*Algorithme d'Euclide étendu : extrait du volume 1 du TAOCP (The Art Of Computer Programming, l'Art de la Programmation des Ordinateurs) de Donald E. Knuth, bas de la page 13 et suivantes (Denise Vella-Chemla 19.12.2021)*

L'induction mathématique peut également être utilisée pour prouver des choses à propos des *algorithmes*. Considérons la généralisation suivante de l'algorithme d'Euclide.

**Algorithme E** (*Algorithme d'Euclide étendu*). Étant donnés deux entiers positifs $m$ et $n$, on calcule leur plus grand commun diviseur $d$, et on calcule aussi deux entiers $a$ et $b$ non nécessairement positifs tels que $am + bn = d$.

**E1**. [Initialiser.] On fait les affectations $a' \leftarrow b \leftarrow 1$, $a \leftarrow b' \leftarrow 0$, $c \leftarrow m, d \leftarrow n$.

**E2**. [Diviser.] Soient $q$ et $r$ le quotient et le reste, respectivement, de $c$ divisé par $d$ (on a $c = qd + r$ et $0 \leq r < d$).

**E3**. [Reste zéro ?] Si $r = 0$, l'algorithme s'arrête ; nous avons dans ce cas $am + bn = d$ comme souhaité.

**E4**. [Boucle.] On fait les affectations $c \leftarrow d$, $d \leftarrow r, t \leftarrow a', a' \leftarrow a$, $a \leftarrow t - qa, t \leftarrow b', b' \leftarrow b, b \leftarrow t - qb$, et on retourne en E2. ∎

Si on supprime les variables $a, b, a'$, et $b'$ de cet algorithme et qu'on utilise $m$ et $n$ pour les variables auxiliaires $c$ et $d$, on retrouve notre vieil algorithme, 1.1E[1]. La nouvelle version fait un peu plus, en déterminant les coefficients

[1]Depuis 1950, le mot algorithme a été très fréquemment associé à l'algorithme d'Euclide, un processus pour trouver le plus grand commun diviseur de deux nombres qui apparaît dans les *Éléments* d'Euclide (Livre 7, Propositions 1 et 2) ; il sera instructif de montrer l'algorithme d'Euclide ici :

**Algorithme E** (*Algorithme d'Euclide*). Étant donnés deux nombres entiers positifs $m$ et $n$, trouver leur *plus grand commun diviseur*, qui est, le plus grand entier positif qui divise à la fois $m$ et $n$.

**E1**. [Trouver le reste.] Diviser $m$ par $n$ et appeler le reste $r$ (on aura $0 \leq r < n$).

**E2**. [Est-ce zéro ?] Si $r = 0$, l'algorithme s'arrête ; $n$ est la réponse.

**E3**. [Réduction] Affecter $m \leftarrow n, n \leftarrow r$, et retourner à l'étape E1. ∎

$a$ et $b$. Supposons que $m = 1769$ et $n = 551$ ; on a successivement (après l'étape E2) :

| $a'$ | $a$ | $b'$ | $b$ | $c$ | $d$ | $q$ | $r$ |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1769 | 551 | 3 | 116 |
| 0 | 1 | 1 | −3 | 551 | 116 | 4 | 87 |
| 1 | −4 | −3 | 13 | 116 | 87 | 1 | 29 |
| −4 | 5 | 13 | −16 | 87 | 29 | 3 | 0 |

La réponse est correcte : $5 \times 1769 - 16 \times 551 = 8845 - 8816 = 29$, le plus grand commun diviseur de 1769 et 551.

Le problème est de *prouver* que cet algorithme fonctionne correctement, pour tout $m$ et $n$. On peut essayer d'appliquer la méthode de l'induction mathématique en définissant $P(n)$ comme l'assertion "L'algorithme E fonctionne correctement pour $n$ et tous les entiers $m$". Pourtant, cette approche ne marche pas si facilement, et nous avons besoin de démontrer quelques faits auxiliaires. Après une petite étude, on trouve que quelque chose doit être démontré à propos de $a, b, a'$, et $b'$, et le fait approprié est que les égalités

$$a'm+b'n = c, \qquad am+bn = d \tag{6}$$

sont toujours vérifiées à chaque fois que l'étape E2 est exécutée. Nous pourrions prouver ces égalités directement en observant qu'elles sont vraies de façon certaine la première fois qu'on arrive à l'étape E2, et que l'étape E4 ne change pas leur validité[2]

