CARACTÉRISATION DE LA PENSÉE ALGORITHMIQUE EN MATHÉMATIQUES POUR LA RECHERCHE EN DIDACTIQUE DES MATHÉMATIQUES

Marie-Frédérick ST-CYR[1]
Université de Sherbrooke

Hassane SQUALLI[2]
Université de Sherbrooke

Fatima BOUSADRA[3]
Université de Sherbrooke

Résumé. La pensée algorithmique est au cœur de l'activité mathématique. Elle est aussi présente dans l'activité informatique. Dans ce texte, nous l'abordons en mobilisant la théorie de l'objectivation de Radford (2011). Dans cette perspective, la pensée algorithmique en mathématiques renvoie à une manière d'agir et de réfléchir dans des activités mathématiques qui font intervenir au moins un algorithme pouvant être exécuté par un agent du traitement de l'information. Sur la base d'une analyse des écrits scientifiques et professionnels, nous avons élaboré un cadre de référence visant à caractériser la pensée algorithmique en mathématiques pour un usage en didactique des mathématiques. Ce cadre est structuré autour de trois classes d'activités caractéristiques de la pensée algorithmique en mathématiques qui ont été identifiées.

Mots-clés. Pensée mathématique, pensée algorithmique, pensée informatique, théorie de l'objectivation.

Abstract. Algorithmic thinking is at the heart of mathematical activity. It is also present in computing activity. In this text, we approach it by mobilizing Radford's theory of objectification (2011). From this perspective, algorithmic thinking in mathematics refers to a way of acting and thinking in mathematical activities that involve at least one algorithm that can be executed by an information-processing agent. These activities include the automation of algorithms. On the basis of an analysis of scientific and professional literature, we have developed a framework to characterize algorithmic thinking in mathematics for use in mathematics didactics. This framework is structured around three classes of activities that have been identified as characteristic of algorithmic thinking in mathematics.
Keywords. Mathematical thinking, algorithmic thinking, computational thinking, theory of objectification.

Resumen. El pensamiento algorítmico está en el corazón de la actividad matemática. También está presente en la actividad informática. En este texto, lo abordamos basándonos en la teoría de la objetivación de Radford (2011). Desde esta perspectiva, el pensamiento algorítmico en matemáticas se refiere a una forma de actuar y pensar en actividades matemáticas que implican al menos un algoritmo que puede ser ejecutado por un agente de procesamiento de la información. Estas actividades incluyen la automatización de algoritmos. Sobre la base de un análisis de la literatura científica y profesional, hemos desarrollado un marco para analizar y caracterizar el pensamiento algorítmico en matemáticas para su uso en la educación matemática. Este marco se estructura en torno a tres clases de actividades que se han identificado como características del pensamiento algorítmico en matemáticas.

Palabras clave. Pensamiento matemático, pensamiento algorítmico, pensamiento computacional, teoría de la objetivación.

[1] marie-frederick.st-cyr@usherbrooke.ca

[2] hassane.squalli@usherbrooke.ca

[3] fatima.bousadra@USherbrooke.ca

Introduction

L'enseignement-apprentissage des mathématiques évolue pour répondre aux besoins d'une société de plus en plus axée sur les technologies numériques (par exemple l'intelligence artificielle, les réseaux sociaux, la science des données). Au cœur de ces technologies numériques se trouvent des algorithmes et des personnes qui effectuent une activité algorithmique dans laquelle elles imaginent, conçoivent et étudient ces algorithmes. L'activité algorithmique[1] est de ce fait inévitable dans le milieu scolaire et partout dans le monde (Kadijevich et al., 2023). Considérant que le concept d'algorithme est enraciné intrinsèquement dans la culture mathématique depuis des centaines d'années (Modeste, 2012), on peut penser que l'enseignement-apprentissage des mathématiques au primaire et au secondaire doit jouer un rôle important dans l'étude et la conceptualisation de ce type d'activité. Notre travail s'inscrit dans cette vision. Nous inscrivons la pensée algorithmique en mathématiques (PAM) dans un contexte d'enseignement-apprentissage au primaire et au secondaire. Nous définissons cette forme de pensée comme une manière d'agir et de réfléchir dans des activités mathématiques faisant intervenir au moins un algorithme pouvant être exécuté par un agent du traitement de l'information (St-Cyr, 2022). L'agent de traitement de l'information peut être humain (p. ex., l'algorithme d'addition tel qu'enseigné au primaire est destiné à un autre humain) ou machine (p. ex., un algorithme informatique permettant d'analyser une base de données est destiné à un ordinateur).

Ainsi, nous supposons que sous l'angle de la didactique des mathématiques, la pensée algorithmique en mathématiques serait une forme singulière de la pensée mathématique qui se déploie dans des activités mathématiques faisant intervenir au moins un algorithme pouvant être exécuté par un agent du traitement de l'information. L'objectif de ce travail étant de caractériser la pensée algorithmique en mathématiques dans le contexte de l'enseignement-apprentissage des mathématiques au primaire et au secondaire, nous allons commencer par présenter les fondements théoriques qui orientent notre proposition. Outre la visée de problématisation de la pensée algorithmique en mathématiques, ces fondements permettront également de mettre en évidence l'originalité de notre travail d'un point de vue scientifique. Nous étayons ensuite l'approche méthodologique utilisée. Finalement, nous exposons un cadre de référence opérationnalisant une caractérisation de la pensée algorithmique en mathématiques en illustrant les différentes caractéristiques par des exemples[2].

1. Problématique

Depuis une dizaine d'années, l'activité algorithmique occupe une place de plus en plus importante dans plusieurs milieux scolaires à travers le monde, notamment au regard des liens qu'elle entretient avec l'activité informatique. Citons à titre d'exemples, l'Estonie, l'Angleterre, la France, la Finlande, les États-Unis, le Japon, la Corée, la Chine et l'Australie (Barma, 2018 ; Kotsopoulos et al., 2017 ; Wing, 2017). Bien qu'elle puisse apparaître de différentes manières dans les programmes de formation et dans les classes, elle est souvent présente dans le domaine des mathématiques. Toutefois, malgré sa présence dans plusieurs régions du monde (Kadijevich

[1] Nous utilisons le terme activité algorithmique qui est étroitement lié au terme pensée algorithmique. Il s'agit d'un choix fréquent en mathématiques, mais d'autres personnes autrices se réfèrent plutôt à la pensée computationnelle qui a une signification très proche selon plusieurs (Kadijevich et al., 2023). Nous utiliserons donc pensée algorithmique tout au long du texte, mais nous resterons fidèles aux termes utilisés par les personnes autrices citées.

[2] Le présent article est fondé sur le mémoire de maîtrise dans lequel nous avons élaboré une caractérisation opératoire de la pensée algorithmique en mathématiques (St-Cyr, 2022).

et al., 2023), il ne semble pas y avoir un accord sur ce que recouvre la pensée algorithmique spécifiquement en mathématiques. Pourtant, les travaux de recherche sur cette forme de pensée ne cessent de se multiplier dans plusieurs domaines montrant la nécessité de développer cette forme de pensée en mathématiques (van Borkulo et al., 2020). En ce sens, Stephens et Kadijevich (2020) affirment, dans l'encyclopédie de l'enseignement des mathématiques, qu'il n'y a pas de consensus sur ce que recouvre la pensée computationnelle. En effet, la définition de la pensée computationnelle semble varier d'un chercheur à l'autre (Cansu et Cansu, 2019). Cela renforce la confusion au regard de la pensée algorithmique en mathématiques qui est tantôt confondue avec la pensée computationnelle et tantôt avec la pensée informatique.

À ce flou terminologique et conceptuel entourant la pensée algorithmique en mathématiques, s'ajoute l'absence de recherche qui s'intéresse explicitement à la pensée algorithmique spécifiquement en mathématiques (Kadijevich et al., 2023). Ce manque est d'ailleurs mis de l'avant par des équipes de recherche qui travaillent sur la pensée computationnelle (Kalelioglu et al., 2016 ; Lyon et Magana, 2020 ; Shute et al., 2017). Or, une caractérisation permettant de mieux définir les attributs de la pensée algorithmique en mathématiques est essentielle autant pour le milieu de la pratique que pour la recherche en didactique des mathématiques puisqu'elle permettrait de circonscrire cet objet de recherche sur les plans théorique et méthodologique. Cette caractérisation pourrait également baliser l'enseignement-apprentissage de cette forme de la pensée en mathématiques.

2. Objectif de la recherche

L'objectif de cet article est de proposer une caractérisation de la pensée algorithmique en mathématiques qui pourrait être développée chez des élèves du primaire ou du secondaire. L'originalité de notre proposition vient entre autres que nous considérons la pensée algorithmique en mathématiques comme une unité indivisible qui se déploie dans des activités mathématiques faisant intervenir le concept d'algorithme (dites activités algorithmiques en mathématiques) pouvant être exécuté par un agent du traitement de l'information. Nous reconnaissons qu'elle existe dans d'autres sphères d'activités (notamment en sciences informatiques), mais la pensée algorithmique en informatique sort du cadre de ce travail ; nous nous intéressons spécifiquement à l'activité algorithmique en mathématiques. Notre travail ne vise donc pas à mettre en rapport la pensée algorithmique en mathématiques avec la pensée informatique comme l'ont fait Knuth (1985), Modeste (2012) et Shute et al. (2017), mais plutôt à décrire, de manière forcément non exhaustive, cette forme de la pensée mathématique qui se déploie dans des activités mathématiques où figurent un ou plusieurs algorithmes. Cette posture nous distingue également de recherches qui visent à caractériser la pensée informatique ou computationnelle, pour ensuite identifier les caractéristiques qui sont aussi présentes en mathématiques (Kallia et al., 2021 ; Weintrop et al., 2016).

