ALGORITHMIQUE ET ALGÈBRE : LE CAS DU PARAMÈTRE

Jean-Marc LEGRAND[1]
CREN, Inspé de l'Académie de Nantes

Résumé. Algorithmique et algèbre partagent de nombreux concepts, méthodes ou objectifs, dont notamment la recherche de généricité s'appuyant nécessairement sur la variable dans un rôle de paramètre. Ce concept de paramètre est un obstacle épistémologique probable. Nous proposons une situation de généralisation de motif informatisée qui permettra d'étudier la construction de ce concept (le paramètre et sa représentation dans un certain langage), par des élèves de cycle 4. Les premiers résultats semblent confirmer la difficulté de cette construction. Ils soulignent aussi l'importance de faire vivre les tensions fixe-variable ou unique-multiple induites par le paramètre et la généricité. Pour cela, faire faire une généralisation par un programme semble être une piste à exploiter, notamment parce que cela nécessite une exploration du langage dans lequel on représente le paramètre.

Mots-clés. Algorithmique, algèbre, généralisation de motif, paramètre, variable, langage.

Abstract. Algorithmic and algebra share many concepts, methods or objectives, including the aim of genericity which necessarily relies on the variable in a parameter role. This concept of parameter is a potential epistemological obstacle. We will propose a situation of a computerized pattern generalization which will allow us to study the construction of this concept (the parameter and its representation in a certain language), by cycle 4 students. The first results seem to confirm the difficulty of this construction. They also underline the importance of bringing to life the fixed-variable or unique-multiple tensions induced by the parameter and genericity. Making do the generalization by a program seems to be an interesting approach, especially because it requires an exploration of the language in which we represent the parameter.

Keywords. Algorithmic, algebra, pattern generalization, parameter, variable, language.

Introduction

L'entrée dans l'algèbre élémentaire, en cycle 4, est un passage souvent difficile pour les élèves[2]. En outre, une initiation à l'algorithmique et à la programmation est à présent prévue dans les programmes de l'école primaire et du collège. Or ces deux domaines semblent partager des démarches et concepts communs, en particulier la recherche de généricité et la nécessité de la variable dans un rôle de paramètre. Dans cet article, nous interrogerons cette proximité et proposerons une situation de généralisation de motif informatisée afin d'étudier en quoi une activité d'apprentissage de l'algorithmique pourrait contribuer à la construction du concept de paramètre. On s'intéressera notamment aux tensions induites par une situation de généralisation, et comment les élèves explorent le langage donné afin de régler ces tensions et aboutir au paramètre et à sa représentation[3].

[1] jean-marc.legrand@univ-nantes.fr

[2] Pour une revue de ces difficultés et des recherches qui leur sont consacrées, voir par exemple Kahane (2002), Bronner et Squalli (2021, p. 5), Pilet et Grugeon-Allys (2021, p. 2)…

[3] Nous reprenons et complétons ici des éléments donnés dans Legrand (2022).

1. Algorithmique et algèbre : faire-faire et généralisation

1.1. Proximité algorithme-algèbre

Les mots « algorithme » et « algèbre » ont une origine historique commune, en référence au nom latinisé du mathématicien Perse al-Khwārizmī pour le premier, et au terme « *al-jabr* » utilisé par al-Khwārizmī pour décrire une *« opération qui consiste à faire disparaître un terme négatif qui figure dans l'un des membres d'une équation, en ajoutant la valeur absolue de ce terme aux deux membres »* (Farès, 2015). Mais algorithmique et algèbre partagent aussi des objectifs, des raisonnements et des concepts.

Objectifs

Selon Grugeon-Allys *et al.* (2012), l'algèbre est notamment mobilisée

> *comme outil de résolution via la modélisation, pour résoudre des problèmes « arithmétiques » formulés en langue naturelle sous forme d'équations et, au-delà, pour résoudre des problèmes intra ou extra mathématiques sous forme de relations fonctionnelles entre données et variables* (Grugeon-Allys *et al.*, 2012, p. 6).

Ainsi, on peut considérer que l'algèbre élémentaire est la branche des mathématiques qui vise à la recherche et l'étude de procédures de résolution de problèmes modélisables par des équations polynomiales.

L'algorithmique, quand à elle, consiste en la production et l'étude des algorithmes, vus comme

> *procédure[s] de résolution de problème, s'appliquant à une famille d'instances du problème et produisant, en un nombre fini d'étapes constructives, effectives, non-ambiguës et organisées, la réponse au problème pour toute instance de cette famille »* (Modeste, 2012, p. 25).

Ainsi, pour l'algorithmique comme pour l'algèbre, l'objectif est de produire et d'analyser non pas des solutions, mais des méthodes organisées permettant de résoudre des problèmes. En fait, il ne s'agit pas de *faire*, mais de *faire-faire* (Samurçay & Rouchier, 1985).

Raisonnement et pensée

Selon Pilet et Grugeon-Allys 2021), *« le raisonnement algébrique consiste à représenter les relations entre les données et les nombres non connus du problème et à utiliser un traitement formel pour le résoudre »*. D'un autre côté, pour Wing (2011) :

> *La pensée informatique est le processus de pensée impliqué dans la formulation des problèmes et de leurs solutions, de sorte que les solutions sont représentées sous une forme qui peut être exécutée efficacement par un agent de traitement de l'information* (Wing, 2011, p. 20).

Ainsi, pensée informatique et raisonnement algébrique partagent la capacité à *« savoir réfléchir aux tâches à accomplir pour résoudre un problème en termes d'étapes et d'actions »* (Tchounikine, 2017), mais aussi la nécessité d'une formalisation de ces étapes, dans un langage interprétable de façon effective et non ambiguë par l'*« agent d'information »*, ce que Samurçay et Rouchier (1985) nomment *« dispositif d'exécution »*. Celui-ci peut être une machine (et pas forcément un ordinateur), *« un système matériel qui obéit aux lois de la physique [...] et plus généralement tous les systèmes physiques, pour lequel nous avons défini un protocole d'interaction, qui nous permet d'échanger des données »* (Dowek, 2011), mais aussi un humain ou une combinaison des deux.

Pensée et généralisation

Un autre élément qui nous semble à l'interface entre algorithmique et l'algèbre est l'importance de la généralisation. En effet, Bronner et Squalli (2021) affirment que *« La généralisation est un processus essentiel dans l'activité mathématique et tout particulièrement en algèbre »* en précisant que plusieurs auteurs la reconnaissent *« comme étant une composante très importante de la pensée algébrique »*. De la même façon, la généralisation, c'est-à-dire l'extension de la solution d'une instance d'un problème à toutes les instances de ce problème, est aussi une des composantes essentielles de la pensée informatique. Dans ces deux domaines de pensée, la généralisation consiste en l'identification de motifs, de régularités, de relations, et en l'exploitation de ces éléments. Radford (2006a) précise ainsi que généraliser en algèbre — et cela s'applique aussi à l'algorithmique —, c'est :

1) identifier une régularité dans l'ensemble des instances produites,
2) être conscient que cette régularité peut s'étendre à l'ensemble des instances possibles,
3) pouvoir exprimer cette régularité pour trouver n'importe quelle instance.

Généralisation et langage

Le point 3) met l'accent sur un des élément fondamentaux en algèbre comme en algorithmique : il s'agit d'exprimer la généralisation dans un certain langage. Il y a ici en jeu une conversion de registre sémiotique (Duval, 1993) entre le langage de description du problème et le langage de description de l'algorithme ou de la procédure. Radford (2014) souligne que la pensée algébrique en situation de résolution de problèmes est définie par trois caractéristiques :

(a) le caractère indéterminé, le fait que des nombres non connus sont impliqués dans le problème,
(b) la dénotation, le fait de nommer ou symboliser ces nombres indéterminés,
(c) l'analycité, le traitement des quantités inconnues comme si elles étaient des nombres connus.

Il considère aussi trois niveaux de généralisation algébrique (Radford, 2003) :

- le niveau *« factuel »* dans lequel la généralité se limite à des actions, sans nommer le paramètre, la quantité non déterminée ;
- le niveau *« contextuel »* pour lequel le paramètre est nommé ou exprimé par des mots, des expressions des gestes ;
- le niveau *« symbolique »* où les actions et les paramètres sont exprimés dans un langage formel, proche ou identique au registre algébrique.

Nous nous intéressons ici à la généralisation algébrique symbolique, tout en précisant comme Radford qu'on ne s'intéresse pas au seul cas de la lettre.

Les quatre concepts de l'informatique

Ainsi lorsqu'on parle d'algèbre élémentaire ou d'informatique :

- on recherche la conception d'une méthode ou d'une procédure transformant les données d'un problème en une solution ;
- cette méthode doit s'appliquer à une famille d'instances du problème, c'est-à-dire que l'on recherche une méthode générique (le plus possible) ;
- cette méthode est exécutable par un « dispositif d'exécution », il s'agit de faire-faire par autrui (élève scribe pour les babyloniens, mathématicien pour Diophante ou juriste pour

al-Khwārizmī) ou par une machine ;

- cette méthode doit donc aussi être écrite dans un langage spécifique interprétable de façon non ambiguë par le dispositif d'exécution.

On retrouve ici *« les quatre concepts de l'informatique »* de Dowek (2011) : un algorithme, traitant des *données* ou des informations, écrit dans un certain *langage* afin d'être exécuté sur une certaine *machine*. Ainsi, on peut sans doute affirmer que algèbre et informatique, pensée algébrique et pensée informatique, relèvent en partie de mêmes raisonnements, de mêmes objectifs, et de mêmes concepts. Parmi ces concepts, il en est un qui nous paraît fondamental, celui de « nombres indéterminés », soit les paramètres.

1.2. La longue construction du paramètre

Ce concept essentiel, *la variable dans un rôle de paramètre*[4], est impliqué dès que l'on parle de généralisation, en algorithmique comme en algèbre. Ces paramètres, *« ce qui fait la force de l'algèbre »* selon Chevallard (1989), sont *« les variables du système [mathématisé] dont les valeurs sont supposées connues »*. La représentation sous forme de symboles de ces paramètres ne viendra qu'à la fin du XVI[e] siècle avec Viète puis Descartes. Il nous semble important de comprendre la genèse de ce concept, et de s'interroger sur sa construction si tardive.

Premières généralisations algébriques : les Babyloniens

Première partie de la tablette (début détruit) :

2 [Le nombre est **4;10**. Quel est son inverse ?
Procède comme suit.
Forme l'inverse de **10**, tu trouveras **6**.
Multiplie **6** par **4**, tu trouveras **24**.
Ajoute **1**, tu trouveras **25**.
Forme l'inverse de **25**] tu trouveras **2;24**.
[Multiplie **2;24** par **6**], tu trouveras **14;24**.
[L'inverse est **14;24**]. Telle est la façon de procéder.

3 [Le nombre est **8;20**]. Quel est son inverse ?
[Pro]cède comme suit.
[Forme l'inverse de **20**], tu trouveras **3**.
Mul[tiplie **3** par **8**], tu trouveras **24**.
[Ajoute **1**], tu trouveras **25**.
[Forme l'inverse de **25**] tu trouveras **2;24**.
[Multiplie **2;24** par **3**], tu trouveras [**7;12**.
L'inverse est **7;12**. Telle est la façon de procéder].

