PUBLICATIONS MATHÉMATIQUES ET INFORMATIQUES DE RENNES

JEAN-BAPTISTE LAGRANGE

Les contraintes dans l'utilisation d'un langage de commande portant sur des variables et expressions

Publications de l'Institut de recherche mathématiques de Rennes, 1993, fascicule 3
« Fascicule de didactique des mathématiques », , p. 61-86

<http://www.numdam.org/item?id=PSMIR_1993___3_61_0>

LES CONTRAINTES DANS L'UTILISATION D'UN LANGAGE DE COMMANDE PORTANT SUR DES VARIABLES ET EXPRESSIONS

Jean-Baptiste LAGRANGE
IUFM de Rennes

Abstract

Difficulties encountered by pupils using an informatic system as a tool to help mathematics learning, may result from constraints specific to those systems, which are not directly integrated. With many between those means, pupils have to use a control language in order to create generic objects by way of variables and expressions. Analysing researches in that field, one may distinguish two kinds of constraints: operationality of the language, and calculability of the objects.

This paper reports a work of inquiry about initial representations of pupils which do not integrate those constraints, and about tasks and domains where those representations are likely to appear. Inquiry is made through observation of 10th and 11th grade pupils. Problems are set, where distance is great between problem's objects and informatic variables, and therefore, variable's type is String in some problems, and Boolean in another.

Observation indicates that many pupils use String and Ordinal expressions to traduce actions in a way close to ordinary communication, and think of Boolean expressions from everyday "conditions". In consequence, pupils do not give a meaning to assignment. String functions arguments are thought of as information with a vague statute. Confrontation with constraints, by way of problems with that distance between problem's objects and informatic data, appears to be productive.

From those results, directions are deduced in order to analyse pupil's task before a given use of computer in mathematics teaching, and in order to build a suitable preliminary initiation into the informatic tool.

Résumé

Parmi les difficultés rencontrées par des élèves lors de l'utilisation de systèmes informatiques pour des apprentissages mathématiques, nous nous intéressons à celles qui résultent de la non-intégration de contraintes spécifiques de ces systèmes. Dans de nombreuses situations, les élèves ont à créer des objets génériques en formant des expressions sur des variables dans un langage de commande. En analysant des recherches sur ces situations, deux types de contraintes se dégagent: l'opérationnalité du langage, et la calculabilité des objets.

A partir de l'observation d'élèves de l'option informatique des lycées, nous avons travaillé à mettre en évidence des représentations initales résultant d'une insuffisante prise de conscience des contraintes. Dans les problèmes posés aux élèves pour ce travail, les objets sont considérés par leurs propriétés intuitives, et les variables informatiques qui les codifient doivent quant à elles, être "calculées" selon des règles propres à leur type (chaîne, booléen...).

L'observation fait apparaître que beaucoup d'élèves utilisent les expressions chaînes et ordinales pour traduire des actions en s'inspirant du mode habituel de communication. Les expressions booléennes sont comprises seulement lorsqu'elles interviennent dans des conditionnelles. En conséquence, ces élèves ne donnent pas de signification à l'affectation. Parallèlement, les arguments des fonctions chaînes sont conçus comme des informations au statut vague. Des problèmes comme ceux posés pour le travail d'observation, permettent aux élèves de rencontrer les contraintes et d'en prendre conscience progressivement.

Ces résultats conduisent à dégager des orientations pour l'analyse préalable à l'utilisation d'un outil informatique donné dans une situation d'enseignement des mathématiques, et pour la mise en place de l'initiation informatique appropriée à cette situation.

1. Introduction

Les interactions entre apprentissages mathématiques et activités informatiques constituent un champ largement ouvert, allant de l'utilisation de logiciels ne demandant en principe pas de connaissance informatique, jusqu'à la programmation sur des objets mathématiques (nombres, figures, fonctions...). Un grand nombre d'enseignants, de chercheurs, ont placé des espoirs dans les apports des activités informatiques aux apprentissages mathématiques. Parmi ces espoirs, peu se sont d'ores et déjà concrétisés, et par conséquent l'analyse des spécificités pour l'élève de l'environnement informatique reste encore largement à faire.

Un moyen informatique dans une situation d'action, participe à l'environnement face auquel le sujet développe ses connaissances afin de contrôler la situation, et que Brousseau (1989) qualifie de milieu de référence. Il apporte en effet des facilités et contraintes nouvelles dans la confrontation du sujet au milieu. Ces facilités et contraintes, constituent ainsi un nouveau "cadre" à côté des cadres numériques, graphiques... (Douady, 1986) qui peut permettre un éclairage différent sur les notions,

et la mise en oeuvre de procédures nouvelles. Cependant certaines contraintes de l'outil informatique peuvent aussi se révéler sans rapport avec le contenu mathématique visé, et, si elles ne sont pas intégrées spontanément, faire écran à l'activité mathématique.

Dans toute une gamme de tâches, allant de l'utilisation de calculatrices programmables ou de tableurs à la production de tracés géométriques, le dispositif se commande par un langage utilisant des variables, et l'élève a à créer des objets variables par le biais d'expressions. Les difficultés rencontrées par des élèves dans ces tâches (même s'ils ont déjà eu une initiation aux variables et expressions dans l'enseignement de l'algèbre), conduisent à penser que des contraintes liées à ces notions dans un langage de commande sont spécifiques et non intégrées spontanément: un témoignage de ces difficultés est rapporté par Balacheff, Capponi, (1989), à partir de l'observation d'élèves de 14-15 ans dans l'utilisation d'un tableur.

Le but de cet article est d'identifier des contraintes spécifiques des langages de commande comportant des variables et des expressions, d'apporter un éclairage sur les conduites et représentations des élèves débutants confrontés à ces contraintes, et de rechercher des conditions didactiques permettant d'agir sur ces représentations.

Au paragraphe 2, nous analysons brièvement plusieurs recherches sur différents types de langages: une recherche récente (Artigues, 1991) concernant l'utilisation d'un langage de commande sur des objets géométriques, une étude (Samurçay, 1985) de l'utilisation d'un langage de programmation impératif sur des variables numériques, et d'autres concernant la compréhension de la gestion des variables (Dupuis, Guin, 1989), et de la notion de fonction (Lagrange 1986) dans l'utilisation d'un langage de commande applicatif.

Au paragraphe 3, deux types de contraintes sont identifiés à partir de cette analyse, et les objectifs de recherche sont developpés.

Les paragraphe 4 et 5 rapportent l'étude expérimentale menée en fonction de ces objectifs. Les résultats sont discutés au paragraphe 6.

2. Les difficultés des élèves à travers plusieurs observations (Méta-analyse)

2.1. Expressions sur des objets géométriques et contraintes du dispositif

Artigues (1991) a étudié l'utilisation, dans le cadre de l'enseignement de la géométrie, d'un logiciel (EUCLIDE) constitué d'un jeu de fonctions définies dans le langage LOGO, dont les arguments et les valeurs sont des objets géométriques. Le langage permet de définir et afficher des éléments géométriques variables, calculés à l'aide d'expressions. Les élèves de

13-14 ans (8ème grade) avaient pour tâche d'utiliser le langage afin d'obtenir le tracé d'une configuration géométrique demandée.