Maintenant on est prêt à montrer que l'algorithme E est valide, par induction sur $n$ : si $m$ est un multiple de $n$, l'algorithme fonctionne correctement de façon évidente, puisqu'on obtient le résultat immédiatement à E3 la première fois. Ceci est toujours le cas quand $n = 1$. Le seul cas restant est quand $n > 1$ et que $m$ n'est pas un multiple de $n$. Dans un tel cas, l'algorithme consiste à effectuer les affectations $c \leftarrow n, d \leftarrow r$ après la première exécution, et puisque $r < n$, *on peut supposer par induction* que la valeur finale de $d$ est le pgcd de $n$ et $r$. Par l'argument fourni en Section 1.1, les paires $\{m, n\}$ et $\{n, r\}$ ont les mêmes diviseurs communs, et, en particulier, elles ont le même diviseur commun. Par conséquent, $d$ est le pgcd de $m$ et $n$, et $am + bn = d$

[2]Voir l'exercice 6. *Note de la traductrice* : les exercices n'étant pas fournis ici, on a consigné les renvois vers les exercices en notes de bas de page.

par (6).

La phrase en italique dans la preuve ci-dessus illustre le langage conventionnel qui est si souvent utilisé dans une preuve inductive : quand on exécute la partie (b) de la construction, plutôt que de dire “Nous supposerons maintenant $P(1), P(2), ..., P(n)$, et avec cette supposition, nous démontrerons $P(n+1)$”, nous disons souvent simplement “Nous allons maintenant démontrer $P(n)$ ; supposons par induction que $P(k)$ est vraie à chaque fois que $1 \le k < n$.”

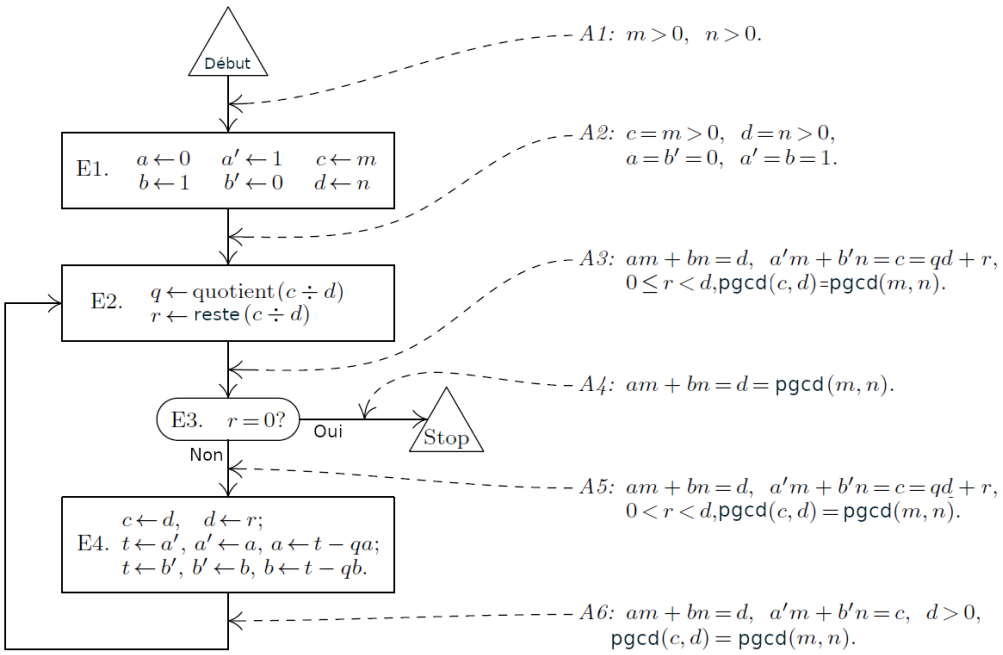


FIG. 4. Organigramme pour l'algorithme E, étiqueté par des assertions qui prouvent la validité de l'algorithme.