4 [Le nombre est **16;40**. Quel est son inverse ?
[Pro]cède comme suit.
Forme [l'inverse de **6** ;**40**], tu trouveras **9**.
Multiplie [**9**]par **10**, tu trouveras **1;30**.
[Aj]oute [**1**], tu trouveras **2;30**.
Forme [l'inverse de **2**]**;30**] tu trouveras **24**.
[Multiplie **24** par **9**, tu trouveras **3;36**.
[L'inverse est **3;36**]. Telle est la façon de procéder].

***Figure 1** : VAT6505 - (entre −1900 et −1600), à partir de (Chabert,2010).*

Algorithme 1 : VAT6505 inverse d'un nombre régulier

Données : x, y, z avec $x = y + z$, et y un nombre régulier dont on connaît $\bar{y}$
Résultat : $\bar{x}$ (tel que $x \times \bar{x} = 60^n$)

```
début
    ȳ ← inverse(y);
    t ← ȳ ×z;
    u ← t + 1;
    ū ← inverse(u);
    v ← ū × ȳ ;
fin
retourner v
```

***Figure 2** : VAT6505 - Transcription en pseudo-langage.*

Dès les babyloniens, on peut constater la volonté de généraliser les méthodes de résolution de problème. Dans l'extrait de la tablette VAT6505 (*cf.* figure 1), on peut observer trois instances de la résolution d'une famille de problème : trouver l'inverse d'un nombre régulier (en base sexagésimale). La méthode est strictement identique et peut être représentée par l'algorithme de la figure 2, mais s'applique pour trois paramètres différents. L'algorithme — *« telle est la façon de procéder »* — n'est pas présenté de façon générique : la charge de la construction de la

[4] Nous utilisons cette expression afin d'évoquer le concept de paramètre à la fois dans un cadre algébrique et dans un cadre algorithmique.

méthode générale est laissée à l'étudiant scribe. On trouve ici ce que Balacheff (1987) nomme des exemples à caractère générique, le nombre donné prenant un statut de nombre générique : il est ici *« en tant que représentant caractéristique d'une classe d'individus »*. Ainsi, chez les babyloniens, il n'y a pas le caractère d'analycité de Radford. Si certains nombres indéterminés sont parfois nommés en fonction de leur nature, on ne les manipule pas comme s'ils étaient connus. Ces dénominations de *« ce que l'on recherchait »* ont longtemps été représentées par *« ce que l'on désirait »* : ce que Descartes nommera *« l'inconnue »* fut ainsi désignée par un *« tas »* (*« hau »*) pour les égyptiens, *« le bien, l'héritage, la possession »* pour les mathématiciens arabes, ou encore la couleur pour les indiens (Serfati, 1997).

Diophante et la première symbolisation de l'inconnue

Chez Diophante, le nombre inconnu, l'*arithme* (« une quantité indéterminée d'unités »), dispose d'un symbole (ς) et ce symbole est manipulé comme un nombre, comme s'il était connu. Ainsi peut-on parler de l'inverse de l'arithme, alors que les babyloniens ne pouvaient envisager que l'inverse d'un certain nombre donné. Radford (1991) donne l'exemple d'une expression telle que ςλγMη : celle-ci désigne le nombre formé de 33 (λγ) arithmes (ς) auxquels on ajoute 8 (η) unités (le M servant à désigner les nombres déterminés). Même si on s'accorde à dire que le langage mis en place par Diophante n'est pas algébrique, mais plutôt pré-algébrique, les critères de dénotation et d'analycité sont bien présents. Serfati (1997) souligne que :

> *Dès lors que l'écriture contenait une représentation symbolique de l'inconnue, ainsi suivie à la trace, le passage d'une ligne à la suivante dans la succession des équations présenta un caractère automatique extrêmement commode, la mécanique « aveugle » du calcul* (Serfati, 1997, p. 40).

Cependant, si, contrairement aux babyloniens, les problèmes sont posés de façon générique, leur résolution reste décrite à partir d'un exemple générique. Ainsi, dans les *Arithmétiques*, la Proposition I.1 est : *« Partager un nombre proposé en deux nombres dont la différence est donnée »*, mais Diophante la résout en prenant comme nombres génériques 100 pour le « nombre donné » et 40 pour la différence. La résolution de ce problème est constituée du texte suivant :

> *Posons que le plus petit nombre est un arithme ; donc, le plus grand nombre est 1 arithme plus 40 unités. En conséquence, la somme des deux nombres devient deux arithmes plus 40 unités. Or, les 100 unités données sont cette somme ; donc 100 unités sont égales à deux arithmes plus 40 unités. Retranchons les semblables des semblables, c'est-à-dire 40 unités de 100 et, de même, 40 unités de 2 arithmes plus 40 unités. Les deux arithmes restants valent 60 unités, et chaque arithme devient 30 unités. Revenons à ce que nous avons proposé : le plus petit nombre sera 30 unités ; tandis que le plus grand sera 70 unités, et la preuve est évidente* (Farès, 2017 p.17).

Avec notre notation actuelle, il s'agit de trouver les nombres entiers x et y tels que $x+y=a$ (a est le nombre donné) et $x-y=b$ (b est la différence). Si on désigne le plus petit nombre par l'arithme ς, la résolution du problème instancié devient alors :

$$\begin{aligned}
(x+\varsigma=100)\wedge(x-\varsigma=40) \quad & (1)\\
x=\varsigma+40 \quad & (2)\\
x+\varsigma=2\varsigma+40 \quad & (3)\\
2\varsigma+40=100 \quad & (4)\\
2\varsigma=100-40 \quad & (5)\\
2\varsigma=60 \quad & (6)\\
\varsigma=30 \quad & (7)
\end{aligned}$$

On peut alors suivre aisément le processus aboutissant à la détermination du « plus petit nombre », et le changement d'instance est rendu plus facile que dans la démarche babylonienne,

puisqu'au lieu d'une suite de calculs arithmétiques, on garde la trace des opérations à effectuer sur l'arithme. Il faut préciser néanmoins que Diophante n'utilisait qu'un seul symbole pour l'arithme, il ne disposait donc pas d'un deuxième symbole pour représenter ici « le plus grand nombre ». Cependant, d'autres représentations symboliques, qui sont plutôt considérées comme des abréviations, sont présentes dans le langage de la « théorie arithmétique » mis en place par Diophante, afin de représenter les « espèces » : le nombre carré, le nombre cube, le nombre bi-carré, etc. Ces éléments du langage de la théorie sont définis en début d'ouvrage, et Diophante différencie bien le langage de description du problème (en langue naturelle) du langage de la représentation et de la résolution du problème (créé pour la théorie) : une étape importante de ce qu'il nomme « la voie » (*oidos*) consiste à traduire le problème dans le langage de la théorie. La méthode générale prônée par Diophante pour résoudre les problèmes arithmétiques présentés commence ainsi selon Christianidis (2007) par « l'invention », soit le transfert du problème (dans sa version instanciée) dans l'environnement de la « théorie arithmétique », c'est-à-dire en opérant notamment une conversion du langage de l'énoncé vers le langage de la théorie, pour transformer le problème en une équation[5]. Donnons deux exemples notables de cette conversion de registres de représentation sémiotique (Duval, 1993) : le nombre carré du problème (« *tetragônos* ») se nomme « *Dynamis* » dans la théorie et est représenté par la désignation abrégée Δ^{Y} , tandis que le nombre cube « *kybos* » sera nommé « *Kybos* » dans la théorie, et représenté par K^{Y}. La création d'un langage spécifique sur lequel on opère est une caractéristique essentielle de l'algèbre, tout comme de l'algorithmique.

Al-Khwārizmī et les débuts de l'algèbre

Cependant, l'utilisation de symboles n'est pas nécessaire pour respecter le critère d'analycité, ni même la généricité. Chez al-Khwārizmī, mathématicien perse du IX^{e} siècle considéré comme étant à l'origine de l'algèbre moderne, aucun symbole n'est présent : son algèbre n'est ni syncopée (comme chez Diophante) ni symbolique (comme dans l'algèbre moderne), elle est rhétorique. Il définit cependant des termes de son langage algébrique, introduits dans la partie théorique de son ouvrage *« kitāb al-jabr wa al-muqābala »* : *« l'inconnue — indifféremment désigné par « racine » (jidhr) ou par « chose » (shay') ; son carré (māl) ; le nombre ('adad) [...] ; les opérations : addition, soustraction [...] ainsi que celle d'al-jabr (restauration) et al-muqābala (réduction) »* (Rashed, 2007). Comme chez Diophante, une étape essentielle consiste à représenter le problème dans le *« langage de la théorie »*, c'est-à-dire une équation (du second degré) exprimée avec les termes *shay'*, *māl*, etc. Cependant, contrairement à Diophante, chez al-Khwārizmī, les équations précèdent le problème : avant de traiter des problèmes dans son ouvrage, il établit les six formes canoniques des équations du premier et second degré ($ax^2=bx$, $ax^2=c$, $bx=c$, $ax^2+bx=c$, $ax^2+c=bx$, $ax^2=bx+c$, où a, b et c sont des rationnels positifs) et indique ce que Farès (2015), appelle *« la formule algorithmique de la solution pour chacun des types »*. Cependant, là encore, même si les équations sont génériques, elles ne sont ni formulées ni résolues avec des paramètres : l'équation générique $ax^2=bx$ est ainsi formulée « des *māl* égalent des *shay'* », et résolue pour plusieurs exemples exemplaires : $x^2=5x$, $\frac{1}{3}x^2=4x$, $5x^2=10x$, …

Descartes et Viète, enfin le paramètre !

Le paramètre et sa représentation n'apparaîtra qu'à la fin du XVI^{e} siècle, avec Viète puis

[5] Les étapes suivantes étant la « disposition », la transformation de l'équation afin de déterminer le nombre inconnu, puis le calcul des nombres cherchés à partir de l'arithme, et éventuellement une preuve ou vérification.

Descartes. Viète désignera les inconnues et leurs puissances avec une voyelle (majuscule) suivie d'un mot marquant la puissance à laquelle elle est élevée (Macbeth *et al.*, 2004). Ainsi, le carré d'une inconnue s'écrira « *A quadratum* », le cube d'une autre « *E cubum* ». Les paramètres seront eux désignés par une consonne suivie d'une annotation de l'espèce (« *B plano* », « *D solido* »). Les nombres représentés restent ainsi fortement liés aux objets qu'ils représentent. Descartes se débarrassera de ces questions liées à l'homogénéité en ne considérant que les symboles (fin de l'alphabet pour les inconnues, début pour les paramètres) et que :

> *lorsque l'unité est déterminée, […] elle peut être sous-entendue partout où il y a trop ou trop peu de dimension ; comme s'il faut tirer la racine cubique de* $aabb-b$ *il faut penser que la quantité aabb est divisée une fois par l'unité, & que l'autre quantité b est multipliée deux fois par la même* (Descartes, 1897).

Considérant donc que $a^2b^2-b=\frac{a^2b^2}{1}-b\times 1\times 1$: on n'obtient ainsi que des nombres « cubes ».

Les nombres deviennent indépendant des objets dont ils représentent une propriété et peuvent ainsi être manipulés sans considération de l'espèce. En outre, la notation introduite pour les puissances va permettre ultérieurement à Fermat de faire une interprétation mathématique d'une extension du langage pour les puissances non entières, ce qui ne pouvait se faire avec des nombres « typés ».

