Leçon 1 : Exemples de méthodes et outils pour la correction des programmes

Niveau : MP2I Pré-requis : Notion de fonction, Récursivité

Sources : Ellipses MP2I

La conjecture de Syracuse est un problème ouvert de mathématiques :

La suite u définie par : $\begin{cases} u_0 = a \in \mathbb{N}^* \\ u_{n+1} = \begin{cases} \frac{u_n}{2} \text{ si } u_n \text{ est pair} \\ 3u_n + 1 \text{ sinon} \end{cases} \end{cases}$

finie-t-elle toujours par le cycle 1,2,4 ? (toujours = pour tout $a \in \mathbb{N}^*$)

Cela revient alors à savoir si l'algorithme

```
Syracuse (a):
    u = a
    Tant que u est une nouvelle valeur:
        Si u est pair:
            u ← u/2
        Sinon:
            u ← 3u+1
    Renvoyer u
```

Renvoie toujours 1, 2 ou 4.

I - Terminaison

Une première question est de savoir si Syracuse finit (ne boucle pas à l'infini) sur toute entrée.

Définition 1 : Prouver la terminaison d'un algorithme revient à prouver que sur toute entrée il termine.

Remarque 2 : On se limite parfois aux entrées valides (pour Syracuse, $a \in \mathbb{N}^*$ par exemple)

Exemple 3 :

```
Tant que a > 0:
    a = a - 1
```

Termine sur toute entrée si on autorise pas a à valoir $+\infty$, sinon ne termine pas sur toute entrée.

Pour prouver la terminaison, on va utiliser la technique du variant.

Définition 4 : Un variant est une fonction des variables, à valeurs dans $\mathbb{N}$ qui décroit strictement :
- à chaque passage dans la boucle pour les algorithmes itératifs
- à chaque appel récursif pour les algorithmes récursifs.

Exemple 5 : La fonction pgcd(a,b)

```
pgcd(a,b):
    Tant que min(a,b) > 0
        Si a < b:
            b = b - a
        Sinon:
            a = a - b
    Renvoyer max(a,b)
```

qui calcule le pgcd de a et b pour $a, b \in \mathbb{N}$ admet comme variant $a + b$ (qui est en effet toujours positif, et décroit à chaque fois).

Propriété 6 : Si une boucle a un variant, alors elle s'exécute un nombre fini de fois. De même pour un algorithme récursif.

Exemple 7 : pgcd(a,b) termine pour tout $(a,b) \in (\mathbb{N}^*)^2$.

Remarque 8 : La technique du variant suffit toujours, mais le variant peut être difficile à trouver.

Exemple 9 : On définit ack(m,n) pour $m, n \in \mathbb{N}$ par

```
ack(m,n):
    Si m = 0
        Renvoyer n+1
    Sinon si n = 0
        Renvoyer ack(m-1, 1)
    Sinon
        Renvoyer ack(m-1, ack(m, n-1))
```

représentant la fonction définie par $\begin{cases} ack(0,n) = n+1 \\ ack(m,0) = ack(m-1,1) \\ ack(m,n) = ack(m-1, ack(m,n-1)) \end{cases}$

Définition 10: On dit qu'un ordre est un bon ordre si toute suite décroissante est stationnaire.

Propriété 11: L'ordre produit de bons ordres et l'ordre lexicographiques de bons ordres sont des bons ordres.

Définition 12: On étend dans la définition du variant aux fonction à valeurs dans un bon ordre.

Propriété 13: La propriété 6 reste vrai avec notre définition étendue du variant.

Exemple 14: Pour ack, (m, n) est un variant dans $\mathbb{N}^2$ avec l'ordre lexicographique, donc ack termine.

II - Correction partielle

Une autre question pour Syracuse est de savoir si on peut tomber sur un autre cycle que 1,2,4 et donc renvoyer autre chose que 1, 2 ou 4.

Définition 15: On appelle spécification d'un algorithme deux propriétés P_1 sur les entrées (pré-condition) et P_2 sur les sorties (post-condition).

Exemple 16: Pour Syracuse, P_1 : « $a \in \mathbb{N}^*$ » et P_2 : « Syracuse(a) $\in \{1, 2, 4\}$ »

Définition 17: On dit qu'un algorithme est partiellement correct si pour toute entrée vérifiant la pré-condition, si l'algorithme termine, la sortie vérifie la post-condition.

Exemple 17: L'algorithme de l'exemple 5 est partiellement correct si la pré-condition est « $a \in \mathbb{N}, b \in \mathbb{N}$ » et la post-condition « pgcd(a, b) renvoie le PGCD de a et b ».

