REPERES - IREM. N° 119 - avril 2020

RECURRENCE ET RECURSIVITE A L'INTERFACE DES MATHEMATIQUES ET DE L'INFORMATIQUE (*)

Nicolás LEÓN, Simon MODESTE
IMAG, Université de Montpellier,
CNRS, Montpellier, France

Résumé : L'inclusion de contenus d'informatique au sein des programmes de mathématiques est une opportunité pour réfléchir à des questions émergeant des points de vue épistémologique et didactique sur les relations entre ces deux domaines d'activité scientifique. En particulier, les notions de récurrence et récursivité sont porteuses d'un intérêt majeur, par leur importance pour les deux disciplines, par les nombreuses difficultés que rencontrent les élèves et étudiants qui les apprennent, mais aussi par la dialectique qui s'établit entre elles, et qui reste jusqu'à présent relativement peu explorée de ce point de vue-là. Nous présentons le concept d'induction structurelle, qui permet d'apporter un éclairage sur le lien entre récurrence et récursivité, et montrons quelques exemples d'application en logique, informatique et mathématiques, susceptibles d'intéresser les professeurs chargés de l'enseignement du raisonnement par récurrence ou, dans le futur, de la récursivité.

Introduction

La récente introduction de contenus d'informatique (algorithmique et programmation) dans les programmes de mathématiques pour les collèges et les lycées français soulève plusieurs questions sur les moyens et les objectifs d'une telle modification curriculaire : quelle relation existe-t-il entre l'informatique et les mathématiques et quelles en sont les conséquences didactiques ? Quels concepts, méthodes et modes de pensée partagent ces deux disciplines ? Comment mieux promouvoir les synergies et éviter les (possibles) discordances ? Ces questions sont étudiées dans le cadre du projet de recherche ANR DEMaIn (Didactique et Épistémologie des interactions entre Mathématiques et Informatique).

Parmi les nombreux concepts qui se situent au carrefour des mathématiques et de l'informatique, nous nous intéressons particulièrement aux notions de récurrence et de récursi-

(*) Publication réalisée avec le soutien financier de l'ANR, projet DEMaIn <ANR-16-CE38-0006-01>

vité. Les procédures incluant des appels récursifs en programmation, les fonctions μ-récursives dans la théorie de la calculabilité, les arbres en théorie des graphes, les formules bien formées de la logique du premier ordre, les entiers naturels, ne constituent qu'une liste très modeste d'exemples d'objets que l'on étudie à l'aide de la récurrence et de la récursivité, ce qui montre à quel point l'usage de ces deux notions est répandu tant en mathématiques qu'en informatique. De plus, le fait qu'elles fassent partie de la plupart des curriculums de licence en mathématiques, informatique et ingénierie témoigne de leur importance dans chacune de ces disciplines.

Nous voudrions, à terme, proposer des situations didactiques adaptées à l'apprentissage de la récurrence et la récursivité. Mais quelle est la portée de ces notions ? Les entend-on de la même manière en mathématiques et en informatique ? Quel liens précis se tissent entre elles ? Des réponses diverses à ces questions donneront lieu à des approches d'enseignement différentes. Ainsi, pour parvenir à proposer des situations didactiques, il nous faut d'abord arriver à une compréhension approfondie des notions concernées, incluant les définitions qu'en donnent les ouvrages de référence, les problèmes qui les impliquent traditionnellement, tout comme les usages qu'en font les mathématiciens et les informaticiens. En résumé, il nous faut développer une étude épistémologique qui puisse étayer nos choix didactiques ultérieurs.

Du point de vue méthodologique, notre étude épistémologique inclut deux volets principaux : l'analyse d'ouvrages académiques et la réalisation d'entretiens auprès de chercheurs en mathématiques et en informatique.

Dans l'analyse d'ouvrages, nous nous intéressons aux définitions, usages et liens entre cinq termes clés pour notre recherche : récursivité (*recursiveness*), récurrence (*recurrence*), induction (*induction)*, itération (*iteration*) et boucle (*loop*). Nous essayons de trouver des éléments communs aux différentes approches, ainsi que des disparités qui pourraient indiquer des visions divergentes sur la signification des concepts.

D'autre part, les entretiens auprès de chercheurs ciblent les pratiques en lien avec la récurrence et la récursivité. Les chercheurs interrogés ont été choisis en essayant de couvrir un large spectre de domaines de recherche et le format de l'entretien est semi-dirigé, le but étant d'approfondir notre compréhension des aspects spécifiques de l'utilisation des concepts qui se manifesteraient de manière spontanée. Les réponses que les chercheurs ont données lors des entretiens seront utilisées dans cet article pour appuyer nos observations (nous citerons trois chercheurs, qui seront nommés « C1 » jusqu'à « C3 »).

En ce qui concerne le plan de cet article, nous commencerons par un aperçu sommaire de notre objet d'étude. Les définitions de la section 1 serviront de référence par la suite.

Dans la section 2 nous ferons le point sur l'état de l'art en didactique de la récurrence et de la récursivité. Nous constaterons que le lien entre les deux notions a été peu exploré du point de vue de leur enseignement.

Ensuite, dans la section 3 nous détaillerons les résultats de l'enquête, en nous centrant sur le concept d'induction structurelle, qui permet d'articuler récurrence et récursivité. Nous montrerons plusieurs exemples de cette articulation.

Finalement, nous présenterons les conclusions de cette étude préalable, ainsi que les perspectives de réflexion.

1. — Cadrage de l'objet d'étude

1.1 *Récurrence*

Nous appellerons « raisonnement par récurrence » un type de raisonnement que l'on applique pour prouver des propriétés des entiers naturels. L'ensemble d'entiers naturels, noté $\mathbf{N}$, est l'un des ensembles qui, muni d'une constante notée 0 et d'une fonction $\sigma : \mathbf{N} \longrightarrow \mathbf{N}$, satisfait les propriétés suivantes [1] :

1. L'image de σ ne contient pas 0.
2. La fonction σ est injective.
3. (Récurrence) Si S est un sous-ensemble de $\mathbf{N}$ tel que i) $0 \in S$, et ii) pour tout $n \in \mathbf{N}$ $n \in S$ implique $\sigma n \in S$, alors $S = \mathbf{N}$.

Le « schéma de récurrence classique » s'en déduit par application de la dernière propriété, en prenant l'ensemble $S = \{n \in \mathbf{N} \mid P(n)\}$, où P est un prédicat quelconque défini sur $\mathbf{N}$:

Théorème 1 (Schéma de récurrence classique). Soit $P(n)$ un prédicat qui dépend de $n \in \mathbf{N}$. Si
1. $P(0)$ et
2. pour tout $k \in \mathbf{N}$ $P(k) \Rightarrow P(\sigma k)$,
alors pour tout $n \in \mathbf{N}$ $P(n)$.

Nous appellerons « raisonnement par récurrence » toute application du schéma de récurrence classique. Nous allons montrer dans la section 3 qu'il existe des généralisations de ce type de raisonnement qui sont plus pertinentes pour travailler dans des ensembles différents de celui des entiers naturels. Nous emploierons d'autres dénominations pour ces généralisations, car il nous semble utile de bien distinguer ce cas particulier emblématique.

Voici un exemple d'application du raisonnement par récurrence.

Exemple 1. Démontrons par récurrence que, pour tout $n \in \mathbf{N}$, $4^n - 1$ est divisible par 3. Soit $P(n)$ le prédicat « $3 \mid 4^n - 1$ ».

1. $P(0)$, car $4^0 - 1 = 0$, et $3 \mid 0$.
2. Soit $k \in \mathbf{N}$ et supposons $3 \mid 4^n - 1$. On a

$$\begin{aligned} 4^{k+1} - 1 &= 4 \times 4^k - 1 \\ &= 4 \times (4^k - 1 + 1) - 1 \\ &= 4 \times (4^k - 1) + 4 - 1 \\ &= 4 \times (4^k - 1) + 3 \end{aligned}$$

Comme 3 divise $4^k - 1$ par hypothèse, $4 \times (4^k - 1) + 3$ est la somme de deux nombres divisibles par 3, et donc il est aussi divisible par 3. Ainsi, on a bien

$$\forall k \in \mathbf{N}\ P(k) \Rightarrow P(k + 1)$$

Nous concluons, par récurrence classique, que pour tout $n \in \mathbf{N}$ $P(n)$ [2] .