Les erreurs les plus nombreuses sont moyennement ou peu en rapport avec le contenu mathématique: en fait, les élèves ne prennent pas en compte certaines contraintes, proprement informatiques, du langage:

- Le mode de définition des objets. Certains élèves calquent leur utilisation de l'instruction `SOIT variable <expression>` sur le mode de définition mathématique: *Soit* `variable` *telle que* `<propriété>`

  Or, dans EUCLIDE, l'instruction `SOIT variable <expression>` est en fait une affectation, impliquant un changement d'état du dispositif[1], alors que la proposition *Soit* `variable` *telle que* `<propriété>` est quant à elle, un élément du discours (méta)mathématique visant à faciliter la suite de l'exposé, et les règles régissant `<propriété>` peuvent être beaucoup plus souples que dans le logiciel. Par exemple, *Soit* D *tel que* B *milieu de* AD ne se traduit pas directement dans EUCLIDE; il faut employer la fonction SYMP qui rend le symétrique d'un point donné par rapport à un point donné:

  ```
  SOIT "D SYMP :B :A
  ```

- La distinction entre affichage et définition. Certains élèves emploient la primitive `DES <objet>` qui affiche `<objet>` à l'écran en lieu et place de la primitive `SOIT`.

  L'affichage d'un objet, qui peut être une variable ou une expression est une opération tangible pour l'élève. La définition (qui est en fait une affectation) modifie l'état interne du dispositif, mais ne produit aucun effet externe. Les élèves ne distinguent donc pas spontanément état interne et effets externes.

- Les contraintes syntaxiques. Par exemple, l'ordre des arguments des fonctions à plusieurs variables est critique, à la différence du langage géométrique habituel.

  L'appel d'une fonction à plusieurs variables produit un calcul dans le dispositif, où les arguments sont traités selon leur position. Les élèves se réfèrent en fait au contexte géométrique habituel où la particularisation des objets se fait par d'autres moyens: par exemple, dans la proposition *le symétrique de* `A` *par rapport à* `B` les deux points apparaissent chacun avec leur statut qui n'est pas lié à leur position.

- La nécessité de définir tous les objets utilisés (en utilisant en fait l'affectation).

---

[1]Voir Annexe 1.

Le contexte géométrique n'impose pas de donner un nom systématiquement à tous les objets utilisés. Par exemple A B C D étant des points donnés distincts, on peut parler du *point d'intersection des droites* AB *et* CD sans donner un nom à ces droites. Un expressions analogue en EUCLIDE serait:
`INTDD (DRPP :A :B) (DRPP :C :D)`[2].
Ce type d'expression composée n'appartenant pas au répertoire initial des élèves, il leur faut nommer explicitement chaque droite en l'affectant à une variable, ce qui est une contrainte dont ils ne peuvent apercevoir la signification.

2.2. Les contraintes dans le calcul d'expressions algébriques

Il est intéressant de confronter ces résultats sur un langage "géométrique", à ceux d'une étude sur la notion de variable dans la programmation informatique (Samurçay, 1985). Cette étude est basée sur les réponses à des épreuves écrites concernant la programmation du calcul d'expressions algébriques pour des valeurs données des variables (par exemple, programmer le calcul de x^n-y^n pour des valeurs de x, y, et n entrées par l'utilisateur). Chez des élèves de lycée après 15h de programmation, on rencontre comme ci-dessus des difficultés liées à l'affectation et à la signification des expressions:

- Initialisation des variables par lecture plutôt que par affectation

  Les élèves font peu de différence entre les variables externes (dont la valeur est déterminée par l'utilisateur, donc doit être "lue") et les variables internes (compteur de boucle, accumulateur, dont la valeur initiale doit être fixée par affectation). Ils utilisent peu l'affectation, préférant l'instruction de lecture à laquelle ils donnent plus facilement un sens.
- Signification de l'expression booléenne constituant la condition d'arrêt

  Pour beaucoup d'élèves, cette expression sert à indiquer que "le calcul est fini", sans qu'ils aient conscience de la nécessité du calcul par l'ordinateur d'une valeur de vérité pour la condition d'arrêt (par exemple, pour le calcul du produit de deux entiers x, y par addition itérée, un élève propose la boucle `repeat.... until x*y`).

La difficulté à donner un sens à l'affectation est commune aux deux situations. De même, comme les expressions géométriques, l'expression booléenne de la condition de

---

[2] `INTDD :D1 :D2` rend le point d'intersersection des droites `:D1` et `:D2`
`DRPP :A :B` rend la droite passant par les points `:A` et `:B`

sortie d'itération est formée en s'inspirant du contexte mathématique, sans prise de conscience des contraintes du calcul par le dispositif.

2.3. Le fonctionnement du dispositif dans un langage applicatif

Les observations ci-dessus pourraient sembler liées à des contraintes spécifiques des langages impératifs, où l'affectation `variable←<expression>` est le mode d'évolution de l'état interne du dispositif. Dans d'autres types de langages de commande, l'évolution de l'état interne se fait par d'autres constructions: par exemple, dans un langage applicatif, si l'on a défini une procédure `proc`, l'instruction `proc <expression>` provoque l'évaluation de `<expression>` et la transmission de la valeur résultante à la procédure `proc`[3].

Dans ce cadre, des difficultés à concevoir l'évolution du dispositif apparaissent également. Dupuis et Guin (1989) ont demandé à des élèves de programmer des tracés sous forme de procédures s'emboîtant. Un tracé `DESSIN2` mettant en jeu le tracé précédent `DESSIN1` à une échelle inférieure, par exemple 1/3, une solution conforme au caractère applicatif du langage consiste à appeler dans la définition de `DESSIN2` :C, la procédure `DESSIN1` avec l'argument `:C/3`. En fait la majorité des élèves préfère définir `DESSIN2` avec deux arguments `:C` et `:D`, où `:D` est l'argument avec lequel `DESSIN1` est appelé dans `DESSIN2`. Ils fixent les valeurs respectives de ces arguments, seulement au moment de l'exécution en calculant eux même la valeur de `D` à partir de celle qu'ils donnent à `C` (par exemple, ils exécutent `DESSIN2 12 4`).

Cette conduite est tout à fait analogue à celle des élèves observés par Samurçay (1985) qui préfèrent dans leur majorité initialiser les variables par lecture plutôt que par affectation. Elle témoigne de l'incertitude éprouvée par les élèves sur la façon dont le dispositif gère les expressions, lorsque celles-ci sont des arguments de procédures. Par exemple `DESSIN1` comportant l'instruction `Avance :C/3`, les élèves ne savent pas l'effet de cette instruction lors de l'appel de `DESSIN1` avec l'argument `:C/3`, alors même qu'ils savent "réduire" l'expression algébrique $\frac{\frac{c}{3}}{3}$ en $\frac{c}{9}$.

2.4. Affichage et état interne du dispositif

---

[3]En annexe 1, nous développons davantage les différences de principes des langages impératifs et applicatifs.

