LES OBJETS ET LES TYPES DANS UN ENSEIGNEMENT DE LA PROGRAMMATION S'ADRESSANT A DES DÉBUTANTS

Jean Baptiste LAGRANGE

1. LES PROGRESSIONS « NUMÉRIQUE D'ABORD » ET LEURS LIMITES

Beaucoup de progressions s'adressant à des débutants, sont basées sur l'hypothèse selon laquelle les premiers traitements peuvent être enseignés indépendamment du type des objets sur lesquels ils portent, et sans que la question de la codification des données se pose. En fait cette hypothèse conduit à ce que, dans les premiers problèmes, les objets sont tous « déjà codifiés » sous la forme d'entités numériques. Dans les problèmes directs, c'est-à-dire avant qu'interviennent les alternatives et l'itération, l'élève a en fait à établir une « formule » algébrique associant des données numériques et à la traduire dans le langage de programmation ; éventuellement, il a à deviner l'outil mathématique adéquat (*les congruences, par exemple, dans le problème du « jour où tombe Noël »* [ARSAC 80]).

Plus loin dans la progression, on introduit des traitements non triviaux (alternatives en chaînes, itération), sur des entités numériques avec des problèmes tels que sommation, tri d'entiers, factorielle, exponentiation... Ces problèmes imposent des méthodes de conception qui semblent, au moins dans le secondaire, dépasser les capacités d'une bonne partie du public auquel on s'adresse (*Hypothèse de récurrence* [ARSAC 80] *par exemple*). En effet, même si l'élève sait effectuer un calcul correct « à la main », à partir de la définition d'un objet mathématique tel que, par exemple, l'exponentiation, cette connaissance ne l'aide en rien à la conception d'un algorithme ; celle-ci suppose une compréhension de la définition mathématique à un niveau supérieur, acquise seulement plus tard et par une partie des élèves seulement.

Jusque là, le type « chaîne de caractères » apparaît essentiellement dans les instructions de sortie à l'écran, ce qui donne aux chaînes un statut d'objets « affichables, mais pas calculables ». Le type booléen n'apparaît pas, bien que, sans le savoir, les élèves manipulent des écritures du type booléen dans les alternatives. On peut même dire que, dans une certaine mesure, les élèves apprennent à se passer du type booléen, en construisant des structures alternatives plus complexes que s'ils avaient ce type à leur disposition.

Les traitements non triviaux étant supposés installés chez les élèves, on pense ensuite, dans un démarche de généralisation, « transférer » à des entités non numériques les algorithmes (et les méthodes de création) supposées acquises dans le cadre numérique. Mais ce transfert se heurte à des difficultés importantes : d'une

part, vu le niveau des méthodes employées, l'acquisition ne peut être considérée comme effective pour tous les élèves, d'autre part, dans leurs schémas spontanés de résolution, les traitements peuvent difficilement être séparés des objets sur lesquels ils portent (*pour un exemple des difficultés de transfert des structures algorithmiques d'un type d'objet à un autre, voir* [SAMURCAY ROUCHIER 90]).

2. LA CONSTITUTION DE REPRÉSENTATIONS MENTALES comme enjeu d'un enseignement de l'informatique à des débutants.

L'absence de représentations mentales adéquates du dispositif « ordinateur+langage de programmation » est le principal handicap du sujet débutant face à une tâche de programmation. En particulier [HOC 77], les méthodes de programmation descendantes ne peuvent fonctionner avec des représentations trop frustes : le raisonnement au niveau général du problème, que supposent ces méthodes, implique en effet une organisation de la solution en modules, qui doivent pouvoir être développés séparément sans remettre en cause le schéma d'ensemble ; pour envisager cette organisation, le sujet doit impérativement disposer de représentations articulant des capacités d'ensemble du dispositif, et des traitements particuliers. De même la capacité à « laisser en attente la représentation des données » suppose des représentations adaptées des objets et de leurs rapports avec les traitements. [1]

Un enjeu essentiel de l'enseignement à des débutants est donc d'aider à ce que leurs représentations, en s'adaptant aux contraintes de la programmation, évoluent, se diversifient, se coordonnent, et pour cela il est nécessaire que le débutant rencontre les limites de ses représentations spontanées et donc qu'il soit confronté à des problèmes de programmation :

- où la situation proposée lui permette « d'évoquer » des schémas de solutions à partir de Systèmes de Représentation et de Traitement [2] qu'il a construits pour des situations connues ou familières ;
- où la programmation impose de réviser ces schémas de solution , et en même temps d'adapter et de diversifier ses Systèmes de Représentation et de Traitement.