Pour une discussion plus approfondie sur le schéma de récurrence et sa relation avec les axiomes de Peano, voir CORI et LASCAR (2003b), POINCARÉ (2017) et LOMBARDI (2011) [3] .

1.2 *Récursivité*

Il est beaucoup plus difficile de donner une définition de la récursivité, même dans ce cadre préliminaire. Cette difficulté apparaît de manière assez flagrante dans les réponses des chercheurs en mathématiques et informatique que nous avons interviewés dans le cadre de notre étude (voir section 3). Nous allons d'abord

1. Il existe d'autres ensembles, non isomorphes à $\mathbf{N}$, satisfaisant ces propriétés. On les appelle ensembles d'entiers non standard.

2. Il aurait été également envisageable de prouver cette propriété en faisant appel aux congruences modulo 3, ou bien à l'identité $a^n - b^n = (a - b)\ \sum_{k=0}^{n-1} a^{n-1-k} b^k$.

3. Nous remercions l'un des relecteurs pour avoir attiré notre attention sur les deux dernières références.

donner quelques exemples de contextes où l'on retrouve la récursivité en mathématiques et en informatique :

— Une définition est dite *récursive* ou *impredicative* lorsque l'objet défini (*definiendum*) intervient dans le texte le définissant (*definiens*). Par exemple, dans la définition – très courante en informatique – « une liste est soit la liste vide, soit un couple composé d'un premier élément et d'une liste », le mot que l'on définit, « liste », apparaît dans sa propre définition. Il y a, bien sûr, des conditions à remplir pour qu'une définition récursive ne conduise pas à une régression à l'infini stérile : informellement, il faut que le *definiendum* apparaisse dans le *definiens* dans une version « plus simple » de lui-même (bien sûr, une bonne partie du travail consiste à préciser ce que l'on veut dire par « plus simple »).

— En théorie de la calculabilité, les fonctions primitives récursives sont définies à partir d'un ensemble de fonctions de base par l'application réitérée des opérations de composition et de récursion primitive [4]. Elles représentent une partie considérable des fonctions dites « effectivement calculables », celles dont les valeurs peuvent être obtenues par une procédure purement « mécanique » [5].

— Un algorithme ou un programme est dit récursif si dans son corps on fait appel à l'algorithme ou programme lui-même.

Par exemple, l'algorithme 1, qui calcule la factorielle d'un nombre entier positif n. Notons que l'appel à l'algorithme se fait sur une instance de son argument plus petite que l'instance de départ (on passe du calcul de `fact(n)` au calcul de `fact(n – 1)`). Observons également qu'il est nécessaire, pour pouvoir faire l'appel de l'algorithme, qu'on le définisse comme une fonction.

Algorithme 1 : Factorielle d'un entier naturel n

```
fact (n)
si n = 0 alors
|  retourner 1
sinon
|  retourner n*fact (n – 1)
fin
```

— Un autre cadre où l'on rencontre la récursivité est l'étude des suites. Une suite $(u_n)_{n \in \mathbf{N}}$ est dite récursive si sa ou ses premières valeurs sont données explicitement et les termes successifs sont calculés suivant un certain nombre de règles à partir des valeurs des termes précédents. Par exemple, la célèbre suite de Fibonacci, donnée par :

$$u_0 = 1 \qquad u_1 = 1$$
$$\forall n \geq 2 \quad u_n = u_{n-1} + u_{n-2}$$

4. Les fonctions de base sont :

— Pour $(m, c) \in \mathbf{N}^2$, la fonction *constante* c_m, donnée par $c_m(x_1, ..., x_m) = c$.

— Pour $m \in \mathbf{N}$ et $i \in \{1, ..., m\}$, la fonction de *projection* p_i^m, donnée par $p_i^m(x_1, ..., x_m) = x_i$.

— La fonction *successeur* σ, donnée par $\sigma(x) = x + 1$.

Étant donné la fonction $h : \mathbf{N}^k \to \mathbf{N}$, et les fonctions $g_1, ..., g_k$, telles que pour tout $i \in \{1, ..., k\}$ $g_i : \mathbf{N}^m \to \mathbf{N}$, la fonction $f : \mathbf{N}^m \to \mathbf{N}$ est la *composition* des fonctions précédentes si pour tous $x_1, ..., x_m \in \mathbf{N}$

$$f(x_1, ..., x_m) = h(g_1(x_1, ..., x_m), ..., g_k(x_1, ..., x_m))$$

Étant donné les fonctions $h : \mathbf{N}^{m+2} \to \mathbf{N}$ et $g : \mathbf{N}^m \to \mathbf{N}$ la fonction $f : \mathbf{N}^{m+1} \to \mathbf{N}$ s'obtient des fonctions précédentes par *récursion primitive* si pour tous $x_1, ..., x_{m+1} \in \mathbf{N}$

$$f(0, x_2, ..., x_{m+1}) = g(x_2, ..., x_{m+1})$$
$$f(x_1 + 1, ..., x_{m+1}) = h(x_1, ..., x_{m+1}, f(x_1, ..., x_{m+1}))$$

5. Il y a des fonctions que l'on s'accorderait à classer comme « effectivement calculables » qui ne sont pas récursives primitives, un exemple célèbre étant la fonction d'Ackermann. L'ajout de l'opération de *minimisation non-bornée* aux opérations de composition et récursion primitive permet de retrouver toutes les fonctions calculables par une machine de Turing, que l'on appelle également les *fonctions récursives générales*, *fonctions récursives* ou tout simplement *fonctions calculables*. Pour une introduction très accessible à la calculabilité et aux fonctions récursives, nous recommandons au lecteur l'ouvrage de Wolper, 2001.

On voit bien que ces contextes divers présentent des aspects voisins, quoique distinguables, de la récursivité. Par exemple, l'algorithme 1 permet de calculer les valeurs de la fonction factorielle, qui est primitive récursive. En même temps, cette fonction peut être exprimée par une définition récursive : « La factorielle d'un nombre est soit 1, si le nombre est 0, soit le produit du nombre et la factorielle de son prédécesseur », ou encore par une suite dont les termes correspondent aux images de la fonction :

$$u_0 = 1$$
$$\forall n \geq 1 \quad u_n = n \times u_{n-1}$$

Une bonne partie des différents usages du mot « récursivité » peut être capturée grâce à la notion de *clôture* d'un ensemble par une famille de fonctions. Avant de préciser le sens de cette notion, nous en donnons un exemple :

Exemple 2. Considérons l'ensemble $\mathscr{F}(\mathbf{R}, \mathbf{R})$ des fonctions réelles d'une variable réelle. On dit qu'un sous-ensemble $E \subset \mathscr{F}(\mathbf{R}, \mathbf{R})$ est « clos » par l'opération de somme de fonctions si pour tout $f, g \in E$ on a aussi $f + g \in E$. On définit de façon analogue la clôture par l'opération de produit de fonctions. Il se trouve que l'ensemble des fonctions polynomiales $\mathbf{R}[x]$ peut être défini comme le plus petit sous-ensemble de $\mathscr{F}(\mathbf{R}, \mathbf{R})$ (i) contenant les fonctions constantes et la fonction identité, et (ii) clos par les opérations de somme et produit. Un autre point de vue, aboutissant au même ensemble défini, met l'accent sur la construction des objets de $\mathbf{R}[x]$. Il s'agit de penser à $\mathbf{R}[x]$ comme l'ensemble que l'on engendre en partant des fonctions constantes et de l'identité, et en appliquant successivement les opérations de somme et de produit aux fonctions que l'on obtient à chaque étape.
Par exemple, on peut construire le polynôme $P(x) = 2x + 3$ comme suit :

— $P_0(x) = x$ est dans $\mathbf{R}[x]$ car c'est l'identité.

— $P_1(x) = 2$ et $P_2(x) = 3$ sont dans $\mathbf{R}[x]$ car ce sont des fonctions constantes.

