L'algorithmique : abstraire ou réifier ?

Michel GAUTHIER
maître de conférences en informatique, HDR
faculté des sciences et techniques de Limoges

Pour éviter toute interprétation fallacieuse, je demanderai d'abord de ne considérer ces réflexions que comme l'expression d'une longue expérience professionnelle dans l'enseignement de la programmation, sous les divers aspects fondamentaux, théoriques, méthodologiques et pratiques. Elles n'ont aucune raison d'être applicables sans digestion préalable à l'enseignement des mathématiques, ce n'est pas un plaidoyer en faveur de l'enseignement de l'algorithmique, ni d'ailleurs contre, et ma situation assure une totale indépendance vis-à-vis des choix de l'administration. Je ne fais que vous les offrir en espérant qu'elles vous seront utiles.

❧

Si vous me demandez de dire ce qu'il faut enseigner d'algorithmique au lycée dans le cadre des mathématiques, ce n'est pas de ma compétence, dans les deux sens du mot. Je peux préciser quelles connaissances et quels savoir-faire pourraient être utiles à la compréhension de l'informatique quelques années plus tard, ce qui ne constituerait qu'un aspect du paysage, où plusieurs voies peuvent conduire. Je peux aussi rendre compte d'expériences et des conséquences qui en ont été tirées, pour tenter d'éviter le renouvellement d'erreurs précédemment identifiées.

Si vous attendez de moi de vous définir ce qu'est l'algorithmique, vous risquez d'être déçus. Après quarante années d'activité autour de ce sujet, j'ai tout au plus accumulé quelques idées solides sur ce que ce n'est pas et fabriqué quelques hypothèses sur ce que ça pourrait être. Peut-être aussi y a-t-il plusieurs approches, éventuellement incompatibles.

Je sais notamment qu'elle ne consiste pas en une traduction en français des mots réservés d'un langage de programmation, même choisi pour ses qualités pédagogiques, ni en une imitation de ses structures. Cette orientation a fait assez de mal pour qu'on ne la rejette pas immédiatement. Je sais aussi qu'elle est destinée à une transmission d'information entre humains, sans intervention d'une machine, et qu'une traduction parfois complexe est toujours nécessaire en direction d'une machine ou d'un langage de programmation.

Quelques aspects historiques autour de la programmation peuvent aider à maintenir un cap cohérent. Nous sommes dans les années 1965-1975. Il existe des langages de programmation déjà éloignés de la machine et les compilateurs qui les traduisent en un code exécutable. La conception des programmes utilise largement les organigrammes, pertinemment rejetés ensuite quoique pas pour les raisons ordinairement invoquées.

➢ ☾ ☽ ➢

Premier jalon, 1966, appelé « théorème de Böhm et Jacopini ». Il dit « on peut transformer tout organigramme en algorithme utilisant seulement la répétition tant-que, l'instruction conditionnelle si-alors-sinon et des variables logiques ». Révolutionnaire, mais —comme toute révolution— immédiatement pervertie par le dogmatisme et l'incompétence, ici le remplacement du « on peut » théorique par un « on doit » pratique. Mais en 1966, on n'a pas encore l'outillage intellectuel pour comprendre pourquoi les organigrammes posent problème, seulement pour observer qu'ils en posent.

Il en résultera le langage Pascal, et ses quinze ans de domination pédagogique justifiée, ainsi que le thème de la « programmation structurée » avec ses deux orientations trop faiblement compatibles : structurer le programme ou structurer la pensée du programmeur ? La même ambiguïté survit toujours, et seuls se posent vraiment la question ceux pour qui l'essentiel est de structurer la pensée. Instruire plutôt qu'enseigner, diraient certains étymologistes. L'essentiel est de transmettre le procédé de calcul de manière à permettre au destinataire de calculer, pas d'obtenir une qualité formelle du texte transmis. Selon la formule classique, cela ira encore mieux en le disant.

Dans une logique où la formalisation est devenue politiquement incorrecte (logique dont je ne discuterai pas ici si elle est pertinente), il est totalement normal de ne pas imposer une forme à l'algorithmique mais de privilégier la compréhension de ce qui fait qu'on peut diffuser le procédé de calcul. Inversement, il est totalement impossible de programmer en conservant la même logique. Il faut donc trouver un équilibre entre obligations administratives et obligations techniques. L'algorithmique pourrait constituer un domaine d'expérimentation de cet équilibre, qui n'a aucune raison d'être le même dans toutes les classes.

❧

Deuxième jalon, 1970, la « définition axiomatique du langage de programmation Pascal ». Révolutionnaire aussi. Un langage de programmation n'est pas défini par le code généré par le compilateur, mais par une définition formelle, permettant de l'articuler avec des mathématiques.

