$, $m_1(b) = m_0(b) - 1$. $(a, b - 1) <_\times (a, b)$

Dans les deux cas $\psi(m_1) <_\times \psi(m_0)$.¨

Raphaël Charrondière
raphael.charrondiere@inria.fr

Exercice 2 Champ de bataille

On décrit informellement l'algorithme suivant : on a une grille d'entiers naturels ($\mathbb{N}$) de taille finie (c'est-à-dire un tableau 2D). L'algorithme cherche une case où appliquer une règle R, il l'applique et recommence jusqu'à ne plus trouver de case où appliquer la règle. Les voisins d'une case, sont les 4 cases adjacentes par les arrêtes de la case.

1 On propose la règle R suivante : Si la valeur de la case n'est pas zéro et qu'une des cases voisines a une valeur strictement supérieure, on décrémente la valeur de la case. Proposer un variant pour prouver que l'algorithme fini.

`Correction :` On se place dans l'ensemble $\mathbb{N}$ muni de l'ordre strict $<$ usuel. On défini le variant $\varphi(m) =$«la somme de la valeur de toute les cases». À chaque fois que l'on applique la règle R, le variant diminue de 1.

2 On choisi maintenant la règle suivante : si la case A a une valeur v_A et un de ses voisins a une valeur strictement inférieur alors on décrément la valeur de A et on fixe la valeur de ses voisins a $v_A - 1$. Proposer un variant pour montrer que l'algorithme fini.

`Correction :` On considère l'ensemble des multi-ensembles sur $\mathbb{N}$ finis, muni de l'ordre $<_{DM}$ du cours, chapitre 2. On défini le variant $\varphi(m) =$«le multi-ensemble contenant toutes les cases de la grille».

	x	
y	v_A	z
	w	

	v_A-1	
v_A-1	v_A-1	v_A-1
	v_A-1	

On illustre ci contre la grille autour de la case A avant et après l'application de la règle, on note les deux multi-ensembles E_0 et E_1. Puisque la règle R est appliquée, on a $x, y, z, w < v_A$.

$\max\{x, y, z, w, v_A\} = v_A$. On montre que $E_1 <_{DM} E_0 : \max(E_1 \setminus (E_0 \cap E_1)) = \max(v_A - 1) = v_A - 1 < \max(E_1 \setminus (E_0 \cap E_1)) = \max\{x, y, z, w, v_A\} = v_A$

3 Peut-on généraliser ? C'est-à-dire, quelque soit la règle R, l'algorithme finira-t-il ? *Justifiez votre réponse !*

`Correction :` On ne peut pas généraliser. Tout simple voilà un contre-exemple : la règle R est «on incrémente la valeur de la case».

Raphaël Charrondière
raphael.charrondiere@inria.fr

Exercice 3 Graphes et cycles

Dans cet exercice, les graphe sont orientés. On rappelle qu'un graphe orienté $\mathcal{G}$ se définit par $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ un ensemble de nœuds et d'arcs. Les nœuds sont des entiers $\mathcal{N} = [\![1, k]\!]$. Les arcs sont des paires d'entiers $(u, v) \in \mathcal{N} \times \mathcal{N}$. Cherchez la définition de graphes dans n'importe quel encyclopédie/cours pour plus de détails. On propose l'algorithme suivant pour connaitre les nœuds appartenant à un cycle.

```
   Entrées : G un graphe
   Output : La liste des nœuds appartenant à un cycle
 1 N ← l'ensemble des nœuds de G;
 2 A ← l'ensemble des arcs de G;
 3 n ← nombre de nœuds de G;
 4 r ← ∅;
 5 t ← Tableau2D(n,n);
 6 Remplir t avec des 0;
 7 pour i de 1 à n faire
 8 |   pour (u, v) ∈ A faire
 9 |   |   pour n ∈ N faire
10 |   |   |   si t[n, u] == 1 OU n == u alors
11 |   |   |   |   t[n, v] ← 1;
12 |   |   |   fin
13 |   |   fin
14 |   fin
15 fin
16 pour i de 1 à n faire
17 |   si t[n, n] == 1 alors
18 |   |   r ← r ∪ {n};
19 |   fin
20 fin
21 retourner r
```

Algorithme 2 : Détection de cycles

1 À quoi correspond le tableau t ?

`Correction :` Le tableau t enregistre s'il y a un chemin d'un nœud à un autre. On le justifie dans la question suivante.