Dans ce même contexte d'un langage applicatif, nous avons nous-même observé des conduites témoignant d'incertitudes quant aux rôles respectifs des instructions d'affichage (qui agissent sur l'écran, mais ne modifient pas l'état interne du dispositif) et du mécanisme d'évolution de cet état interne. On a demandé à des étudiants en mathématiques post DEUG (3ème année d'université) d'indiquer l'effet de la composition de procédures et de fonctions à partir de la question suivante: «CARRE (N) étant une procédure *affichant* le carré de son argument sur le terminal, et RAC (N) une fonction *rendant* comme résultat la racine carrée de son argument, quel est l'effet des écritures suivantes CARRE (RAC (256)) et RAC (CARRE (-25)) ?» Il est clair que la première écriture conduit au calcul de la racine carrée de 256, soit 16, le carré de cette valeur, soit 256 étant ensuite affiché à l'écran. La seconde écriture conduit à l'affichage du carré de -25, soit 625, ensuite la fonction RAC étant appelée sans argument, le message d'erreur correspondant est affiché à l'écran. Les réponses des étudiants montrent qu'ils n'aperçoivent pas dans leur majorité le fonctionnement spécifique de RAC et de CARRE dans leur composition. En effet, 6 réponses sur 13 indiquent une erreur à l'exécution de CARRE (RAC (256)), et 9 réponses sur 13 ne décrivent pas de façon appropriée l'effet de RAC (CARRE (-25)) (Lagrange, 1986).

3. Identification de contraintes et objectifs de recherche

L'ensemble des observations précédentes s'interprète en terme de contraintes non intégrées dans les représentations qu'ont les élèves du dispositif informatique.

Nous nous inspirons de méthodes de recherche utilisées pour étudier l'acquisition des structures algorithmiques en programmation. Par exemple, des erreurs de programmation de débutants confrontés à la coordination d'alternatives dans un programme, ont pu être interprétées en postulant l'existence de modèles implicites du traitement des instructions conditionnelles (Rogalski, Hé, 1989). L'identification de ces représentations est utile à l'enseignant pour analyser a priori la tâche demandée à l'élève, et pour orienter l'activité pédagogique vers la construction de représentations plus adaptées. On a pu ainsi proposer des problèmes où le modèle erroné du traitement des instructions conditionnelles conduit à un programme dont l'exécution ne donne manifestement pas le résultat attendu, confrontant ainsi l'élève à un conflit, et engageant un processus de modification des représentations.

Chez des élèves construisant un traitement à l'aide de variables et d'expression, les représentations du dispositif apparaissent, à partir des observations ci-dessus, marquées par des incertitudes concernant l'effet de ces éléments du langage. Ces incertitudes

viennent de ce que les élèves n'ont pas encore pris suffisamment conscience de deux types de contraintes:

- L'opérationnalité du langage

  Dans un langage de commande, variables et expressions s'insèrent dans un processus de changements d'état d'un dispositif. Le langage est *opérationnel* en ce sens qu'il détermine ce processus. Le changement d'état par affectation caractérise les langages impératifs, alors que les langages applicatifs utilisent les paramètres expression dans les procédures[4]. Les incertitudes chez les élèves se traduisent par des conduites où l'attribution de valeurs aux variables est confiée à l'utilisateur par action sur le périphérique d'entrée (clavier), plutôt qu'au dispositif. Elles se traduisent également par des confusions entre l'information qui apparaît sur le périphérique de sortie (l'écran) et le changement d'état interne du dispositif.

- La calculabilité des objets

  Le langage de commande décrit les changements d'états selon des règles formelles. Ainsi l'utilisation de variables et d'expressions du langage comme solution à un problème dans un domaine donné, suppose de réduire la connaissance sur le domaine à un jeu de fonctions, dont la combinaison permet de rendre compte des relations entre objets dans le domaine. Cette combinaison de fonctions est un *calcul*, ne portant pas nécessairement sur des objets numériques. Par exemple en EUCLIDE, les objets géométriques se *calculent* à partir d'éléments géométriques donnés et d'un jeu de fonctions. Comme vu ci-dessus, certaines règles de calcul des expressions (par exemple la position des arguments des fonctions) peuvent ne pas être intégrées, à cause de leur nature formelle.

Deux types de contraintes sont ainsi identifiés. Les observations rapportées ci-dessus permettent de penser qu'elles ne sont pas intégrées spontanément. L'objectif de l'étude expérimentale qui suit est d'apporter des réponses à trois questions, dont deux sont relatives aux représentations du dispositif, et la troisième au contexte d'apprentissage:

1. Lorsque les élèves ignorent les contraintes d'opérationalité, quel rôle donnent-ils aux différents éléments du langage (affectation, variables, expressions...) ?

---

[4]Voir annexe 1.

2. Comment les élèves pensent-ils l'évaluation des expressions par le dispositif lorsque cette évaluation n'est pas, pour eux, un calcul ?
3. En tenant compte des apprentissages préalables des élèves, existe t'il des tâches, des domaines où on peut leur faire rencontrer assez vite les contraintes, de façon à mettre au jour des représentations erronées et à agir sur elles ?

4. Etude expérimentale

Ces trois questions amènent à construire une expérimentation auprès d'élèves, mettant en jeu une tâche relative à un dispositif informatique. Par souci de généricité nous nous intéressons à des groupements d'élèves ayant reçu un enseignement non différencié. Il se trouve qu'en France, un enseignement d'informatique est offert dans de telles conditions à la fin de la scolarité obligatoire (15-16 ans). Les élèves ont ainsi reçu pendant deux ans une initiation à l'algèbre (expressions et équations simples). Dans ces conditions, comme nous l'expliquons ci-dessous, la tâche la mieux adaptée est la résolution de problèmes de programmation ne mettant pas en jeu d'autre structure algorithmique que la simple séquentialité, et impliquant des types d'objets n'ayant pas reçu un statut algébrique au cours de la scolarité: ordinaux, chaînes, valeurs logiques.

4.1. Tâche

On peut en effet penser que les contraintes d'opérationnalité du langage apparaîtront plus facilement si le sujet a en charge la gestion de la totalité du processus de changements d'états, c'est-à-dire s'il a à résoudre un problème de programmation. La résolution de ce type de problème suppose la construction d'une structure algorithmique qui peut être la simple séquentialité ou une structure plus complexe, telle que l'itération, ou la récursivité. Il est raisonnable de se limiter à des problèmes mettant en jeu la seule séquentialité, car l'acquisition de structures complexes doit s'envisager dans la durée et recèle des difficultés spécifiques (Laborde et alter, 1985; Samurçay, Rouchier, 1990) dont on peut s'attendre à ce qu'elles interfèrent avec les difficultés dues aux contraintes identifiées plus haut.

De même, les difficultés avec l'affectation dans le cadre impératif paraissent plus faciles à mettre en évidence et à analyser que les difficultés de gestion des relations en programmation applicative.

4.2. Objets impliqués dans la tâche et type des variables

Pour que la calculabilité des variables soit effectivement en jeu dans l'activité de programmation, il est important qu'existe une "distance" entre des objets du problèmes,

et les variables informatiques qui les codifient. Les objets doivent pouvoir être considérés par leurs propriétés intuitives, alors que les variables sont "calculées" selon les règles formelles du type (numérique, chaîne, booléen, ...) auxquelles elles appartiennent.

