# VALEUR DIDACTIQUE DES TECHNIQUES RECURSIVES EN PROGRAMMATION

Peter Bender [1]

L'extension de l'introduction des micro-ordinateurs à l'école, si l'on va au-delà des possibilités qu'ils offrent en tant que moyen ou comme outil, se voit justifier en gros selon deux axes. L'un est issu de l'informatique et l'autre de la théorie de la connaissance (ou plus exactement de la direction que lui impriment les spécialistes des ordinateurs). En dépit de contextes argumentatifs différents, c'est dans les deux cas la technique de la "programmation structurée" qui est avancée comme contenu de formation, et ce pour sa valeur potentielle de formation.

Ici dans ce qui suit, nous ne nous intéresserons pas à une définition particulière de cette technique de programmation dans le domaine de l'informatique, mais à l'aspect conceptuel tel qu'il se présente dans la discussion didactique. Je veux aussi pour le moment laisser de côté les objections qu'on lui fait, bien que je les tienne pour significatives. (Ces objections sont, contre l'argument pragmatique, que presque personne n'a ou n'aura dans l'avenir à savoir programmer, à l'école non plus, à l'exception de l'enseignement de l'informatique, et contre l'argument didactique, que la recherche dans le domaine cognitif n'a aucun indice en faveur de l'hypothèse de l'existence de transferts quasi automatiques de procédures de pensée d'un domaine dans un autre, surtout quand l'un des domaines est fortement marqué par le micro-ordinateur et l'autre non.)

On entend principalement par "programmation structurée" l'introduction logique et conséquente de la technique des sous-programmes. A son tour, cette technique sert de préalable à la programmation *récursive*, c'est-à-dire à l'auto-appel des sous-programmes, une technique que certains langages de programmation permettent, comme par exemple le *Logo*. Comme ce langage a été développé pour favoriser la propagation des micro-ordinateurs à l'école et que la propagande en faveur de cet objectif, sous forme de ce qu'on a appelé la philosophie du Logo, a reçu un écho favorable chez bon nombre de pédagogues du monde occi-

[1] GHS, FB 17, Mathematik, Heinrich Plettstr. 40 D 3500 KASSEL

Actes du premier colloque franco-allemand de didactique, pp. 257-65

dental, la technique de la récursivité, avatar de l'idée générale d'itération, a fait son entrée dans la discussion didactique.

L'itération (au sens large) est un concept mathématique (resp. informatique), à qui on accorde suffisamment d'extension (généralité logique), d'étendue (importance et possibilités multiples d'application), et de sens (ancrage dans la pensée de tous les jours, signification dans le monde de la vie) pour qu'elle apparaisse, au sens de Schreiber (1983), comme une idée universelle de la mathématique (et assurément de l'informatique). La question de savoir si l'idée d'itération est déjà suffisamment représentée dans l'enseignement par les exemples mathématiques traditionnels, comme la multiplication vue comme addition itérée, le calcul des intérêts composés, la mesure, le calcul infinitésimal, etc., ou si il est encore nécessaire de la présenter via des programmes adéquats pour ordinateurs, cette question n'a reçu aucun éclaircissement définitif. Dans ce qui suit, je suppose toutefois prise la décision d'enseigner la programmation à l'école, et ce, il va sans dire, dans un langage de programmation évolué. Qu'est-ce alors qui parle en faveur du développement de l'idée d'itération précisément sous la forme de la technique récursive, caractéristique de Logo, et non pas par exemple sous la forme des boucles itératives qui apparaissent dans de nombreux langages de programmation courants? Ce sont ces deux termes de l'alternative qui vont être discutés maintenant, et j'emploierai les termes correspondants de "récursif" et d'"itératif", bien que ceux-ci en tant que concepts mathématiques présentent précisément un caractère d'incompatibilité. En regard des nombreuses questions pédagogiques fondamentales que pose l'"informatisation" de l'école, une étude approfondie faite à partir de cette alternative n'apparaît à vrai dire que comme un problème technique d'importance mineure. Mais ce sont les partisans du langage Logo qui ont toujours souligné la valeur didactique particulière que présente la récursivité (Papert, 1980 : 71, 74; Hoppe, 1984 : 102, 135, 137; Ziegenbalg, 1984; Löthe, 1985b : 198, et de nombreux autres).