— $P_3(x) = 2x$ est dans $\mathbf{R}[x]$ car c'est le produit de P_0 et P_1.

— $P(x) = 2x + 3$ est dans $\mathbf{R}[x]$ car c'est la somme de P_3 et P_2.

Le lecteur pourra se convaincre du fait qu'une construction similaire est possible pour chaque polynôme de $\mathbf{R}[x]$, et notera également que pour un polynôme donné, la construction n'est pas unique.

Nous allons maintenant examiner les deux points de vue que suggère notre exemple de façon plus générale. Nous suivons à peu près la notation et la présentation de MAKINSON (2012).

Nous considérons un ensemble de référence $U \neq \emptyset$ et une fonction $f : U^n \longrightarrow U$ où $n \in \mathbf{N}$. Un ensemble $X \subset U$ est dit *clos* par la fonction f ssi $f(X^n) \subset X$.

Étant donné un ensemble $B \subset U$, nous appelons la *clôture* de B par f l'intersection de tous les ensembles contenant B et qui sont clos par la fonction f (il y a au moins un ensemble qui satisfait cette description, à savoir, U). Nous noterons $f[B]$ la clôture de B par f :

$$f[B] := \bigcap_{\substack{B \subset X \\ f(X^n) \subset X}} X .$$

Il est aisé de montrer que $f[B]$ est également clos par f, ce qui implique qu'il peut aussi être caractérisé comme le plus petit ensemble clos par f contenant B.

Quand la clôture $f[B]$ est présentée de cette manière, on dit qu'elle est définie « par le haut », parce que l'intersection fait intervenir des ensembles « plus grands » $X \supset f[B]$. Il existe, cependant, une autre manière de définir la

clôture qui lui est équivalente, dans le sens où l'on obtient toujours le même ensemble, mais qui fait intervenir des ensembles plus petits que $f[B]$. On appelle parfois cette définition « par le bas », et elle met en avant le processus de construction des éléments de $f[B]$. Pour donner cette définition par le bas, nous définissons d'abord une suite d'ensembles :

$$B_0 = B$$

$$\forall k \in \mathbf{N}\ B_{k+1} = B_k \cup f((B_k)^n)$$

Nous définissons $f^*[B]$ comme l'union de tous ces ensembles :

$$f^*[B] := \bigcup_{k \in \mathbf{N}} B_k$$

Le théorème suivant, que nous énonçons sans démonstration, affirme l'équivalence des définitions par le haut et par le bas de la clôture :

Théorème 2 $f[B] = f^*[B]$

Exemple 3. Dans l'ensemble des entiers naturels ($U = \mathbf{N}$), l'ensemble des nombres pairs P est le plus petit sous-ensemble de $\mathbf{N}$ contenant 0 et clos par la fonction

$$f : \mathbf{N} \longrightarrow \mathbf{N}$$
$$n \longmapsto n + 2$$

Notons que sur cet exemple très simple nous pouvons distinguer les liens entre la construction récursive qu'il décrit et les différents usages du mot « récursif » que nous avions discutés plus haut. Explicitons-les :

1. L'ensemble des nombres pairs peut être défini de façon récursive : « Un nombre pair est soit 0, soit le successeur du successeur d'un nombre pair».
2. On peut utiliser un algorithme récursif pour déterminer si un entier naturel donné est pair ou non (voir l'algorithme 2)

Algorithme 2 : Parité de l'entier naturel n

```
pair(n)
si n = 0 alors
 | retourner true
sinon si n = 1 alors
 | retourner false
sinon
 | retourner pair(n − 2)
fin
```

3. Les nombres pairs sont aussi les termes d'une suite récursive :

$$u_0 = 0$$

$$\forall n \in \mathbf{N}\ \ u_{n+1} = u_n + 2$$

Concluons cette section en mentionnant qu'il est possible de généraliser la définition de clôture pour une famille de fonctions $\mathcal{F} = \{f_1, f_2, ..., f_m\}$, où $m \geq 1$ et pour tout $i \in \{1,\ldots, m\}$, $f_i : U^{n_i} \longrightarrow U$, avec $n_i \in \mathbf{N}$. Nous définissons $\mathcal{F}[B]$ comme l'intersection de tous les ensembles contenant B et qui sont clos par chacune des fonctions $f_i \in \mathcal{F}$. De façon équivalente, on peut dire que $\mathcal{F}[B]$ est le plus petit sous-ensemble de U contenant B et clos par chacune des fonctions $f_i \in \mathcal{F}$. Dans ce contexte, les fonctions $f_i \in \mathcal{F}$ sont souvent appelées « constructeurs ».

2. — Travaux didactiques sur la récurrence et la récursivité

2.1 *Travaux existants sur la récurrence*

En France, le raisonnement par récurrence est introduit dans les programmes de terminale S, où il est attendu des étudiants qu'ils apprennent à « mener un raisonnement par récurrence ». Les programmes indiquent également que ce type de raisonnement doit intervenir tout au long de l'année, et non seulement dans le cadre de l'étude des suites. Or, dans les ressources

d'accompagnement (MEN, 2016), on trouve peu d'applications du raisonnement par récurrence en dehors de l'étude des suites.

La littérature s'adressant aux difficultés de l'enseignement et à l'apprentissage de la récurrence est très vaste. Dès le début du XXe siècle, nous trouvons des documents traitant des problèmes « pédagogiques » liés à ce type de raisonnement. Par exemple, YOUNG (1908) préconise l'utilisation de l'induction mathématique comme moyen d'initier les étudiants à l'étude philosophique de la pensée mathématique et propose une série d'exercices pour montrer l'application du principe, ainsi que des situations non-mathématiques qui illustreraient le fonctionnement du raisonnement par récurrence, par exemple un escalier dont la n-ième marche permet d'accéder à la (n + 1)-ième marche, et il faut pouvoir accéder à la première marche pour pouvoir monter l'escalier. MICHAELSON (2008) présente une revue des publications en langue anglaise sur le sujet. Nous rappellerons ici seulement quelques conclusions tirées par GRENIER (2012) dans son article intitulé « Une étude didactique du concept de récurrence ». Parmi les conceptions erronées des étudiants de licence et master et des enseignants du supérieur, on trouve :

— Une preuve par récurrence ne construit pas d'objets mathématiques, car la propriété que l'on veut démontrer est donnée comme vraie au départ.

— La récurrence ne démontre rien, car on suppose ce que l'on veut démontrer.

— L'initialisation est obligatoirement la première étape.

— Le point à partir duquel l'hérédité est vraie et le rang initial n_0 doivent coïncider.

— Il n'y a pas de lien entre la preuve par récurrence et la preuve par descente infinie.

GRENIER (2012) souligne les liens entre ces conceptions et les choix qui ont été faits pour introduire la récurrence dans différents manuels de lycée et universitaires. De nombreux défauts ont été identifiés dans ces textes, notamment en ce qui concerne la rigueur logique dans l'énoncé des hypothèses du raisonnement par récurrence.

Les difficultés qui émergent dans l'apprentissage du raisonnement par récurrence sont sans doute expliquées au moins partiellement par sa complexité logique et cognitive, comme en atteste la construction historique de cette méthode de preuve. ÉGRÉ (2015) effectue une étude historique et épistémologique de la récurrence et montre bien que les justifications qu'en donnent Poincaré, Frege et les formalistes (par exemple Hilbert) diffèrent à l'évidence les unes des autres, ce qui témoigne de la difficulté de la réduire à un principe qui irait de soi. De façon similaire, mais avec une perspective d'enseignement, PAPERT (1960) étudie les justifications de la récurrence d'après Heyting, Poincaré, Russell et Wittgenstein, pour ensuite proposer une séquence de « paliers de la récurrence », allant du simple principe de *modus ponens* jusqu'à la récurrence classique, et permettant de catégoriser par leur niveaux de complexité les différents précurseurs cognitifs de la récurrence. Nous ne détaillerons pas ici la caractérisation de chaque palier, mais mentionnerons toutefois quelques points critiques dans le chemin qu'ils tracent :

— Le passage du *modus ponens* « p et $p \longrightarrow q$, donc q » , à sa répétition : « p et $p \longrightarrow q$ et $q \longrightarrow r$, donc r ».