Par exemple, pour résoudre un problème impliquant la recherche d'une information pertinente dans un texte, le sujet devra d'abord écarter l'idée d'un « accès direct » à l'information, qui utiliserait les propriétés sémantiques des

---

1. Pour s'exprimer de façon imagée, le débutant serait, face à ces méthodes, dans la situation d'avoir à faire le plan d'une maison sans connaître la fonction des pièces, les moeurs des habitants, les contraintes de construction.

2. Jean-Michel HOC a introduit dans [HOC 77], et [HOC 87a], les **Systèmes de Représentation et de Traitement** comme modèles des structures mentales que le sujet met en oeuvre à un moment donné d'une tâche donnée: il prend ainsi en compte le fait que ces structures mentales comprennent, de façon organisée, des éléments déclaratifs (particulièrement des schémas de résolution, des règles d'écriture), mais aussi des éléments procéduraux: il s'agit ainsi de "machines mentales" que le sujet fait fonctionner pour anticiper les réponses du dispositif informatique dans une situation donnée. [HOC 87b] étudie plus particulièrement l'emploi par un sujet non programmeur, face à une tâche de programmation nouvelle, de schémas de solution issus de S.R.T. correspondant à un processus de résolution familier (apprentissage par analogie).

éléments du texte, puis ayant par conséquent conçu son schéma de solution comme itératif, il devra donner un statut informatique aux éléments de l'itération : par exemple, l'index sur les éléments du texte devra être distingué de l'élément qu'il pointe, et se voir reconnaître son statut numérique (ordinal) avec les fonctions adéquates (suivant, précédent...).

Conséquences pour les progressions « numérique d'abord »...

Le schéma de progression (numérique d'abord) décrit dans ses grandes lignes ci-dessus ne répond pas à cet enjeu : en effet, les représentations et procédures mentales auxquelles le sujet peut faire appel dans les problèmes de ce type de progression sont celles que l'enseignement des mathématiques a installées ; compte tenu du faible potentiel horaire généralement constaté dans les cursus d'informatique, il semble illusoire de penser agir en profondeur sur ces représentations.

Par exemple, s'agissant du raisonnement par récurrence, sur des objets typiques de l'enseignement des mathématiques, il est clair que la programmation apporte un éclairage supplémentaire à cette notion, mais par contre, on ne voit pas comment, dans le secondaire par exemple, elle pourrait offrir un raccourci permettant de gagner plusieurs années sur l'enseignement des mathématiques.

... et pour l'emploi des progiciels

Un autre schéma souvent avancé est celui où l'on développerait l'usage de progiciels et/ou la pratique du système d'exploitation à l'aide d'interfaces utilisateur comme alternative, ou comme complément à la programmation. Il faut bien voir que les progiciels et les interfaces utilisateur des systèmes d'exploitation sont conçus pour être, au moins dans leurs usages les plus courants, **compatibles** avec les représentations spontanées. C'est pourquoi, s'il semble clair que l'usage de progiciels peut aider la démarche de programmation, en fournissant un cadre "illustratif" de spécificités du traitement informatique, elle ne saurait la remplacer.

Par exemple, l'utilisation de la fonction de recherche d'une chaîne donnée dans un traitement de texte illustre les capacités du dispositif informatique, et ses limitations. De la même façon, la connaissance de ces progiciels permet de donner un cadre intuitif à des problèmes de programmation, comme par exemple, la « justification » d'une ligne de texte en plaçant des espaces jusqu'à obtenir une longueur donnée. Cette connaissance des possibilités des progiciels ne remplace pas la programmation des algorithmes correspondant, qui suppose une intégration bien plus poussée des spécificités du dispositif (en particulier la nécessité et les contraintes d'un langage).