Il est certain que le fait d'atteindre un certain niveau de compétence en programmation, en liaison avec le passage à des classes de problèmes déterminés, pousse à employer la récursivité. Bien sûr, toute récursion peut aussi se programmer sous forme d'itération (tout comme l'inverse), mais le travail sur une structure arborescente point trop primitive fait à l'aide de boucles itératives peut être fort coûteux, favoriser les erreurs et rendre le déboguage difficile. Si l'on part de la philosophie du Logo, avec son orientation vers l'intelligence artificielle, la récursivité apparaît certainement au programmeur avancé comme une technique appropriée,

tout au moins dans la perspective d'une conception dynamique des langages de programmation, telle qu'elle a dominé jusque dans les années soixante-dix. En ce qui concerne l'enseignement scolaire, qui est plutôt organisé en fonction de représentations éducatives traditionnelles, j'estime que la récursivité, même dans le cas d'une "informatisation" plus poussée de l'école, n'a qu'une importance mineure : pour les exemples où elle présente un avantage authentique pour la programmation, on manque d'analyses didactiques tant soit peu convaincantes à partir desquelles pourrait se justifier son incorporation dans les matières canoniques de l'enseignement de formation générale. Le fait de dire que la programmation de ces exemples (tours de Hanoï, dessins de diagrammes arborescents, etc.) est une forme de résolution de problèmes propice à l'apprentissage de la technique de la récursivité, met partiellement en évidence un cercle vicieux dans l'argumentation, et, à mon avis, ne vaut pas pour la majorité des élèves d'un niveau scolaire donné.

Pour Hoppe (1984 : 102) et Löthe (1985 : p.198 entre autres), la programmation récursive est "proche des mathématiques" et contribue à dépasser le "principe de travail de von Neumann" de l'ordinateur traditionnel, avec son parcours linéaire et séquentiel de la suite des instructions. A la base de cette appréciation, il y a certainement un échelonnement dans le confort que présentent les langages de programmation, avec peut-être d'un côté les langages assembleurs, où une opération élémentaire de calcul doit être combinée à l'aide de plusieurs instructions séparées, jusque peut-être de l'autre le Lisp, où il est possible d'écrire directement une expression récursive. Cet échelonnement se laisse d'ailleurs étendre au cas des langages ou des systèmes de programmation dans lesquels par exemple des équations polynomiales sont automatiquement résolues dès qu'elles sont posées, ce qui rend en ce sens ces langages ou ces systèmes encore "plus proches des mathématiques". Pourquoi donc devoir choisir justement le langage Logo sur cette échelle de langages, et de plus comme premier (et seul) langage?

Il n'y a pas de langage de programmation qui me semble mieux adapté que les autres pour prévenir le danger psychique émanant de l'ordinateur qui, infaillible, sans être sujet au moindre doute, sans ressentir la moindre émotion, exécute instruction après instruction, usurpant le mode de pensée du programmeur et l'entraînant dans le dévoiement de sa "pensée linéaire". La contribution qu'apporte la technique de la récursivité au combat contre ce danger n'a qu'une ampleur modeste, d'autant plus - et ceci est commun à toutes ces finesses de programmation - que ce qu'on

dépense peut-être en moins quand on veut programmer doit être réinvesti quand on veut regarder le fonctionnement du programme. Il est possible que l'effort qu'il faille alors faire soit faible pour le programmeur averti, et d'ailleurs largement inutile, mais pour le débutant, cet effort est pendant longtemps important, ne serait-ce tout pragmatiquement par exemple pour le déboguage, mais aussi pour des considérations de fond. Sans cette exigence de regard sur le fonctionnement des programmes, il ne serait pas d'ailleurs possible de justifier pourquoi l'élève (l'apprenant) a à programmer et non à se contenter de se servir simplement de programmes déjà existants ou de logiciels.

Et je tiens également l'apport de la programmation à l'élaboration des concepts (mathématiques par exemple) en tant que troisième phase après l'approche en compréhension du concept et sa formalisation, comme moins évident avec un langage de programmation "proche des mathématiques" qu'il ne peut l'être avec précisément un langage qui oblige l'élève à voir clairement, étape par étape, comment s'élabore linéairement un algorithme. En exprimant ceci avec anthropomorphisme et exagération : avec un langage de programmation "proche des mathématiques", notre "tutee" d'ordinateur est trop intelligent et n'a pas, pour l'élève, assez d'exigence en matière d'explication.