— Le passage à une suite de répétitions : « p_0 et $p_0 \longrightarrow p_1$ et $p_1 \longrightarrow p_2$ et ... et $p_9 \longrightarrow p_{10}$, donc p_{10} ».

— L'abrègement de la suite d'implications, avec un quantificateur borné : « p_0 et, pour

tout $k \in \{0, 1, ..., 9\}$, $p_k \rightarrow p_{k+1}$, donc p_{10} ».

— L'abrègement de la suite d'implications, avec un quantificateur borné par une borne « très large » : « p_0 et, pour tout $k \in \{0, 1, ..., 10^{10^{10}}\}$ $p_k \rightarrow p_{k+1}$, donc $p_{10^{10^{10}}}$ ».

— L'extension du raisonnement au nombre quelconque « n »: « p_0 et, pour tout $k < n$, $p_k \rightarrow p_{k+1}$, donc p_n ».

Cette dernière étape correspond au passage à l'*infini potentiel* des entiers naturels, le nombre n qui apparaît pouvant être aussi grand que l'on veut. Nous ajouterions encore une étape, à savoir, le passage de l'infini potentiel à l'infini actuel des entiers naturels « p_0 et pour tout k $p_k \rightarrow p_{k+1}$, donc pour tout n, p_n ». Il s'agit là d'un véritable saut dans l'abîme, car le fait de savoir que l'on peut appliquer un raisonnement pour un nombre quelconque mais « instantiable » n et parvenir à affirmer p_n, n'est *a priori* pas équivalent, du point de vue cognitif, à pouvoir d'un seul coup faire une affirmation portant sur tous les (c.-à-d. sur toute l'*infinité des*) entiers naturels[6] .

Encore de nos jours, les mathématiciens, les informaticiens et les philosophes ont des regards très divers sur la justification et même sur le contenu de la récurrence. Quel que soit le « bon » fondement du principe, il ne s'agit certainement pas d'une intuition simple. Quoi qu'il en soit, la séquence de paliers de PAPERT (1960) et les erreurs identifiées par GRENIER (2012) montrent bien la sophistication du raisonnement par récurrence et permettent de comprendre qu'il y a de bonnes raisons pour que les étudiants se sentent dérangés par sa forme et doutent de sa validité.

2.2 *Travaux existants sur la récursivité*

La récursivité est une autre notion qui a attiré l'attention des didacticiens et chercheurs en éducation, en raison des nombreuses difficultés auxquelles les étudiants sont confrontés lorsqu'ils doivent apprendre à la maîtriser.

L'article de revue le plus complet que nous avons trouvé est celui rédigé par RINDERKNECHT (2014). Il contient plus de 200 références sur l'enseignement et l'apprentissage de la récursivité. MCCAULEY, GRISSOM, FITZGERALD et MURPHY (2015) écrivent eux aussi un article de revue sur les recherches au sujet de l'enseignement et l'apprentissage de la récursivité, se centrant sur celles qui présentent une dimension empirique. Ils décrivent les différentes difficultés que rencontrent les élèves et étudiants qui apprennent la récursivité, les erreurs et les conceptions erronées (*misconceptions*) les plus fréquentes chez eux, et les représentations mentales (*mental models*) qu'ils se font de la récursivité. Ensuite, les auteurs montrent que, malgré le fait que les différents travaux aboutissent parfois à des conclusions divergentes, quelques stratégies d'enseignement semblent donner généralement de bons résultats, par exemple : l'utilisation d'analogies et d'exemples concrets, la diversification des situations d'application, et l'application de méthodes de traçabilité ou prédiction de ce que font les programmes structurés récursivement.

6 D'ailleurs, il ne l'est pas, non plus, du point de vue logique, comme argumente HOFSTADTER (1999) dans son chapitre sur la théorie typographique des nombres : l'axiome de récurrence de Peano est fondamental pour pouvoir prouver même les propriétés les plus banales dès lorsque l'on veut les affirmer pour la totalité des entiers naturels. Par exemple, alors qu'on a une manière de démontrer $0 + n = n$ pour chaque instance de n en un nombre fini de pas et sans utiliser l'axiome de récurrence, il est impossible de montrer $\forall n \in \mathbf{N}\ 0 + n = n$ sans y faire appel.

Comme il est montré dans BARON et BRUILLARD (2011) beaucoup de travail sur la didactique de l'informatique a été fait pendant les années 1980. En France, cet intérêt au niveau de la recherche coïncide avec la mise en place d'un enseignement optionnel d'informatique dans l'enseignement secondaire. En ce qui concerne la didactique de la récursivité, on constate, notamment, que la question de l'ordre d'introduction adéquat entre les boucles itérative et récursive dans un cours de programmation est soulevée. Cette opposition se produit en miroir des débats entre les informaticiens sur les avantages des langages impératifs et récursifs (privilégiant les boucles itérative et récursive, respectivement). ARSAC (1992) compare cette dichotomie avec celle entre les fonctions constructives et implicites, qui divisait les mathématiciens au début du XXème siècle.

Revenons sur l'exemple de la fonction factorielle pour illustrer les différences entre les boucles itérative et récursive. Nous avons déjà montré un algorithme récursif (voir l'algorithme 1) calculant la factorielle d'un entier naturel *n*. Comparons-le avec l'algorithme 3, qui a, lui, une structure itérative. Notons que l'algorithme itératif exige d'utiliser une variable auxiliaire v qui cumule les résultats des multiplications successives, ainsi qu'une variable i qui mémorise à quelle étape de la boucle on se situe.

Algorithme 3 : Factorielle d'un entier naturel *n* (version itérative)

```
factit (n)
v=1
pour i ← 1 à n faire
  |  v=v*i
fin
retourner v
```

Faudrait-il enseigner d'abord l'itération ou la récursivité ? Si on ne prend en compte que l'aisance des élèves, on aurait du mal à défendre la récursivité : plusieurs auteurs lui accordent une difficulté supérieure (BENDER, 1988 ; MURNANE, 1992), et ceux qui ont étudié au plus proche sa didactique reconnaissent bien sa complexité :

> Nous avons cherché à construire une autre approche dans laquelle la structure récursive peut apparaître comme une réponse à un certain type de problèmes [...] Si on accorde quelque crédit à la thèse selon laquelle il serait important de lier à chaque connaissance (nouvelle) une situation fondamentale composée d'un ou plusieurs problèmes, associée à un mode de mise en rapport avec ce(s) problème(s), ce qui est notre cas, il ne nous semble pas que nous ayons isolé, pour le moment les éléments de cette situation. (ROUCHIER, 1990, p.40)

Devant cet état de l'art, il semblerait que la question est close. Cependant, nous considérons au moins deux arguments qui nuiraient à cette conclusion : i) la récursivité apparaîtra tôt ou tard dans le parcours d'un étudiant de mathématiques, d'informatique ou de sciences, alors peut-être gagnerait-on du temps en la lui faisant rencontrer plus tôt, et ii) le fait que la récursivité puisse être plus complexe que l'itération du point de vue cognitif impliquerait en même temps qu'elle est plus intéressante en termes du développement du potentiel cognitif des élèves.

La suppression de l'option d'informatique dans les années 1990 a sans doute rendu ce type de recherche moins pertinent et viable en France. On repère également un recul des recherches en didactique de l'informatique au niveau international depuis les années 1990, suivi d'un regain d'intérêt assez récent. Si l'on ajoute à cet état des choses les évolutions du curriculum, des langages de programmation et des équipements informatiques utilisés dans les collèges et lycées, on peut légi-

timement se demander si la récursivité doit être réservée aux seuls étudiants de la spécialité NSI (la récursivité est explicitement au programme de terminale), ou si elle pourrait aussi être enseignée à un plus grand nombre d'élèves du secondaire.

2.3 *Vers une étude du lien entre récurrence et récursivité*

Quelques auteurs ont mis en avant le lien entre récurrence et récursivité et ont discuté ses possibles conséquences pour l'enseignement. ANDERSON (1992) signale que les problèmes qui peuvent être résolus récursivement sont ceux qui ont une solution inductive, dans le sens où il existe des cas de base qui ont une solution simple, et où tout cas plus complexe peut être exprimé en termes d'un ou plusieurs cas réduits qui sont plus proches des cas de base. On voit moins clairement, par contre, le rapport qui s'établit entre cette solution inductive et la récurrence en tant que méthode de preuve.