3. Fondements théoriques et conceptuels : qu'est-ce que la pensée pour nous ?

En nous intéressant à la pensée algorithmique en mathématiques, nous nous inscrivons dans les recherches qui s'intéressent aux différentes formes de la pensée mathématique (par exemple la pensée algébrique, statistique, fonctionnelle, etc.). Pour approcher la notion de pensée mathématique, nous nous appuyons sur la théorie de l'objectivation (Radford, 2021) une théorie socioculturelle d'inspiration vygotskienne. Dans cette théorie, Radford (2015) nous invite à distinguer la pensée dans son sens anthropologique et la pensée dans son sens subjectif. Dans son sens anthropologique, la pensée relève d'une synthèse codifiée des pratiques sociales, culturelles et historiques (Radford, 2015). Elle s'est construite au fil du temps au travers de « l'agir des

individus et [des] pratiques sociales à l'intérieur desquelles ces individus agissent » (Radford, 2015, p. 336). Cette synthèse résulte de la pratique sociale, les singularités des actions se voient reflétées dans ce qui devient reconnu comme une même manière d'agir et de réfléchir, se constituant ainsi en un prototype d'actions et de réflexions (Radford, 2015). La pensée est dès lors une capacité toujours latente d'agir ou de réfléchir d'une certaine manière. Dans son sens subjectif, la pensée se réfère à la pensée de l'individu (Radford, 2015). Cette forme de pensée est donc colorée par le bagage de l'individu dans toute sa singularité. Bien que cette forme de pensée soit individuelle, elle est intimement liée à la pensée anthropologique. En effet, c'est par la pensée subjective qu'une possibilité de penser culturelle peut se réaliser dans une activité.

Pour l'instant, **nous considérons uniquement la pensée dans son sens anthropologique** afin de décrire, le mieux possible, la pensée algorithmique en mathématiques qui s'est construite dans les activités mathématiques au fil du temps et au travers des cultures et des sociétés. En ce sens, en inscrivant notre travail dans la théorie de l'objectivation, nous soutenons que la pensée algorithmique en mathématiques est une manière d'agir et de réfléchir dans des activités algorithmiques en mathématiques. C'est donc dans les activités algorithmiques en mathématiques que la pensée algorithmique en mathématiques prend forme. Ces activités algorithmiques en mathématiques ont deux caractéristiques : (1) ce sont des activités mathématiques et (2) elles font intervenir au moins un algorithme. En prenant en compte certaines idées de Modeste (2012), de Wing (2017) et de Venant (2018), et en cohérence avec notre position épistémologique, nous définissons la pensée algorithmique en mathématiques comme une manière d'agir et de réfléchir dans des activités mathématiques faisant intervenir au moins un algorithme pouvant être exécuté par un agent de traitement de l'information (un humain ou une machine). En reprenant des idées de Denning (2017), de Modeste (2012) et de Thomas (2020), nous considérons l'algorithme comme une suite d'étapes logiques, non arbitraires et ne nécessitant pas le jugement humain, placées dans un ordre permettant de résoudre un problème d'un certain type en un nombre fini d'étapes.

4. Approche méthodologique

Pour atteindre ces objectifs, nous avons suivi une approche méthodologique qui se divise en deux grandes étapes. Nous avons élaboré une caractérisation de la pensée algorithmique en mathématiques et nous avons confronté celle-ci en analysant le programme de mathématiques de l'Ontario (province du Canada qui a intégré en 2020 le codage dans son programme de mathématiques).

4.1. Élaboration de la caractérisation

Dans un premier temps, nous avons élaboré une caractérisation de la pensée algorithmique en mathématiques en effectuant une recherche documentaire pour identifier des classes d'activités algorithmiques en mathématiques ainsi que les principales caractéristiques de ces activités (concepts, raisonnements et manières de représenter).

Nous avons sélectionné des articles publiés dans des revues de référence, en français et en anglais, publiés depuis 2006 (année correspondant au point de vue proposé par Wing, point tournant pour la recherche sur la pensée computationnelle et la pensée informatique). Nous avons exploré les bases de données ERIC, APA PsychInfo et Érudit. Nous avons utilisé des mots clés ainsi que leur traduction anglophone (et leur ramification pour optimiser la recherche) liés à :

- la pensée algorithmique : pensée computationnelle, pensée informatique, codage, programmation

- aux mathématiques
- à l'éducation : enseignement, école, primaire, secondaire, élèves, apprentissage, éducation.

Parmi les articles identifiés dans les bases de données, nous avons conservé ceux qui respectaient les critères d'inclusion : inclure les mathématiques, inclure la notion d'algorithmique (pensée algorithmique, pensée informatique, pensée computationnelle, programmation, codage, etc.), inclure l'enseignement-apprentissage au primaire ou au secondaire. Puis, au fil des lectures, les articles cités qui semblaient pertinents ont été ajoutés aux articles lus. Au final, les articles proposant des caractéristiques de la pensée algorithmique en mathématiques que nous avons étudiés plus en profondeur sont au nombre de dix-sept.

Toutefois, dans ces articles peu de tâches algorithmiques en mathématiques ont été repérées. Nous avons alors exploré des sites où des personnes enseignantes et conseillères pédagogiques proposent des tâches ayant le potentiel de faire émerger une activité algorithmique en mathématiques. Nous avons analysé un total de 29 tâches en faisant une analyse phénoménologique ; c'est-à-dire en analyse notre propre pratique. Nous avons fait le choix d'arrêter les analyses lorsque nous avons atteint la saturation théorique en ce qui a trait aux types d'activités que nous effectuons en réalisant les tâches et en ce qui a trait aux caractéristiques algorithmiques (concepts, raisonnements et représentation). Les tâches que nous avons sélectionnées et analysées sont des tâches prototypiques. Finalement, nous avons vérifié cette caractérisation avec notre équipe de direction et deux personnes externes, expertes de la programmation informatique dans les écoles, afin d'avoir un consensus sur les caractéristiques ciblées ainsi que sur les classes d'activités essentielles identifiées.

4.2. Confrontation de la caractérisation

Dans un deuxième temps, une confrontation de notre caractérisation avec les propositions théoriques principales ainsi qu'une validation empirique par l'analyse du programme ontarien ont été effectuées. D'abord, notre recherche est bâtie sur des travaux et des recherches précédents qui s'intéressent à la pensée algorithmique en mathématiques. Bien que nous n'ayons trouvé aucun travail qui ait construit une caractérisation de la pensée algorithmique en mathématiques directement, plusieurs travaux sont intimement liés et doivent être inclus, du moins en partie, dans notre caractérisation. Parmi ces travaux, certains proposent des caractérisations en incluant les mathématiques (Kallia et al., 2021 ; Shute et al., 2017) ou des aspects de la pensée algorithmique propre à l'automatisation (Brennan et Resnick, 2012 ; Weintrop et al., 2016). Nous avons vérifié que ces caractérisations sont incluses dans la nôtre. Si ce n'est pas entièrement le cas, nous avons pris deux voies : (1) intégrer les éléments manquants ou (2) justifier pourquoi ces éléments ne sont pas inclus dans notre propre caractérisation.

Ensuite, pour vérifier le caractère opératoire de notre proposition, nous avons effectué une analyse critique du programme de l'Ontario. Ce programme est choisi puisqu'il a récemment été conçu volontairement pour développer la pensée informatique chez les élèves en intégrant plusieurs éléments propres à la pensée algorithmique en mathématiques. Nous avons mobilisé la caractérisation élaborée pour nous assurer qu'elle soit réellement utilisable.

Dans le cadre de cet article, nous présentons uniquement la caractérisation qui a émergé de ce travail. Nous ne présentons pas les confrontations avec les autres travaux ni l'analyse du programme de l'Ontario.