3. UTILISER LA «CODIFICATION» PAR DES TYPES NON NUMÉRIQUES (CHAÎNES, BOOLÉENS).

Dans un enseignement d'informatique destiné à des débutants, on peut se fixer raisonnablement comme objectif d'**obtenir des élèves qu'ils considèrent la programmation comme la description d'un calcul**, ce qui constitue une rupture et un progrès par rapport à une conception spontanée du programme comme suite

d'actions sur l'environnement. En fonction de ce qui précède, et de l'analyse développée dans [LAGRANGE 90], il est clair qu'on doit s'efforcer de trouver des situations où le calcul porte sur des entités non assimilables aux quantités numériques des mathématiques ; on a donc à concevoir des problèmes de modélisation simple de situations connues à l'aide de chaînes de caractères et de booléens. Des propriétés de chacun de ces types, on peut déduire des problèmes génériques, pour lesquels il restera à trouver "une situation connue" convaincante.

Par exemple, pour le type chaîne, on peut penser à des problèmes de manipulation sur des mots ainsi qu'à des problèmes de « recherche translation » à partir d'une chaîne matérialisant une relation entre données et résultats (voir [LAGRANGE 90]*). Dans le cas des booléens, on peut envisager la construction de « fonctions logiques » à partir de situations impliquant des objets à deux états. Des problèmes peuvent également faire intervenir le choix d'un type et la conversion de type : ainsi des entités numériques peuvent avoir à être représentées par la chaîne de leurs chiffres, et une conversion booléen-chaîne peut s'envisager pour régler les questions d'interaction avec l'utilisateur du programme dans le cas où le programme est un calcul sur des booléens*.

4. QUELQUES CONSÉQUENCES DIDACTIQUES

Envisager un type, comme un ensemble structuré de fonctions

Un type de données est pour nous l'analogue d'une théorie en mathématiques : les fonctions de base en sont les axiomes, les fonctions construites à l'aide des fonctions de base en sont les théorèmes. **La programmation de ces fonctions à l'aide des fonctions de base structure le type et en assure la cohérence**, de la même façon que la démonstration des théorèmes dans une théorie mathématique. **Elle permet de faire fonctionner la dialectique outil-objet** [DOUADY 86] : les fonctions sont construites comme outils dans des situations, avant d'être reconnues comme des entités générales (présentes dans le langage de programmation), et étudiées pour leurs propriétés.

Un exemple est la fonction `position` *dans les chaînes de caractères qui intervient dans les problèmes de recherche d'occurrences et d'appartenance, et qui se construit à l'aide des fonctions de base de ce type, et de l'itération.*

Prendre en compte les difficultés des élèves, comme manifestations de représentations "en train de se construire"

Une stratégie mettant en jeu la codification des données ne vise pas à être une "voie royale" supprimant toute difficulté. Au contraire, les hésitations des élèves, leurs premiers essais infructueux sont l'indice que leurs représentations spontanées entrent en contradiction avec les contraintes d'un dispositif informatique, et il est important de travailler sur ces difficultés. Nous avons décrit plusieurs difficultés dans [LAGRANGE 90] : choix du type des données, conception des éléments généraux du langage (affectation, notation fonctionnelle...) quand ils portent sur un type donné, sémantique des fonctions d'un type donné, articulation avec des traitements...

Une attention particulière doit être portée par l'enseignant à la caractérisation de ces difficultés et des représentations sous-jacentes. Cette caractérisation doit en effet en premier lieu lui permettre la construction de problèmes adaptés. Si l'on n'y prend garde, dans de nombreux exemples ou exercices proposés aux élèves, les réponses exactes peuvent être compatibles avec des représentations erronées, et donc contribuer à les renforcer. *Par exemple dans l'écriture (correcte) de la sous-chaîne calculée à partir d'une chaîne donnée de longueur* `L` sans *son dernier caractère* sous-chaîne(chaîne, 1,L-1)`,` *l'écriture* `L-1` *peut parfaitement être comprise par les élèves comme codant l'action d'ôter le dernier caractère (alors que c'est l'argument correspondant à la longueur de la sous-chaîne extraite). Dans ce cas, il paraît important de prévoir des exercices où la sous-chaîne à extraire ne commence pas au premier caractère.*

La caractérisation des représentations sous-jacentes aux difficultés rencontrées doit permettre aussi à l'enseignant de fonder son intervention auprès de l'élève. Si l'enseignant ne peut s'appuyer sur cette connaissance des représentations de l'élève, ou au moins sur des hypothèses dans ce domaine, il intervient essentiellement par des rappels à la syntaxe qui conduisent l'élève à améliorer l'écriture sans que sa conception soit modifiée.