LERON et ZAZKIS (1986) détaillent davantage la nature des relations entre récursivité du coté de l'informatique, et récurrence (*induction*) du coté des mathématiques, et décrivent plusieurs exemples qui leur permettent d'illustrer les différents regards (définition d'un objet mathématique, définition d'un algorithme, démonstration) que l'on peut avoir sur les applications des deux notions. Ils spéculent aussi sur les conséquences didactiques de ces relations : ils font l'hypothèse que la maîtrise préalable de la récursivité pourrait faciliter l'apprentissage de la preuve par récurrence.

POLYCARPOU (2006) explore la corrélation entre la compréhension des définitions inductives et des preuves par récurrence chez des étudiants de licence en informatique. Ses résultats montrent, d'une part, que les étudiants qui avaient une meilleure compréhension des définitions inductives de structures, obtenaient des meilleurs résultats dans l'application d'une preuve par récurrence, et d'autre part que ceux qui comprenaient moins les définitions inductives avaient tendance à appliquer mécaniquement le raisonnement par récurrence, sans se préoccuper du fait qu'ils étaient en train d'établir. Finalement, DRYSDALE (2011) argumente que le raisonnement par récurrence classique sur $\mathbf{N}$, où le pas inductif consiste à établir $P(k+1)$ sous l'hypothèse $P(k)$, est moins pertinent pour les informaticiens que l'application d'autres schémas inductifs, notamment l'induction structurelle, qui correspond plutôt, selon lui, à la récurrence forte sur $\mathbf{N}$. Il présente ainsi une approche différente à l'enseignement du raisonnement par récurrence qui consiste à le fonder non pas sur le dernier axiome de Peano, mais plutôt sur le fait que « la récursivité fonctionne » (c'est-à-dire que les programmes définis récursivement atteignent le résultat souhaité), principe avec lequel les étudiants d'informatique devraient être plus à l'aise.

Ce qui ressort de ces articles est une insistance sur l'idée que non seulement il existe un lien entre la récurrence et la récursivité du point de vue mathématique et informatique, mais aussi qu'il serait envisageable de tirer profit de ce lien dans l'enseignement des deux notions. L'exploration plus approfondie des caractéristiques de la relation entre récurrence et récursivité au sein du savoir savant des mathématiciens, informaticiens et logiciens devient alors un outil permettant d'éclairer les conditions sous lesquelles il serait possible, effectivement, d'implémenter des situations didactiques visant l'enseignement conjoint des deux notions.

3. — Résultats de l'enquête

3.1 *Fondements mathématiques de la récurrence et la récursivité*

Le raisonnement par récurrence dans les

entiers naturels a beau être le schéma d'induction le plus fréquemment enseigné aux niveaux secondaire et de licence, il n'est pas le plus utilisé ni en logique ni en informatique, et même son omniprésence en mathématiques doit être nuancée :

> ... ma formation initiale est en théorie des nombres, en arithmétique. Géométrie arithmétique. Donc les problèmes portent sur les nombres, donc on pourrait s'attendre à ce qu'il y ait beaucoup de récurrence, et je pense n'avoir jamais fait aucune preuve par récurrence dans le domaine. (C1)

De plus, des schémas d'induction plus généraux peuvent être plus pertinents pour attaquer des problèmes qui concernent des structures autres que celle des entiers naturels. Nous allons décrire le schéma d'induction structurelle, qui semble très pertinent pour travailler dans la plupart des structures récursives que l'on trouve fréquemment en informatique et dans certaines branches des mathématiques et qui a, d'ailleurs, comme cas particulier, le raisonnement par récurrence dans **N**.

Soit U un ensemble de référence. Nous dirons qu'une fonction $f : U^n \longrightarrow U$ *préserve* une propriété P ssi pour tout $(x_1, ..., x_n) \in U^n$, si chaque x_i possède la propriété P , alors $f(x_1, ..., x_n)$ possède la propriété P :

$$\forall x_1, ..., x_n \in U \quad P(x_1) \wedge ... \wedge P(x_n) \Rightarrow P(f(x_1, ..., x_n))$$

Théorème 3. (Induction structurelle) Soit P une propriété définie sur U, $m \geq 1$, $\mathcal{F} = \{f_1, f_2, ..., f_m\}$ une famille de fonctions et $B \subset U$. Si

1. tous les éléments de B ont la propriété P , et
2. chaque fonction $f_i \in \mathcal{F}$ préserve la propriété P

alors tout les éléments de $\mathcal{F}[B]$ ont la propriété P .

Démonstration. La validité du schéma découle directement de la définition de $\mathcal{F}[B]$. Soit $X = \{x \in U \mid P(x)\}$. L'hypothèse 1 indique que $B \subset X$, tandis que l'hypothèse 2 indique que X est clos par chacune des fonctions $f_i \in \mathcal{F}$. $\mathcal{F}[B]$ étant l'intersection de tous les ensembles remplissant ces deux conditions, nous concluons que $\mathcal{F}[B] \subset X$.

Nous voyons de cette manière qu'à chaque ensemble défini récursivement $\mathcal{F}[B]$ correspond un schéma de preuve inductif qui permet de démontrer que tous les éléments de B possèdent une certaine propriété pourvu que i) les éléments de B la possèdent et ii) les fonctions $f_i \in \mathcal{F}$ qui « construisent » $\mathcal{F}[B]$ à partir de B la préservent. Nous allons voir dans 3.2 plusieurs exemples illustrant cette dualité. En particulier, nous verrons que les entiers naturels peuvent être construits récursivement, d'une façon très proche de celle que nous venons de décrire. La récurrence classique apparaîtra alors comme un cas particulier d'induction structurelle.

Faut-il parler de raisonnement par récurrence quand il s'agit de l'application d'un schéma comme celui de l'induction structurelle ? Il semble que, dans l'usage, parfois on confond les deux notions, et parfois on les distingue :

> Après, tout dépend de quel type de récurrence on parle. Si tu prends uniquement la récurrence où le cas m implique le cas $m+1$, ou si tu prends quelque chose de plus large, ou quelque chose qui est induit par la structure de l'objet que tu utilises, ça dépend. Je pense qu'on peut faire plusieurs distinctions, tout dépend de ce que tu appelles récurrence. (C3)

Nous avons choisi d'appeler « raisonnement par récurrence » l'application des schémas d'induction *sur les entiers naturels*, mais il ne s'agit, bien sûr, que d'une convention.

Il est possible de donner un cadre encore plus général pour le raisonnement inductif. L'induction structurelle est un cas particulier de l'induction bien fondée. Cette dernière, que l'on appelle aussi induction nœthérienne, s'applique sur des ensembles bien fondés, c'est-à-dire, des ensembles sur lesquels on a défini une relation d'ordre (possiblement partielle) telle qu'il n'existe pas de suite infiniment décroissante. Nous ne donnerons pas de détails sur l'induction bien fondée dans cet article, mais il faut dire qu'elle n'a pas qu'un intérêt théorique, c'est-à-dire que parfois elle permet de prouver des faits qui deviennent très complexes à traiter si l'on veut faire appel à l'induction structurelle :

> Les entiers tu les définis comme ça. Tu dis zéro appartient à **N**, et si n – on suppose que c'est notre ensemble d'entiers – et si n appartient à **N**, alors $n+1$ appartient à **N**. Et tu vois bien que le schéma d'induction suit exactement cette forme-là. Donc elle colle, elle est structurelle, on dit. Donc ça, c'est l'induction qu'on connaît très bien, mais on sait très bien que c'est pas suffisant, si tu veux démontrer, je sais pas, des choses sur le PGCD, par exemple, si tu prends le schéma d'induction structurelle, tu vas pas démontrer grand-chose.Tu n'arriveras pas, ça va être compliqué. Donc, il te faut une induction un peu différente, par cas, il faut ce qu'on appelle des ordres d'induction bien fondés et qui reposent sur des ordres bien fondés, c'est-à-dire qu'il y a pas de suite infinie, tu fais x inférieur à y, inférieur à z, et cetera, il n'y a pas de suite infinie dans l'ensemble... (C2)