2 On peut découper le code en trois blocs : ligne 1 à 6 , (B1) ligne 7 à 15 une première boucle et (B2) ligne 16 à 20. On ne va pas rentrer dans les détails des boucles imbriquées ligne 8 à13. De même les variants de boucles ne sont pas très intéressants. Donner un invariant pour la boucle du bloc (B1). *Bien sûr justifiez votre réponse en expliquant clairement. Il n'est pas utile de trop rentrer dans les détails.*

`Correction :` On propose l'invariant $I(m_0, m) \equiv$«Pour toute paire de nœud (u, v), $m(t)[u, v] = 1$ si et seulement si il existe un chemin non vide de u à v de taille $m(i) - 1$» En effet :

Initialisation : en entrant dans la boucle i vaut 1 et t ne contient que des zéros.

Récurrence : on note m_1 l'état mémoire au début d'une itération et m_2 l'état mémoire à la fin de cette itération. On suppose $P(m_0, m_1)$ et on doit démontrer $P(m_0, m_2)$. L'hypothèse est donc $m_1(t)[u, v] = 1$ ssi il existe un chemin non vide de u à v de longueur inférieur à $m_1(i) - 1$.

Dans l'itération pour deux nœuds a et b $t[a, b]$ est vaudra 1 en sortie si : SOIT $t[a, b]$ était déjà à 1, SOIT (a, b) est un arc (cas $n == u$ ligne 10), SOIT si il existe un nœud c tel que $t[a, c] = 1$ et (c, b) est un arc du graphe, autrement dit si il existe un nœud c tel qu'il existe un chemin non vide de a à c de taille inférieur à $i - 1$ et (c, b) est un arc du graphe, autrement dit si il existe un chemin de a à b de taille inférieur à i et supérieur à i. De plus $m_1(i) = m_0(i) + 1$, donc ces trois cas prouvent bien $P(m_0, m_2)$

Raphaël Charrondière
raphael.charrondiere@inria.fr

3 Donner un invariant pour la boucle du bloc (B2). *Cette question doit être très facile si vous avez compris l'algorithme et en particulier le rôle de t.*

`Correction :` L'invariant sera $I(m_0, m)$ =«Pour tout nœud $i < m(n)$, $i \in r$ ssi i appartient à un cycle».

Preuve : Il nous faut une propriété sur t. Avec l'invariant précédent on peut affirmer «Pour toute paire de nœuds (u, v), $t[u, v] = 1$ ssi il existe un chemin non vide de u à v» En effet, tout chemin sans cycle d'un graphe ne peut contenir que $n - 1$ arcs maximum (sinon il passe deux fois par le même nœud), et donc tout cycle ne peut contenir que n arcs.

Cette propriété nous permet d'affirmer qu'un nœud u appartient à un cycle ssi $t[u, u] = 1$

Exercice 4 — Plus longue sous-suite

Soit deux mots A=$a_1a_2 \dots a_k$, B=$b_1b_2 \dots b_m$, une plus longue sous-suite commune (abrégée PLSSC) est un mot C=$c_1 \dots c_n$ tel que C est une sous-suite de A et une sous-suite de B, et pour tout autre sous-suite commune D, C sera plus longue que D.

Une sous-suite de A est le mot A auquel on a retiré des lettres

Par exemple «ananas» et «amnésie» ont pour plus grande sous-suite commune «ans». De manière plus abstraite «abcbdab» et «bdcaba» ont pour PLSSC «bcab» ou «bdab».

On propose l'algorithme suivant :

```
   Entrées : a, b des mots
   Output : Un tableau t
 1 n ←longueur(a);
 2 m ←longueur(b);
 3 t ← Tableau2D(n+1,m+1);
 4 Remplir t avec des 0;
 5 pour i de 1 à n faire
 6 |   pour j de 1 à m faire
 7 |   |   compLettre ← a[i] == b[j];
 8 |   |   t[i, j] = max(t[i − 1, j], t[i, j − 1], t[i − 1, j − 1] + compLettre);
 9 |   fin
10 fin
11 retourner t
```

Algorithme 3 : Longueur PLSSC

1 Prouvez que l'algorithme 3 termine

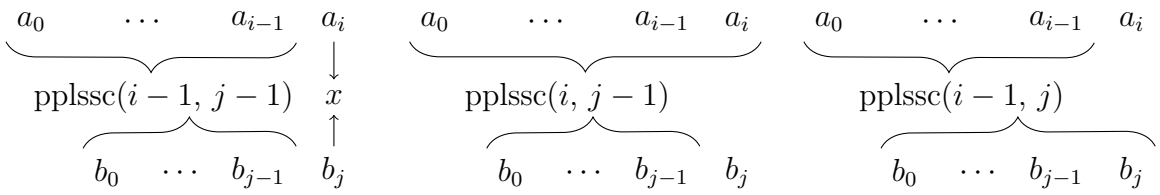
`Correction :` La preuve est assez immédiate : un variant pour la boucle lignes 6-9 est $V(\mathfrak{m}) = \mathfrak{m}(m) - \mathfrak{m}(j) + 1$ et pour la boucle lignes 5-10 $V(\mathfrak{m}) = \mathfrak{m}(n) - \mathfrak{m}(i) + 1$