Paramètre et informatique

Ces paramètres vont permettre à Pascal de décrire un algorithme générique de divisibilité. Il construira par ailleurs sa *Pascaline*, premier calculateur mécanique opérationnel dont la généricité du mécanisme lui permet d'effectuer des calculs dans des bases autres que décimale. Plus tard, Ada Lovelace créera un langage manipulant des représentations de variables informatiques en séparant les rôles (« *data* » : entrée ou paramètre ; « *result variable* » : sortie ou inconnue ; « *working variable* » : variable temporaire) et en précisant l'état courant de la machine de Babbage. Les symboles utilisés sont de la forme ${}^{e}V_{n}$, où *n* indique le numéro de la variable et *e* l'état, c'est-à-dire l'indication du nombre d'opérations effectuées. En 1954, dans les premières versions du langage Fortran (pour « *Formula translator* »), les variables, quel que soit leur rôle, sont représentées par une séquence d'un ou deux caractères alphanumériques, le premier indiquant le type de la variable (entier ou « flottant »). Plus tard, dans les langages informatiques, on manipulera des mots[6] facilitant entre autre la lecture et la compréhension du programme.

Le paramètre, fondateur et obstacle

Ainsi, le paramètre semble être un concept non seulement fondateur de l'algèbre moderne, puisqu'il permet d'exprimer des solutions génériques, mais aussi un concept essentiel pour l'informatique, puisque ces solutions génériques pourront être exécutables par une machine, sans nécessiter d'interprétation. En effet, si le *« dispositif d'exécution »* (Samurçay & Rouchier, 1985) est un humain, les exemples génériques (et toute la part d'implicite qu'ils portent) sont interprétables : « Si le nombre est 15 alors tu fais 15-1 » pourra être interprété et adapté par un humain si le nombre n'est plus 15 mais 342. En revanche, un dispositif d'exécution automatisé ne peut faire cette interprétation.

Si le paramètre est un concept fondamental, on peut se demander, à la suite de Furinghetti et

[6] Créer des variables s'apparente à la création de nouveaux mots d'un langage, dont le sens est localisé au problème dans lequel elles sont mobilisées.

Paola (1994), si la longue période séparant les premières dénominations d'inconnues de la représentation des paramètres n'impliquerait pas un obstacle épistémologique. En effet, l'absence de nécessité du paramètre lorsque le dispositif d'exécution est un humain n'est pas la seule raison de l'arrivée tardive de sa représentation avec un symbole. Ely et Adams (2012) soulignent que, pour les mathématiciens grecs, *« identifier un objet à sa représentation ou remplacer un objet déterminé par un objet susceptible d'être déterminé était impensable »*. Serfati (2010) met en avant la difficulté de dépasser la dialectique unique-multiple, puisqu'il s'agit *« de combiner au sein du système symbolique deux concepts jusqu'alors considérés comme opposés, l'arbitraire et le fixe, ou l'un et le multiple ou même, peut-être de manière plus significative, l'indéterminé et le singulier »*. Dans le cas de l'inconnue, la représentation unique de l'inconnue est identifiable à la valeur de cette inconnue, puisque cette valeur est en quelque sorte « déjà là » (*cf.* figure 3). Alors que dans le cas du paramètre, un objet est identifiable à de multiples valeurs possibles, qui ne deviendra déterminé que lorsque le calcul deviendra effectif. Dès lors, comment concevoir que le « donné » puisse être en même temps arbitraire ? Radford (2006b) estime que cette rupture épistémologique a été rendue possible par la recherche nécessaire de méthodes *« claires et systématiques »* exprimées sous forme d'une séquence de symboles *« que vous pouvez manipuler à la manière d'une machine, de façon efficace »* : la nécessité didactique s'accompagne d'une nécessité économique.

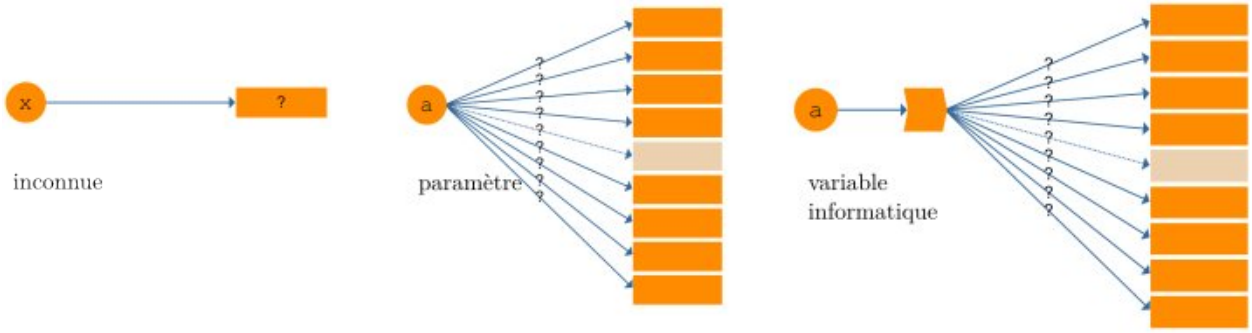


Figure 3 *: Indirections (ou dénotation) des variables.*

1.3. Généralisation de motifs et informatique

Ainsi, d'une part, la généralisation est une des caractéristiques essentielles de l'algèbre comme de l'algorithmique. D'autre part, le paramètre est nécessaire pour une généralisation symbolique (algébrique ou algorithmique) : comme le précise Radford (2006a), il ne s'agit pas seulement de capter la généralité, mais aussi d'identifier l'élément sémiotique lié à l'expression par des signes de cette généralité. De plus, non seulement *« l'apprentissage de l'algèbre élémentaire dans le secondaire reste un passage délicat pour beaucoup d'élèves alors que c'est un moment décisif de leur cursus scolaire »* (Pilet & Grugeon-Allys, 2021), mais aussi, selon Coppé et Grugeon (2009), *« on trouve peu de problèmes de généralisation [...] et presque jamais en activité d'introduction »*. Enfin, on remarque que dans les situations de généralisation de motifs, la généralisation symbolique est soit imposée (il faut l'exprimer avec des lettres), soit rendue potentiellement nécessaire par une situation fictive de communication différée vers un individu — donc un dispositif d'exécution humain, comme dans l'exemple donné par Bronner (2015)[7].

Nous avons donc envisagé la possibilité d'explorer la construction du paramètre et de sa symbolisation au travers d'une situation de généralisation de motifs informatisée. En effet, faire-

[7] *Les chaînes du joaillier* : des chaînes sont constituées de mailles carrées, il s'agit « d'envoyer un message au bijoutier » lui expliquant comment déterminer le nombre de tiges nécessaires pour former ces chaînes.

faire une généralisation de motifs par une machine, un dispositif d'exécution automatisé pour lequel la description de la méthode se fait dans un langage (formel) θ , « rend nécessaire l'existence d'une représentation des paramètres dans θ ainsi que celle d'expressions de θ manipulant ces représentations de paramètres comme s'ils étaient connus » (d'après Legrand, 2022). Les élèves doivent *« avoir recours à des quantités indéterminées des modes idiosyncrasiques ou spécifiques, culturellement et historiquement évolués, de représentation/symbolisation de ces quantités indéterminées et de leurs opérations, tout en manipulant ces quantités indéterminées de manière analytique »*, ce qui correspond aux caractéristiques de la pensée algébrique proposées par Radford (2018, p. 8, traduction libre).

2. Une situation de généralisation de motif informatisée[8]

2.1. Le Cadre de l'Apprentissage par Problématisation (CAP)

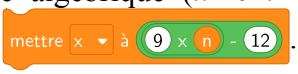
Nous nous situons dans le *Cadre de l'Apprentissage par Problématisation*, dans lequel on s'intéresse à la construction de savoirs *raisonnés*, par la mise en tension de faits et de nécessités (Fabre & Orange, 1997). Dans ce cadre, on considère l'activité scientifique :

> *comme la co-construction articulée de trois registres : le registre des modèles qui correspond aux explications construites ; le registre empirique des « faits » à expliquer ou permettant d'expliquer (issus d'observations, d'expériences ou modèles factualisés) et considérés comme pertinents pour le problème travaillé ; le registre explicatif c'est-à-dire les présupposés théoriques acceptés (rationalité des élèves, types d'explications reconnues par la communauté scientifique, principes structurants des disciplines notamment)* (Hersant & Orange Ravachol, 2015).

Les faits peuvent être des éléments mobilisés (des connaissances anciennes, des données du problèmes) ou construits par l'expérience ou le raisonnement. Leur articulation avec les nécessités, les explications construites (registre des modèles), permettent la construction de savoirs scientifique « apodictiques » : il ne s'agit pas de savoir que *c'est vrai*, mais que, dans un cadre donné, *cela ne peut pas être autrement*.

Dans la situation mise en œuvre, nous cherchons ainsi à construire la nécessité du paramètre dans un problème de généralisation de motif. Il s'agit :

> *d'établir la nécessaire existence d'un objet représentant une certaine propriété non définie a priori, ainsi que la nécessaire existence, dans le registre de représentation sémiotique de la construction du problème (θ), d'un symbole permettant de manipuler cet objet comme s'il était connu, dans une expression mettant en relation la valeur que dénote l'objet et au moins une autre valeur du système mathématisé* (Legrand, 2022).

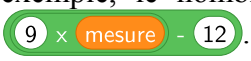
En référence au *« langage de la théorie »* de Diophante (Christianidis, 2007), θ désigne le registre sémiotique dans lequel on produit une méthode de résolution du problème. Ce registre peut être algébrique ($x=9n-12$) ou celui du dispositif d'exécution choisi, dans notre cas *Snap!*[9] : 

.

La résolution d'un problème de généralisation de motifs ne nécessite pas obligatoirement un paramètre symbolisé et une expression manipulant ce symbole. Dans le cadre de notre expérimentation, un élève explique ainsi à son binôme « on fait moins deux », ce qu'un humain peut interpréter. Un autre groupe a produit l'écrit « on prend la mesure, multiplier par deux on

[8] Nous reprenons et développons dans cette partie des éléments donnés dans Legrand (2022).

[9] https://snap.berkeley.edu/

soustraie 1 multiplier par 3 la mesure et soustraire 9 ». Ce groupe mobilise bien *« un objet représentant une certaine propriété non définie* a priori *»* (Legrand, 2022, p. 4) mais le représente dans une expression rhétorique, et non dans un registre sémiotique algorithmisable. On peut aussi trouver des exemples à caractère générique : $9\times3+(9-2)\times6$ est décliné pour plusieurs valeurs exemplaires ($4\times3+(4-2)\times6$ et $7\times3+(7-2)\times6$). Or, si le dispositif d'exécution est automatisé, de telles solutions sont impossibles. Un dispositif d'exécution automatisé, pour lequel on décrit les instructions à exécuter dans un langage θ, nécessite bien une représentation du paramètre dans θ ainsi qu'une représentation dans θ de l'expression mettant en relation le paramètre et une valeur recherchée. Par exemple, le nombre total d'hexagones dans notre situation s'exprime dans *Snap!* par la relation 

.

2.2. Présentation de la situation

Nous avons donc choisi de mettre en œuvre une situation de généralisation de motifs informatisée, dont nous allons donner une rapide description[10].

Le motif que nous avons choisi de mettre au travail représente la première itération d'un triangle de Sierpinski (TS) formé par des pièces jointives hexagonales (*cf.* figure 4). Nous définissons la *mesure* comme étant le nombre d'hexagones constituant un côté du triangle de base. Dans la suite de ce texte, TS6 désignera le tracé du motif pour une mesure de 6 hexagones. $TS(6)$ représentera le script permettant de produire le tracé. Dans l'exemple de la figure 4, le nombre d'hexagones nécessaires pour tracer le TS6 est 42.