Les appels récursifs de procédure peuvent masquer le déroulement des calculs (voir déjà Oberschelp, 1977 : 45) - c'est d'ailleurs là leur fonction: ils doivent dégager le programmeur de l'obligation d'étudier ce déroulement de façon suivie. Cette fonction qui est la leur se révèle être sur ce point anti-didactique, tout au moins en ce qui concerne les novices en programmation, c'est-à-dire dans un large domaine de l'enseignement donné à l'école. Qu'on veuille bien se rappeler que le Logo a été développé spécialement pour les enfants, non à partir de langages "interactifs" avec ajout délibéré de techniques récursives, mais bien comme une simplification du langage Lisp qui est lui "porteur de récursivité".

Personne ne contestera certes que la conception récursive d'une expression est plus difficile à percevoir concrètement que ne l'est sa conception itérative, ces conceptions étant entendues au sens suivant: Soit l'expression $a_n = f(a_{n-1})$ (avec $n \in \mathbb{N}$). La concevoir itérativement, c'est s'en représenter la construction totale ainsi: $a_0$ , $a_1 = f(a_0)$ , $a_2 = f(a_1)$, ....., $a_n = f(a_{n-1})$. Contrairement à cela, la conception récursive signifie: pour tout n on a $a_n = f(a_{n-1}) = f^2(a_{n-2}) = \ldots\ldots = f^n(a_0)$, et ensuite $a_1 = f(a_0)$, $a_2 = f(a_1)$,.....,$a_n = f(a_{n-1})$. Procéder itérativement veut donc dire

partir avec une valeur connue, obtenir à partir d'elle une deuxième valeur avec l'opérateur f, puis une troisième, etc..Procéder récursivement consiste à considérer une quantité inconnue cherchée comme donnée quand elle est obtenue par l'opérateur f à partir d'une autre quantité, d'ailleurs également inconnue. Tandis que l'"itératif" - et avec raison - ne se pose même pas la question de savoir s'il va vers son but, le "récursif", du moins s'il n'est que peu versé en mathématiques, va s'étonnant de voir qu'une quantité inconnue puisse devenir connue à partir du moment où elle est ramenée à une autre quantité tout aussi inconnue. Il doit se représenter étape après étape la reconstruction de ce retour en arrière et se trouve continuellement sur un terrain apparemment peu sûr. Tandis que l'"itératif" peut oublier après chaque étape les résultats antérieurs, le "récursif" doit se représenter toute la chaîne des retours en arrière puisque, dès qu'il tombe sur une quantité vraiment connue, il a à déterminer, par marche en avant cette fois, toutes les autres quantités.

C'est exactement sous cette forme qu'apparaît la suite d'instructions en laquelle l'interpréteur transforme une structure de contrôle récursive. Tout à fait banalement, on ne peut échapper au mode linéaro-séquentiel de travail de l'ordinateur quand on veut regarder le fonctionnement d'une technique récursive. De plus, il y aura toujours une itération à la clef, mais celle-ci ne sera pas développée comme une boucle, dans la structure interne du programme toutes les instructions seront exécutées spécialement, ce qui soit dit en passant conduit, lors d'appels récursifs nombreux, à des besoins en mémoire énormes. La récursivité terminale (où l'auto-appel de la procédure se fait avec la dernière instruction) n'exige pourtant pas d'envisager la récursivité en ce sens, car dans ce cas, toutes les instructions de la procédure sont exécutées directement, avant que celle-ci ne soit appelée de nouveau avec d'autres paramètres. On n'a donc pas alors besoin de se représenter une nouvelle construction d'une suite d'instructions, suivie des élaborations correspondantes, et pour de telles procédures la conception itérative est adéquate.

Tout n'est cependant pas récursivité terminale dans ce qui en a l'air: ainsi, il n'y a pas récursivité terminale quand dans l'affectation dans laquelle se produit l'auto-appel de la procédure, celle-ci se voit suivie de l'exécution d'une opération, comme par exemple la multiplication dans l'instruction 'RENVOI: N*FACT: N-1' lors du calcul de factorielles avec une procédure 'FACT'.

On rapporte toujours que les apprenants peuvent (au moins) manier la récursivité (terminale) (Hanninger, 1982 : 35, Lavallade, 1985 : 59, Treadaway, 1985 : 47, et d'autres), tandis que

