5. HERCULE CONTRE L'HYDRE

Le combat d'Hercule contre l'Hydre représente, comme le théorème de Goodstein, un exemple de propriété qui met en jeu des entiers et ne peut se démontrer sans utiliser l'infini. Hercule doit vaincre l'Hydre, mais, chaque fois qu'il lui coupe une tête, il en repousse de nouvelles ! Les règles du combat sont les suivantes : Hercule coupe une seule tête à chaque coup. S'il coupe une tête directement reliée au corps de l'Hydre, la tête ne repousse pas ; en revanche, si la tête coupée n'est pas directement reliée au corps, alors l'ensemble des têtes situées à côté de la tête coupée, c'est-à-dire provenant de la bifurcation commune au-dessous, se reproduit en *n* nouveaux exemplaires, si Hercule porte son *n*-ième coup du combat. Le nombre de tête de l'Hydre semble tendre vers l'infini irrémédiablement. Pourtant, un théorème énonce que, quelle que soit la forme initiale de l'Hydre et quelle que soit la stratégie d'Hercule, il finira toujours par tuer l'Hydre en coupant toutes les têtes. Comme le théorème de Goodstein, le théorème de l'Hydre se démontre en utilisant des ordinaux et, à nouveau, Kirby et Paris ont prouvé qu'il ne peut pas être démontré dans le système de Peano, c'est-à-dire par récurrence.

PREMIÈRE COUPURE
En brun, la partie excitée par la coupe. Cette partie est dupliquée.

DEUXIÈME COUPURE
En brun, la partie excitée par la deuxième coupe. Cette partie est ajoutée deux fois.

TROISIÈME COUPURE
En brun, la partie excitée par la troisième coupe. La partie excitée est ajoutée trois fois. S'il coupe la tête bleue reliée directement au corps de l'hydre, elle ne repousse pas.

Coupera-t-on toutes les têtes de l'hydre? Oui, et ce résultat se démontre avec les ordinaux.

récurrence. Il ne donne toutefois aucun exemple explicite d'une telle propriété.

Le second théorème d'incomplétude comble cette lacune, en livrant un exemple : c'est une propriété qui code, dans le langage de l'arithmétique, le fait que le principe de récurrence n'est pas contradictoire avec lui-même. Ce résultat, fort remarquable, continue de fasciner après bien des années. Pourtant, du point de vue de l'arithmétique, il nous laisse un peu sur notre faim, car la «propriété qui code dans le langage de l'arithmétique le fait que le principe de récurrence n'est pas autocontradictoire» manque de clarté et ne correspond guère à ce qu'un arithméticien a l'habitude d'appeler une propriété d'arithmétique...

Il a fallu attendre cinquante ans avant de trouver des propriétés d'arithmétique plus communes qui ne soient pas démontrables par récurrence. Le premier exemple a été une propriété de type combinatoire isolée par Paris et Harrington en 1978, suivi du théorème de Goodstein, établi quelques années plus tard. C'est à ce jour l'exemple le plus simple d'une propriété d'arithmétique, qui se démontre à l'aide de l'infini, et dont il est prouvé qu'elle ne peut être démontrée par récurrence.

En un sens, un tel résultat constitue un argument puissant en faveur de l'utilisation de l'infini en mathématiques puisqu'il montre que certaines propriétés ne mettant en jeu que des objets finis ne peuvent être démontrées qu'à partir de l'infini : on aurait donc tort de se priver d'une telle possibilité. D'un autre côté, l'exemple de la graine 4 montre que le théorème de Goodstein met en jeu des entiers gigantesques, beaucoup plus grands par exemple que le nombre d'atomes dans l'univers, et nous pouvons aussi nous interroger sur l'usage de tels entiers et leurs rapports avec les entiers «de tous les jours»...

Patrick DEHORNOY est professeur de mathématiques à l'université de Caen.

G. GOODSTEIN, *On the Restricted Ordinal Theorem*, in *J. Symb. Logic*, n° 9, pp. 33-41, 1944.

L. KIRBY et J. PARIS, *Accessible Independence Results for Peano Arithmetic*, in *Bull. London Math. Soc.*, n° 14, pp. 285-293, 1982.