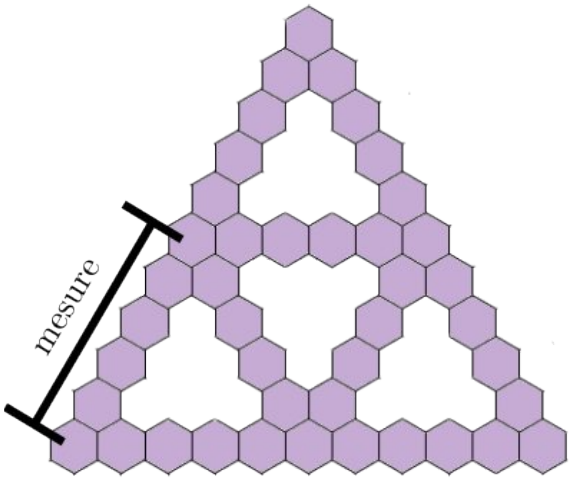


Figure 4 : Le motif "triangle de Sierpinski" de mesure 6.

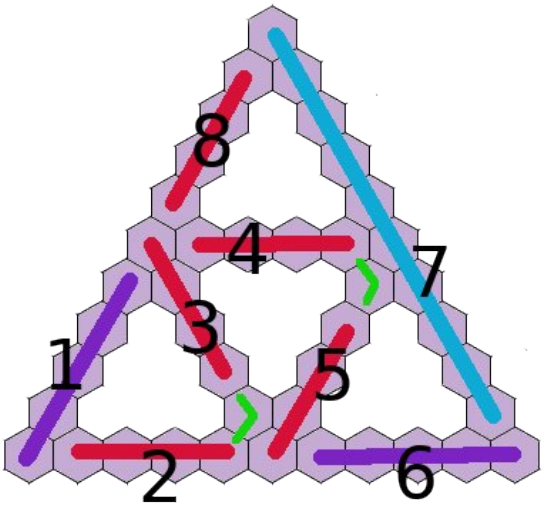


***Figure 5** : Séquence de boucles traçant un TS.*

[10] Voir en annexe le script pour le TS4 et le script générique attendu.

Problème donné : vers la nécessaire généralisation d'un script

Le problème donné aux élèves est de déterminer quel est le plus grand TS que l'on peut construire avec 35 hexagones, puis avec 1 400, 1 654 ou 1 710 hexagones[11].

Pour résoudre la première question, les élèves vont pouvoir construire et dénombrer différents motifs à la main, en utilisant une grille hexagonale.

En revanche, pour les quantités d'hexagones suivantes, la méthode manuelle devient impossible à mettre en œuvre. Il est alors proposé aux élèves de faire construire et dénombrer le tracé de n'importe quel TS en utilisant un programme, écrit dans l'environnement de programmation graphique par blocs (EPGB) *Snap!* (Legrand, 2019). Quatre phases suivront : les trois premières ont comme objectif, pour les élèves, d'aboutir à la réalisation d'un script générique de tracé et dénombrement des TS afin de résoudre le problème donné ; la tâche prescrite lors de la quatrième étant de trouver une méthode permettant « d'aller plus vite que l'ordinateur ».

Caractéristiques du script

On ne demande pas aux élèves de construire le script de tracé, relativement complexe pour des débutants. Le script permettant de tracer un TS4 est fourni, les élèves doivent le modifier en fonction des tâches prescrites. Le script choisi est une séquence de boucles traçant le TS *sans trou ni chevauchement*, c'est-à-dire en traçant chaque hexagone un à un de façon contiguë, sans jamais tracer deux hexagones au même endroit. Ainsi, dans l'illustration du tracé d'un TS6 donné figure 5, le script commence par tracer successivement 5 hexagones (boucle 1), puis après une rotation 4 hexagones (boucle 2). Une nouvelle rotation sera suivie d'un hexagone unique, puis d'une rotation avec tracé, de nouveau, de 4 hexagones (boucle 3), et ainsi de suite. Le programme donnant à voir son exécution, ce mode de tracé permet un *« feed-back élaboré de vérification »* (Bosc-Miné, 2014). Ainsi, la validité du script peut-être déterminée par l'élève grâce au tracé produit d'une part, et d'autre part à la valeur du nombre d'hexagones total dénombrés. De plus, la localisation de l'erreur est en partie possible grâce à la rétroaction fournie par le tracé (un « trou » ou la manifestation de la superposition de deux hexagones par une couleur plus foncée).

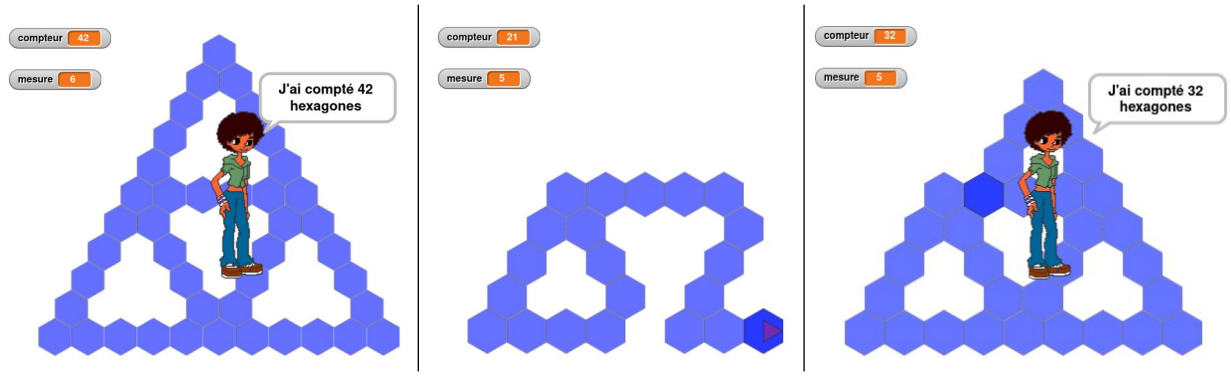


***Figure 6**: Exemples de rétroactions.*

La figure 6 montre trois exemples de rétroactions. La première est une rétroaction valide pour le

[11] Si n est le nombre d'hexagones disponibles et x la mesure du TS recherché, les valeurs de n (variable didactique) sont choisies pour éviter que la solution erronée $x = E\left(\frac{n}{9}\right)+1$ soit toujours valide. Ainsi, si $x = 9p+r$, la solution précédente est valide si $r<6$, mais invalide si $r \geqslant 6$. On choisit donc des valeurs représentant les deux cas. Par exemple 1400 ($r=5$), 1608 ($r=6$ et il ne reste aucun hexagone), 1654 ($r=7$), 1710 ($r=0$)…

tracé d'un TS6 ; la deuxième est une rétroaction invalide d'un TS5 en cours d'exécution, avec une erreur sur la boucle 4 (un hexagone en trop) ; et la troisième une rétroaction invalide pour un TS5 avec une erreur sur la boucle 7 (un hexagone en moins, confirmé par la valeur affichée).

Afin d'alléger l'écriture, nous utiliserons dans la suite de ce texte « b_i » pour désigner indifféremment « la boucle *i* », « l'expression donnant le nombre d'itérations de la boucle *i* » ou « le nombre d'itération de la boucle *i* ». « b_i^n » désignera de la même façon la boucle ou le nombre d'itération de la boucle *i* de l'instance $TS(n)$ (soit le script traçant et dénombrant un TS de mesure *n*).

Ces boucles peuvent être classées en trois familles :

- les boucles b_1 et b_6, dont le nombre d'itérations est $mesure-1$;
- les boucles b_2, b_3, b_4, b_5 et b_8, dont le nombre d'itérations est $mesure-2$;
- la boucle b_7, dont le nombre d'itérations est $2\times mesure-2$.

Lorsqu'il n'y a pas d'ambiguïté, nous confondrons la boucle et sa famille.

Réalisation du script générique

Dans un premier temps, le script traçant le TS4 étant fourni, il faut lui « apprendre à compter », soit modifier ce script pour qu'il trace le TS4 mais aussi dénombre les hexagones nécessaires pour le tracé. Cette phase permet aux élèves de se familiariser avec la structure du script proposé, et de rencontrer une première variable informatique, dans un rôle de compteur (Sajaniemi, 2005). Précisons ici que nous parlerons de « variable » pour désigner « une représentation dans un système sémiotique donné d'un objet représentant *a priori* un objet indéterminé, soit non encore déterminé mais susceptible de l'être ». La définition de la variable informatique qui sera institutionnalisée avec les élèves à l'issue de cette phase sera « une variable informatique est un nom (ou un symbole) qui fait référence à une zone de mémoire, contenant la représentation d'une certaine donnée. Elle permet de stocker une valeur et de la réutiliser plus tard, ou de la modifier ».

Dans une deuxième phase, les élèves devront dupliquer et modifier le script afin de tracer et dénombrer d'autres instances de TS : les TS5, TS6, TS7, puis TS11 et TS22. Cette phase vise à amener les élèves à identifier les relations entre instances puis entre instance et mesure.

La phase 3 est la phase de généralisation : les élèves doivent modifier leur script afin qu'il trace et dénombre n'importe quelle instance de TS, la mesure étant choisie par l'utilisateur au lancement du script, et mémorisée dans la variable *mesure*. Faire-faire ce tracé générique par un programme écrit dans *Snap!* implique donc pour les élèves de construire la nécessaire existence :

- d'un objet *o* dénotant une valeur arbitraire non définie *a priori*, ici la valeur du paramètre *mesure* entrée par l'utilisateur ;
- disposant d'une représentation $R_\theta(o)$ dans le registre de représentation sémiotique θ dans lequel on exprime l'algorithme permettant de résoudre le problème, ici *Snap!*, donc $R_\theta(o)=$  ;
- dont cette représentation est manipulable dans une expression de θ déterminant la relation entre la mesure et le nombre d'itération de la boucle b_i, comme si la valeur de $R_\theta(o)$ était connue.

Dans notre cas, les expressions attendues sont donc :

- $b_1=$  ,
- $b_2=$  ,
- $b_7=$  ou $b_7=$  ou équivalent.

Dans le registre sémiotique A de l'algèbre, $R_A(o)=n$, $b_1=n-1$, $b_1=n-2$ et $b_7=2n-2$ ou $b_7=2(n-1)$ ou équivalent.

Une fois le programme générique construit, les élèves peuvent rechercher les réponses au problème initial en procédant par essai-erreur. À la suite de ces essais, les temps d'exécution des scripts s'amplifiant, il est proposé aux élèves de trouver une façon de dénombrer les hexagones d'un TS quelconque en allant « plus vite que l'ordinateur », avec pour objectif d'amener les élèves à formuler la solution sous forme algébrique : $9n-12$ ou toute formule équivalente.

Une phase supplémentaire visera à demander aux élèves de produire plusieurs formules équivalentes permettant d'obtenir le nombre d'hexagones total d'un TS quelconque, en les justifiant et en implémentant ensuite ces formules dans un script *Snap!* permettant de les tester, tout en amenant les élèves à être confrontés à des difficultés liés à la structure profonde des expressions.