De manière à établir cette distance, nous avons fait le choix de problèmes portant d'une part sur des mots (en tant que suite signifiante de caractères) et d'autre part sur des valeurs logiques représentatives d'une situation concrète. En effet, pour les élèves que nous considérons, compte tenu de leurs apprentissages antérieurs en arithmétique puis en l'algèbre, la notion de calcul, en tant que manipulation formelle permettant de résoudre des problèmes, s'entend appliquée à des objets numériques représentatifs de grandeurs mesurables. Par contre, modifier un mot selon une règle donnée, combiner des propositions logiques pour prendre une décision, font partie pour les élèves, d'activités intuitives qui n'appellent pas de leur part l'emploi de formules, et donc pour lesquels la calculabilité est à construire.

Un mode de traitement nouveau pour des objets familiers
(traiter des mots à l'aide des fonctions sur les chaînes)

Les chaînes de caractères sont l'équivalent informatique des notions intuitives de mot ou de texte; elles en diffèrent en ce sens que le traitement informatique des chaînes est une simple décomposition-recomposition du mot (un calcul sur le mot) ne faisant intervenir aucune forme de sémantique. Les fonctions à partir desquelles sont formées les expressions réalisant ce traitement peuvent être limitées à trois "primitives" (voir Annexe 2).

A la différence des expressions algébriques de l'enseignement mathématique, les expressions chaîne sont formées de variables de type hétérogène et utilisent la notation fonctionnelle préfixée. Le traitement des chaînes met ainsi en jeu la notion d'expression de façon différente et plus générale.

Des calculs sur des objets non encore algébrisés
Les ordinaux

Traiter des mots à l'aide des fonctions sur les chaînes, implique par ailleurs d'utiliser un (des) index sur des chaînes, c'est à dire une(des) variable(s) de nature ordinale. D'un point de vue informatique, ces variables sont de type entier, comme les variables cardinales, mais on peut penser que des questions de calculabilité vont se poser pour les élèves. En effet, les ordinaux n'apparaissent pas dans la scolarité obligatoire avec un statut algébrique, on rencontre en mathématiques des variables ordinales seulement

beaucoup plus tard, par exemple avec les changements d'indice dans les suites. La construction d'une compréhension "calculable" des ordinaux peut donc s'étudier en parallèle avec celle des chaînes.

Les booléens

Le type booléen est celui des variables logiques, c'est à dire des variables à deux valeurs (`Vrai` et `Faux`). Un jeu de "connecteurs propositionnels" (`ET`, `NON`, `OU`...) permet de former des expressions logiques, concrétisant ainsi la notion de calcul sur des valeurs logiques. Considérer les propositions logiques comme calculables conduit par conséquent à opérer une réduction drastique du contenu sémantique qui leur est attaché dans le contexte intuitif.

4.3. Les élèves

De façon à éliminer le biais dû à des motivations ou à des acquisitions particulières, l'étude expérimentale a été menée auprès d'un public sans sélection particulière, participant au cursus scolaire commun. Les observations ont eu lieu au cours d'un enseignement d'informatique optionnel, mais ouvert à tous les élèves de l'enseignement secondaire long français: l'option informatique des lycées. Les élèves observés sur les chaînes de caractères (observation rapportée ci-dessous paragraphes 5.1 et 5.2) étaient en première année (15/16 ans), les élèves observés sur les booléens (observation rapportée ci-dessous paragraphe 5.3) étaient en seconde année (16/17 ans). L'enseignement reçu, conforme aux textes officiels (MEN 87) comportait une initiation aux types observés, mais sans prise en compte particulière des contraintes repérées ci-dessus.

5. Problèmes et analyse des réponses des élèves

Pour deux problèmes sur les mots, et un problème sur les valeurs de vérité, correspondant chacun à des contraintes spécifiques, nous présentons la tâche demandée, les élèves auxquels il a été posé, et nous analysons les réponses (succès et types de formulations erronées).

5.1. Opérationnalité du langage avec les chaînes et les ordinaux

Tâche

Le problème ONJOURB (énoncé figure 1) porte principalement sur l'opérationnalité du langage. Les expressions à former u[illegible]ent la fonction sous-chaîne, pour calculer le premier caractère du mot donné et la chaîne "reste", puis la concaténation. La difficulté vient de la complexité de l'expression résultat: elle s'exprime sous forme d'une

expression composée sur plusieurs niveaux (solution de droite, figure 1), ou peut être décomposée à l'aide d'affectations à des variables intermédiaires (solution de gauche, figure 1).

Le problème **ONJOURB**

Ecris un programme: l'utilisateur entre un mot au clavier, l'ordinateur affiche le mot avec la première lettre passée à la fin.

Exemples de solutions

```
Lire Mot
L ← longueur (Mot)
Début ← souschaîne(Mot,1,1)
Reste←souschaîne(Mot,2,L-1)
Résultat ← Reste + Début
Afficher Résultat
```

```
Lire Mot
Résultat←
 souschaîne(Mot,2,longueur(Mot)-1)
 +souschaîne(Mot,1,1)
Afficher Résultat
```

Figure 1

Elèves

Le problème a été posé à une classe de 10 élèves de première année d'option informatique dans le cadre d'une épreuve sur papier, suivie d'entretiens en présence de l'ordinateur. Les élèves avaient eu 25 heures d'enseignement, comportant une initiation aux fonctions sur les chaînes.

Analyse des réponses

Problème ONJOURB	Succès 5/10		
Types d'écritures erronées	Fréquence	Interprétation	Contrainte non-intégrée
Expression chaîne isolée dans une ligne de programme	1/10	Rôle de l'affectation ignoré, conception préopératoire des expressions et sous-expressions chaînes	Opérationnalité du langage
Expression complexe syntaxiquement incorrecte, s'interprétant en terme d'actions successives	4/10		

Figure 2

Le dépouillement des réponses (figure 2) montre qu'un élève sur deux réussit: Ces élèves construisent une affectation pour chaque partie du mot, comme dans la solution

de gauche (figure 1), mais évitent le calcul du résultat par concaténation en faisant afficher successivement la variable "reste", puis la variable "début".

Parmi les formulations erronées (figure 2), on rencontre chez un élève une ligne de programme comportant l'expression isolée `sous-chaîne(Mot,1,1)` sans affectation ni instruction d'affichage. Ici l'élève indique sa volonté de "séparer" ou "extraire" le premier caractère: l'expression n'est pas comprise comme un objet potentiel, mais comme agissant sur des objets internes au dispositif, sans que soit perçue la nécessité de nommer le résultat de l'action.

Dans la résolution de problèmes analogues sur les chaînes, on rencontre d'ailleurs assez souvent ce type d'erreur sous une forme moins évidente: l'élève affecte l'expression à une variable qui n'est pas réutilisée dans la suite, ou construit l'affichage de l'expression, sans que cet affichage ait une signification dans la résolution. On peut penser que là aussi, pour l'élève, c'est l'expression qui agit en tant que telle pour modifier l'état, mais l'attention de l'élève a été attirée par les messages d'erreur, ou les remarques du professeur, sur la nécessité de ne pas laisser d'expression isolée. L'élève a intégré la contrainte syntaxique sans améliorer sa compréhension du traitement des expressions par le dispositif.