Si on examine cet argument de très près et que l'on change légèrement de point de vue, on peut envisager une méthode générale applicable pour démontrer la validité de n'importe quel algorithme. L'idée est de prendre un organigramme d'un certain algorithme et d'étiqueter chacune des flèches par une assertion à propos de l'état courant au moment où le calcul traverse cette flèche. Voir la Fig. 4, où les assertions ont été étiquetées $A1, A2, ..., A6$. (Toutes ces assertions s'appuient sur la stipulation supplémentaire que les

variables sont des entiers ; cette stipulation a été omise pour gagner de la place). $A1$ fournit les suppositions intiales en entrée de l'algorithme, et $A4$ est l'assertion que nous espérons prouver à propos des valeurs en entrée $a, b$, et $d$.

La méthode générale consiste à prouver, pour chaque boîte de l'organigramme, que

*si une assertion attachée à n'importe quelle flèche entrant dans la boîte est vraie avant que l'opération dans cette boîte ne soit effectuée, alors toutes les assertions sur les flèches sortant de la boîte sont vraies après avoir procédé à l'opération.* (7)

Ainsi, par exemple, nous devons prouver que soit $A2$ soit $A6$ avant E2 implique $A3$ après E2. (Dans ce cas $A2$ est une assertion plus forte qu'$A6$ ; c'est-à-dire qu'$A2$ implique $A6$. Donc nous avons seulement besoin de prouver qu'$A6$ avant $E2$ implique $A3$ après. Notons que la condition $d > 0$ est nécessaire dans $A6$ juste pour prouver que l'opération E2 a toujours du sens.) Il est également nécessaire de montrer que $A3$ et $r = 0$ implique $A4$ ; qu'$A3$ et $r \neq 0$ implique $A5$ ; etc. Chacune des preuves requises est très évidente.

*Une fois que l'assertion (7) a été démontrée pour chaque boîte, il s'ensuit que toutes les assertions sont vraies durant l'exécution de l'algorithme.* Car nous pouvons maintenant utiliser l'induction sur le nombre d'étapes du calcul, dans le sens du nombre de flèches traversées dans l'organigramme. Pendant qu'est traversée la première flèche, celle sortant de "Début", l'assertion $A1$ est vraie puisque nous supposons toujours que nos valeurs en entrée respectent les spécifications ; donc l'assertion sur la première flèche traversée est correcte. Si l'assertion qui étiquette la $n^{\text{ième}}$ flèche est vraie, alors par (7) l'assertion qui étiquette la $(n+1)^{\text{ième}}$ flèche est aussi vraie.

En utilisant cette méthode générale, le problème de prouver qu'un algorithme donné est valide consiste essentiellement la plupart du temps à inventer les bonnes assertions à mettre dans l'organigramme. Une fois que ce saut a été fait, c'est quasiment automatique d'amener les preuves que chaque assertion menant à une boîte implique logiquement chaque assertion qui en sort. En fait, c'est quasiment automatique d'inventer les assertions elles-mêmes, une fois que peu d'assertions parmi celles qui sont difficiles ont été découvertes ; par conséquent c'est très simple dans notre exemple d'écrire principalement que $A2$, $A3$, et $A5$ doivent être vraies si seulement $A1$, $A4$, et $A6$ sont données.

Dans notre exemple, l'assertion $A6$ est la partie créative de la démonstration ; tout le reste devrait, en principe, être fourni mécaniquement. Par conséquent, aucune tentative n'a été faite pour fournir des preuves formelles détaillées des algorithmes qui suivent dans ce livre, à un niveau de détail tel que celui qu'on trouve dans la Fig. 4. Il suffit d'établir les assertions inductives clés. Ces assertions apparaissent soit dans la discussion qui suit un algorithme, soit elles sont données comme des remarques entre parenthèses dans le texte de l'algorithme lui-même.

Cette approche pour prouver la correction des algorithmes a un autre aspect qui est même plus important : elle reflète la façon dont nous comprenons un algorithme. Rappelons que dans la Section 1.1, on a attiré l'attention de la lectrice pour qu'elle ne s'attende pas à lire un algorithme comme un extrait de roman ; un ou deux tests de l'algorithme sur quelques données d'exemples étaient recommandées. Cela a été recommandé expressément parce qu'une exécution de l'algorithme aide la personne à formuler mentalement les différentes assertions. L'auteur estime que nous ne comprenons vraiment pourquoi un algorithme est valide que quand nous avons atteint le point auquel nos cerveaux ont implicitement rempli toutes les assertions, comme cela a été fait dans la Fig. 4. Ce point de vue a des conséquences psychologiques importantes pour la bonne communication d'algorithmes d'une personne à une autre : cela implique que les assertions clés, celles qui ne peuvent pas facilement être déduites par un automate, devraient toujours être énoncées explicitement quand quelqu'un explique un algorithme à quelqu'un d'autre. Quand l'algorithme E est mis en avant, l'assertion $A6$ devrait être mentionnée également.