2.3. Méthodologie

La situation a été mise en œuvre par une enseignante expérimentée dans deux classes de quatrième n'ayant eu qu'une initiation à *Scratch*. Comme dans le cadre de l'apprentissage par problématisation, on s'intéresse au processus de construction et de résolution du problème, les seuls scripts finaux créés par les élèves sont insuffisants pour l'analyse. Nous avons donc conçu un dispositif de captation automatisé des traces de programmation sous *Snap!* (Legrand, 2019). L'environnement de programmation a ainsi été instrumenté afin de capter, stocker puis représenter les événements de programmation produits par les élèves. Avec ce dispositif, on peut reconstituer, représenter et parcourir l'« histoire[12] du programme », quelle que soit la situation de programmation. Dans notre cas, les événements sont ensuite filtrés afin de ne conserver que ceux ayant un rôle dans notre étude : ici, les modifications des boucles b_i, les exécutions et leur rétroaction, ainsi que les modifications impliquant des variables. On ne s'intéresse pas spécifiquement à la modification de la structure du programme, les élèves n'ayant normalement pas à la modifier.

Ces événements sont ensuite représentés sous forme d'épisodes « action - effets de l'action ». Nous disposons des traces de programmation de tous les binômes, et trois groupes dans chaque classe ont été en outre filmés, les événements de programmation ayant été intégrés aux transcriptions des interactions langagières. L'analyse de ces épisodes, avec ou sans interactions langagières captées, permet de construire des espaces de contraintes (Legrand, 2022 ; Choquet *et al.*, 2022) représentant les tensions entre faits et nécessités. Nous décrirons ici celles qui sont en jeu dans la phase de généralisation, et qui sont susceptibles d'amener les élèves à la construction du concept de paramètre.

[12] *« Ensemble d'événements, évolution concernant une personne ou une chose ».* https://www.cnrtl.fr/definition/histoire

3. La construction du paramètre : manifestations de la tension fixe-variable

Nous nous intéressons ici à la phase de généralisation du script : il s'agit de modifier le script du tracé afin de tracer et dénombrer n'importe quel TS dont la mesure est entrée par l'utilisateur. Cette entrée est stockée dans la variable *mesure* , ce qui est précisé et rappelé aux élèves. À ce stade, les élèves ont déjà construit plusieurs instances du script à partir du $TS(4)$: il leur était demandé préalablement de construire les $TS(5)$, $TS(6)$, $TS(7)$ puis $TS(11)$ et enfin $TS(20)$, $TS(21)$ ou $TS(22)$. Ils doivent à présent construire $TS(n)$ (où n est la mesure du TS choisi par l'utilisateur).

Il s'agit d'une phase de généralisation algébrique : les élèves doivent identifier une régularité dans l'ensemble des instances traitées, prendre conscience que cette régularité s'applique à toutes les instances possibles, et surtout exprimer cette régularité pour tracer et dénombrer n'importe quelle instance (Radford, 2006a). Pour rappel, cela revient à exprimer dans *Snap!* les relations $b1=n-1$, $b2=n-2$, $b_7=2n-2$, ce qui donne :

- $b_1=$ 

,
- $b_2=$ 

,
- $b_7=$ 

ou $b_7=$ 

.

Nous allons nous focaliser sur deux groupes : le groupe 46m, dont seules les traces de programmation sont récoltées, ainsi que le groupe 46e, pour lequel traces de programmation et interactions langagières ont été captées. Ces deux groupes sont les seuls de cette classe à avoir abouti à l'utilisation de la variable *mesure* dans leur script, et ils illustrent à eux deux une grande partie des phénomènes observés sur l'ensemble des groupes. Nous verrons par la suite quelles tensions sont en jeu lors de cette phase de généralisation, et comment elles sont réglées.

3.1. Première tension : un pour tous (script)

Un même script pour plusieurs mesures (46e)

La première tension qui apparaît est la nécessité d'avoir *un seul* script pour tracer *tous* les TS. Pour le groupe 46e, l'élève E2 se demande ainsi « comment on change ça » (S4, 392) en indiquant la partie « initialisation » ou « commentaire » du script correspondant au $TS(22)$ pour lequel « c'est là qu'il y a 22, mais pour celui-là [le script $TS(n)$] on sait pas » : comment changer ce qui est fixe et connu pour représenter quelque chose de variable et de non encore connu ? E1 renchérit ensuite :

398. Script[13] : START - ASK <<Quelle est la mesure du triangle de Sierpinski que tu veux dessiner ?>>
400. E1 : *Et là par exemple si je mets huit.*
401. Script : ANSW <<8>>
402. E1 : *Il faut que | il m'en fasse huit 'fin un de mesure huit.*
403. E1 : *Mais sauf que là ça va m'en faire un | un de mesure vingt-deux.*
404. Script : STOP receiveKey(478)
405. E2 : *Ah oui.*

[13] Les événements de programmation sont directement intégrés dans la transcription, mentionnés par « *Script* ». « *START* » est le codage du lancement d'un script. « *ASK* » celui d'une demande d'entrée par l'utilisateur. « *ANSW* » la valeur entrée par l'utilisateur. « *STOP* » l'arrêt d'un script, déclenché par l'utilisateur. Pour les boucles $b1i0(VAL)$: rpt ***10*** fois <<**8**>> indique que l'expression donnant le nombre d'itérations de la boucle **1** était « **8** » et a été remplacée par « **10** ». Ce qu'on écrira $b1 \leftarrow 10$.

406. E1 : *Donc il faut trouver un programme.*

407. E1 : *Pour aller pour toutes les mesures* [geste englobant avec la souris].

Le script $TS(n)$ est à ce moment issu d'une duplication de la structure du $TS(22)$ — donc $b_1=21$, $b_2=20$, $b_7=42$ — , avec l'ajout de la demande d'entrée de la mesure par l'utilisateur. E1 pointe la première tension, entre la nécessité que le script trace le TS dont la mesure est donnée par l'utilisateur (ici 8) et le fait qu'un script donné trace un TS donné (ici 22). Il faut donc trouver « *un* programme » permettant de tracer « *toutes* les mesures » : un même script doit tracer toutes les instances de TS.

Une entrée différente pour un tracé différent (46e et 46m)

Cette nécessité implique que lorsque la mesure change, le tracé doit changer aussi. Cela entre en contradiction avec les essais produits par les élèves des deux groupes. 46e, ultérieurement, tentera ainsi $n=22$[14], suivi de $n=3$ puis $n=8$, ce qui produira toujours un TS22. De même, le groupe 46m — qui lui s'est basé sur le script $TS(4)$ — procède à deux tests consécutifs : le premier confirmant que lorsque $n=4$, leur script trace bien le TS4, le deuxième rendant compte que si l'entrée est $n=8$, leur script trace *toujours* le TS4. Ultérieurement, ils feront deux tests similaires avec $n=4$ suivi de $n=5$; plus tard, à partir d'un script traçant le TS8, ils effectueront les tests pour $n\in\{8,9,5,8\}$ (dans l'ordre), et, après une nouvelle modification, pour $n\in\{8,8,9,8\}$. Lors de tous ces tests impliquant un script instancié, non générique, les élèves semblent ainsi avoir bien conscience qu'un même script doit produire des résultats différents suivant l'entrée. Malgré tout, un seul test ne semble pas leur suffire : obtenir un TS8 pour $n=8$ puis $n=9$ n'implique pas chez les élèves l'invalidation *a priori* du script pour $n=5$. Ces vérifications supplémentaires non nécessaires peuvent laisser penser que les élèves sont perturbés par cette tension, voire qu'ils s'attendent à ce que cette tension se règle d'elle-même, ce qui serait une occurrence de la « pensée magique » que de nombreux élèves prêtent aux artefacts informatiques : ils font ce qu'on attend d'eux qu'ils fassent. Notons que faire de multiples tests avec des entrées différentes pour un script générique n'a pas le même sens : il ne s'agit plus de vérifier si « ça change ou pas », mais plutôt de vérifier « si ça change comme prévu ».

3.2. Deuxième tension : un pour tous (nombre) ou l'insuffisance du nombre

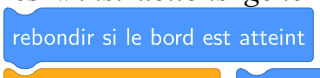
Ce qui change ou pas : le nombre d'itérations des boucles et sa représentation dans (46e)

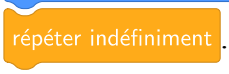
Les élèves ont donc à résoudre la tension entre deux éléments contradictoires :

- lorsqu'on modifie l'entrée, le tracé change ;
- lorsqu'on modifie l'entrée, on ne modifie pas le script.

En outre, lors des séances précédentes, ils ont été amenés à construire la nécessité de modifier le nombre de répétitions de chacune des boucles : les modifications faites au script ne concernent pas la structure même du script (la succession de ses instructions) mais les valeurs des nombres de répétitions des boucles. Il ne s'agit pas de déterminer les instructions nécessaires au tracé, mais les valeurs spécifiques de certaines de ces instructions, ici les boucles. Cela est montré par les traces de programmation, puisque dans la plupart des cas, lorsque les élèves doivent passer d'une instance à une autre, ils modifient l'ensemble des valeurs des boucles avant de lancer une exécution validant ou non leur modification. En reprenant le vocabulaire utilisé par Komis *et al.* (2017), les élèves remplacent une *instruction spécifique* par une autre. Komis *et al.* différencient

[14] Ils lancent donc l'exécution du script et entrent la valeur « 22 » lors de la demande d'entrer la mesure : ils testent le script $TS(n)$ pour vérifier s'il trace correctement le TS22.

les *« instructions générales »* des *« instructions spécifiques »*. Par exemple, le bloc d'action 

est une instruction générale, tout comme le bloc de contrôle 

. 

et 

sont en revanche des instructions spécifiques : leur effet dépend d'une certaine valeur. Pour passer du script $TS(4)$ au script $TS(5)$, les élèves vont ainsi transformer l'instruction spécifique de la boucle b_1, en passant de 

à 

. Les élèves du groupe 46e précisent ainsi « c'est euh répéter neuf fois » (S4, 332), considérant l'instruction 

. Ils s'intéressent ensuite à la *valeur* de cette instruction : « c'est pas neuf fois c'est euh » (S4, 332) puis « nan c'est pas dix » (S4, 334) et « là du coup c'est neuf » (S4, 353). Ainsi, changer l'instruction spécifique revient à changer la valeur de cette instruction, et E1 l'explicite en remplaçant toutes les boucles par 

tout en affirmant « mais euh tu vois c'est juste faut changer ça » (S4, 355). E1 joint alors l'action à la parole, en changeant l'ensemble des boucles afin d'obtenir un $TS(20)$ ($b_1=19$, $b_2=18$, $b_7=38$). E1 n'est plus à la recherche d'instructions spécifiques résolvant le problème en cours (l'instance ou l'ensemble d'instances à construire) : elle recherche *une* instruction qui serait valide pour toutes les instances. De plus, les ◯ marquent *ce qui doit changer* lors de l'exécution, et E1 simule *ce qui devrait se passer* si on choisit de tracer un TS20. Ainsi, le nombre comme valeur définissant une instruction spécifique n'est pas suffisant, il faut trouver par quoi le remplacer.

L'insuffisance du nombre (46e)

Tout en terminant ces modifications, E1 entame alors la discussion suivante :

473. E1 : *Mais | on marque quoi à la place ?*
474. E2 : *Ah | je sais pas on prend quoi comme mesure du coup ?*
475. Script : $b\,8\,i\,0\,(VAL)$: rpt ** fois <<18>>
476. Script : AFFBL_motion
477. E1 : *Bah euh aucun.*
478. E1 : *Il faut que ça aille euh toutes les mesures possibles (claque des doigts).*
479-481. E2 : *Bah, ça veut dire qu'il faut eu un multiple? Et euh | de tout ça ?*

...