5. Caractérisation de la pensée algorithmique en mathématiques

Nous présentons notre caractérisation de la pensée algorithmique en mathématiques. La caractérisation relève d'une réflexion théorique ainsi que d'une analyse phénoménologique à partir de la résolution de problèmes algorithmiques prototypiques c'est-à-dire qui appartiennent à des classes de problèmes. Nous présentons des élèves fictifs pour illustrer des activités algorithmiques possibles en classe de mathématiques. Comme mentionné, selon nous la pensée algorithmique en mathématiques se déploie dans des activités algorithmiques en mathématiques. Par conséquent, nous avons d'abord identifié des activités algorithmiques en mathématiques que nous avons classifiées selon trois classes que nous jugeons essentielles au développement de la pensée algorithmique en mathématiques :

- Les activités de mobilisation d'un algorithme existant : Il s'agit d'activités mathématiques où une personne fait appel à un algorithme qui existe déjà pour réaliser une tâche ou résoudre un problème mathématique. Elle doit mobiliser l'algorithme de manière intentionnelle (donc, ne pas se limiter à l'appliquer sans réfléchir). Un exemple d'une telle activité est la transposition (au sens de Balacheff, 1994) d'un algorithme mathématique dans le langage de programmation.
- Les activités de conception d'algorithmes : Il s'agit d'activités mathématiques dans lesquelles une personne construit un algorithme en s'appuyant, ou non, sur au moins un algorithme existant. Un exemple d'une activité de conception est lorsque les élèves conçoivent des algorithmes personnels pour effectuer des additions.
- Les activités d'étude d'algorithmes : Il s'agit d'activités mathématiques dans lesquelles une personne analyse (ou compare) la structure d'un algorithme en s'appuyant sur ses caractéristiques (efficacité, complexité, parallélisme, optimisation, se termine en un nombre fini d'étapes, etc.). Un exemple d'une telle activité pourrait être l'analyse et la comparaison de différents algorithmes d'additions pour d'une part mieux les comprendre et distinguer leurs points communs et leurs différences.

En nous appuyant sur ces trois classes d'activités, nous faisons ressortir des composantes essentielles de la pensée algorithmique en mathématiques. Bien que nous considérons la pensée comme une totalité dynamique, nous avons, en nous appuyant sur le travail de Squalli (2015) pour la pensée algébrique et de Robert (2018) pour la pensée fonctionnelle, mis en lumières les caractéristiques essentielles de la pensée algorithmique en mathématiques selon trois composantes interreliées dans ce que nous appelons une caractérisation de la pensée algorithmique en mathématiques :

- Les registres de représentation utilisés et les manières d'opérer : le langage mathématique, le langage informatique, le pseudocode et le passage d'un registre à l'autre
- Les raisonnements utilisés : le raisonnement séquentiel, le raisonnement itératif, le raisonnement récursif, le raisonnement conditionnel et le raisonnement par analyse de l'erreur
- Les concepts en jeu et leur signification : opérateur (mathématique, chaîne de caractères ou autres structures), l'égalité comme une relation équivalence ou un opérateur, variable (mathématique et informatique) ainsi que fonction et procédure

5.1. Les registres de représentation utilisés et les manières d'opérer

Un algorithme peut être représenté et communiqué de différentes manières. En nous inspirant de travaux précédents (Bråting et Kilhamn, 2021 ; Briant et Bronner, 2015 ; Modeste, 2012), nous avons identifié trois manières principales de représenter les algorithmes en mathématiques : le registre mathématique, le registre informatique et le registre du pseudocode. Nous présentons tour à tour ces trois manières de représenter des algorithmes et nous terminons par discuter des manières de passer d'un registre de représentation à un autre.

Le registre mathématique

Le registre mathématique comprend des expressions mathématiques formelles contenant des symboles alphanumériques et des expressions provenant du langage naturel. Les expressions mathématiques formelles ont l'avantage d'être synthétiques et univoques, alors que les expressions dans le langage naturel permettent de soutenir, de justifier et d'expliquer les expressions mathématiques formelles.

Le registre informatique

Le registre informatique vise à communiquer un algorithme à une machine ; donc à faire exécuter celui-ci par la machine. La machine ne parle pas le même langage que les humains et fait exactement ce qu'on lui demande ; ni plus ni moins. L'algorithme doit donc absolument être décomposé en étapes élémentaires. De plus, bien que la machine apporte de nouvelles perspectives, elle apporte de nouvelles contraintes dont des règles syntaxiques différentes de celles présentes dans le pseudocode et dans le langage mathématique (Bråting et Kilhamn, 2021 ; Modeste, 2012). Également, certains concepts, dont le concept de variable, n'ont pas la même signification dans le registre informatique que dans le registre mathématique. Ceci peut amener des défis supplémentaires aux élèves qui seront à considérer ultérieurement.

Le registre du pseudocode

Le pseudocode est une représentation intermédiaire entre le registre mathématique et le registre informatique (Modeste, 2012). Il utilise souvent des codes (SI, RÉPÉTER, etc.) rappelant la structure d'un algorithme informatique, mais en retirant les contraintes liées au registre informatique (Briant et Bronner, 2015 ; Modeste, 2012). Ces codes peuvent ressembler davantage au langage naturel chez un novice et davantage à un langage de programmation ou à un langage mathématique chez un expert. Ils peuvent aussi prendre des formes plus schématiques comme un diagramme de flux. Également, ces codes, tout comme la structure de l'algorithme, peuvent varier selon les contraintes du langage informatique qui sera utilisé par la suite.

Le pseudocode peut permettre de passer du registre mathématique au registre informatique (Briant et Bronner, 2015), mais il peut également permettre de passer du registre informatique au registre mathématique ou de réfléchir à l'algorithme informatique sans revenir à l'algorithme mathématique. Le pseudocode est un lieu de réflexion pour construire l'algorithme en réduisant les règles de syntaxe du langage informatique. Il permet aussi l'étude d'un algorithme existant et peut faciliter la mobilisation d'un algorithme existant en le transposant dans un registre intermédiaire.

Le pseudocode diffère de l'algorithme mathématique en se rapprochant plus du langage informatique. Le pseudocode utilise un langage semblable au langage informatique (si, sinon, répéter, etc.) et ne vise pas à démontrer l'algorithme. Il permet de faciliter l'écriture du programme par la suite. Nous illustrons ces faits à l'aide d'un exemple, mais il est à noter que le pseudocode peut toujours être utilisé pour faciliter le passage du registre mathématique au registre informatique et vice-versa.

Le passage d'une représentation à l'autre

Pour passer d'un registre à un autre, l'algorithme doit être transformé. Briant et Bronner (2015) parlent d'une double transposition ; c'est-à-dire d'une transposition commençant par le registre mathématique et allant vers le pseudocode, puis une deuxième allant du pseudocode vers le registre informatique. Pour eux, la transposition va uniquement dans cette direction. Selon nous, il est autant possible de transformer le registre informatique en pseudocode que l'inverse. En effet, un élève pourrait être plus familier avec le langage de la machine, ou les contraintes de ce langage pourraient faciliter sa conception. Puis, pour expliquer son algorithme à un collègue, il pourrait souhaiter transformer son algorithme en pseudocode ou en langage mathématique. Pour cette raison, nous préférons parler d'un va-et-vient entre chacun des registres plutôt qu'une transposition unidirectionnelle commençant par le registre mathématique et se terminant dans le registre informatique comme Briant et Bronner (2015) le proposent.

5.2. Les raisonnements utilisés

Raisonnement séquentiel

Avoir un raisonnement séquentiel renvoie à la capacité à : (1) décomposer le problème ou la solution en sous-étapes et en étapes élémentaires (Selby, 2014 ; Shute et al., 2017), (2) déterminer l'ordre des étapes (Selby, 2014) et (3) exécuter les étapes dans le bon ordre.

(1) Décomposer en sous-étapes et en étapes élémentaires, signifie que lorsqu'une personne fait face à un problème, elle cherche à identifier les grandes étapes du problème, puis pour chacune de ces grandes étapes, elle identifie des sous-étapes jusqu'à identifier chacune des étapes élémentaires à effectuer. Pour des problèmes très complexes, il peut arriver qu'elle itère plusieurs fois avant d'arriver à l'identification des étapes élémentaires. Par ailleurs, dans une activité de mobilisation d'un algorithme ou d'étude d'un algorithme, une personne peut, à partir d'un algorithme, identifier les grandes étapes ainsi que les étapes élémentaires de l'algorithme.

(2) Déterminer l'ordre des étapes signifie qu'une fois les étapes identifiées, la personne met les étapes en ordre, et ce, que ce soit les étapes élémentaires ou les grandes étapes.

(3) Exécuter les étapes dans le bon ordre signifie qu'une fois l'ordre des étapes déterminé et/ou compris, la personne exécute les étapes dans cet ordre. Elle doit être en mesure de suivre la séquence ainsi que les instructions de contrôle. Elle pourrait aussi faire exécuter ces étapes par une autre personne ou par une machine.

Un exemple d'un raisonnement séquentiel lors d'une activité de conception d'un algorithme pourrait apparaître lors d'une activité pendant laquelle un élève trace un carré en marchant. Il pourrait d'abord observer certaines propriétés caractéristiques du carré, dont les quatre angles droits et les quatre côtés qui sont isométriques (à ce moment elle se retrouve dans le registre mathématique). Il se lève alors et marche cinq pas dans une direction et tourne de 90 degrés. Puis, il marche à nouveau cinq pas dans une direction et tourne de 90 degrés. Il a donc deux côtés de tracés. Il reprend alors deux autres fois la manœuvre pour compléter le carré. Par la suite, il écrit la séquence qu'il vient d'effectuer :

Avancer de cinq pas

Tourner de 90 degrés vers la droite

Avancer de cinq pas

Tourner de 90 degrés vers la droite

Avancer de cinq pas

Tourner de 90 degrés vers la droite

Avancer de cinq pas

Tourner de 90 degrés vers la droite

Cet algorithme est composé de deux étapes élémentaires successives qui se répètent : avancer de cinq pas et tourner de 90 degrés vers la droite. La dernière étape qui est de tourner de 90 degrés vers la droite n'est pas nécessaire puisqu'avant de faire cette étape, le carré est déjà dessiné. Toutefois, elle ne nuit pas non plus.