Au cœur de la définition, le concept de type : un ensemble de valeurs, un jeu d'opérations, les propriétés qui les relient, à quoi on ajoutera ultérieurement les comptes-rendus en cas d'opération impossible. Autrement dit, traduit en termes mathématiques classiques, c'est une structure algèbrique. L'algèbre, l'algorithmique et la programmation sont donc demi-sœurs, même si l'autre parent les a rendues différentes. Penser faire de l'algorithmique sans identifier explicitement les opérations disponibles, c'est irréaliste dès que l'on dépasse un niveau très élémentaire d'application.

Conséquence de cette parenté et de cette formalisation, la possibilité de prouver les algorithmes et les programmes. Puisque les types sont définis formellement, puisque les instructions sont associées à des règles de déduction (analogue à des implications entre des emplacements et des instants distincts), on peut assembler tout cela en une preuve, au sens mathématique du mot. Sans doute la preuve effective d'un programme effectif est-elle inaccessible, mais des preuves partielles et une démarche algorithmique fondée sur la possibilité de prouver peuvent être envisagées. Au cœur de tout cela, le moteur du fonctionnement est la récurrence. Un algorithme, ce n'est plus l'exécution d'une suite d'instructions, c'est l'enchaînement d'une suite de propriétés logiques.

Arrivés à ce point, on notera que là réside la difficulté des organigrammes : un algorithme en tant-que et si-alors-sinon a de bonnes propriétés de déduction, alors qu'un organigramme est terriblement complexe à maîtriser de ce point de vue. Pratiquement, les seules autres instructions valables pour structurer programme ou algorithme satisfont l'idée « je constate ici la propriété que j'attends là ». Le raccourci est du niveau de complexité de $A \Rightarrow A$, mais l'enjeu est de savoir exprimer ce qu'on attend. Cela est parfois difficile mais comment y échapper ?

❧

Troisième jalon à observer, plus difficile à dater, quoiqu'avant 1975, plus difficile à nommer comme concept, appelé encapsulation dans les langages de programmation. Schématiquement, l'idée est qu'il faut cacher, rendre inaccessibles, tous les aspects de représentation informatique, pour ne laisser à disposition que ce qui exprime exactement la structure du monde extérieur —algèbrique—. On parle à ce sujet de « programmation modulaire », sœur puînée de la programmation structurée. Si l'encapsulation reste un concept en dehors de la programmation, ce qui est discutable, il est étranger à l'algorithmique. S'il est étranger au thème de ce jour, pourquoi en parler ? Eh bien, justement pour affirmer cette extranéité. L'algorithmique fabrique une abstraction du monde réel, et tout ce qui conduit ensuite vers une mise en œuvre dans une machine arrête la démarche vers l'abstrait et la redirige vers le concret, autrement dit commence à réifier.

Il y a là un aspect fondamental : à un certain moment apparaît un maximum dans la démarche de l'observation du réel vers la mise en œuvre du calcul. Le niveau algorithmique, je propose de le définir comme étant ce maximum. Avant lui, on est dans une démarche mathématique et plus précisément algèbrique (une démarche géométrique du genre « fabriquer le point tel que… » est du point de vue algorithmique de nature algèbrique). Après lui, on passe dans la démarche de programmation. Il constitue une frontière entre les mathématiques et l'informatique. Frontière assez floue en fait, peuplée de travailleurs bi-nationaux et facile à traverser (et je dirais beaucoup trop, mais c'est un avis personnel), mais frontière tout de même, et qui ne fait ni un programmeur d'un mathématicien ni l'inverse.

Ceci permet notamment de comprendre pourquoi la démarche de traduction en français d'un langage de programmation est définitivement vouée à l'échec : un algorithme ne peut être qu'indépendant de la programmation, mais aussi plus abstrait que le plus abstrait des langages de programmation. Ceci définit aussi une exigence vis-à-vis de l'enseignement de l'algorithmique. Il ne peut pas, il ne doit pas être une préparation à la programmation, une sorte de supplément d'âme, d'alibi culturel. Enseigner l'algorithmique, c'est approcher et enseigner l'algèbre en tant que science expérimentale.

❧

Un dernier jalon pourrait être joint pour satisfaire à la mode : quid du concept d'objet ? Nous sommes maintenant un peu plus tard, vers 1980. Ce concept n'apporte guère plus que l'encapsulation, dont nous avons dit qu'il n'a pas à être considéré par l'algorithmique. Il apporte deux choses, la structure appelée héritage et l'idée que la duplication (plus connue sous le nom informatique d'affectation) n'est pas une opération universellement disponible.

Du point de vue algorithmique, l'existence de la structure informatique appelée affectation doit donc être constatée et pas présupposée. Il existe des domaines où elle existe naturellement et d'autres où elle n'existe pas, ce qui compliquerait tout si on n'avait pas la chance que toutes les abstractions mathématiques ont les propriétés qui permettent l'utilisation de l'affectation. Les objets, fils de l'encapsulation, n'ont pas plus que leur mère de place dans l'algorithmique. On peut les oublier ou, mieux, ne pas en parler si personne ne pose la question.