493-496. E2 : *Ah je sais pas | heum | bah un truc qui finit par un zéro | parce que zéro c'est un multiple de tout.*
497. *E1 :* Ah non ça | oui mais euh.
498. E1 : *Oui mais si on marque par exemple euh dix ça va le faire euh chais pas combien de fois.*
499. Script : $b\,1\,i\,0\,(VAL)$: rpt *10* fois <<>>
500. E1 : *'fin ça va pas euh XXX pour tous.*
501. E1 : *Je pense qu'il y a un bloc qui existe.*
502. E1 : $b\,1\,i\,0\,(VAL)$: rpt ** fois <<10>>
503. Script : AFFBL_variables
504. E1 : *XXX.*
505. E2 : *(encore).*
506. E1 : *On se prend (les variables).*
507. E1 : *Les variables.*

E1 semble chercher un signe, un symbole ou une expression de *Snap !* que l'on pourrait mettre « à la place » des valeurs spécifiques pour « toutes les mesures possibles » : cet espace ◯ peut s'apparenter à un marque-place, une marque indiquant une « place à prendre », ne désignant pas ici un inconnu, mais un donné, donc un paramètre (*placeholder*). E2, quant à elle, reste dans le registre sémiotique des nombres liés à l'instance (« on prend quoi comme mesure ? »). Elle cherche un nombre qui pourrait les représenter tous (« un multiple de tout ça ») et propose un signe (« un truc ») dont l'écriture termine par « 0 », puisque « zéro c'est un multiple de tout »[15]. E1 invalide alors cette hypothèse, puisque si un nombre est utilisé comme représentation de la valeur du nombre d'itération, cela ne peut correspondre qu'à un seul tracé, à une seule instance.

3.3. Troisième tension : une pour tous (expression)

Ce groupe met en avant les faits suivants :

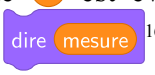
- un même script doit tracer plusieurs instances ;
- si la mesure change, le tracé change ;
- ce qui fait changer le tracé est le nombre d'itérations de chacune des boucles ;
- utiliser l'écriture d'un nombre comme expression de la valeur du nombre d'itérations ne permet de tracer qu'une instance.

E1 en conclut que, sans doute, « il y a un bloc qui existe », c'est-à-dire qu'il existe une expression dans *Snap!* qui représente la valeur du nombre d'itération des boucles pour toutes les instances de TS. Cette expression étant instanciée, évaluée, au moment de l'exécution, lorsque la mesure est connue, comme E1 l'a simulé en remplaçant les ◯ par les valeurs correspondant à un TS20. On passe ainsi du registre des nombres à celui des expressions numériques dans un algorithme. Cela souligne aussi les différences sémantiques entre les instructions 

et 

: la première est spécifique tandis que la seconde est paramétrée et ne devient spécifique qu'à l'exécution, au moment où la valeur de 

est évaluée. Cela peut être une difficulté didactique : comment différencier 

et 

[16] ?

L'expression donne un nombre (46m)

Généraliser le script traçant les TS revient donc à déterminer une expression dans *Snap!* qui représenterait le nombre d'itérations d'une boucle, ce nombre dépendant de la mesure entrée. Il faut une expression dans *Snap!* représentant la relation entre le nombre d'itérations d'une boucle et la mesure.

Le groupe 46m va ainsi explorer différentes expressions pour la boucle b_1 : 

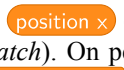
, puis en utilisant 

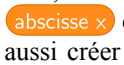
, 

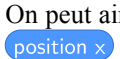
ou encore 

. Ces expressions seront à chaque fois testées pour une mesure entrée de 8. Elles

[15] Cette utilisation du zéro ou d'écritures de puissances de dix est présente chez plusieurs groupes.

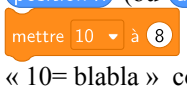
[16] Profitons-en pour signaler une autre difficulté didactique des EPGB comme *Snap!* ou *Scratch* : il est possible d'avoir des variables différentes portant le même nom, ou même dont le nom est la représentation d'un nombre. On peut ainsi nommer une variable 

(ou 

dans *Scratch*) qui est différente de la variable interne 

(ou

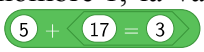
dans *Scratch*). On peut aussi créer une variable 

, et utiliser une instruction comme 

ou même 

, ce qui pourrait amener à considérer les propositions « 10=8 » ou « 10= blabla » comme étant vraies !

sont des propositions dont la valeur est soit « vrai », soit « faux », mais cela ne produit pas d'erreur à l'exécution car dans *Snap!*, comme dans la plupart des langages impératifs, la valeur booléenne « vrai » est représentée par le nombre 1, la valeur « faux » par le nombre 0. Ainsi, 

représente la valeur 6, et 

la valeur 5. Les différents essais des élèves vont aboutir respectivement au tracé de un, zéro, un et un hexagone. Mais quel est le sens de ces expressions pour les élèves ?

En premier lieu, les élèves de ce groupe, comme pour le groupe 46e, sont bien à la recherche d'une *expression* mettant en jeu un ou des nombres, et non plus à la recherche d'*un* nombre qui permettrait de tracer toutes les instances. Ils testent en premier lieu les opérateurs booléens, ou plutôt sans doute les opérateurs *qui ne sont pas des opérateurs arithmétiques* (*cf.* figure 7).

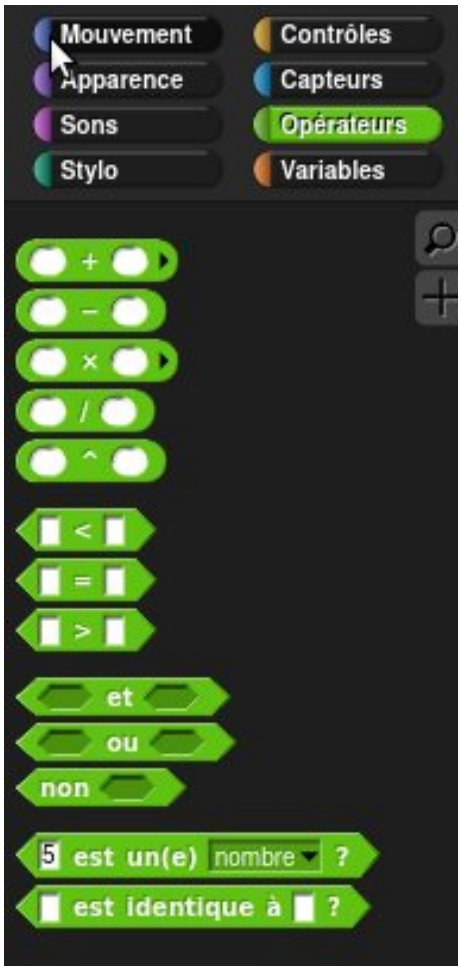


***Figure 7** : Blocs « opérateurs » disponibles.*

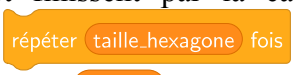
En outre, la première expression utilise la valeur 8, qui est aussi la valeur de la mesure qui est testée : il s'agit bien d'une représentation de la relation entre le nombre d'itérations et la mesure. Ici, comme chez les babyloniens, on peut supposer que le 8 prend un statut de *nombre générique* (que nous noterons NG dans la suite de ce texte). Dans le deuxième essai, ce NG nous semble apparaître de façon implicite : il existe une relation entre 4 et 2 permettant d'obtenir la valeur 8. Nous estimons qu'il y a ici, probablement, une *perte du caractère générique du nombre*, ce que nous noterons PNG. Dans la troisième expression, la valeur 5 est la valeur par défaut de cette instruction, ce qui pourrait suggérer que cette valeur représente la mesure qui sera entrée, puisque ce 5 est présent « automatiquement ». C'est pour nous un *nombre potentiellement*

générique (NPG). Enfin, pour la dernière expression, il n'y a plus de représentation d'un nombre, remplacée par un s*igne qui pourrait être porteur de généricité* (SG). Les élèves semblent donc bien être à la recherche d'une expression incluant la représentation de la mesure.

Enfin, on peut raisonnablement penser que, pour ces élèves, les expressions utilisées n'ont pas le sens qu'un expert leur donnerait. « Répéter 8 est identique à 8 » est sans doute interprété comme « répéter le nombre de fois qu'il faut pour la mesure comme on le ferait si la mesure était 8 » : ils cherchent une expression dans *Snap!* qui correspondrait au sens voulu, ils ne cherchent pas à comprendre le sens de l'expression. Enfin, on peut raisonnablement penser que pour ces élèves, les expressions utilisées n'ont pas le sens qu'un expert leur donnerait. « Répéter 8 est identique à 8 » est sans doute interprété comme « répéter le nombre de fois qu'il faut pour la mesure comme on le ferait si la mesure était 8 » : ils cherchent une expression dans *Snap!* qui correspondrait au sens voulu, ils ne cherchent pas à comprendre le sens de l'expression.

L'utilisation des expressions mobilisant les opérateurs  étant invalidée, les élèves vont ensuite explorer les opérateurs arithmétiques  . Toujours en entrant une mesure de 8 pour leurs tests, ils expérimentent successivement pour b_1 les expressions  ,  ,  , puis $b_1 \leftarrow$  et $b_2 \leftarrow$  . Pour finir, mais sans tester, ils écriront $b_1 \leftarrow$  , $b_2 \leftarrow$  tout en fixant toutes les boucles restantes à  . Ils annuleront alors toutes leurs modifications en rechargeant le programme de base de la séance.

Ici encore, les élèves semblent se focaliser sur la recherche d'une expression qui « fonctionnerait », plutôt que sur le sens des expressions envisagées. Ceci est confirmé par exemple par l'utilisation successive de 4×2 et 2×4, comme s'ils s'attendaient à un résultat différent — alors qu'on peut estimer qu'en quatrième, le fait numérique $4\times2=2\times4$ est construit. On remarque là encore la présence récurrente des deux PNG 4 et 2, les élèves semblent donc toujours à la recherche d'une expression représentant la relation entre le nombre d'itérations et la mesure, impliquant une représentation de la mesure. Mais jusqu'ici, ni la relation ni la représentation de la mesure ne sont visibles[17].

Les élèves du groupe 46e, quant à elles, recherchent elles-aussi un « bloc » (une expression de *Snap!*) qui représenterait la relation entre le nombre d'itérations et la mesure, mais elles explorent le langage en examinant l'ensemble des blocs disponibles dans chaque catégories. Elles commencent et finissent par la catégorie « variables », en testant en premier lieu  puis  pour la boucle b_1. Il peut paraître surprenant que les élèves aient abandonné le  qui les aurait peut-être engagées sur la voie de l'expression  : l'erreur de tracé est aisément interprétable, et avant même de tester E1 remarque que « là [la boucle $b_1=$ ] c'est pas le même truc que ta mesure » (S4, 541). E1 semble ainsi considérer que  dénote la valeur de la mesure puisqu'elle prend en compte le fait que pour une mesure donnée, le nombre d'itérations de la boucle b_1 est différent du nombre donné par la mesure.  est bien pour cette élève une représentation dans *Snap!* de la valeur de la mesure, mais, malheureusement, les tests avec  ont abouti à une erreur d'exécution dont l'origine n'a pas pu être identifiée[18]. La rétroaction étant dans ce cas une absence de tracé, les élèves semblent avoir invalidé l'utilisation de cette expression et ont donc exploré d'autres

[17] On peut noter que la recherche de l'expression semble primer sur l'identification de la relation pour le groupe 46m comme pour la moitié des groupes analysés.