Ainsi, après avoir pressenti la séquence, l'élève a effectué la séquence avec son corps pour ensuite l'écrire. Il a donc décomposé le problème en étapes élémentaires, placé ces étapes dans l'ordre et exécuté l'algorithme. Cet élève était dans une activité de conception.

Raisonnement itératif

Avoir un raisonnement itératif renvoie à la capacité à : (1) déterminer la séquence d'instruction à répéter (reconnaître les régularités), (2) initialiser la (ou les) variable(s) (si nécessaire), (3) déterminer la condition d'arrêt ou le nombre de répétitions et (4) incrémenter.

(1) Déterminer la séquence d'instructions à répéter : pressentir puis reconnaître les régularités pour déterminer une séquence d'instructions qui se répète (Weintrop et al., 2016). Dans une activité de mobilisation ou d'étude, la personne peut identifier la structure itérative au sein de l'algorithme et déterminer la séquence d'instructions qui se répète.

(2) Initialiser : déterminer les valeurs que la (ou les) variable(s) doivent avoir au début de l'itération et fournir l'instruction à la personne ou au programme qui effectue la tâche. La personne identifie l'initialisation de la variable et l'exécute au bon moment. Elle pourrait aussi, après l'avoir identifiée, vérifier si ce choix est judicieux et s'il n'y a pas une autre manière d'initialiser qui serait plus efficace.

(3) Déterminer la condition d'arrêt : la séquence d'instructions doit se répéter un nombre fini de fois. Il s'agit de déterminer le nombre de fois que la séquence se répète ou la condition permettant à l'agent processeur de sortir de la boucle au bon moment. Ce nombre d'itérations, ou cette condition d'arrêt, doit être communiquée clairement. Pour parvenir à déterminer la condition d'arrêt, il faut anticiper et visualiser l'évolution des variables à l'intérieur de la boucle.

(4) Incrémenter : déterminer la valeur de la (ou des) variable(s) qui est (sont) modifiée(s) et qui permet(tent) de compter le nombre d'itérations ou d'atteindre la condition d'arrêt. Pour y parvenir, la modification de la variable (ou les) est (sont) anticipée(s) pour que l'agent processeur puisse sortir de la boucle exactement au bon moment.

En revenant à l'exemple du carré, il aurait été possible de proposer une solution qui inclut une itération. En effet, un élève pourrait remarquer que la séquence « avancer de cinq pas » et « tourner de 90 degrés vers la droite » se répète quatre fois. Par conséquent, il pourrait simplifier son algorithme en effectuant une itération. Dans ce cas, il n'y a pas de variable, donc il n'y a pas d'initialisation à faire. Toutefois, la condition d'arrêt consiste à arrêter après avoir effectué quatre fois ces deux étapes. Finalement, il n'y a pas d'incrémentation à prévoir non plus.

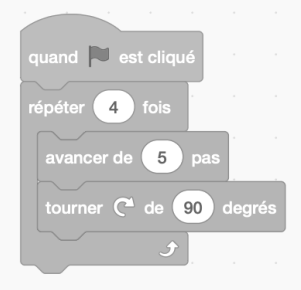

Figure 1. *Programme Scratch pour le dessin d'un carré (itératif).*

Un autre exemple d'un raisonnement itératif peut survenir dans une activité au travers de laquelle un élève conçoit un algorithme pour calculer le factoriel d'un certain nombre entier n. Il pourrait d'abord passer par le registre mathématique pour réfléchir au concept et se rappeler de la définition de la factorielle. Puis, il pourrait analyser un exemple du calcul de 4!. Il pourrait écrire : $1 \cdot 2 \cdot 3 \cdot 4 = 24$. Il remarque que le calcul de 4 ! consiste à multiplier systématiquement les nombres de 1 jusqu'à 4. En effectuant le calcul papier crayon, il écrit les étapes suivantes :

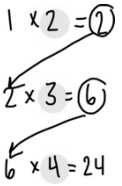

Figure 2. *Trace du calcul de 4 !*

Par les flèches effectuées et par les nombres ombragés, il pressent une régularité. Il choisit alors de tester avec un autre nombre pour vérifier cette régularité. Ensuite, il écrit l'algorithme en pseudocode :

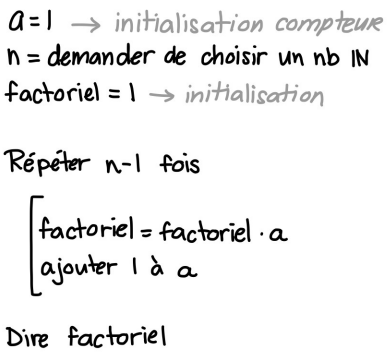

Figure 3. *Algorithme du calcul de n ! en pseudocode (itératif).*

On constate que l'élève a :

- Déterminé la séquence à répéter
- Initialisé (a=1 et factoriel =1)
- Déterminé que le programme doit itérer (n-1) fois
- Incrémenté (ajouter 1 à a)

L'algorithme aurait ensuite pu être transformé dans le registre informatique. La structure aurait aussi pu prendre une forme différente en étant récursive plutôt qu'itérative, comme nous allons le voir maintenant.

Raisonnement récursif

Le raisonnement récursif, qui est parfois confondu avec le raisonnement itératif, consiste à reconnaître qu'une séquence d'instructions (ou d'actions) qui se répète un certain nombre de fois, puis à créer une fonction (ou une procédure) avec cette séquence d'instructions à l'intérieur de laquelle la fonction (ou la procédure) s'appelle elle-même. Le raisonnement récursif diffère du raisonnement itératif de par la manière dont la répétition est faite. En effet, la récursivité demande d'appeler une fonction (ou une procédure) à l'intérieur d'elle-même ; il s'agit plutôt d'une composition de fonctions (ou de procédure).

Le raisonnement récursif renvoie à la capacité à : (1) déterminer une séquence d'instructions qui se répète, (2) déterminer la (ou les) condition(s) de terminaison qui est appelée(s) cas de base et (3) appeler la séquence elle-même.

(1) Déterminer une séquence d'instructions qui se répète : pressentir puis reconnaître les régularités pour déterminer une séquence d'instructions qui se répète et qui pourrait être décrite de manière autonome sous la forme d'une fonction ou d'une procédure informatique.

(2) Déterminer la (ou les) condition(s) de terminaison : déterminer une (ou plusieurs) condition(s) qui permettra(ont) de mettre fin à l'appel de la fonction elle-même. Pour y parvenir, le moment où la récursion doit arrêter est anticipé en suivant les étapes de la récursivité et en suivant l'évolution des variables.

(3) Appeler la séquence elle-même : déterminer la manière d'appeler la fonction elle-même pour qu'elle entre dans une récursivité.

Un exemple d'un raisonnement récursif apparaît dans une activité dans laquelle une personne choisit de mobiliser la définition du calcul du factoriel d'un nombre pour concevoir un algorithme qui effectue ce calcul de manière récursive. Par définition le factoriel d'un nombre entier n est : $n!=n\cdot(n-1)!$. Cette manière d'écrire est déjà récursive puisque pour calculer le factoriel de n, le factoriel de (n-1) est nécessaire, puis pour calculer le factoriel de (n-1), le factoriel de (n-2) est nécessaire et ainsi de suite jusqu'à ce que n soit égale à 1. Il est alors possible d'écrire un algorithme récursif en mobilisant la définition.

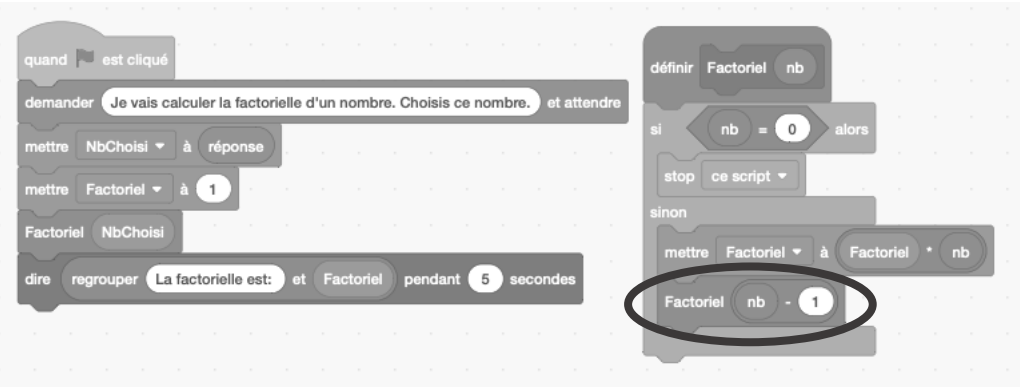

Figure 4. *Programme Scratch pour calculer n ! (récursif — la fonction se rappelle elle-même).*

Dans l'algorithme écrit dans le programme Scratch, une procédure (appelé bloc dans Scratch) a été créée. Cette procédure prend en entrée un nombre (nb) et ne renvoie rien en sortie (c'est la manière dont Scratch est conçu). Par conséquent, pour qu'une valeur puisse être fournie en sortie, une variable (factoriel) a été créée dans le programme principal afin d'enregistrer la valeur lorsque la procédure sera exécutée. La récursivité apparaît dans le bloc factoriel encerclé au

moment où il s'appelle lui-même. Dans ce cas, la personne qui conçoit le programme est dans une activité de mobilisation d'un algorithme, puisqu'elle connaît déjà les étapes à réaliser (par la définition du calcul de n !). Également, la condition de terminaison (n=1) est la même que lorsque la personne calcule manuellement la factorielle d'un nombre. Toutefois, elle doit ajuster cette condition pour qu'elle soit comprise par la machine et doit déterminer la manière dont la fonction se rappellera elle-même. De plus, la personne est en mesure d'anticiper que le nombre (nb) qui est pris en entrée dans le bloc factoriel encerclé diminue de 1 à chaque fois que la fonction est appelée puisque la fonction appelle (nb -1). Ces capacités à anticiper et à suivre les étapes d'un algorithme récursif sont fondamentales au raisonnement récursif.