Le seul véritable apport du concept d'objet s'appelle « héritage ». Il est difficile à définir en quelques mots et nous mènerait trop loin aujourd'hui. On peut cependant déduire son absence d'intérêt algorithmique de sa nature de construction par spécialisation « un S est un cas particulier de R obtenu en lui ajoutant… » qui le rend inapplicable à tout travail d'abstraction. On peut construire un monde par héritage, mais pas comprendre un monde existant. La controverse informatique l'oppose à une construction par généralisation, où on constate l'existence dans diverses structures d'ensembles communs d'opérations, démarche évidemment naturelle en mathématique et facile à appliquer à l'algorithmique.

❧

Pour illustrer tout cela, quelques exemples que j'utilise pour illustrer certains de mes cours bac+3 en informatique.

❧

Le document du comité des programmes propose un calcul d'intérêts composés. Je suis presque certain que 99 % des programmes le mettant en œuvre seront faux. Ils seront faux parce que traitant les valeurs financières comme des réels. Peut-être n'est-ce pas si grave pour une programmation de débutants. C'est en revanche une erreur fondamentale pour une programmation professionnelle ou pour l'algorithmique. Une valeur financière, c'est un nombre entier d'un atome qui est une fraction décimale de l'unité principale, décime, centime, millime,… Très peu de langages[1] possèdent un tel type, certes, mais cela impose que la vision algorithmique conserve cette propriété de nombre entiers de centimes et les opérations associées. Abstraire par un réel est une faute grave, même si traduire ensuite en réels de la machine peut parfois être acceptable si on maîtrise la grandeur de l'approximation.

❧

[1] Pour ceux qui voudraient en savoir plus, je suggère d'aller sur le site de l'IUT d'Aix-en-Provence chercher les cours de langage Ada de Daniel Feneuille, ou sur celui de l'Université de Limoges mes divers aide-mémoire à l'intention des étudiants de l'année 3 de la licence d'informatique. Ces compléments ne sont pas utiles pour comprendre et enseigner l'algorithmique au lycée..

Récemment, j'avais donné un projet de programmation mettant en œuvre des numéros de téléphone. Plusieurs étudiants ont proposé une représentation par un entier. Étonnant, parce qu'en fait si on ajoute un zéro devant, la signification n'est plus la même. Ce n'est dont certainement pas un entier, même si c'est une suite de chiffres décimaux. Savoir distinguer les suites de chiffres ayant une nature numérique des autres est algorithmiquement essentiel.

❧

En continuant dans la même veine, observons le calendrier. Qu'est-ce qu'une année ? Pour comprendre ce qu'est 2009, essayons d'ajouter, de multiplier ou de soustraire 2005 et 2009. Multiplier, ça a apparemment une cohérence mathématique (des « années au carré ») mais peu de sens réel. Ajouter, ça n'a aucun sens ni mathématique ni réel. Soustraire, ça a tous les bons sens voulus, à condition de comprendre que le résultat est un nombre d'années. Remarquez aussi que 2009 + 2005 a un sens si 2009 est un numéro d'année et 2005 un nombre d'années. Et finalement multiplier des numéros d'année n'a effectivement aucun sens.

L'algorithmique doit obtenir ce niveau d'abstraction. Appelons ça comme on veut, des jalons et des distances (accompagnées d'un sens), des points et des vecteurs, une géométrie affine à une dimension sur $\mathbb{Z}$, des numéraux ordinaux et des numéraux cardinaux, peu importe mais la structure doit être à ce niveau de détail. Et cette structure est la même que celle du calendrier (la suite des jours) par rapport au nombre de jours, ou du repèrage horizontal des pixels d'un écran par rapport à un déplacement entre les colonnes…

❧

Autre idée essentielle, une valeur numérique ayant un caractère de grandeur possède une unité. L'algorithmique doit conserver non seulement l'indication de grandeur physique (longueur, temps, puissance, travail,…) mais aussi l'unité utilisée. On n'ajoute jamais une masse et un poids, et on n'ajoute qu'avec précaution des yards et des mètres (certains y ont perdu une sonde martienne). Là encore, peu de langages de programmation permettent de conserver ce niveau de précision.

❧

Autrement dit, il faut être extrêmement attentif à bien caractèriser les nombres utilisés. Se mettre à la remorque des langages de programmation, logiciels ou calculettes ce n'est jamais le bon choix, c'est même parfois dangereux. En revanche, un algorithme agissant comme abstraction, et seulement comme abstraction, est une aide précieuse pour un éventuel programmeur final.

☾ ☽

En renouvelant mon souhait que tout ceci puisse être utile.