[18] Il s'agit d'une erreur d'indexation dans le tableau des variables que rien dans les traces ne permet d'expliquer, mais qui est apparue à d'autres reprises chez certains groupes.

solutions. Lors de la séance suivante, elles vont néanmoins re-mobiliser dès le début la variable pour exprimer la relation entre le nombre d'itération et la valeur de la mesure.

3.4. Quatrième tension : une pour tous (relation)

Arrivés à ce stade, les élèves semblent avoir construit les nécessités suivantes :

- il faut (et il suffit de) changer les instructions b_i (le nombre d'itérations de la boucle n° i) pour créer une instance valide de $TS(n)$;
- il y a une relation entre b_i et la mesure n ;
- il existe des expressions $S_\theta(i)$ dans *Snap!* permettant de représenter les relations entre le nombre d'itérations b_i et la mesure n (et ces expressions ne sont pas l'écriture d'un nombre) ;
- il existe une expression R_θ dans *Snap!* représentant la valeur de la mesure entrée ;
- $S_\theta(i)$ contient R_θ.

Il est donc nécessaire d'identifier la relation boucle-mesure, et de l'exprimer dans le « langage de la théorie ». Rappelons ici les relations valides, pour une mesure n : $b_1=n-1$; $b_2=n-2$; $b_7=2n-2$.

Recherche du pareil et du différent : le moins un ou la mesure (46m, 46e)

La construction de différentes instances de script permet aux élèves d'établir différents faits. Ainsi par exemple *a)* $b_1^5=4$, *b)* $b_1^6=5$, *c)* $b_2^5=3$ et *d)* $b_2^6=4$. Ces faits peuvent en amener de nouveaux :

- de $(a\wedge b)$ on peut établir une régularité : $b_1^5=5-1$ et $b_1^6=6-1$;
- de $(c\wedge d)$, une autre régularité peut émerger : $b_2^5=5-2$ et $b_2^6=6-2$;
- de $(a\wedge c)\wedge(b\wedge d)$, une autre : $b_1^5=5-1$ et $b_2^5=4-1$ et $b_1^6=6-1$ et $b_2^6=4-1$.

Le groupe 46m semble avoir établi les deux premières régularités, puisque, à la suite de leurs recherches infructueuses exposées précédemment, ils testent, toujours avec une mesure entrée de 8, $b_1\leftarrow$ (8 - 1) et $b_2\leftarrow$ (8 - 2). On retrouve ici encore le 8 en tant que nombre générique (NG) : ce nombre ne change pas au sein de l'instance, mais il change si on change d'instance (pour une entrée de 9, on devrait avoir $b_1\leftarrow$ (9 - 1)). En revanche, la relation change au sein de l'instance (elle n'est pas la même pour b_1 et b_2) mais reste identique quand on change d'instance (on a toujours « moins un » pour b_1 et « moins deux » pour b_2, quelle que soit l'instance) (*cf.* figure 8).

Le groupe 46e semble voir en premier lieu la troisième régularité : remobilisant la représentation de la mesure envisagée en fin de séance 4, les élèves testent $b_i\leftarrow$ (mesure - 1) (pour toutes les boucles). Pour eux, le « moins un » est le « pareil » de toutes instances et de toutes les boucles au sein d'une instance. Après plusieurs essais pour différentes mesures, E1 explique, en montrant la trace écrite constituée en groupe classe au début de la séance :

140. E1 : *Mais non ils sont euh désolé mais euh.*
141. E1 : *Moi pour moi le plus il marche pas parce que regardes (montre trace écrite classe).*
142. E1 : *La prof elle a dit | par exemple pour la mesure huit.*
143. E1 : *Après quoi la boucle d'en dessous elle fait sept.*
144. E2 : *Ha.*
145. E1 : *La mesure cinq la boucle d'en dessous elle fait quatre.*

146. E1 : *C'est différent.*
147. E1 : *Sauf que là | du coup.*
148. E1 : *Ça fait tout le temps la mesure euh.*
149. E1 : *Du coup il faudrait faire euh.*
150. E1 : *Moins un.*
151. Script : $b\,1\,i\,0\,(DROPVAL)$: rpt *[mesure − 1]* fois
152. E1 : *Et là il faudrait pas mettre mesure mais il faudrait mettre euh.*
153. Script : $b\,2\,i\,0\,(NEWVAL)$: rpt [** − 1] fois
154. E1 : *Celui-là euh.*
155. E1 : *Les nombres de la boucle.*
156. E2 : *Les nombres de la boucle c'est-à-dire euh.*
157. E2 : *Ouai tu mets euh (montre trace écrite classe).*
158. E1 : *Ou au pire on fait moins deux.*
159. Script : $b\,2\,i\,0\,(VAL)$: rpt [− *2*] fois <<1>>
160. E1 : *Puisque là elle fait moins deux.*
161. Script : $b\,2\,i\,0\,(NEWVAL)$: rpt [*mesure* − 2] fois
162. E2 : *Ouai.*
163. E2 : *Vas-y on essaye avec moins deux.*
164. Script : $b\,3\,i\,0\,(NEWVAL)$: rpt ** fois
165. E1 : *Et là c'est moins deux aussi.*
166. Script : $b\,0\,i\,1\,(VAL)$: mesure − *2*<<1>>

E1 oppose ainsi le différent (« c'est différent » s'appliquant aux deux boucles b_1 et b_2) et le pareil (« sauf que là du coup ça fait tout le temps la mesure »), ce qui l'amène à basculer sur les deux premières régularités ($a \wedge b$ et $c \wedge d$), ce qui est confirmé par le « puisque là elle fait moins deux » suivi de la modification $b_2 \leftarrow$ mesure - 2. Ce groupe a mis en tension l'unicité (ou la généricité) de la relation (« moins un ») et l'unicité (ou la généricité) de la mesure. La mobilisation d'un nouveau fait (« là elle fait moins deux ») leur permettant de régler cette tension : l'unique est la mesure, le multiple est la relation (*cf.* figure 8). Cependant, des groupes confrontés à des constructions de relations similaires se sont retrouvés en difficulté pour aboutir à la généralisation.

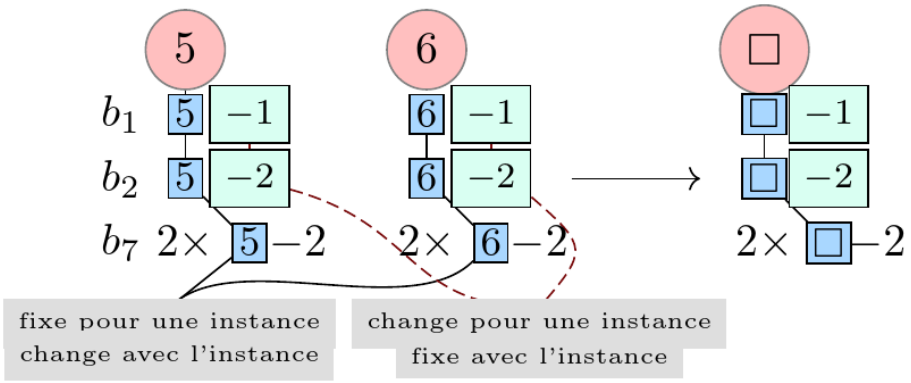


Figure 8 *: Des faits permettant d'identifier un schéma.*

Recherche de régularité : la perte de généricité

En effet, d'autres élèves ont identifié la même régularité que le groupe 46e (« moins un »), mais

l'ont formulé avec un nombre générique — qui va du coup coup perdre sa généricité : pour 8, $b_1 \leftarrow$ (8 - 1) mais $b_2 \leftarrow$ (7 - 1). Le script est valide, puisqu'il trace bien un TS8, mais la généricité du 8 va se perdre avec le 7, ce qui va rendre d'autant plus difficile l'identification d'une représentation de la mesure (*cf.* figure 9). En effet, les élèves peuvent exprimer la régularité identifiée pour trouver n'importe quelle instance (« on fait moins un puis encore moins un »), ils ne peuvent exprimer cette généricité dans un langage non ambigu comme *Snap!* ou le langage algébrique. D'autres variantes encore plus problématiques, en tout cas rendant l'identification d'une expression unique de la généricité plus difficile, sont aussi présentes, comme par exemple l'expression de la relation en fonction des valeurs d'une autre instance (le $TS(4)$ pour la Figure 10).

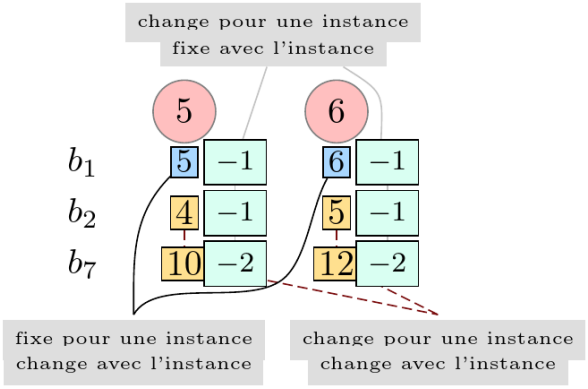


Figure 9 *: Des faits perturbant l'identification du schéma.*

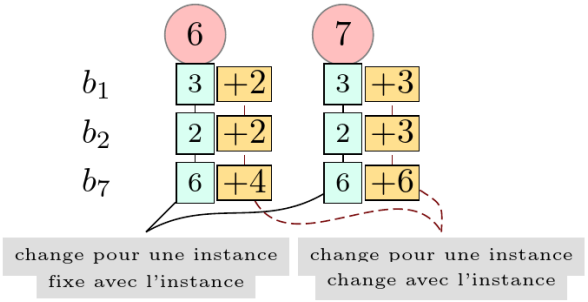


Figure 10 *: Des faits rendant l'identification du schéma peu probable.*

L'insuffisance du nombre, le retour (46m)

Le groupe 46m va aussi être confronté à cette perte de généricité. En effet, s'ils ont correctement identifié les relations boucle-mesure pour b_1 et b_2, celle pour b_7 ne l'est pas. Ils commencent ainsi par l'exprimer de façon partielle : $b_7 \leftarrow$ (8 + ◯) (« 8 plus quelque chose »). Le tracé étant erroné pour b_7, ils vont tester en premier lieu (10 - 2). On a ici encore une perte de généricité, même si le NG 8 est encore présent implicitement ($10-2=[NG]$). Ils garderont ensuite le « moins deux » en testant (5 - 2), (14 - 2), pour aboutir à une solution valide, mais ayant perdu la généricité : $b_7 \leftarrow$ (16 - 2) ($[2\times NG]-2$). Ils testeront de nouveau avec plusieurs valeurs de la mesure (8, 9, 5, 8) comme s'ils s'attendaient — sait-on jamais ! — à une adaptation autonome du script.

Tout comme le groupe 46e, ils vont alors exprimer « ce qui change » par un marque-place ◯, avant de simuler ce qui devrait se passer. Ils fixent $b_1 \leftarrow$ (◯ - 1), $b_2 \leftarrow$ (◯ - 2) et $b_7 \leftarrow$ (◯ - 2). Ici, les ◯ ne sont pas des signes et ne sont pas destinés à être remplacés par une même valeur, puisque suivant les boucles ils seront remplacés par 8 ou par 16. Puis ils remplacent les marque-place par les valeurs attendues (respectivement 8, 8 et 16), et font de nouveau de multiples tests, avec 8 ou 9 en valeur entrée pour la mesure. Cette nouvelle tension fixe-variable, associée à la présence de ces marque-place est possiblement ce qui les amènera à substituer les ◯ par des (mesure), aboutissant à (mesure - 1), (mesure - 2) et (mesure - 2). Le tracé erroné les amènera alors à chercher une nouvelle relation pour b_7, ce qu'ils n'auront pas le temps trouver pour cause de fin de séance.