3.2 *Exemples d'application*

3.2.1 *Les formules du calcul des propositions*

Nous suivons à peu près la présentation de CORI et LASCAR (2003a). Soit $V = \{A, B, C, ...\}$ un ensemble de variables propositionnelles et soit $\mathcal{A}$ l'alphabet formé par les variables propositionnelles, les symboles de connecteur propositionnel et les symboles de parenthèses, c'est-à-dire :

$$\mathcal{A} = V \cup \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\} \cup \{), (\} .$$

On suppose que cette union est disjointe. Soit $\mathcal{M}(\mathcal{A})$ l'ensemble des suites finies d'éléments de $\mathcal{A}$, que l'on appelle aussi l'*ensemble des mots sur* $\mathcal{A}$. Nous voulons distinguer parmi tous les mots de $\mathcal{M}(\mathcal{A})$ ceux qui représentent des formules bien formées. Par exemple, si A et B sont des variables propositionnelles, nous voudrions étudier les expressions du type $A \Rightarrow (B \Rightarrow A)$, tandis que nous serions moins intéressés par celles du type $)((A \wedge \Leftrightarrow B$. Pour formaliser cette distinction, nous considérons la famille de fonctions $\mathcal{F} = \{f_1, f_2, f_3, f_4, f_5\}$ où

$$\begin{aligned}
f_1 : \mathcal{M}(\mathcal{A}) &\longrightarrow \mathcal{M}(\mathcal{A}) \\
F &\longmapsto \neg F \\
f_2 : \mathcal{M}(\mathcal{A})^2 &\longrightarrow \mathcal{M}(\mathcal{A}) \\
(F, G) &\longmapsto (F \wedge G) \\
f_3 : \mathcal{M}(\mathcal{A})^2 &\longrightarrow \mathcal{M}(\mathcal{A}) \\
(F, G) &\longmapsto (F \vee G) \\
f_4 : \mathcal{M}(\mathcal{A})^2 &\longrightarrow \mathcal{M}(\mathcal{A}) \\
(F, G) &\longmapsto (F \Rightarrow G) \\
f_5 : \mathcal{M}(\mathcal{A})^2 &\longrightarrow \mathcal{M}(\mathcal{A}) \\
(F, G) &\longmapsto (F \Leftrightarrow G)
\end{aligned}$$

Nous pouvons définir récursivement l'ensemble des *formules bien formées* $\mathcal{F}$ comme le plus petit sous-ensemble de $\mathcal{M}(\mathcal{A})$ contenant V et clos par chacune des fonctions $f_i \in \mathcal{F}$. C'est-à-dire

$$\mathcal{F} := \mathcal{F}[V]$$

Nous profitons maintenant de la structure récursive de $\mathcal{F}$ pour démontrer par induction struc-

turelle quelques propriétés que satisfont ses éléments.

Proposition 1. Toutes les formules bien formées ont le même nombre de parenthèses ouvrantes « (» et fermantes «) ».

Démonstration. Pour chaque formule $F \in \mathcal{F}$, désignons par $o(F)$ son nombre de parenthèses ouvrantes et par $c(F)$ son nombre de parenthèses fermantes. Appelons P la propriété « avoir le même nombre de parenthèses ouvrantes et fermantes ».

1. *Cas de base :* Si $F \in V$, alors on a bien $P(F)$, car $o(F) = c(F) = 0$.
2. *Pas inductif* pour f_1 : f_1 n'ajoute pas de parenthèse à son argument, alors si $F \in \mathcal{F}$ est telle que $o(F) = c(F)$, on a bien : $o(f_1(F)) = c(f_1(F))$. Donc, f_1 préserve P .

   Pas inductif pour f_2 : Si $F, G \in \mathcal{F}$ sont telles que $o(F) = c(F)$ et $o(G) = c(G)$, alors

   $$o((F \wedge G)) = o(F) + o(G) + 1 = c(F) + c(G) + 1 = c((F \wedge G))$$

   donc f_2 préserve P. On obtient de la même manière que les fonctions f_3, f_4 et f_5 préservent P . Nous concluons, par induction structurelle, que tous les éléments de $\mathcal{F}$ ont la propriété P .

Pour la proposition suivante, nous avons besoin du concept d'équivalence logique. Deux formules bien formées F et G sont dites *logiquement équivalentes*, ce que l'on note $F \equiv G$, ssi leurs valeurs de vérité coïncident pour toutes les distributions de valeurs de vérité des variables propositionnelles dans V (pour des rappels sur le calcul des valeurs de vérité, nous renvoyons le lecteur vers CORI et LASCAR (2003a)). La relation $\equiv$ est une relation d'équivalence sur $\mathcal{F}$. Nous supposons prouvées les équivalences logiques suivantes.

Pour toutes formules bien formées F et G, on a :

$(F \Rightarrow G) \equiv (\neg F \vee G)$

$(F \Leftrightarrow G) \equiv ((F \Rightarrow G) \vee (G \Rightarrow F))$

$(F \wedge G) \equiv \neg(\neg F \vee \neg G)$ (Loi de De Morgan)

Dans la démonstration de la proposition 2, on utilise également le fait que l'équivalence logique est préservée par les constructeurs $f_i \in \mathcal{F}$, ce que nous établissons comme lemme, sans en écrire la démonstration :

Lemme 1. Pour toutes $F, G \in \mathcal{F}$, si $F \equiv F'$ et $G \equiv G'$, alors $\neg F \equiv \neg F'$ et pour chaque symbole de connecteur propositionnel $\alpha \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$ $(F \alpha G) \equiv (F' \alpha G')$.

Proposition 2. Toute formule bien formée est logiquement équivalente à une formule n'incluant comme connecteurs propositionnels que $\neg$ ou $\vee$.

Démonstration. Soit $\mathcal{F}_{\neg,\vee}$ l'ensemble des formules bien formées n'incluant comme connecteurs propositionnels que $\neg$ ou $\vee$. Appelons P la propriété « être logiquement équivalent à une formule F de $\mathcal{F}_{\neg,\vee}$ ».

1. *Cas de base :* Soit $F \in V$. Alors F ne contient aucun connecteur propositionnel, donc elle possède la propriété P .
2. *Pas inductif pour* f_1 : Soit $F \in \mathcal{F}$. Si $F \equiv F'$, où $F' \in \mathcal{F}_{\neg,\vee}$, alors $\neg F \equiv \neg F'$, et $\neg F' \in \mathcal{F}_{\neg,\vee}$. Ainsi, f_1 préserve la propriété P .

   Pas inductif pour f_2, f_3, f_4, f_5 : Soient $F, G \in \mathcal{F}$. Si $F \equiv F'$ et $G \equiv G'$, où F', $G' \in \mathcal{F}_{\neg,\vee}$, alors :

   (a) $(F \wedge G) \equiv (F' \wedge G') \equiv \neg(\neg F' \vee \neg G')$ et $\neg(\neg F' \vee \neg G') \in \mathcal{F}_{\neg,\vee}$. Ainsi, f_2 préserve la propriété P .

(b) $(F \vee G) \equiv (F' \vee G')$ et $(F' \vee G') \in \mathscr{F}_{\neg,\vee}$. Ainsi, f_3 préserve la propriété P .

(c) $(F \Rightarrow G) \equiv (F' \Rightarrow G') \equiv (\neg F' \vee G')$ et $(\neg F' \vee G') \in \mathscr{F}_{\neg,\vee}$. Ainsi, f_4 préserve la propriété P.

(d) $(F \Leftrightarrow G) \equiv (F' \Leftrightarrow G') \equiv (F' \Rightarrow G') \wedge (G' \Rightarrow F') \equiv \neg(\neg(\neg F' \vee G') \vee \neg(\neg G' \vee F'))$ et cette dernière formule appartient à $\mathscr{F}_{\neg,\vee}$. Ainsi, f_5 préserve la propriété P .

Nous concluons, par induction structurelle, que tous les éléments de $\mathscr{F}$ ont la propriété P .