Un lecteur attentif aura noté un trou béant dans notre dernière preuve de l'algorithme E, pourtant. Nous n'avons jamais démontré que l'algorithme s'arrête ; tout ce que nous avons démontré, c'est que *s*'il s'arrête, il donne la bonne réponse !

(Notons, par exemple, que l'algorithme E a toujours du sens si on autorise ses variables $m, n, c, d$, et $r$ à prendre des valeurs de la forme $u + v\sqrt{2}$, où $u$ et $v$ sont des entiers. Les variables $q, a, b, a', b'$ doivent conserver une valeur entière. Si nous démarrons la méthode avec $m = 126\sqrt{2}$ et $n = 2010\sqrt{2}$, disons, cela calculera un "plus grand commun diviseur" $d = 42\sqrt{2}$ avec $a = +2$, $b = -1$. Même selon cette extension des hypothèses, les preuves

des assertions $A1$ à $A6$ restent valides ; donc toutes les assertions sont vraies tout au long de l'exécution de la procédure. Mais si nous commençons avec $m = 1$ et $n = \sqrt{2}$, le calcul ne s'arrête jamais[3]. Par conséquent une preuve des assertions $A1$ à $A6$ ne prouve pas logiquement que l'algorithme termine.)

Les preuves de terminaison sont habituellement gérées séparément. Mais l'exercice 13 montre qu'il *est* possible d'étendre la méthode ci-dessus dans de nombreux cas importants de telle façon qu'une preuve de terminaison soit incluse comme sous-produit.

Nous avons maintenant prouvé deux fois la validité de l'algorithme E. Pour être strictement logique, nous devrions également essayer de *démontrer* que le premier algorithme dans cette section, l'algorithme I [4], est valide ; en fait, nous avons utilisé l'algorithme I pour établir la correction de toute preuve par induction. Si nous essayons de démontrer que l'algorithme I fonctionne correctement, pourtant, nous sommes confrontés à un dilemme - nous ne pouvons pas vraiment le démontrer sans utiliser à nouveau l'induction ! L'argument serait circulaire.

En dernière analyse, toute propriété des entiers doit être prouvée en utilisant l'induction quelque part dans le processus, parce que si l'on descend jusqu'aux concepts de base, les entiers sont essentiellement *définis* par induction. Par conséquent, nous pouvons considérer comme axiomatique l'idée que tout entier positif $n$ est soit égal à 1 soit peut être atteint en commençant par 1 et en ajoutant itérativement 1 ; cela suffit à démontrer que l'algorithme I est valide. [Pour une étude rigoureuse des concepts fondamentaux à propos des entiers, voir l'article "On Mathematical Induction" de Leon Henkin, *AMM*

[3] Voir l'exercice 12.

[4] **Algorithme I** (*Construire une preuve*). Étant donné un entier positif $n$, cet algorithme fournira en sortie une preuve que $P(n)$ est vraie.

**I1**. [Prouver $P(1)$.] On affecte $k \leftarrow 1$, et on fournit une preuve de $P(1)$.

**I2**. [k = n ?] Si $k = n$, l'algorithme termine ; la preuve requise a été fournie.

**I3**. [Prouver $P(k + 1)$] Fournir une preuve de ce que "Si toutes les assertions $P(1), \ldots, P(k)$ sont vraies, alors $P(k + 1)$ est vraie". Écrire aussi "Nous avons déjà prouvé $P(1), \ldots P(k)$ ; donc $P(k + 1)$ est vraie.".