Les autres réponses erronées sont celles d'élèves qui construisent une seule expression composée. Ces réponses peuvent comporter plusieurs erreurs de syntaxe; elles ne résultent cependant pas d'un simple oubli des règles de syntaxe: pour l'élève la réponse s'explique par la signification qu'il donne à l'expression composée.

Par exemple, une élève écrit

```
A←sous-chaîne(X,2,L + sous-chaîne (1,1))
Afficher A
```

En entretien elle explique sa réponse : "*J'avais écrit* BONJOUR, *donc* X, *à partir de la deuxième, on prend la longueur, plus la première lettre*". La sous-expression `L + sous-chaîne (1,1)`, troisième argument de la fonction sous-chaîne, n'est pas pour l'élève, un simple objet cardinal potentiel: il exprime la façon dont la chaîne résultat doit se terminer. `L` n'est pas seulement le cardinal longueur de la chaîne donnée, mais représente "la chaîne jusqu'à la fin" et le signe + a une signification intermédiaire entre l'addition et la concaténation.

L'observateur lui ayant fait remarquer que `L` n'a pas de valeur; l'élève remplace alors cette variable par `longueur(X)` dans l'expression, ce qui la rend encore plus

complexe. Elle ne pense pas à construire une affectation à la variable L, ce qui aurait permis de maintenir la ligne en l'état.

La compréhension préopératoire des expressions coïncide donc avec l'absence de prise de conscience du rôle de l'affectation : incapable de construire une décomposition à l'aide de l'affectation, l'élève est amené à construire des expressions composées auxquelles il ne peut donner un sens adéquat; réciproquement, sa conception des expressions comme codage d'actions minore pour lui le rôle de l'affectation dans les changements d'état du dispositif.

5.2. Calculabilité des chaînes et des ordinaux

Tâche

Dans le problème BONJROU (figure 3), la décomposition du calcul à l'aide d'affectations est imposée, ainsi que l'emploi de la concaténation pour le calcul du résultat. L'observation projetée portait en effet principalement sur la calculabilité des chaînes et des ordinaux

Le problème BONJROU

On veut un programme tel que: l'utilisateur entre un mot X, puis un nombre N inférieur à la longueur de X. L'ordinateur affiche une chaîne R formée à partir de X, mais où la dernière lettre de X apparaît avancée en Nième position. Exemple: X vaut 'BONJOUR', N vaut 5, R vaut 'BONJROU'.
Complète le programme suivant:

```
Lire X
L ← longueur (X)
A ← sous-chaîne(X,...,...)  Remarque  A est la première partie de X
B ← sous-chaîne(X,...,...)  Remarque  B est la seconde partie de X
C ← sous-chaîne(X,...,...)  Remarque  C est le dernier caractère de X
R ← .......
Afficher R
```

Réponses correctes

```
A ← sous-chaîne(X,1,N-1)
B ← sous-chaîne(X,N,L-N)
C ← sous-chaîne(X,L,1)
R ← A + C + B
```

Figure 3

Elèves

La forme fermée a permis que le problème soit posé à davantage d'élèves que le précédent. Il fait partie d'une épreuve sur papier passée dans trois classes de l'option

informatique des lycées (54 élèves en tout), en fin de première année, soit à l'issue de 75 heures d'enseignement.

Analyse des réponses

Problème BONJROU		Succès 2/54	
Types d'écritures erronées	Fréquence	Interprétation	Contrainte non-intégrée
Erreurs sur position dans A Erreurs sur longueur dans A	23/54 35/54	Expressions comprises comme codage d'actions Difficulté à coordonner les significations cardinales et ordinales de la fonction longueur	Opérationnalité du langage, calculabilité des ordinaux
Erreurs sur position dans B Erreurs sur longueur dans B	34/54 45/54		
Erreurs diverses dans C dont position L-1	40/54 17/54		
Erreurs dans le calcul du résultat R dont concaténation dans l'ordre initial	36/54 12/54	Arguments compris comme des informations au statut imprécis.	Calculabilité des chaînes

Figure 4

La position (ordinale) de départ, et la longueur (cardinale), demandées pour le calcul de A pouvaient sembler faciles pour des élèves ayant eu un enseignement sur les chaînes de caractères, mais les erreurs (figure 4) sont nombreuses aussi bien pour la position que pour la longueur. Un couple relativement courant (8 parmi les 34 réponses erronées) est L, N. En cohérence avec l'analyse faite plus haut, ce couple pourrait signifier "dans le mot tout entier, prendre jusqu'au Nième caractère".

La proportion d'erreur sur la position dans le calcul de B est du même ordre de grandeur que pour le calcul de A. Cette proportion augmente notablement pour la longueur. Celle-ci pose en effet une question difficile: l'expression de la longueur d'une chaîne comprise entre deux positions variables. Il se confirme que l'utilisation des écritures algébriques sur les ordinaux n'est pas maîtrisée par ces élèves, bien qu'ils aient reçu un enseignement d'algèbre, portant, il est vrai, sur des variables représentatives de quantités cardinales.

Dans le calcul de C, les élèves échouent principalement sur la position. Une erreur largement répandue est L-1 (au lieu de L)[5]. Dans d'autres classes et dans d'autres exercices, cette erreur s'est avérée assez courante et résistante. Il faut considérer que L-1 est ici une sous-expression d'une expression composée, structure dans laquelle sont rencontrées des difficultés à donner une signification adéquate aux expressions. Ici L-1 est clairement à mettre en rapport avec l'action de "séparer un caractère à la fin", L étant à nouveau considéré non comme un nombre mais comme "toute la chaîne", ou "la fin de la chaîne".

Le calcul du résultat R est incorrect dans la majorité des réponses (36/54). La réponse erronée la plus courante (12/36) est la concaténation des trois variables A, B et C dans l'ordre initial. Ces élèves indiquent ainsi que le résultat est "formé à l'aide" des sous-chaînes A, B et C, et ne prennent pas conscience de ce que l'ordre des arguments est critique pour le calcul du résultat par le dispositif. Cet exemple de réponse, parmi d'autres, conduit à penser que beaucoup d'élèves ne conçoivent pas les expressions sur les chaînes comme s'insérant dans un calcul, mais comprennent les arguments des fonctions comme des "informations" au statut vague auxquelles le dispositif est supposé donner une signification conforme au but visé.

5.3. Opérationnalité du langage et calculabilité des variables logiques

Tâche

Le problème INVITATION (figure 5) a été introduit pour étudier la capacité d'élèves de l'option informatique ayant reçu un enseignement sur les booléens, à construire des expressions booléennes pour modéliser une situation familière. La variable correspondant à un des "amis" ayant reçu la valeur Vrai si celui est inclus dans "'invitation", et Faux dans le cas contraire, la valeur de vérité de chaque clause se calcule par une expression booléenne faisant intervenir un ou plusieurs connecteurs propositionnels (voir solution pour la clause 1 figure 5). Une particularité du traitement informatique est que le connecteur propositionnel "équivalent" est en fait un cas particulier de l'opérateur de comparaison "égalité" qui s'applique à tous les types (voir solution pour la clause 2 figure 5). Comme pour les chaînes, l'affectation à des variables intermédiaires permet de décomposer des expressions complexes: dans la clause 3 cette décomposition est assez naturelle à partir de l'énoncé de la condition en deux parties.