3.2.2 *Les listes*

Soit $\mathcal{A}$ un alphabet fini quelconque et $\mathscr{M}(\mathcal{A} \cup \{::\})$ l'ensemble des suites finies d'éléments de $\mathcal{A} \cup \{::\}$. Conventionnellement, nous inclurons dans $\mathscr{M}(\mathcal{A} \cup \{::\})$ la suite vide, que nous désignerons par le symbole ε. Considérons la famille des fonctions $\mathcal{F} = \{c_a\}_{a \in \mathcal{A}}$ données par

$$\begin{aligned} c_a : \mathscr{M}(\mathcal{A} \cup \{::\}) &\longrightarrow \mathscr{M}(\mathcal{A} \cup \{::\}) \\ \varepsilon &\longmapsto a \\ x &\longmapsto a :: x \end{aligned}$$

Notons que la fonction c_a « ajoute un $a ::$ » au début de n'importe quelle suite de caractères non vide, et transforme la suite vide en la suite ayant comme seul caractère a. Nous définissons *l'ensemble des listes sur* $\mathcal{A}$, $\mathscr{L}(\mathcal{A})$, comme le plus petit sous-ensemble de $\mathscr{M}(\mathcal{A} \cup \{::\})$ contenant ε et clos par chacune des fonctions c_a, c'est-à-dire :

$$\mathscr{L}(\mathcal{A}) := \mathcal{F}[\varepsilon]$$

Du point de vue d'un informaticien, il n'est pas anodin qu'un ensemble comme celui des listes puisse être défini récursivement. Si l'on veut écrire un programme s'appliquant sur des objets qui appartiennent à une structure de données récursive, l'approche récursive semble beaucoup plus adaptée que l'itérative :

Par exemple, en Java, les gens ils font des listes, ils écrivent les listes comme des listes chaînées, c'est assez facile d'écrire des listes comme ça, en fait, c'est comme ça que tu vas les écrire, et derrière ils écrivent une palanquée de programmes impératifs, alors du coup, pour parcourir une liste avec un programme itératif, t'es obligé d'avoir un pointeur, t'es obligé de déplacer le pointeur. C'est faisable, bien sûr, ils y arrivent bien, mais, tu vois, t'as envie de leur dire « écoutez, votre structure est récursive, et donc, ça serait bien d'être plus en adéquation avec la structure de données », donc tout dépend de la structure de données... (C2)

Observons que la structure récursive de $\mathscr{L}(\mathcal{A})$ nous permet de définir aisément par récursion structurelle des fonctions ayant $\mathscr{L}(\mathcal{A})$ comme domaine. Par exemple, l'opération de concaténation de deux listes, notée @ et donnée par :

$$\forall y \in \mathscr{L}(\mathcal{A})\ \varepsilon @ y = y$$
$$\forall a \in \mathcal{A}\ \forall x, y \in \mathscr{L}(\mathcal{A})\ c_a(x) @ y = c_a(x @ y)$$

Nous pouvons profiter de la dualité récursivité-récurrence pour prouver l'associativité de l'opération de concaténation :

Proposition 3. La concaténation est associative. C'est-à-dire :

$$\forall x, y, z \in \mathscr{L}(\mathcal{A})\ \ (x @ y) @ z = x @ (y @ z)$$

Démonstration. Nous procédons par induction structurelle sur la variable x :

1. *Cas de base :* Soient $y, z \in \mathscr{L}(\mathcal{A})$. Nous avons $(\varepsilon @ y) @ z = y @ z = \varepsilon @ (y @ z)$

2. *Pas inductif :* Soient $x, y, z \in \mathscr{L}(\mathcal{A})$ et $a \in \mathcal{A}$. Si

$$(x @ y) @ z = x @ (y @ z)$$

alors

$$
\begin{aligned}
(c_a(x)@y)@z &= c_a(x@y)@z \\
&\qquad \text{[définition de @]} \\
&= c_a((x@y)@z) \\
&\qquad \text{[définition de @]} \\
&= c_a(x@(y@z)) \\
&\qquad \text{[hypothèse d'induction]} \\
&= c_a(x)@(y@z) \\
&\qquad \text{[définition de @]}
\end{aligned}
$$

Nous concluons que pour tous $x, y, z \in \mathcal{L}(\mathcal{A})$, $(x@y)@z = x@(y@z)$.

De façon similaire, nous définissons la fonction d'inversion d'une liste, *inv*, donnée par

$$\text{inv}(\varepsilon) = \varepsilon$$

$$\forall a \in \mathcal{A}\ \forall x \in \mathcal{L}(\mathcal{A})\ \ inv(c_a(x)) = inv(x)@a$$

Nous invitons le lecteur à écrire les démonstrations par induction structurelle des propositions suivantes :

Proposition 4. ε est un élément neutre pour l'opération @. C'est-à-dire

$$\forall x \in \mathcal{L}(\mathcal{A})\ x@\varepsilon = \varepsilon@x = x$$

Proposition 5. $\forall x, y \in \mathcal{L}(\mathcal{A})$

$$inv(x@y) = inv(y)@inv(x)$$

Proposition 6. $\forall x \in \mathcal{L}(\mathcal{A})\ inv(inv(x)) = x$

3.2.3 *Les entiers naturels*

La phrase « un entier naturel est soit 0, soit le successeur d'un entier naturel » exprime bien le caractère récursif de l'ensemble des entiers naturels. Bien que sa formalisation dans l'axiomatique de Peano puisse cacher cette idée plus intuitive sur sa structure, dans la théorie des ensembles on retrouve la construction récursive de **N**. Nous donnons les idées générales de cette construction, où les entiers naturels sont définis comme des ensembles (pour plus de détails voir, par exemple, HALMOS (1974)).

On définit 0 comme l'ensemble vide :

$$0 := \emptyset$$

Et pour tout ensemble n on définit son successeur $S(n)$ comme l'ensemble contenant n lui-même, tous les éléments de n, et rien d'autre. C'est-à-dire :

$$S(n) := n \cup \{n\}$$

Ceci permet de construire chaque entier naturel :

$$1 := S(0) = \{\emptyset\}$$

$$2 := S(1) = \{\emptyset, \{\emptyset\}\}$$

$$3 := S(2) = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$$

$$\vdots$$

Mais cela n'implique pas la possibilité de construire l'ensemble contenant chaque ensemble obtenu par ce procédé. Il faut alors faire appel à l'axiome d'infinité (faisant partie des axiomes de Zermelo-Fraenkel pour la théorie des ensembles), qui indique « qu'il existe un ensemble contenant le successeur de chacun de ses éléments et l'ensemble $\emptyset$ ».

On voit bien le parallèle entre cette construction et celles que nous avons évoquées pour les formules bien formées du calcul des propositions et pour les listes : {0} joue le rôle de l'ensemble de base, tandis que l'opération successeur joue le rôle des fonctions de construction $f_i \in \mathcal{F}$. Il est alors très naturel que le raisonnement par récurrence classique apparaisse comme une application du schéma d'induction structurelle au cas spécifique où l'ensemble qui a été construit récursivement est **N** : pour prouver que tous les éléments de **N** possèdent une propriété, il suffit de mon-

trer que 0 la possède et que l'opération successeur la préserve.

3.2.4 *La programmation fonctionnelle*

Les structures définies récursivement revêtent une importance capitale pour la programmation fonctionnelle. Dans ce paradigme de programmation, les programmes sont constitués de fonctions qui ressemblent beaucoup à leurs analogues mathématiques, en ce sens qu'elles prennent une entrée (input) et fournissent une sortie (output). Ces fonctions peuvent, dans leur exécution, faire appel à d'autres fonctions, où à elles-mêmes. Parmi les avantages des programmes fonctionnels, on compte le fait que les fonctions n'ont pas d'effet autre que le calcul de leur résultat, ce qui élimine une source de bugs informatiques (HUGHES, 1989).

Nous donnons par la suite un exemple simple et informel du rôle de l'induction structurelle dans la programmation fonctionnelle et orientons le lecteur vers BURSTALL (1969) pour obtenir plus de détails sur le sujet.