Raisonnement conditionnel

Ce type de raisonnement se réfère à la capacité de prendre une décision basée sur certaines conditions (Brennan et Resnick, 2012) dans le but de choisir une voie plutôt qu'une autre selon ces conditions. Il s'agit de déterminer les options et les conditions permettant d'accéder à ces options. En informatique l'expression utilisée est souvent le « si alors, sinon ».

Le raisonnement conditionnel renvoie à la capacité à : (1) déterminer les options, (2) déterminer les critères (les conditions) menant à ce choix et (3) déterminer l'ordre dans lequel présenter ces choix.

(1) Déterminer les options : pressentir qu'un choix est à faire en fonction de certaines contraintes puis déterminer les chemins possibles à emprunter.

(2) Déterminer les critères (les conditions) menant à ce choix : pour chacun des chemins possibles à emprunter, déterminer les conditions nécessaires pour que chacun soit emprunté.

(3) Déterminer l'ordre dans lequel présenter ces choix : déterminer l'ordre dans lequel les choix seront présentés en étant conscient de l'impact de cet ordre sur leur réalisation et sur l'efficacité de l'algorithme éventuellement.

L'exemple du calcul du factoriel d'un nombre n de manière récursive présenté précédemment illustre un raisonnement conditionnel. En effet, dans le bloc factoriel (encerclé dans la figure 4), la personne a d'abord déterminé les options : (1) le nombre est égal à zéro, alors l'algorithme s'arrête ou (2) le nombre est différent de zéro, alors la machine effectue le produit de factoriel avec ce nombre (nb) et incrémente la valeur de ce nombre (nb) en lui soustrayant 1. Cette personne a identifié les contraintes au même moment que les options. Effectivement, elle a déterminé la condition que le nombre soit égal à zéro (nb=0) pour que le premier chemin soit pris (et mette ainsi fin à la récursivité) et sinon, c'est-à-dire pour toutes les autres valeurs de n, le second chemin sera emprunté.

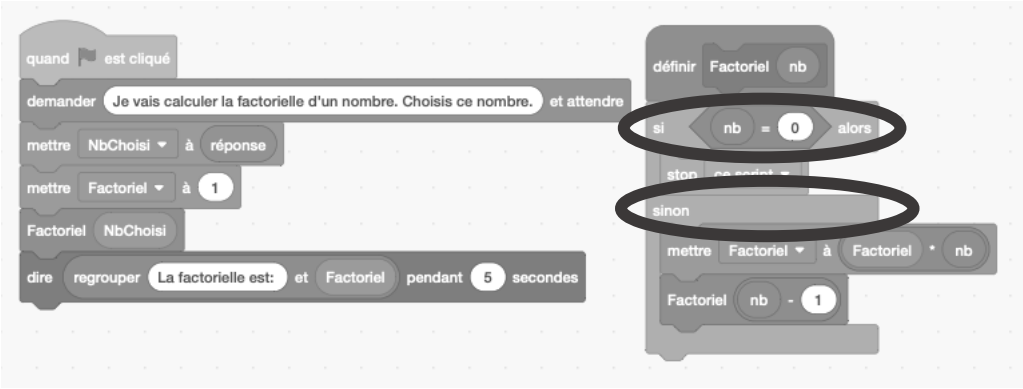

Figure 5. *Programme Scratch pour calculer n ! (raisonnement conditionnel).*

Raisonnement par analyse de l'erreur

Le cinquième raisonnement est le raisonnement par analyse de l'erreur (souvent lié au débogage ou la correction de l'algorithme) (Brennan et Resnick, 2012 ; Shute, 2017 ; Weintrop et al., 2012). Il s'agit d'un raisonnement au cœur de l'activité informatique, mais également important en algorithmique dans la mesure où nous incluons l'automatisation à notre définition de l'activité algorithmique.

Avoir un raisonnement par analyse de l'erreur renvoie à la capacité à : (1) déterminer si le programme fait effectivement la tâche qu'il doit faire en fournissant une réponse exacte, (2) déterminer la source de l'erreur et (3) corriger l'erreur.

(1) Déterminer si le programme fait effectivement la tâche qu'il doit faire en fournissant une réponse exacte : pour y parvenir, il est possible de tester l'algorithme en utilisant certaines valeurs judicieusement choisies. Les tests peuvent se faire sur l'algorithme entier ou sur une partie de l'algorithme. Certains langages informatiques possèdent des outils de débogage pour aider le programmeur à identifier son erreur.

(2) Déterminer la source de l'erreur : deux types d'erreurs peuvent survenir, c'est-à-dire une erreur au niveau de l'algorithme lui-même (de sa structure) et une erreur de langage (surtout dans le cas où l'algorithme est informatisé). Lorsque l'erreur est au niveau de la structure, l'algorithme peut fournir une réponse sans soulever d'erreur. La tâche revient donc au programmeur, qui anticipe la solution attendue pour une certaine entrée, de constater l'écart entre la réponse attendue et la réponse fournie. Il cherche ensuite l'endroit où l'erreur est située en utilisant différentes stratégies (tester l'algorithme entier ou des sous-algorithmes ; analyser la structure dans le langage de la machine ou en analysant le pseudocode ou la réflexion mathématique sous-jacente) pour déterminer l'endroit où se produit l'erreur. Par conséquent, lorsque l'erreur est structurale, elle peut relever de l'algorithme, mais aussi de la réflexion mathématique. En revanche, lorsque l'erreur provient du langage, l'algorithme ne fournira généralement pas de solution et un message d'erreur s'affichera en fournissant des indices sur la source de l'erreur.

(3) Corriger l'erreur : si l'erreur est d'ordre structural, un questionnement sur les étapes à suivre et leur impact sur la solution est nécessaire. Il peut parfois arriver qu'il soit nécessaire de revisiter une partie (plus ou moins grande) de l'algorithme. Si l'erreur est d'ordre du langage, il s'agit de chercher la solution à notre problème (dans le guide ou dans une communauté) ; généralement une personne a déjà rencontré et documenté le problème.

Il est à noter que plusieurs stratégies existent pour prévenir et corriger plus facilement les erreurs comme concevoir une étape à la fois et tester au fur et à mesure de la conception.

Un exemple d'un tel raisonnement pourrait survenir dans le calcul, de manière récursive, du factoriel d'un nombre. Il aurait été possible d'initialiser la variable Factoriel à la valeur du nombre choisi plutôt qu'à 1. Ce choix aurait pu s'expliquer par le fait que le calcul du Factoriel est n(n-1)(n-2) ; c'est-à-dire qu'il commence par n… Le premier facteur, nombre initial, est effectivement le nombre choisi (n). Ceci aurait amené à toujours obtenir la factorielle du nombre en question multiplié par ce nombre. Des tests auraient pu être effectués pour prendre conscience de l'erreur et identifier sa source. Différentes manières auraient pu être choisies pour résoudre le problème, dont initialiser la variable à la valeur 1.