Conclusion

Une des difficultés rencontrées par les élèves lors de leur entrée dans l'algèbre élémentaire peut être la construction du concept de variable dans un rôle de paramètre. La généralisation peut-être vue comme étant une des caractéristiques essentielle de l'algèbre comme de l'algorithmique, or le paramètre est essentiel à la généricité. Si l'on considère l'épistémologie de cet objet d'un point de vue à la fois phénoménologique et sémiotique, il se distingue notamment de l'inconnue par la tension unique-multiple ou fixe-variable qui la caractérise : si l'inconnue est *un* objet mathématique associé à *un* symbole désignant *une* valeur déjà présente et qu'il s'agit de découvrir, le paramètre est lui aussi une association objet-représentation unique, mais désignant *toute valeur possible* parmi un ensemble. La généralisation informatisée de motif crée la nécessité du paramètre et de sa représentation, et on peut voir dans les premiers résultats de l'expérimentation menée à quel point cette tension fixe-variable est présente et délicate à dépasser. Dans une généralisation informatisée, cette nécessité est présente mais aussi alliée à la possibilité d'explorer librement le langage de la théorie choisi — ce qui n'est pas le cas pour une communication, différée ou non, avec un humain. C'est selon nous une piste possible pour permettre aux élèves de dépasser cet obstacle épistémologique : cela amène à faire vivre nécessairement cette tension fixe-variable, ou plutôt ces tensions. Soulignons pour terminer une question possible et une ouverture. D'une part, l'exploration du registre sémiotique dans un EPGB comme *Snap!* est certes possible, mais elle est limitée et peut-être problématique. Il est impossible pour les élèves de tester « leurs » propres mots : le typage des blocs empêche une écriture telle que 

ou 

: est-ce une contrainte facilitant et encadrant l'exploration, ou cela peut-il limiter la construction du concept à la seule découverte d'un « bloc qui existe » ? De plus, dans un EPGB, contrairement à ce qu'il se passe dans la plupart des langages informatiques, les variables sont définies dans l'environnement et non dans le programme : la déclaration de la variable est un acte indépendant du script (ou de l'ensemble des scripts), cela pourrait rendre difficile une analyse logique des langages algébrique et informatique[19].

D'autre part, une spécificité de la variable informatique n'a pas été exploitée : comme représenté dans la figure 3, la variable informatique dans un rôle de paramètre est un objet faisant référence à un emplacement mémoire, cet emplacement pouvant contenir la représentation d'une des valeurs possibles : cette indirection vers un objet « tangible » peut-elle faciliter la construction du concept de paramètre ?

Références bibliographiques

Bronner, A. & Squalli, H. (2021). La généralisation dans la pensée algébrique. *Revue québécoise de didactique des mathématiques, 3*(0). https://rqdm.recherche.usherbrooke.ca/ojs/ojs-3.1.1-4/index.php/rqdm/article/view/32

Balacheff, N. (1987). Processus de preuve et situations de validation. *Educational Studies in Mathematics, 18*(2), 147-176. https://doi.org/10.1007/BF00314724

[19] Analyse suggérée par F. Arzarello et S. Bagossi, commentaires à une communication, colloque ETM7, Strasbourg, 2022.

Bosc-Miné, C. (2014). Caractéristiques et fonctions des feed-back dans les apprentissages. *L'année psychologique, 114*(2), 315-353.

Bronner, A. (2015). Développement de la pensée algébrique avant la lettre : Apport des problèmes de généralisation et d'une analyse praxéologique. Dans L. Theis (éds.), *Actes du colloque EMF2015-GT3* (pp. 247-264).

Chabert, J.-L. (éds.) (2010). *Histoire d'algorithmes : Du caillou à la puce*. Belin.

Chevallard, Y. (1989). Le passage de l'arithmétique à l'algébrique dans l'enseignement des mathématiques au collège. Deuxième partie : Perspectives curriculaires : La notion de modélisation. *Petit x, 19*, 45-75.

Christianidis, J. (2007). The way of Diophantus : Some clarifications on Diophantus' method of solution. *Historia Mathematica, 34*(3), 289-305. https://doi.org/10.1016/j.hm.2006.10.003

Coppé, S., & Grugeon, B. (2009). *Le calcul littéral au collège. Quelle articulation entre sens et technique ?* Colloque de la CORFEM. https://halshs.archives-ouvertes.fr/halshs-00959612

Descartes, R. (1596-1650) année du texte. (1897). *Œuvres de Descartes. Tome 6 / publiées par Charles Adam et Paul Tannery*. https://gallica.bnf.fr/ark:/12148/bpt6k3411414j

Dowek, G. (2011). *Les quatre concepts de l'informatique*. https://edutice.archives-ouvertes.fr/edutice-00676169

Duval, R. (1993). Registres de représentation sémiotique et fonctionnement cognitif de la pensée. *Annales de Didactique et de Sciences Cognitives, 5*, 37-65.

Ely, R., & Adams, A. (2012). Unknown, placeholder, or variable : What is *x*? *Mathematics Education Research Journal, 24*. https://doi.org/10.1007/s13394-011-0029-9

Fabre, M., & Orange, C. (1997). Construction des problèmes et franchissements d'obstacles. *Aster, 24*(1), article 1. https://doi.org/10.4267/2042/8668

Farès, N. (2015). Al-Khwārizmī et le fondement axiomatique de l'algèbre. *Lebanese Science Journal, 16*, 107-126.

Farès, N. (2017). Diophante et l'algèbre. Dans *Naissance et développement de l'algèbre dans la tradition mathématique arabe, Dār al-Fārābī*. Beyrouth, 2017. https://hal.archives-ouvertes.fr/hal-01741649

Furinghetti, F. & Paola, D. (1994). Parameters, unknowns and variables: a little difference? Dans *Proceedings of the 18th International Conference for the Psychology of Mathematics Education, vol. 2*. (p. 375).

Grugeon-Allys, B., Pilet, J., Chenevotot-Quentin, F. & Delozanne, É. (2012). Diagnostic et

parcours différenciés d'enseignement en algèbre élémentaire. *Recherches en Didactique des Mathématiques, hors-série algèbre*, 26.

Hersant, M. & Orange Ravachol, D. (2015). Démarche d'investigation et problématisation en mathématiques et en SVT : Des problèmes de démarcation aux raisons d'une union. Inquiry based learning in mathematics and earth science education: from issues dividing to reasons for union. *Recherches en éducation*.
https://doi.org/10.4000/ree.7533

Komis, V., Touloupaki, S. & Baron, G.-L. (2017). Analyse cognitive et didactique du langage de programmation *ScratchJr*. Dans J. Henry, A. Nguyen & E. Vandeput (éds.), *L'informatique et le numérique dans la classe : Qui, quoi, comment ?* Presses Universitaires de Namur (pp. 109-121).

Legrand, J.-M. (2019). Captation Automatisée et Visualisation de Traces de Programmation dans un Environnement de Programmation Graphique par Blocs. *RJC EIAH 2019*. https://hal.archives-ouvertes.fr/hal-02615243

Legrand, J.-M. (2022, à paraître). Algorithmique et entrée dans l'Algèbre élémentaire : la (difficile) construction du concept de « paramètre ». *Actes du VII^e Symposium international d'étude sur le Travail Mathématique (ETM)*. 27 juin - 2 juillet 2022, Strasbourg Université.

Choquet, C., Grau, S., Hersant, M., Legrand, J.-M. & Zebiche, N. (2022, à paraître). Le travail mathématique à l'aune du cadre de l'apprentissage par problématisation : travail mathématique et processus de problématisation chez les élèves. *Actes du VII^e Symposium international d'étude sur le Travail Mathématique (ETM)*. 27 juin - 2 juillet 2022, Strasbourg Université.

Macbeth, D., Department, G. F. P., & for Social Research, N. S. (2004). Viète, Descartes, and the Emergence of Modern Mathematics. *Graduate Faculty Philosophy Journal, 25*(2), 87-117.

Modeste, S. (2012). *Enseigner l'algorithme pour quoi ? Quelles nouvelles questions pour les mathématiques ? Quels apports pour l'apprentissage de la preuve ?* [Thèse de doctorat, Université de Grenoble].
https://tel.archives-ouvertes.fr/tel-00783294/document

Pilet, J. & Grugeon-Allys, B. (2021). L'activité numérico-algébrique à la transition entre l'arithmétique et l'algèbre. *Éducation et didactique, 15*(15-2), 9-26.
https://doi.org/10.4000/educationdidactique.8580

Radford, L. (1991). Diophante et l'algèbre pré-symbolique. *Bulletin AMQ*.

Radford, L. (2003). Gestures, Speech, and the Sprouting of Signs: A Semiotic-Cultural Approach to Students' Types of Generalization. *Mathematical Thinking and Learning, 5*, 37-70. https://doi.org/10.1207/S15327833MTL0501_02

Radford, L. (2006a). *Algebraic thinking and the generalization of patterns : A semiotic perspective, 1*.

Radford, L. (2006b). The Cultural-Epistomological Conditions of the Emergence of Algebraic

Symbolism. Dans F. Furringhetti, S. Kaijser & C. Tzanakis. *Proceedings of the 2004 History and Pedagogy of Mathematics Conference & ESU4*. Uppsala, Sweden (pp. 509-524) (Plenary Lecture).
https://www.academia.edu/34611951/The_Cultural_Epistomological_Conditions_of_the_Emergence_of_Algebraic_Symbolism

Radford, L. (2014). The Progressive Development of Early Embodied Algebraic Thinking. *Mathematics Education Research Journal, 26*(2), 257-277.
https://doi.org/10.1007/s13394-013-0087-2

Radford, L. (2018). The Emergence of Symbolic Algebraic Thinking in Primary School. Dans C. Kieran (éds.), *Teaching and Learning Algebraic Thinking with 5- to 12-Year-Olds*. Springer International Publishing (pp. 3-25).

https://doi.org/10.1007/978-3-319-68351-5_1

Rashed, R. (2007). *Le commencement de l'algèbre*. Blanchard.

Sajaniemi, J. (2005). Roles of Variables and Learning to Program. *Proceedings of the 3rd Panhellenic Conference « Didactics of Informatics »*.

Samurçay, R. & Rouchier, A. (1985). De «faire» à «faire faire» : Planification d'actions dans la situation de programmation. *Enfance, 38*(2), 241-254.
https://doi.org/10.3406/enfan.1985.2883

Serfati, M. (1997). *La constitution de l'écriture symbolique mathématique*. [Thèse de doctorat, Université Paris I].
https://tel.archives-ouvertes.fr/tel-01252590

Serfati, M. (2010). Symbolic revolution, scientific revolution: Mathematical and philosophical aspects. Dans M. Van Dyck & A. Heeffer (éds.), *Philosophical Aspects of Symbolic Reasoning in Early Modern Mathematics*. College Publications (p. 103).

Tchounikine, P. (2017). *Initier les élèves à la pensée informatique et à la programmation avec Scratch*.
https://lig-membres.imag.fr/tchounikine/PenseeInformatiqueEcole.html

Wing, J. (2011). Research Notebook: Computational Thinking - What and Why? *The Link, 3*, 20-23.