---

[5]Il a été vérifié que, à une exception près, les élèves qui donnent L-1 comme position du dernier caractère d'une chaîne de longuer L, donnent bien 1 et non 0 comme position du premier caractère.

Le problème INVITATION

J'ai 5 amis : Marie, Marc, Luc, Janine et Jean.
Je souhaite les inviter à dîner, mais il y a des incompatibilités d'humeur et des préférences, que je traduis par les "clauses" suivantes :

- clause 1 : "Marie et Jean ne s'entendent pas : il ne faut pas les inviter ensemble ".
- clause 2 : "Marc et Marie ne viendront pas l'un sans l'autre : si j'invite l'un , il faut que j'invite l'autre".
- clause 3 : "Si j'invite Janine, il faut que j'invite Luc ou Jean, mais on ne peut pas les inviter tous les trois ensemble"

On veut écrire un programme qui permet de savoir si une invitation (c'est-à-dire un groupe d'amis à inviter) est possible :
Ecris un programme qui pose 5 questions "On invite Marie (O/N)" ... puis calcule et affiche la valeur de vérité de chaque clause (c'est-à-dire si la clause est ou non respectée).

Solution (Calcul des clauses sous forme Booléenne)

```
Clause1 ← NON (Marie ET Jean)

Clause2 ← (Marie = Marc)

Clause3.1 ← NON Janine OU ( Luc OU Jean)
Clause3.2 ← NON (Janine ET Luc ET Jean)
Clause3 ← Clause3.1 ET Clause3.2
```

Figure 5

Elèves

Le problème a été posé dans une épreuve sur papier à une classe de 9 élèves lors de leur seconde année d'option informatique, après une centaine d'heures d'enseignement, comportant une initiation au type booléen; en particulier, les élèves avaient construit une expression booléenne pour rendre compte du fonctionnement d'un circuit électrique simple (deux interrupteurs en série formant un circuit en parallèle avec un troisième interrupteur). Comme dans beaucoup de classes de seconde année, les élèves étaient, à une exception près, issus de classes scientifiques.

Analyse des réponses

L'analyse des protocoles (figure 6) montre qu'en fait, les élèves ont abandonné la résolution par construction d'expressions booléennes, pour l'utilisation de structures alternatives

SI <condition> ALORS <action1> SINON <action2>.

Les réponses au problème INVITATION				
Protocole	Clauses programmées	Forme de programmation	Connecteurs employés	Type des données
A1	1 (correct) 2 (faux)	Affichage conditionnel avec condition composée	ET	numérique
A2				booléens
A3	1,2,3 (manque initialisation du résultat)	Alternatives incomplètes en chaîne, remplacement		booléen dans l'algorithme, numérique dans le programme
A4 (3 élèves)	1,2(correct) 3 (faux)	Remplacement conditionnel avec condition composée	ET, DIFFERENT DE	numérique
A5				booléens
A6	1 (confus)	Affichage conditionnel avec condition composée	ET	booléens
A7	1,2(correct) 3 (faux)	(??) conditionnel avec condition composée	ET, DIFFERENT DE + (addition)	numérique

Figure 6

Certains élèves emploient cependant des connecteurs propositionnels pour former des *conditions composées*, ce qui leur permet d'éviter la lourdeur des alternatives en chaîne. Dans certains cas, l'alternative est *incomplète* (pas de partie SINON...) la partie <action> de l'alternative étant constituée par l'affectation d'une valeur de vérité. Le programme est ainsi davantage conforme à l'énoncé que le simple affichage d'un message, et permet plus simplement de construire une décomposition par alternatives successives dans le cas de la clause 3.

Si l'on croise les variables *condition* et *action*, on obtient les quatre solutions types de la figure 7. La solution type *Alternatives complètes en chaîne, affichage* est celle qui met le moins en jeu les connaissances sur les valeurs logiques. Elle n'est pas présente dans les protocoles: elle devient en effet très lourde à partir de la clause 2.

Problème INVITATION.

Quatre solutions-types utilisant la structure alternative, classées selon la partie <condition>, *et la partie* <action>

	Action: affichage d'un "message"	Action: affectation d'une valeur de vérité

Condition simple	*Alternatives complètes en chaîne, affichage*
	`SI Marie`
	`  ALORS SI Jean`
	`  ALORS AFFICHER`
	`  'Clause 1 respectée'`
	`  SINON AFFICHER`
	`  'Clause 1 non respectée'`
	`SINON AFFICHER`
	`'Clause 1 non respectée'`
		Alternatives incomplètes en chaîne, remplacement
		`Clause1 ← VRAI`
		`SI Marie ALORS`
		`  SI Jean ALORS`
		`  Clause1 ← FAUX`

Condition composée	*Affichage conditionnel avec condition composée*
	`SI Marie ET Jean`
	`  ALORS AFFICHER 'Clause 1 non respectée'`
	`  SINON AFFICHER 'Clause 1 respectée'`
		Remplacement conditionnel avec condition composée
		`Clause1 ← VRAI`
		`SI Marie ET Jean`
		`  ALORS Clause1 ← FAUX`

Figure 7

A l'autre extrémité du tableau, la solution type *Remplacement conditionnel avec condition composée* est assez proche d'une solution par calcul et affectation d'une expression booléenne telle que celle de la figure 5. Le passage à cette dernière forme pose cependant aux élèves plusieurs difficultés:

- le type "booléen" des variables: les élèves abandonnent en effet le type booléen à partir du moment où ils passent à l'écriture effective d'un programme, la cause principale de ce changement de type étant qu'ils sont peu à l'aise avec les procédures d'entrée (lecture) et de sortie (affichage) sur ce type (dernière colonne, figure 6).
- le connecteur de négation (NON): les connecteurs relevés dans les protocoles sont ceux auxquels les élèves peuvent donner un sens à partir des conditionnelles du langage courant ou des alternatives (quatrième colonne, figure 6). Ce n'est pas le cas de la négation. En effet la structure alternative permet de se passer de ce connecteur,

SI NON<condition> ALORS <action1> SINON <action2>
s'écrivant plus simplement
SI <condition> ALORS <action2> SINON <action1>.
De même, le connecteur DIFFERENT DE peut être utilisé directement pour nier l'égalité.

- l'affectation d'une expression booléenne à une variable logique. En effet si les élèves se représentent l'affectation d'une constante (Vrai ou Faux) à une variable logique, ils éprouvent des difficultés à donner une signification à l'affectation des expressions booléennes. En fait, ils ont donné une signification aux expressions booléennes à partir des premières alternatives sur le modèle des conditionnelles du langage courant, et cette signification n'est pas celle d'objet booléen potentiel.

La première difficulté concernant l'entrée-sortie de valeurs booléennes est en fait peu résistante: dans la suite du travail (sur ordinateur) les élèves utilisent le type booléen après qu'on leur ait indiqué comment résoudre ces questions. Ils éprouvent davantage de difficulté avec les connecteurs non familiers NON et OU (par exemple, ils ne simplifient pas l'expression NON(Marie DIFFERENT DE Marc)). L'affectation d'une expression booléenne reste peu employée, les élèves utilisant plutôt l'affectation conditionnelle de valeurs logiques constantes:

SI <expression booléenne>
ALORS <variable logique> ← VRAI
SINON variable logique ← FAUX.