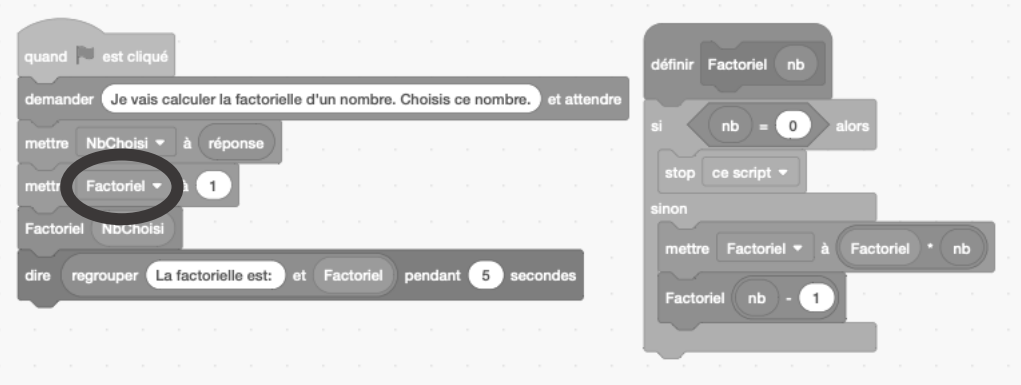

Figure 6. *Programme Scratch pour calculer n ! (raisonnement par analyse de l'erreur).*

5.3. Les concepts en jeu et leur signification

Nous allons maintenant présenter la composante des concepts en jeu et leur signification de notre caractérisation de la pensée algorithmique en mathématiques. Nous avons identifié quatre concepts essentiels pour décrire la pensée algorithmique en mathématiques : les opérateurs, la variable, le symbole d'égalité comme une relation d'équivalence ou comme un opérateur et la fonction. Nous présentons chacun de ces concepts en mettant en lumière leur signification qui varie d'un registre à l'autre et en illustrant par des exemples.

Les opérateurs

Les opérateurs permettent d'effectuer des manipulations numériques sur les chaînes de caractères et sur toutes autres structures de données (Brennan et Resnick, 2012). Ils permettent « la transformation des problèmes ; ces opérateurs sont attestés par des productions et des comportements » (Modeste, 2012, p. 56). L'addition, la soustraction, la multiplication et la division sont des opérateurs arithmétiques de base en mathématiques. Toutefois, il existe plusieurs autres opérateurs mathématiques tels que le sinus, la puissance et la partie entière. Des opérateurs proviennent également de l'algèbre booléenne comme le ET, le OU et le NON ainsi que toutes leurs compositions. La concaténation permettant de joindre deux chaînes de caractères est un exemple d'opérateur pouvant s'appliquer aux chaînes de caractères. Dans tous les cas, ces opérateurs peuvent déjà exister dans le langage utilisé ou ils peuvent être définis par le concepteur.

Un exemple d'opérateur dans le calcul du PGCD est présent pour mentionner que les deux conditions (nombre 1 modulo compteur est égal à zéro et nombre 2 modulo compteur est égal à zéro) sont respectées. Si seulement l'une des deux conditions est respectée, ce n'est pas suffisant. Également, le symbole plus grand que (>) est un opérateur mathématique permettant de vérifier une condition. Finalement, il y a plusieurs opérations d'affectations (mettre nombre 1 à…) qui seront discutées plus amplement dans la prochaine section.

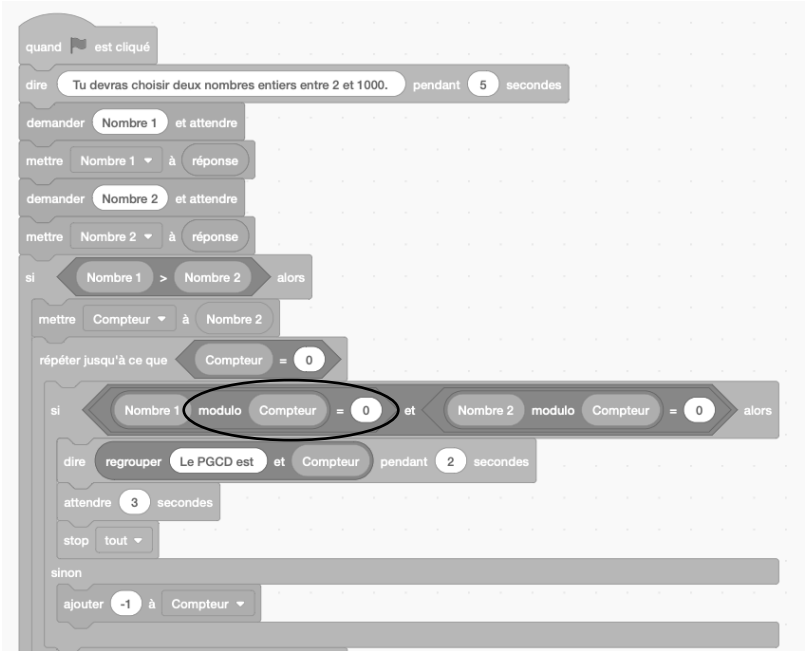

Figure 7. *Programme Scratch pour le calcul du PGCD d'un nombre (opérateur).*

La relation d'équivalence et l'opération d'affectation

Le symbole d'égalité se prête à différentes significations selon le contexte. En mathématiques, ce symbole représente généralement une relation d'équivalence (Bråting et Kilhamn, 2021). Cette idée d'équivalence est fondamentale notamment en algèbre et représente une relation symétrique (Briant, 2013 ; Modeste, 2012). Par ailleurs, en arithmétique, le symbole d'égalité peut jouer le rôle d'opérateur en symbolisant « donner la réponse ». En informatique, le symbole d'égalité[1], lorsqu'il vise à vérifier si une contrainte est vraie ou fausse, permet de poser une question plutôt qu'à représenter une relation d'équivalence. Il joue alors le rôle d'un opérateur comme le ET et le OU tel qu'illustré dans l'exemple suivant :

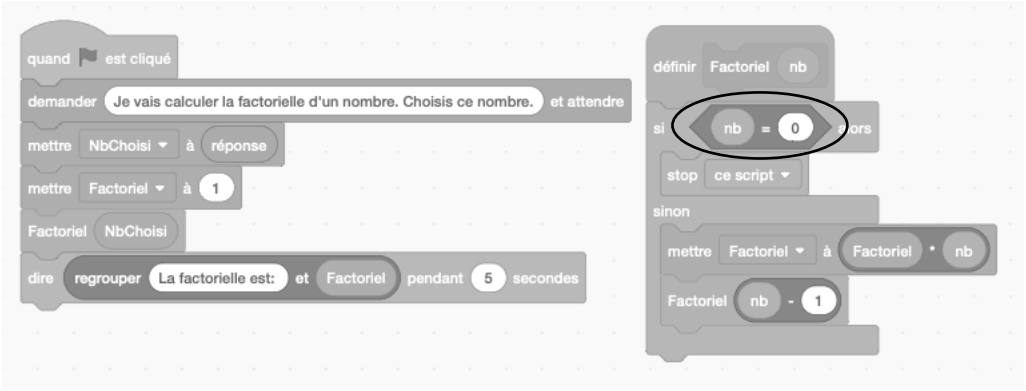

Figure 8. *Programme Scratch pour calculer n ! (symbole d'égalité comme opérateur).*

Cette distinction n'est pas la seule. En informatique, le symbole d'égalité, « mettre variable à… » ou « ← » (ou autre dans d'autres langages) peuvent représenter une opération d'affectation. Comme la variable représente un espace mémoire (Haspekian et Nijimbéré, 2016), l'opération d'affectation permet d'enregistrer une valeur (numérique ou non) à cet endroit. Par conséquent, nous plaçons une valeur dans cet espace mémoire. Une particularité est que cette valeur peut changer dans un même problème pour éviter d'enregistrer de l'information inutilement et ainsi

[1] Selon le langage, ça pourrait varier et notamment être « == ».

de diminuer l'efficacité d'un programme (Modeste, 2012). Par exemple, pour ajouter 4 à la valeur emmagasinée dans la variable x, nous pourrions écrire : x ← x+4). Si la valeur de x était 5, alors elle deviendrait égale à 9. Cette manipulation n'est pas valide en mathématiques au sens d'une relation d'équivalence, puisqu'elle n'est pas symétrique (Briant, 2013 ; Modeste, 2012), mais elle l'est en informatique puisque la variable, x dans ce cas, n'a pas la même signification tout comme l'opération d'affection et la relation d'équivalence.

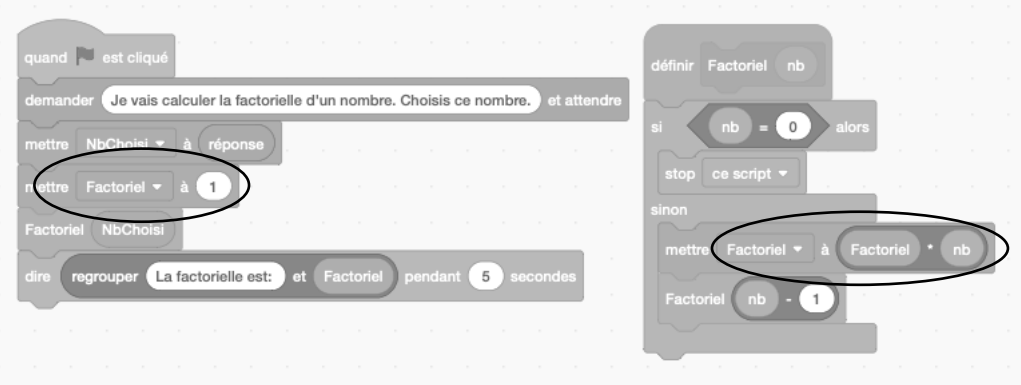

Figure 9. *Programme Scratch pour calculer n ! (opération d'affectation).*

La variable

La variable est un concept central autant en informatique qu'en mathématiques. Or, elle n'a pas exactement la même signification dans les deux registres (Bråting et Kilhamn, 2021 ; Briant, 2013 ; Modeste, 2012). En mathématiques, la variable est un symbole utilisé pour « représenter un élément non spécifié ou inconnu d'un ensemble et marquer sa place dans une expression mathématique » (Modeste, 2012, p. 249). De plus, la variable mathématique permet de représenter des relations (Bråting et Kilhamn, 2021) en pouvant être remplacée par une infinité de valeurs tant qu'elle reste la même du début à la fin du problème (Briant, 2013).