II-1) Correction partielle des algorithmes impératifs

Pour prouver la correction partielle des langages impératifs on utilise un invariant de boucle.

Définition 18: Un invariant de boucle est une propriété qui est vrai avant la boucle, et si elle est vraie quand on commence un tour de boucle, alors elle l'est quand on la finit.

Propriété 19: Si un invariant de boucle est valide, alors il est vrai après la boucle, et la condition d'arrêt de la boucle elle est fausse.

Exemple 20: Pour pgcd, « $PGCD(a,b) = PGCD(a_0, b_0)$ » où a_0 et b_0 sont les valeurs initiales de a et b, est un invariant de boucle valide. A la fin de l'éxécution, on a donc $\min(a,b) = 0$ et $PGCD(a,b) = PGCD(a_0, b_0) = PGCD(\min(a,b), \max(a,b)) < PGCD(0, \max(a,b)) = \max(a,b)$. D'où l'éxécution de l'exemple 17.

II-2) Correction partielle des algorithmes récursifs

Définition 21: Un ordre bien fondé est un ordre où toutes parties non vides à un élément minimal (plus grand que personne)

Propriété 22: Les ordres produit et lexicographiques d'ordres bien fondés sont bien fondés.

Théorème 23: Soit $(A, \leqslant)$ un ensemble muni d'un ordre bien fondé et P une propriété sur A. Alors $(\forall x \in A, (\forall y \in A, y \leqslant A \Rightarrow P(y)) \Rightarrow P(x)) \Rightarrow \forall x\ P(x)$

Remarque 24: Cela étend le principe de récurrence forte sur $\mathbb{N}$

On utilise cela pour la correction partielle des algorithmes récursifs.

Exemple 25: exp(a, n) pour $a, n \in \mathbb{N}$ renvoie a^n

```
exp(a,n):
    Si n = 0
        Renvoyer 1
    Sinon
        x = exp(a, n/2)
        Si n est pair
            Renvoyer x * x
        Sinon
            Renvoyer a * x * x
```

La propriété $P(n)$: « exp(a,n) = a^n » vérifie les hypothèses du théorème 23. Donc $\forall n$, exp(a,n) = a^n, et ce pour tout $a \in \mathbb{N}$.

Ne rien écrire hors du cadre.

III - Correction

La conjecture de Syracuse dit donc que notre fonction Syracuse termine, et quand elle termine est correcte (donc elle est partiellement correcte).

Définition 26 : Quand un programme termine sur toute entrée valide et est partiellement correcte, on dit qu'il est correcte, ou encore totalement correcte.

Exemple 27 :

```
fusion (l1, l2):
    res = Tableau de taille |l1|+|l2|
    i = 0
    j = 0
    Tant que i < |l1| et j < |l2|
        Si l1[i] ≤ l2[j]
            res[i+j] = l1[i]
            i = i+1
        Sinon
            res[i+j] = l2[j]
            j = j+1.
    Tant que i < |l1|
        res[i+j] = l1[i]
        i = i+1
    Tant que j < |l2|
        res[i+j] = l2[j]
        j = j+1.
    Renvoyer res

Tri_fusion (l):
    Si |l| = 1
        Renvoyer l
    Sinon
        Renvoyer fusion (Tri_fusion(l[|l|/2 :]), Tri_fusion(l[: |l|/2]))
```

Développement 1 : Correction totale de Tri-fusion

Néanmoins ce n'est pas toujours facile. La conjecture de Syracuse est toujours un problème ouvert. Et c'est parfois même pire

Théorème 28 : La correction partielle et la terminaison sont indécidables

Développement 2 : Preuve du théorème 28

IV - Outils

* Typer : Le fait d'utiliser un typage fort comme en OCaml permet d'éviter beaucoup d'erreurs bêtes

* Programmer défensivement, en utilisant la bibliothèque assert. Il permet de vérifier qu'à un moment donné du code, les hypothèses sont satisfaites, et ainsi de ne pas avoir simplement un algorithme qui ne fonctionne pas globalement avec une erreur possible à chacune des 1000 lignes de codes

* Faire des tests tout au long de la programmation, en utilisant au maximum la modularité détectera les erreurs tôt dans le processus de création d'un programme

* Utiliser des logiciels tels que GDB ou valgrind permettra de trouver où se situe les fuites mémoires (valgrind) et de retracer les erreurs et d'inspecter la mémoire au milieu d'une exécution

* Commenter. Ou plutôt COMMENTEZ ! Cela est primordial pour déclarer la spécification des fonctions et rendre le code compréhensible, donc débogable.

Ne rien écrire hors du cadre.