Kurland, Pea (1983, d'après Haussmann 1985:148) arrivent, dans leurs recherches avec des enfants entre 8 et 12 ans, à la conclusion que ces derniers n'ont pas de vue suivie du principe de récursivité, même s'ils l'utilisent. Quant à des succès avec la récursivité complète, on n'en entend pas du tout parler. Ces résultats plaident plutôt pour la force de la conception itérative, d'autant plus que Anzai, Uesato (1982, d'après Haussmann 1985 : 147) constatent que les apprenants comprennent mieux les formulations récursives quand ils sont familiarisés avec les structures itératives. Au total, je tiens pourtant les bases statistiques de ces résultats pour beaucoup trop étroites et il y aurait à examiner ce que chacun des auteurs entend d'ailleurs par succès ou échec. Mais, après cette analyse de difficulté et ces "constatations" empiriques, qu'est-ce qui s'opposerait à travailler dans l'enseignement de formation générale, si on y tient, avec un langage de programmation itératif, avec lequel , à mon avis, on ferait assez face aux demandes mathématiques et informatiques dans l'enseignement, ou bien si on tient absolument à la récursivité essentielle, n'introduire cette dernière que plus tard (voir un avis contraire chez Löthe, 1985 : 198)?

Les tenants du Logo, même les modérés parmi eux, craignent - avec raison - que le passage à leur langage de programmation ne soit alors trop souvent rejeté. De fait, plus on avance à l'école, plus les contenus qui suggèrent une programmation récursive apparaissent, en raison de leur caractère interchangeable, de leur isolement et du manque d'engagement qu'ils suscitent chez maîtres et élèves, comme de plus en plus suspects. Dans le domaine de l'école élémentaire, de tels contenus se laissent plus facilement justifier, par exemple en invoquant un objectif d'acquisition de stratégies cognitives. Il en serait autrement si l'importance culturelle des mathématiques et de l'informatique comme sources principales de problèmes à résoudre à l'aide de programmes en Logo, était significativement augmentée, mais je tiens cette vue idyllique d'une jeunesse avide de s'adonner aux mathématiques et à l'informatique pour une projection irréaliste des dadas de beaucoup de didacticiens des mathématiques ou de partisans de l'"ordinateur à l'école". Mais ce serait là une condition nécessaire pour obtenir un certain niveau, dans lequel les différences dans les styles de programmation pourraient avoir une influence sensible sur les contenus (choix, façon de les traiter sans l'intervention du maître) ou même sur les styles de pensée.

Ainsi Fischer (1977 : 73) et Hoppe (1984 : 116) voient dans la récursivité un cas particulier de la stratégie générale qui consiste à ramener un problème à des sous-problèmes plus simples et à ré-

soudre ensuite ceux-ci (stratégie de la réduction), tout en acceptant qu'ici le problème en tant que tel demeure le même après l'étape récursive, mais en le désignant alors comme plus simple, parce que plus proche de la condition d'arrêt. En fait, la simplicité du sous-problème ne joue ici aucun rôle, puisqu'il n'est pas du tout résolu, mais au contraire traité bien prudemment comme une boîte noire (Hoppe 1984 : 117). C'est beaucoup plus la relation existant entre le sous-problème et le problème de départ qui donne pour l'essentiel la solution. Ce n'est pas une réduction de problème, c'est l'introduction d'une relation grâce au pas que marque la récursivité, qui est ici la stratégie employée, ce qui peut se comparer à l'établissement de la formule donnant la somme d'une progression géométrique finie: on donne la progression $1 + q + q^2 + \ldots + q^n$, avec $q \neq 1$ et $n \in \mathbb{N}$; on la multiplie par q et on forme la différence entre la nouvelle progression et l'ancienne:

$$(q - 1)(1 + q + q^2 + \ldots + q^n) = q + q^2 + \ldots + q^{n+1} - 1 - q - \ldots - q^n$$
$$= q^{n+1} - 1$$

d'où la formule cherchée : $1 + q + q^2 + \ldots + q^n = (q^{n+1} - 1)/(q - 1)$.

En résumé: il est bien vrai qu'avec chaque étape de la récursivité la fin du calcul se rapproche, mais l'exécution d'un algorithme auquel se réfèrent Fischer et Hoppe, est cependant un travail de calcul effectué (éventuellement par l'ordinateur) *après que* la solution du problème a été obtenue dans son principe. Pour la solution du problème lui-même on peut dire que si on la compare avec les difficultés de la stratégie de réduction, que je ne veux en aucun cas minimiser (recherche et/ou résolution de sous-problèmes d'un problème ne sont que trop souvent elles-mêmes des problèmes), la stratégie de récursivité, dans son principe, a des exigences plus grandes. En regard de la structure de sous-problèmes, la relation récursive a plutôt un caractère externe, et sa donnée résout le problème sur un plan situé en amont (méta-plan). Les obstacles qui s'opposent à cette façon de comprendre la solution du problème sont connues de chaque enseignant des classes supérieures et de chaque universitaire didacticien des mathématiques qui a une fois essayé d'enseigner le principe de l'induction complète.