Les deux dernières difficultés sont liées toutes les deux à une conception des expressions logiques découlant des conditionnelles telles qu'on les utilise dans le langage courant. La difficulté à utiliser certains connecteurs est à mettre en relation avec une méconnaissance du calcul sur les valeurs logiques (calculabilité des booléens non intégrée), la difficulté avec l'affectation relevant davantage de conceptions ne prenant pas en compte la capacité du dispositif à exécuter des transformations sur des valeurs logiques (Opérationnalité du langage sur les booléens).

6. Discussion

Interprétation des résultats

(Réponses aux questions posées au paragraphe 3.)

1. Une fraction notable des élèves observés considère les expressions et sous-expressions chaînes comme des formes inspirées du langage habituel (description d'objet, codage d'action...), et non comme des objets potentiels. Ne percevant pas la

nécessité de nommer les résultats des actions, ils minorent le rôle de l'affectation sur les chaînes

De même, les élèves ont des difficultés à donner une signification à l'affectation d'une expression logique à une variable booléenne . Ils ne comprennent pas les expressions logiques comme des valeurs booléennes potentielles, mais comme des *conditions* ayant une signification seulement lorsqu'elles constituent la première partie dans une alternative.

2. Les difficultés à exprimer un calcul sur des ordinaux conduisent assez généralement les élèves à donner aux variables ordinales une signification pré-opératoire où les ordinaux sont mal dissociés des caractères ou des chaînes. Dans les expressions formées à partir de la fonction sous-chaîne ou de la concaténation, les élèves ne manifestent généralement pas de prise de conscience des contraintes formelles relatives aux arguments. Ils fournissent en fait "toutes les informations nécessaires" au dispositif, le statut de ces informations restant imprécis.

   Dans le problème sur des valeurs logiques, les élèves emploient seulement les connecteurs propositionnels auxquels ils peuvent donner le plus facilement un sens à partir des conditionnelles du langage courant. Ils pensent ainsi l'évaluation des expressions logiques à partir d'un raisonnement s'appuyant sur le contexte, et non à partir d'un calcul sur des valeurs booléennes.

3. Les réponses des élèves montrent qu'il est possible, sans mettre en jeu des structures algorithmiques autres que le traitement séquentiel, de faire émerger des représentations initiales ne prenant pas en compte les contraintes d'opérationalité et de calculabilité. Les échecs témoignent de ce que, privilégiant l'acquisition des structures algorithmiques[6], l'enseignement d'informatique reçu par les élèves ne leur a pas fait rencontrer suffisamment ces contraintes. Dans un premier essai d'ingéniérie sur les chaînes de caractères, nous avons montré, par contre, que des

---

[6]Le programme de l'option informatique, première année (MEN 1987) s'organise en quatre parties: *Traitement séquentiel, Traitement conditionnel, Traitement itératif simple, Traitement itératif général.* L'enseignement des structures algorithmiques se fait ainsi sur des données numériques, où la compréhension des notions de variables et d'expression est sous l'influence des acquisitions mathématiques, ensuite, l'enseignement sur les chaînes et sur les booléens consiste à présenter un jeu d'instructions et des exemples types d'utilisation des structures algorithmiques (comptage d'occurrences dans les chaînes, "retournement" d'une chaîne de caractères...). Pour un exemple représentatif voir (Arsac-Mondou et alter 1987).

élèves très en difficulté au départ, pouvaient progresser de façon significative en résolvant des problèmes de programmation faisant effectivement rencontrer ces contraintes (Lagrange, 1991).

Les représentations initiales.

Ayant à faire agir le dispositif sur des objets qui pour eux n'ont pas de statut algébrique, les élèves utilisent le langage de commande comme une instance de la communication habituelle, et comprennent les expressions comme un moyen de transmettre des intentions au dispositif. Ces représentations initiales du langage prennent des "colorations" différentes selon le contexte intuitif dans lequel les élèves considèrent les objets: le langage sur les chaînes est pensé en termes d'action, alors que les booléens sont associés à des "conditionnelles".

Ainsi, face à un problème de traitement par l'ordinateur d'un type d'objet nouveau, le sujet investit sa connaissance extra-informatique des objets dans ses représentations initiales du langage, plutôt que des représentations algébriques ou informatiques formées sur d'autres types d'objets. Il ajuste seulement progressivement ces représentations aux contraintes du dispositif informatique qu'il rencontre.

L'utilisation de l'ordinateur pour l'enseignement des mathématiques

Ce type de conduite paraît assez général pour être pris en compte dans l'analyse didactique de la tâche de l'élève lors de l'utilisation de l'informatique pour des acquisitions en mathématiques. Ainsi, par exemple, un travail de programmation de l'évaluation d'expressions sur des variables numériques, utilisant la décomposition à l'aide de l'affectation (Tall, Thomas, 1991)[7] est une tâche significative pour des élèves débutant en algèbre, car ceux-ci donnent initialement à ces expressions la seule signification d'écriture symbolique à réduire: l'intégration de la contrainte d'opérationnalité, que suppose l'acquisition de l'affectation, permet aux élèves une compréhension plus "fonctionnelle" de ces expressions.

De même, les élèves étudiés par Artigues (1991) dans l'utilisation d'EUCLIDE sont plus influencés par le contexte géométrique que par une connaissance des variables et expressions algébriques. Les contraintes se manifestent sans être totalement

---

[7]Par exemple à partir de l'expression 5+2A, former le programme:

```
Lire A
B←2*A
C←5+B
Afficher C
```

productives: par exemple, l'intégration de l'opérationnalité du langage peut conduire à une certaine rigueur dans l'utilisation du langage géométrique, mais nous avons vu aussi que l'instruction de définition présente des caractéristiques propres à l'affectation informatique, qui entrent en contradiction avec les représentations initiales du langage géométriques, sans qu'elles aient une signification pour l'activité géométrique.

L'existence de contraintes "improductives" impose une initiation des élèves à l'outil informatique, indépendamment de l'activité mathématique proprement dite. Là aussi, représentations initiales et contraintes sont à prendre en compte par le didacticien pour construire une initiation efficace. Par exemple, bien que EUCLIDE soit un sur-ensemble de LOGO, la structure impérative du langage EUCLIDE oblige à distinguer définition et affichage, alors que, en LOGO tortue, toute action a un effet sur l'écran. De plus, on "calcule" des objets géométriques en EUCLIDE, alors que l'on agence des mouvements de la tortue en LOGO. Les contraintes ne sont donc pas les mêmes, et les objets renvoient à des domaines de représentation différents. Il est donc probable qu'une utilisation de LOGO tortue n'est pas la meilleure initiation préalable à l'utilisation d'EUCLIDE, et Artigues (1991) note en particulier que des conduites d'interprétation des feed-back devenues efficaces en LOGO, ne portent plus leur fruits quand l'élève tente de les mettre en oeuvre avec EUCLIDE.