En informatique, la variable représente un espace mémoire auquel il est possible d'affecter une valeur. Or, cette valeur peut varier au sein d'un même problème (Modeste, 2012). Par exemple, pour ajouter 4 à la valeur emmagasinée dans la variable x, nous pourrions écrire : x ← x+4 (x=x+4 ou mettre x à x+4 selon le langage). Si la valeur de x était 5, alors elle deviendrait égale à 9. En changeant de valeur au cours de l'exécution du programme, les variables permettent de mettre en lumière le processus (Bråting et Kilhamn, 2021). En informatique, lorsque l'algorithme est conçu, la variable représente une certaine valeur dans un certain domaine, qui sera ensuite remplacé par une (ou plusieurs) valeur(s) au cours de l'exécution.

De plus, il existe des variables locales et des variables globales en informatique. Dans l'exemple du calcul de la factorielle, la variable nombre (nb), définie dans le bloc Factoriel (à droite dans la figure 10), existe uniquement dans ce processus et nulle part ailleurs. Par conséquent, même si cette variable était rappelée dans le programme principal (programme de gauche), la variable ne serait pas définie. C'est une variable locale. En revanche, les variables définies dans le programme principal (NbChoisi et Factoriel) sont des variables globales qui peuvent être utilisées partout, dont dans le processus (le bloc rose Factoriel), et qui peuvent alors y être modifiées.

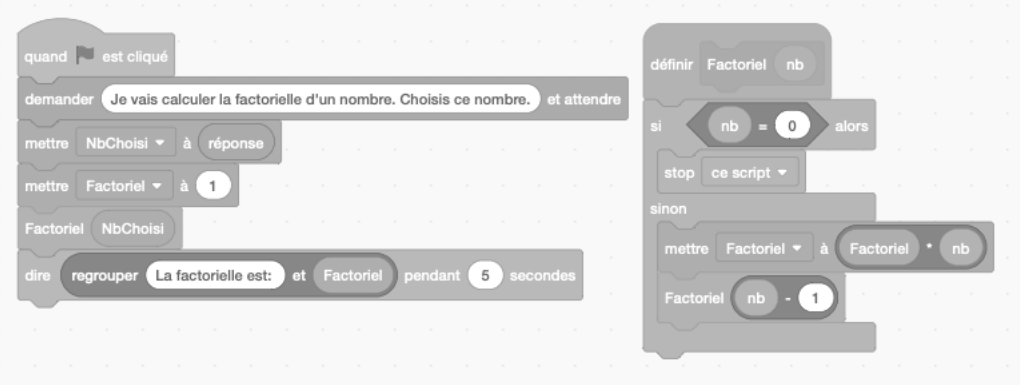

Figure 10. *Programme Scratch pour calculer n ! (variable).*

La fonction et la procédure

Tout comme le concept de variable, le concept de fonction n'a pas la même signification en mathématiques et en informatique. En mathématiques, la fonction est une relation entre une (ou plusieurs) variable(s) indépendante(s) et une variable dépendante. En informatique, le concept de fonction se réfère à un sous-algorithme visant à exécuter une tâche précise ; lequel pouvant être « appelé », une ou plusieurs fois, dans l'algorithme principal. La fonction a l'avantage de faciliter la lecture de l'algorithme principal et d'aider à le structurer.

Toutefois, le concept de fonction en informatique est lié de près au concept de procédure. La procédure, comme la fonction, est un sous-algorithme qui effectue une tâche précise. Contrairement à la fonction, la procédure pourrait ne renvoyer aucune valeur en sortie. Par conséquent, une variable globale doit être définie préalablement (si nous souhaitons une transformation), puis transformée dans la procédure pour que cette transformation soit visible. Au contraire, une fonction pourrait uniquement avoir des variables locales puisqu'elle fournit une valeur en sortie après avoir exécuté le sous-algorithme. Dans Scratch, nous pouvons uniquement définir des procédures. Celles-ci sont appelées des blocs et peuvent prendre plusieurs valeurs en entrée, mais ne renvoient aucune valeur en sortie. Par exemple, dans le calcul de la factorielle d'un nombre, la procédure est représentée par le bloc rose nommé *CalculFactoriel*. Elle prend en entrée un nombre et elle transforme la variable Factoriel. La variable transformée (Factoriel) est définie de manière globale, donc les modifications effectuées dans le processus sont enregistrées dans cet espace mémoire.

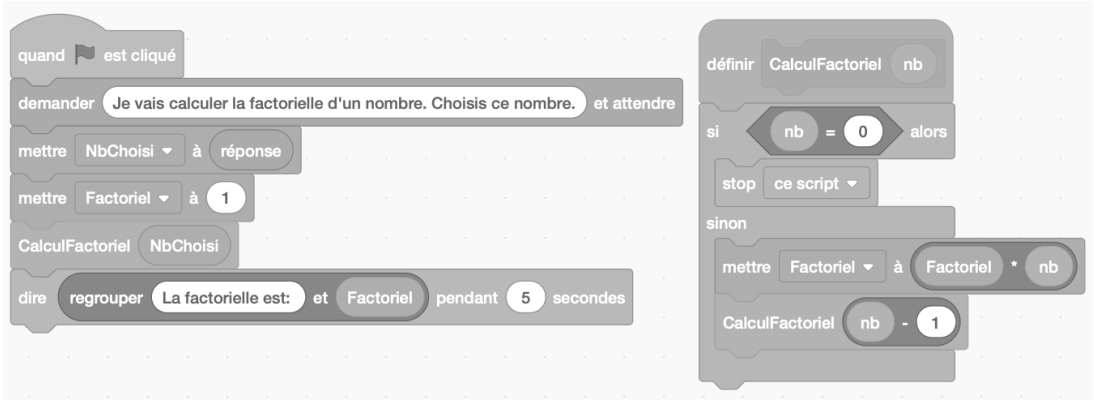

Figure 11. *Programme Scratch pour calculer n ! (procédure).*

Conclusion

Notre objectif était de contribuer aux travaux de recherche en didactique des mathématiques

portant sur la pensée algorithmique en proposant une caractérisation de la pensée algorithmique en mathématiques qui pourrait être développée chez des élèves du primaire ou du secondaire. Cette caractérisation permet de mieux cerner cette forme de pensée d'un point de vue théorique en identifiant les concepts en jeu et leur signification, les raisonnements utilisés et les manières de représenter et d'opérer, puis éventuellement d'un point de vue pratique. Par conséquent, en nous appuyant sur la théorie de l'objectivation, nous proposons une caractérisation de la pensée algorithmique en mathématiques pour la recherche en didactique des mathématiques qui prend forme au travers de trois classes d'activités algorithmiques en mathématiques essentielles : la mobilisation d'un algorithme, la conception d'un algorithme et l'étude d'un algorithme. Rappelons qu'en analysant ces activités, nous avons identifié les représentations et les manières d'opérer sur celles-ci, les raisonnements en jeu ainsi que les concepts et leurs significations. Nous avons toutefois, dans le cadre de ce texte, détaillé uniquement les raisonnements (séquentiel, itératif, récursif, conditionnel et par correction de l'erreur).

Notre travail se distingue d'autres travaux pour trois raisons principales : (1) nous considérons la pensée algorithmique comme une forme de pensée mathématique, (2) nous nous inscrivons dans une approche socioculturelle selon laquelle nous approchons la pensée par l'activité, autrement dit la pensée algorithmique en mathématiques est une forme de la pensée mathématique et (3) nous considérons trois composantes pour décrire de manière la pensée algorithmique dans son sens anthropologique.

Par ailleurs, comme dans la majorité des travaux de recherche, notre travail comporte certaines limites. D'abord, sur le plan méthodologique, nous avons effectué un travail théorique qui a été testé dans l'analyse d'un seul programme de formation, celui de mathématiques de l'Ontario de la maternelle à la 8^{e} année. Bien que ça permette de vérifier la validité de notre proposition et que la caractérisation contienne les propositions principales sur le sujet, il serait pertinent d'élargir les sources de vérification. Également, dans la construction de notre caractérisation nous nous sommes intéressés au curriculum de base dans sa globalité sans distinguer les niveaux scolaires. Notre travail pourrait également servir de base pour proposer une trajectoire d'apprentissage commençant au début du primaire et allant jusqu'à la fin du secondaire. Ce travail reste encore à faire, notamment en considérant le niveau de sophistication de la pensée algorithmique en mathématiques attendu au fil du développement de la pensée mathématique des élèves.