La philosophie du Logo ne voit pas seulement dans la récursivité du Logo une technique de programmation ou une stratégie de résolution de problèmes, elle lui attribue un enracinement dans le monde de la vie. Ainsi Papert utilise-t-il la plaisanterie-problème qui demande ce qu'on doit formuler comme second souhait quand on dispose de deux souhaits: naturellement, c'est la possibilité de formuler encore deux souhaits (Papert, 1980 : 74). C'est une ré-

cursivité sans condition d'arrêt, et il n'est pas permis de demander deux souhaits supplémentaires lors du premier souhait de chacune de ses séquences de souhaits, car celui qui vous les accorde ne saurait venir à bout d'une suite de souhaits ouverts, tout au moins s'il ne sait pas traiter deux souhaits en même temps (Hoyles, Noss, 1985: 174). Autres références au monde quotidien des enfants: n'importe quelle action humaine qui se répète et de nombreux exemples dans des chansons, des poèmes et des contes (Lavallade, 1985: 89). De fait, c'est ici l'ancrage dans la vie de l'idée universelle d'itération qui se manifeste, mais en aucun cas sous la forme particulière de la technique récursive.

Pour Hoppe (1984: 104) (pour moi de façon non prouvée), la formulation récursive (d'un programme qui dessine une spirale) est plus concrète que la formulation itérative. On peut cependant souscrire de façon plus restreinte à sa thèse (Hoppe, 1984: 137) suivant laquelle "la description récursive est propre à rendre certains algorithmes et processus informatiques plus facilement saisissables dans leur ensemble, plus courts et plus élégants dans leur représentation." Cependant à mon avis, cet avantage n'a de portée, s'il en a, que seulement en fin de cursus de l'école de formation générale, quand on a déjà des possibilités de formalisation poussées et, liées à ce bonus, on trouvera les difficultés discutées ci-dessus. Ceci fait que je serais plutôt enclin à refuser à la récursivité du Logo une qualification didactique valable pour l'ensemble du domaine de l'école de formation générale.

## REFERENCES

ANZAI Y., UESATO Y., 1982, Learning recursive procedures by middle-school children. In: *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, Ann Arbour, MI.

FISCHER G., 1977, Das Lösen komplexer Problemaufgaben durch naive Benutzer mit Hilfe des interaktiven Programmierens. *Thèse Hambourg.*

HANNINGER J., 1982, Ein Unterrichtsversuch zu geometrischen Erfahrungen in der Grundschule mit Computerhilfe. *Rapport n°18 du département mathématique*, Pädagogische Hochschule Eßlingen.

HAUSSMANN K., 1985, Iteratives und rekursives Denken beim Lösen mathematischer Probleme. In: *Beiträge zum Mathematikunterricht*, Bad Salzdetfurth: Franzbecker 1985 , pp. 146-149.

HOPPE H.U., 1984, *Logo im Mathematikunterricht - ein Beitrag zur Didaktik des interaktiven Programmierens*. Vaterstetten: IWT Verlag.

HOYLES C., NOSS, R., 1985, *Proceedings of the Logo and Mathematics Education Conference*, London: University 1985.
KURLAND D.M., PEA,R.D., 1983, Children's Mental Models of Recursive LOGO Programs. In : *Proceedings of the Fifth Annual Conference of the Cognitive Science Society*, Rochester N.Y.
LAVALLADE D., 1985, In search of Recursion. In: Hoyles, Noss 1985, pp. 57-60.
LÖTHE H., 1985, Mathematik, Informatik, Computeranwendungen - Probleme und Chancen einer Integration. In : *Beiträge zum Mathematikunterricht*, Bad Salzdetfurth: Franzbecker, pp. 195-198.
OBERSCHELP W., 1977, Zum Verhältnis von Mathematik, Informatik und Philosophie. In : *Schriftenreihe des IDM 16*, 35-61.
PAPERT S.,1980, *Mindstorms. Children, Computers, and Powerful Ideas*. New York: Basic Books. (traduction allemande: Basel, Birkhäuser 1982).
SCHREIBER A., 1983, Bemerkungen zur Rolle universeller Ideen im mathematischen Denken. In : *mathematica didactica 5*, 65-76.
TREADAWAY M., 1985, Logo Developments in Suffolk. In: Hoyles, Noss pp. 45-54.
ZIEGENBALG J., 1984, Programmiersprachen als Träger von Grundideen der Informatik. In : *Der mathematische und naturwissenschaftliche Unterricht 37*, 404-415.