Pour conclure

A partir d'une étude des premiers apprentissages en programmation, l'article propose une analyse des difficultés de l'élève dans son activité de résolution, mettant en rapport ses représentations initiales et les contraintes du langage. Nous souhaitons avoir montré que ce type d'analyse aide à concevoir l'utilisation d'outils informatiques pour l'enseignement des mathématiques, particulièrement dans le cas où l'outil implique un langage de commande. Sur un plan plus général, nous souhaitons avoir montré également l'intérêt que présentent, dans l'état actuel des connaissances sur les spécificités du milieu en présence d'un outil informatique, les études expérimentales centrées sur les difficultés de l'élève.

Jean-Baptiste LAGRANGE
Institut Universitaire de Formation des Maîtres
Rennes FRANCE

Bibliographie

Arsac Mondou, O., Bourgeois, C., Gourtay, M.: 1987, Option Informatique, Classe de Seconde, Nathan, Paris.

Artigues, M.: 1991, 'Analyse de processus d'enseignement en environnement informatique', Petit x, 26, 5-27.

Balacheff, N., Capponi, B.: 1989, 'Tableur et calcul algébrique', Educational Studies in Mathematics 20, 179-210.

Brousseau G.: 1989,'Le contrat didactique: le milieu', Recherche en Didactique des mathématiques 9/3, 309-336

Douady R.: 1986, 'Jeu de cadres et dialectique outil-objet', Recherche en Didactique des mathématiques 7/2, 5-32

Dupuis, C., Guin, D.: 1989, 'Gestion des relations entre variables dans un environnement de programmation LOGO', Educational Studies in Mathematics 20: 293-316

Laborde, C., Balacheff, N., Meijas B.: 1985, 'Problématique et genèse du concept d'itération, une approche expérimentale', Enfance 2, 223-239.

Lagrange, J.B.: 1986 , Apprentissage des concepts informatiques de base.. Rapport de DEA, Université Paris VII.

Lagrange, J.B.: 1991, Représentations mentales et processus d'acquisition dans les premiers apprentissages en informatique. Thèse. Université Paris VII.

M.E.N.: 1987, Option Informatique, Classes de Seconde, première et terminale. CNDP

Rogalski, J., Hé, Y.: 1989, 'Logic abilities and mental representations of the informatical device in acquisition of conditional structures by 15-16 year old students', European Journal of Psychology of Education, 4, 71-82

Samurçay, R., Rouchier, A.: 1990, 'Apprentissage de l'écriture et de l'interprétation de procédures récursives', Recherche en Didactique des mathématiques 10 2.3, 287-326

Samurçay, R.: 1985, 'Signification et fonctionnement du concept de variable informatique chez des élèves débutants' Educational Studies in Mathematics 16, 143-161

Tall, D., Thomas, M.: 1991, 'Encouraging Versatile Thinking in Algebra Using the Computer', Educational Studies in Mathematics 22, 125-147.

Annexe 1: Programmation impérative, et programmation applicative.

Un schéma général de la programmation est le suivant:

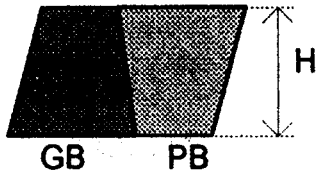
Donnée(s) → [Traitement] →Résultat(s)

Dans l'article, nous considérons seulement des traitements simples qui s'obtiennent comme composition de fonctions primitives du langage. Depuis FORTRAN (FORmula TRANslator), les langages évolués permettent d'exprimer ces traitements simples par une formule. Par exemple, un traitement où les données sont les mesures d'un trapèze (Grande base, Petite base, Hauteur) et le résultat l'aire de ce trapèze, s'exprime

```
Lire GB, PB, H
Afficher (GB+PB)*H/2
```

L'écriture de ce programme est la simple adaptation d'une connaissance mathématique. Par contre, des concepts informatiques sont impliqués à partir du moment où l'on cherche à construire un calcul (ce qui est une nécessité dans les problèmes étudiés dans l'article, à cause de la complexité des "formules" sur des objets non-numériques).

Si l'on reprend l'exemple ci-dessus, la construction peut s'appuyer sur une figure géométrique associant deux trapèzes de même mesure et sur la connaissance d'une formule de calcul de l'aire du parallélogramme.



Une construction impérative.

Elle s'exprime comme une suite d'affectations

```
BaseParallelogramme ← GB+PB
AireParallelogramme ← BaseParallelogramme * H
Resultat ← AireParallelogramme / 2
```

L'affectation `variable←<expression>` peut se décrire par son mécanisme d'exécution: `variable` est une adresse mémoire (une "case"), et l'exécution consiste dans le remplacement de la valeur présente dans cette case par la valeur obtenue par l'évaluation de `<expression>`. Cependant, même à ce niveau élémentaire, le programme s'interprète indépendamment de son exécution, et l'effet de l'affectation (sa "sémantique") est donnée de façon plus juste en considérant qu'elle établit des relations entre les variables du programme.

Le schéma relationnel (ou état interne du dispositif) établi par exemple à la suite de la seconde affectation du programme ci-dessus, est le suivant:

AireParallelogramme ← [Produit] ← BaseParallelogramme ← [Somme] ← PB, GB
← H

Une construction applicative

Elle utilise le passage de la valeur d'une expression comme argument d'une fonction ou d'une procédure:

```
POUR AireParallelogramme (B,H)
 Rends B*H

POUR AireTrapeze (GB,PB,H)
 Rends (AireParallelogramme (GB+PB,H))/2
```

Une description de l'exécution utilise la "substitution" : `AireTrapeze (2,3,4)` se réécrit en `(AireParallelogramme (5,4))/2`, et une interprétation sémantique est donnée par la composition des fonctions, mais la notion d'état interne du dispositif est plus difficile à cerner.

Ce type de construction applicative est employé dans les langages où la programmation se fait par la définition d'un ensemble de fonctions, dans la mesure où l'on s'interdit d'employer l'affectation. C'est pourquoi l'on confond souvent "fonctionnel" et "applicatif". Nous considérons quant à nous, que la construction applicative se conçoit en dehors de la définition de fonctions. Par exemple, les élèves observés par Dupuis, Guin (1989), y sont confrontés dans un contexte de procédures graphiques s'emboîtant. (Voir paragraphe 2.3)

Annexe 2 : Les fonctions sur les chaînes de caractères

Les langages disposant du type "chaînes de caractères" ont en commun trois fonctions qui sont suffisantes pour la définition du type:

- la concaténation (`chaîne1 + chaîne2` dans l'article) permet de former une seule chaîne à partir de deux chaînes,
- la fonction `longueur` s'applique à une chaîne et rend un résultat entier qui peut être considéré soit comme l'ordinal, rang du dernier caractère de la chaîne, soit comme le cardinal, nombre de caractères de la chaîne,
- la fonction `sous-chaîne` a trois arguments, le premier est une chaîne, le second est une position, donc un ordinal, le troisième est un nombre de caractères, donc un cardinal. Le résultat est la chaîne "extraite" dans le premier argument, à partir de la position second argument, et ayant pour longueur le troisième argument.

Pour un exemple de construction à l'aide de ces fonctions voir la figure 1 paragraphe 5.1.