**I4**. [Augmenter $k$.] Augmenter $k$ de 1 et aller à l'étape I2. ∎

**67** (1960), 323-338.]

L'idée derrière l'induction mathématique est donc intimement reliée au concept de nombre. Les premiers européens qui ont utilisé l'induction mathématique pour des preuves mathématiques ont été l'italien Francesco Maurolico, en 1575. Pierre de Fermat a apporté d'autres améliorations, au début du $17^{\text{ième}}$ siècle ; il a appelé sa méthode la "descente infinie". La notion apparaît également clairement dans les derniers écrits de Blaise Pascal (1653). L'expression "induction mathématique" a été apparemment inventée par A. De Morgan au début du dix-neuvième siècle. [Voir *The Penny Cyclopædia* **12** (1838), 465-466 ; *AMM* **24** (1917), 199-207 ; **25** (1918), 197-201 ; *Arch. Hist. Exact Sci.* **9** (1972), 1-21.] On peut trouver une discussion approfondie au sujet de l'induction mathématique dans le livre de G. Pólya's *Induction and Analogy in Mathematics* (Princeton, N.J.: Princeton University Press, 1954), Chapitre 7.

La formulation de la preuve d'algorithme par assertions et induction, comme présentée ci-dessus, est principalement due à R. W. Floyd. Il a souligné qu'une définition sémantique de chaque opération dans un langage de programmation peut être formulée comme une règle logique qui dit exactement quelles assertions peuvent être prouvées après une opération, selon les assertions qui sont vraies avant [voir "Assigning Meanings to Programs", *Proc. Symp. Appl. Math.*, Amer. Math. Soc., **19** (1967), 19-32]. Des idées similaires ont été énoncées indépendamment par Peter Naur, *BIT* **6** (1966), 310-316, qui appelait les assertions des "instantanés généraux". Un raffinement important, la notion d'"invariants", a été introduite par C. A. R. Hoare ; voir, par exemple, *CACM* **14** (1971), 39-45. Des auteurs ultérieurs ont trouvé avantageux d'inverser la direction de Floyd, partant d'une assertion qui devrait être vraie *après* une opération vers la "pré-condition la plus faible" qui devrait être vérifiée avant que l'opération soit effectuée ; une telle approche rend possible la découverte de nouveaux algorithmes dont on a la garantie qu'ils sont corrects, si on part des spécifications de la sortie requise et qu'on travaille en remontant en arrière. [Voir E. W. Dijkstra, *CACM* **18** (1975), 453-457 ; *A Discipline of Programming* (Prentice-Hall, 1976).]

Le concept d'assertions inductives est effectivement apparu sous forme embryonnaire en 1946, au moment où les organigrammes ont été introduits par H. H. Goldstine et J. von Neumann. Leurs organigrammes originaux inclu-

aient des “boîtes d’assertion” qui sont en grande analogie avec les assertions de la Fig. 4. [Voir dans John von Neumann, *Collected Works* **5** (New York: Macmillan, 1963), 91-99. Voir aussi les commentaires préliminaires d’A. M. Turing à propos de la vérification dans *Report of a Conference High Speed Automatic Calculating Machines* (Cambridge Univ., 1949), 67-68 et ses figures; réimprimés avec des commentaires de F. L. Morris et C. B. Jones dans les Annales de l’Histoire de l’informatique [*Annals of the History of Computing* **6** (1984), 139-143.]

*The understanding of the theory of a routine*
*may be greatly aided by providing, at the time of construction*
*one or two statements concerning the state of the machine*
*at well chosen points. ...*
*In the extreme form of the theoretical method*
*a watertight mathematical proof is provided for the assertions.*
*In the extreme form of the experimental method*
*the routine is tried out on the machine with a variety of initial*
*conditions and is pronounced fit if the assertions hold in each case.*
*Both methods have their weaknesses.*

– A. M. TURING, Ferranti Mark I Programming Manual (1950)

*La compréhension théorique d'une procédure*
*peut être grandement améliorée en fournissant, au moment de sa construction*
*une ou deux assertions concernant l'état de la machine*
*en des points bien choisis. ...*
*Dans la forme extrême de la méthode théorique*
*une preuve mathématique étanche est fournie pour les assertions.*
*Dans la forme extrême de la méthode expérimentale,*
*la procédure est essayée sur la machine avec une variété de conditions initiales*
*et on dit qu'elle fonctionne si les assertions sont vérifiées*
*dans chaque cas.*
*Les deux méthodes ont leurs faiblesses.*

**Référence**

[1] Donald Knuth, The Art of Computer Programming, vol. 1, Fundamental algorithms, Third edition, 1997 (pages 13 à 18).