2 Prouver que dans le tableau renvoyé t[i,j] est la longueur du PLSSC des sous-mots $a_0 \dots a_i$ et $b_0 \dots b_j$

`Correction :` On ne va utiliser qu'un seul invariant pour les deux boucles. On pose l'invariant $I(m_0, m) \equiv$«$\forall (k, l) <_l (m(i), m(j))$, $t[i, j]$ est la longueur de la PLSSC de $a_0 \dots a_i$ et $b_0 \dots b_j$» où $<_l$ est l'ordre lexicographique (I signifie aussi «Pour tout les (i,j) que la boucle a visité, $t[i, j]$ est la longeur de la PLSSC etc.»)

En effet en comparant les deux dernières lettres de $a_0 \dots a_i$ et $b_0 \dots b_j$ soit elles sont égales et appartiennent à une plus longue sous chaine commune, dans ce cas on doit calculer la longueur de la PLSSC de $a_0 \dots a_{i-1}$ et $b_0 \dots b_{j-1}$ et ajouter un, soit a_i ou b_j n'appartient pas à la PLSSC (voire la figure ci-dessous). Comme on prend le maximum de tout ces cas, on calcule bien la longueur de la PLSSC.

Raphaël Charrondière
raphael.charrondiere@inria.fr



Cas 1 Cas 2 Cas 3

On utilise maintenant l'algorithme suivant :

Entrées : a, b et t donnés en entrée et sortie de l'algorithme 3
Output : Le PLSSC de a et b

```
1  n ←longueur(a);
2  m ←longueur(b);
3  k ← t[n, m];
4  w ← Mot de longueur(k);
5  i ← n;
6  j ← m;
7  tant que k > 0 faire
8      si t[i − 1, j] == t[i, j] alors
9          i ← i − 1;
10     sinon si t[i, j − 1] == t[i, j] alors
11         j ← j − 1;
12     sinon
13         w[k] ← a[i];
14         i ← i − 1;
15         j ← j − 1;
16         k ← k − 1;
17     fin
18 fin
19 retourner w
```

Algorithme 4 : Longueur PLSSC

3 Montrer que l'algorithme 4 termine.

`Correction :` C'est ici assez spécial il va falloir utiliser la connaissance sur t pour créer un variant. On pose $\phi(m) = m(i) + m(j)$. Le problème est s'assurer que ϕ est à valeur dans $\mathbb{N}$. On va pour cela utiliser l'invariant $P(m_0, m) \equiv m(k) = m_0(t)[m(i), m(j)]$ pour la boucle l.7-18. $P(m_0, m_0)$ est vrai par initialisation de k. Dans une itération de boucle, on pose m_1 l'état mémoire avant l'itération et m_2 après. Il y a trois cas : soit ligne 8 $t[i_1 - 1, j_1] == t[i_1, j_1]$, alors $k_1 = k_2 = t[i_2, j_2]$, soit ligne 10, similairement, sinon comme la longueur du PLSSC diminue quand i_1 seul et j_2 seul diminue, nous sommes dans le cas 1 de la correction de la question 2, donc, en décrémentant en même temps i et j $t[i_2, j_2] = t[i_1, j_1] - 1$ et $k_2 = k_1 - 1$.

Maintenant, nous savons par définition de t que si $i = 0$ ou $j = 0$ alors $t[i, j] = 0$. Donc dans la boucle si i ou j atteint 0, on aura $k = t[i, j] = 0$, la condition de sortie de boucle s'applique, donc ϕ est bien à valeur dans $\mathbb{N}$. ϕ diminue strictement à chaque itération donc ϕ est bien un variant de boucle.

4 Montrer que l'algorithme 4 renvoie ce qu'il faut, c'est à dire le PLSSC.

`Correction :` Il nous faut un invariant. On propose $I(m_0, m) \equiv$« le mot $w[k+1] \dots w[t[n, m]]$ (peut-être vide) est le PLSSC de $a[i] \dots a[n]$ et $b[j] \dots b[m]$»