Voici un exemple de ce phénomène : des élèves donnent à des écritures fonctionnelles un statut d'action sur le dispositif (par exemple un appel de la fonction sous-chaîne est compris comme une « extraction » effective de sous-chaîne) ; par conséquent, dans un premier temps, ces écritures, pour ces élèves, se suffisent à elles-mêmes, et donc ils les placent dans le programme en tant qu'action isolée ; mais une écriture fonctionnelle isolée entraîne une erreur de syntaxe et l'enseignant invitant à corriger cette erreur, les élèves incluent alors l'écriture fonctionnelle dans une instruction d'affichage ou une affectation, qui reste sans signification pour eux. La représentation sous-jacente à l'erreur de syntaxe n'est pas modifiée. L'écriture par contre est devenue conforme à la syntaxe, ce qui rend la mise en évidence de l'erreur de conception encore plus difficile.

Dans le même domaine, **la connaissance des représentations des objets chez l'élève est importante au moment de la construction de traitements non triviaux,** comme l'itération : en effet, les représentations mentales des objets interviennent à deux niveaux : pour la conception d'ensemble du traitement (nécessité ou non d'une itération par exemple), mais aussi pour la conception des différents constituants de l'action : partie condition des alternatives et des itérations, variables itératives...

Penser des situations pédagogiques adaptées aux premiers apprentissages

Compte-tenu de l'hétérogénéité des représentations spontanées dans une classe, de la difficulté qu'éprouve l'enseignant, en situation collective, à prendre connaissance et à analyser les représentations des élèves, on peut penser que les phases de travail dirigé, seul ou en petit groupe, sont des situations pédagogiques privilégiées dans les premiers apprentissages. De nombreuses questions restent cependant à approfondir. Quelles sont les apports spécifiques des phases de

recherche sur papier, et des phases sur ordinateur ? La recherche sur papier est-elle seulement le moment où fonctionnent les représentations spontanées, ou les contraintes du dispositif peuvent-elles déjà se manifester par le passage à l'écriture ? Quelles exigences quant à la forme d'expression des solutions peuvent y contribuer ?

On reproche à la recherche en présence de l'ordinateur de favoriser une programmation « empirique » c'est-à-dire une pratique consistant à compenser les défauts visibles d'un programme mal conçu au départ. Mais ce reproche est-il vraiment fondé pour les problèmes des premiers apprentissages qui ne nécessitent pas vraiment un plan d'ensemble difficile à remettre en cause ? Nous proposons de comparer la présence de l'ordinateur et de ses résultats d'exécution à l'aide qu'apporte au raisonnement de l'élève, la figure en géométrie. La figure, comme les essais sur machine, ne constitue pas une preuve, mais en l'absence d'images mentales basées sur une structuration forte du domaine chez le sujet, ils sont un apport pour le raisonnement. Mais de même qu'en géométrie il faut apprendre aux élèves à faire des figures utiles (suffisamment générales, mettant suffisamment en valeur les éléments importants du raisonnement), il faut sans doute apprendre aux élèves à utiliser de façon pertinente la présence de l'ordinateur. Et, tout comme la figure n'est pas le but du raisonnement, le programme n'a d'intérêt que si l'élève est capable de le justifier.

Jean Baptiste LAGRANGE

Centre de Formation à l'Informatique
et aux applications pédagogiques
Université de CAEN
Annexe Vissol 14032 CAEN Cedex

BIBLIOGRAPHIE

ARSAC J. (1980) *Premières leçons de programmation* CEDIC

DOUADY R. (1986) « Jeux de cadres et dialectique outil-objet » *Recherches en didactique des Mathématiques*, Vol 7.2. Editions La Pensée Sauvage

HOC J.M. (1977) « Role of mental representation in learning a programming language » *Int. Man-Machine Studies* n°9

HOC J.M. (1987a) *Psychologie cognitive de la planification* Editions PUG

HOC J.M. (1987b) « L'apprentissage de l'utilisation des dispositifs informatiques par analogie à des situations familières » *Psychologie Française* n°4

LAGRANGE J.B. (1990) « Des situations connues aux traitements sur des données codifiées » *Actes du Second Colloque Francophone de Didactique de l'Informatique* Presses Universitaires de Namur

SAMURCAY R., ROUCHIER A. (1990) « Apprentissage de l'écriture et de l'interprétation des procédures récursives » *Recherches en didactique des Mathématiques*, Vol 10 2.3. Editions la Pensée Sauvage.