Il serait aussi pertinent de nous intéresser à la formation initiale et continue des personnes enseignantes. Une étude faite dans un contexte québécois recommande notamment de « prévoir du temps de formation et d'expérimentation pour les enseignants dans l'année précédant l'arrivée d'un programme » (Barma, 2021, p. 16). La formation en lien avec l'enseignement de la pensée algorithmique en mathématiques est peu connue, alors tout le travail reste à faire, mais nous pourrons nous appuyer, entre autres, sur notre caractérisation pour baliser les réflexions.

De plus, la caractérisation proposée considère la pensée dans son sens anthropologique, mais pas dans son sens subjectif. Ce choix se justifie par notre volonté de comprendre la pensée algorithmique en mathématiques comme une entité historique, culturelle et sociale qui existe depuis des centaines d'années. Or, en insérant nos travaux dans la théorie de l'objectivation, il serait pertinent d'aller étudier les manifestations de la pensée algorithmique en mathématiques chez les élèves et les personnes enseignantes en étudiant les dimensions empiriques et subjectives de la pensée algorithmique en mathématiques. Ces travaux pourraient éventuellement inclure une dimension affective. Dans un même ordre d'idée, des activités d'enseignement-apprentissage visant à développer la pensée algorithmique en mathématiques pourraient être conçues et analysées à la lumière de notre caractérisation.

Finalement, les recherches s'intéressant spécifiquement à la pensée algorithmique en mathématiques sont peu nombreuses et beaucoup de travail reste encore à faire (Kadijevich et al., 2023). Toutefois, les orientations gouvernementales québécoises laissent croire que cette forme de pensée pourrait être au programme prochainement. Qu'elle apparaisse comme compétence générale ou spécifiquement en mathématiques, la pensée algorithmique, qui est aussi présente en informatique, occupe une place importante en mathématiques en offrant un regard particulier sur cette forme de pensée. Nous sommes conscients qu'inclure la pensée algorithmique en mathématiques dans le programme de formation de l'école québécoise est ambitieux, mais nous croyons essentiel de considérer ce choix, surtout dans la société numérique dans laquelle nous vivons actuellement.

Références bibliographiques

Balacheff N. (1994) Didactique et intelligence artificielle. *Recherches en didactique des mathématiques, 14*(1), 9-42.

Barma, S. (2018). *Rapport final : réaliser une étude de cas multiple qui vise à affiner les connaissances sur l'usage pédagogique ou didactique de la programmation dans les écoles du Québec* (publication nº 118982). Université Laval. https://lel.crires.ulaval.ca/oeuvre/rapport-final-realiser-une-etude-de-cas-multiple-qui-vise-affiner-les-connaissances-sur

Barma, S. (2021). *Rapport final : Colloque sur l'usage pédagogique de la programmation informatique.* Ministère de l'éducation et Université Laval. http://www.education.gouv.qc.ca/fileadmin/site_web/documents/education/PAN_Rapport_Journees-reflexion_-usage-pedagogique-programmation-informatique.pdf

Bråting, K., et Kilhamn, C. (2021). Exploring the intersection of algebraic and computational thinking. *Mathematical Thinking and Learning, 23*(2), 170-185. https://doi.org/10.1080/10986065.2020.1779012

Brennan, K. et Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceeding of the Annual american educational research association meeting 1*, 85-90.

Briant, N. (2013). *Étude didactique de la reprise de l'algèbre par l'introduction de l'algorithmique au niveau de la classe de seconde du lycée français* [thèse de doctorat, Université Montpellier II]. https://tel.archives-ouvertes.fr/tel-01002513/

Briant, N. et Bronner, A. (2015). Étude d'une transposition didactique de l'algorithmique au lycée : une pensée algorithmique comme un versant de la pensée mathématique. Dans L. Theis (dir.), *Pluralités culturelles et universalité des mathématiques : enjeux et perspectives pour leur enseignement et leur apprentissage* (p. 231-246). Université des Sciences et de la Technologie Houari Boumediene.

Cansu, S. K. A. et Cansu, F. K. R. A. (2019). An overview of computational thinking. *International Journal of Computer Science Education in Schools*, *3*(1), 1-11. http://doi.10.21585/ijcses.v3i1.53

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33-39. https://doi.org/10.1145/2998438

Haspekian, M., et Nijimbéré, C. (2016). Favoriser l'enseignement de l'algorithmique en mathématiques : une question de distance aux mathématiques ? *Éducation et didactique*,

10(3), 121-135. https://doi.org/10.4000/educationdidactique.2609

Kadijevich, D. M., Stephens, M. et Rafiepour, A. (2023). Emergence of computational/algorithmic thinking and its impact on the mathematics curriculum. Dans Y. Shimizu et R. Vithal (dir.), *Mathematics Curriculum Reforms Around the World: The 24th ICMI Study* (p. 375-388). Springer.

Kalelioglu, F., Gulbahar, Y. et Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing*, *4*(3), 583-596.

Kallia, M., van Borkulo, S. P., Drijvers, P., Barendsen, E. et Tolboom, J. (2021). Characterising computational thinking in mathematics education: a literature-informed Delphi study. *Research in Mathematics Education, Informa UK Limited, 1-29.* http://doi.org/10.1080/14794802.2020.1852104

Knuth, D.E. (1985). Algorithmic thinking and mathematical thinking. The American Mathematical Monthly, 92(3), 170-181. https://doi.org/10.2307/2322871

Kotsopoulos, D., Floyd, L., Khan, S., Namukasa, I., Somanath, S., Weber, J. et Yiu, C. (2017). A pedagogical framework for computational thinking. *Digital Experiences in Mathematics Education, 3(2),* 154-71. http://doi.org/10.1007/s40751-017-0031-2.

Lyon, J. A. et Magana, A. J. (2020). Computational thinking in higher education: A review of the literature. *Computer Applications in Engineering Education*, *28*(5), 1174-1189. https://doi.org/10.1002/cae.22295

Modeste, S. (2012). *Enseigner l'algorithmique, pourquoi ? Quels apports pour l'apprentissage de la preuve ? Quelles nouvelles questions pour les mathématiques ?* [thèse de doctorat, Université de Grenoble]

Radford, L. (2011). Vers une théorie socioculturelle de l'enseignement-apprentissage : la théorie de l'objectivation. *Éléments, (1),* 1-27.

Radford, L. (2015). Pensée mathématique du point de vue de la théorie de l'objectivation. Dans L. Theis (dir.), *Pluralités culturelles et universalité des mathématiques : enjeux et perspectives pour leur enseignement et leur apprentissage. Acte du colloque EMF2015* (p. 334-335). Université des Sciences et de la Technologie Houari Boumediene.

Radford, L. (2021). *The theory of objectification: A Vygotskian perspective on knowing and becoming in mathematics teaching and learning* (Vol. 4). Brill.

Robert, V. (2018). *Le développement de la pensée fonctionnelle dans les manuels scolaires du 3e cycle du primaire québécois : une analyse praxéologique* [thèse de doctorat, Université de Sherbrooke]. Savoirs UdeS. https://savoirs.usherbrooke.ca/handle/11143/12608

Selby, C.C. (2014). *How can the teaching of programming be used to enhance computational thinking skills?* [thèse de doctorat, University of Southampton]

Shute, V. J., Sun, C. et Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142-158. http://doi.10.1016/j.edurev.2017.09.003

Squalli, H. (2015). La généralisation algébrique comme abstraction d'invariants essentiels. Dans L. Theis (dir.), *Pluralités culturelles et universalité des mathématiques : enjeux et perspectives pour leur enseignement et leur apprentissage. Acte du colloque EMF2015* (p. 334-335). Université des Sciences et de la Technologie Houari Boumediene.

St-Cyr, M-F. (2022). *Élaboration d'une caractérisation opératoire de la pensée mathématique*

algorithmique pour la recherche en didactique des mathématiques [mémoire de maitrise, Université de Sherbrooke]. Savoirs UdeS.

Stephens, M. et Kadijevich, D. M. (2020). Computational/Algorithmic thinking. Dans S. Lerman (Éd.), *Encyclopedia of Mathematics Education* (p. 117-123). Springer International Publishing. https://doi.org/10.1007/978-3-030-15789-0_100044

Thomas, (2020). Algorithms. Dans S. Lerman (dir.), *Encyclopedi*a of mathematics education (p. 44-47). Springer. https://doi.org/10.1007/978-3-030-15789-0

van Borkulo, S., Kallia, M., Drijvers, P., Barendsen, E. et Tolboom, J. (2020). Computational thinking and mathematical thinking: Digital literacy in mathematics curricula. Dans B. Barzel, R. Bebernik, L. Göbel, M, Pohl, H. Ruchniewicz, F., Schacht, D. et Thurm (dir.), *Proceedings of the 14th International Conference on Technology in Mathematics Teaching* (ICTMT 14) (p. 384-385). University of Duisburg-Essen. http://doi.10.17185/duepublico/70781

Venant, F. (2018). Programmer les mathématiques : la pensée informatique à l'école primaire. *Bulletin AMQ*, 58(3), 57-70.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L. et Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. Journal of Science Education and Technology, 25(1), 127-47. http://dx.doi.org/10.1007/s10956-015-9581-5

Wing, J. (2017). Computational thinking's influence on research and education for all. Italian Journal of Educational Technology, 1(1). https://doi.org/10.17471/2499-4324/9