A. LEVY, *Basic Set Theory*, Springer, 1979.

L'infini et l'univers des algorithmes

GILLES DOWEK

On ne peut pas toujours prévoir le nombre d'opérations nécessaires à un calcul. Aussi, pour permettre l'expression de tous les algorithmes, il faut introduire la possibilité que les calculs durent infiniment longtemps.

Un algorithme décompose un calcul en des suites d'opérations élémentaires. Quand un algorithme prescrit un nombre infini d'opérations élémentaires, on n'obtient jamais le résultat. Aussi, faut-il tenter de maîtriser ou de chasser cet infini.

Pour multiplier 35 par 12, nous devons multiplier 5 par 2, poser le 0, retenir le 1, multiplier 3 par 2, ajouter la retenue… au bout de nos efforts, nous obtiendrons le résultat : 420. Cette recette qui décrit l'enchaînement des opérations élémentaires d'une multiplication est dénommée l'algorithme de la multiplication, ou plutôt un algorithme de la multiplication, car il existe d'autres méthodes de calcul possibles. Ainsi, 35 + 35 + 35 + 35 + 35 + 35 + 35 + 35 + 35 + 35 + 35 + 35 est une méthode, un algorithme envisageable pour multiplier 35 par 12, à l'aide d'additions successives. Il en existe bien d'autres.

Ces algorithmes sont des méthodes de calcul qui donnent un résultat après un nombre fini d'opérations et dans un temps fini. L'infini ne semble ainsi jouer aucun rôle dans l'univers des algorithmes, et pourtant la notion d'algorithme ne peut être définie sans utiliser l'infini. L'infini est indispensable, mais il faut le maîtriser.

La définition d'un algorithme

Si nous utilisons des algorithmes depuis l'Antiquité et que le mot algorithme vient du nom du mathématicien Al Khwarizmi qui vécut à Bagdad au IXe siècle, ce n'est qu'au XXe siècle que cette notion a été définie avec précision.

Dans les années 1920, Thoralf Skolem (1887-1963) a proposé une première définition de la notion d'algorithme liée à la récurrence. Ce procédé consiste à définir une fonction en donnant sa valeur en 0 et en indiquant comment passer de sa valeur en n à sa valeur en $n + 1$. Par exemple, la fonction e, qui à chaque nombre entier n associe le nombre 2^n : $e(0) = 1$, $e(1) = 2$, $e(2) = 4$, $e(3) = 8$, $e(4) = 16$, … est définie par récurrence, en indiquant que sa valeur en 0 est 1 et qu'on passe d'une valeur à la suivante en la doublant.

Les fonctions qui sont définies par leur valeur en 0, ou en une autre valeur initiale, et par une opération permettant de passer directement d'une valeur à la suivante sont appelées, selon un terme dû à Rosza Péter, récursives primitives. La fonction $e(n) = 2^n$ est récursive primitive. Elle est définie par $e(0) = 1$ et $e(n + 1) = 2e(n)$. Richard Dedekind, qui avait déjà étudié les définitions par récurrence en 1888, avait montré que l'addition, la multiplication, l'élévation à la puissance étaient des fonctions récursives primitives.

La définition d'une fonction par récurrence donne un algorithme pour le calcul des valeurs de cette fonction. Ainsi, pour calculer la valeur de la fonction e en 10, il suffit de calculer sa valeur en 0, puis en 1, en 2,…, en 9 et enfin en 10. On obtient ainsi $e(0) = 1$, $e(1) = 2$, $e(2) = 4$, …, $e(9) = 512$, et $e(10) = 1\,024$.

On programme aisément de telles fonctions récursives primitives en utilisant des boucles *For*. Chaque itération de la boucle exécute une opération élémentaire ; on peut déterminer le nombre d'itérations nécessaires au calcul en fonction de la valeur que l'on souhaite calculer. Si l'on demande au programme de calculer la fonction e pour n égal à 10, après 10 itérations, 1 024 s'affichera sur l'écran de l'ordinateur.