La programmation fonctionnelle peut être encouragée ou même requise par certains langages de programmation tels que Haskell, Lisp, Scheme et Caml. Nous allons considérer Caml pour cet exemple, évoqué par WALKER (2019). Dans Caml, un type récursif comme celui des arbres binaires peut être défini de la manière suivante :

```
type tree = Leaf | Node of int * tree * tree;;
```

Cela signifie qu'une donnée de type tree peut être soit un arbre qui ne contient qu'une feuille, soit un triplet constitué d'une donnée de type int (entier naturel) et de deux autres arbres. On reconnaît bien une construction de nature récursive (avec un élément de base, et une modalité de construction de nouveaux éléments). On peut s'appuyer sur cette structure pour créer des fonctions récursives qui prendront des arbres comme paramètres, par exemple la fonction booléenne tmember, qui détermine si un certain n de type int est présent dans un arbre t :

```
let rec tmember (t:tree) (n:int) : bool =
  match t with
     Leaf -> false
  | Node (j,t1,t2) -> j = n || tmember t1 n
                      || tmember t2 n
;;
```

La commande match permet de repérer si l'arbre t en entrée est l'arbre à une seule feuille ou un arbre constitué d'un j de type int et de deux arbres, et d'agir différemment selon le cas :

— si l'arbre n'est constitué que d'une feuille, on retourne false (car il ne contient aucun int).

— autrement, si l'arbre est un triplet (j,t1,t2), où j est de type int et t1 et t2 sont de type arbre, on retourne la valeur logique de l'expression j=n *ou* tmember(t1,n) *ou* tmember(t2,n). C'est-à-dire que tmember renverra true si j est l'entier recherché où si l'entier n est présent dans un des deux sous-arbres t1 et t2.

Le lecteur reconnaîtra la structure récursive de la fonction, avec le traitement du cas de base puis du cas générique faisant appel à la fonction tmember elle-même sur deux cas « plus petits ». Comment pourrions-nous prouver que cette fonction est correcte ? Autrement dit, comment pouvons-nous prouver que pour tout couple (t,n), avec t de type tree et n de type int, la fonction fait ce qu'elle est censée faire : renvoyer true si n appartient à t et false s'il n'y

appartient pas ? En fait, il est possible de procéder par induction structurelle :

Le cas de base est celui de l'arbre à une seule feuille. Pour tout n de type int, la commande match identifie l'option Leaf et renvoie false, ce qui est correct, car l'arbre à une seule feuille ne contient aucun int.

Pour le pas inductif, nous supposons que nous avons des arbres t1 et t2, tells que la fonction tmember opère correctement sur tous les deux. C'est-à-dire que pour chaque n de type int, l'appel de fonction tmember t1 n renvoie true ou false correctement, selon que n appartient à t1 ou non, et de même pour t2.

Considérons maintenant un nouvel arbre t=j*t1*t2, construit à partir des arbres précédents et un j de type int. Nous voulons prouver que la fonction opère correctement sur t aussi.

Soit n de type int. Si n n'appartient pas à t, il ne peut pas être égal à j et il ne peut pas appartenir à t1 ni à t2. Par conséquent, le test j=n renvoie false, de même que les appels récursifs tmember t1 n et tmember t2 n, par hypothèse d'induction. Ainsi, tmember t n renvoie false, ce qui correspond à ce que nous attendions.

Si n appartient à t, il doit être égal à j ou il doit appartenir à t1 ou à t2. Ceci étant le cas, le test j=n renverra true, ou l'un des appels récursifs tmember t1 n ou tmember t2 n le fera. Dans tous les cas, tmember t n renvoie true, ce qui correspond à ce que nous attendions.

Cela montre que tmember fonctionne correctement aussi sur t.

Nous concluons, par induction structurelle, que la fonction est correcte pour toute instance.

Conclusions

Notre étude épistémologique nous a permis de mieux cerner les relations entre récurrence et récursivité, notamment à l'aide du concept d'induction structurelle. Nous avons montré que l'induction structurelle constitue une généralisation du raisonnement par récurrence très adéquate pour prouver les propriétés des ensembles définis récursivement, ainsi que celles des fonctions définies sur ces ensembles par récursion structurelle. Les exemples que nous avons fournis montrent l'intérêt de ces ensembles, et par ce fait de l'induction structurelle, dans le cadre de la logique, de l'informatique et des mathématiques.

Nous pensons que ce regard apporte une meilleure compréhension de la récurrence, et une prise de recul qui semble opportune pour les professeurs qui ont à l'enseigner. En particulier, il éclaire la relation entre récurrence et récursivité et montre à quelle point ces deux notions sont imbriquées et indissociables. Nous pensons que mettre cette dialectique au cœur de l'apprentissage peut avoir des conséquences positives pour la compréhension des élèves et étudiants. En outre, elle peut fournir un outil d'analyse des situations didactiques existantes qui visent l'enseignement de la récurrence, ainsi que pour la construction de nouvelles situations. Elle permet aussi d'envisager comment l'outil informatique pourrait être utilisé dans ces situations.

Dans les perspectives de cette étude, il serait intéressant de mieux comprendre les pratiques associées aux notions ciblées, par exemple à partir de l'analyse d'algorithmes récursifs et de preuves par induction structurelle plus complexes. De même, l'analyse du schéma d'induction bien fondée et de ses relations avec l'induction structurelle peut apporter encore un autre élément épistémologique profitable du point de vue didactique.

Références

ANDERSON, O. D. (1992). Induction, recursion, and the Towers of Hanoi. *International Journal of Mathematical Education in Science and Technology*, 23(3), 339-343.

ARSAC, J. (1992). Itération et récursivité. In *Troisième rencontre francophone de didactique de l'informatique* (p. 81-92). Association EPI (Enseignement Public et Informatique).

BARON, G.-L. & BRUILLARD, É. (2011). L'informatique et son enseignement dans l'enseignement scolaire général français : enjeux de pouvoir et de savoirs. In *Recherches et expertises pour l'enseignement scientifique* (T. 1, p. 79-90). De Boeck Supérieur.

BENDER, P. (1988). Valeur didactique des techniques recursives en programmation. In N. BALACHEFF & C. LABORDE (Éd.), *Actes du premier colloque franco-allemand de didactique des mathématiques et de l'informatique*. (p. 257-265). Premier colloque franco-allemand de didactique des mathématiques et de l'informatique. Luminy, France : La Pensée Sauvage.

BURSTALL, R. M. (1969). Proving properties of programs by structural induction. *The Computer Journal*, 12(1), 41-48.

CORI, R. & LASCAR, D. (2003a). *Logique mathématique. 1, Calcul propositionnel, algèbre de Boole, calcul des prédicats : cours et exercices corrigés*. Paris, France : Dunod.

CORI, R. & LASCAR, D.(2003b). *Logique mathématique. 2, Fonctions récursives, théorème de Gödel, théorie des ensembles, théorie des modèles : cours et exercices corrigés*. Paris, France : Dunod.

DRYSDALE, R. L. S. (2011). Mathematical Induction is a Recursive Technique. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education* (p. 269-274). SIGCSE '11. doi :10.1145/1953163.1953246

ÉGRÉ, P. (2015). Le raisonnement par récurrence : quel fondement ? *Gazette des Mathématiciens*, (146), 27-37.

GRENIER, D. (2012). Une étude didactique du concept de récurrence. *Petit X*, (88), 27-47.

HALMOS, P. R. (1974). *Naive set theory*. Undergraduate Texts in Mathematics. New York : Springer-Verlag. Récupérée 2 novembre 2018, à partir de // www.springer.com/gp/book/9780387900926

HOFSTADTER, D. R. (1999). *Gödel, Escher, Bach : an eternal golden braid*. New York, Etats-Unis d'Amérique : Basic Books.

HUGHES, J. (1989). Why functional programming matters. *The computer journal*, 32(2), 98-107.

LERON, U. & ZAZKIS, R. (1986). Computational recursion and mathematical induction. *For the Learning of Mathematics*, 6(2), 25-28.

LOMBARDI, H. (2011). *Épistémologie mathématique*. Paris, France : Ellipses.

MAKINSON, D. (2012). *Sets, logic and maths for computing* (2e éd.). Undergraduate Topics in Computer Science. London : Springer-Verlag. Récupérée 24 septembre 2018, à partir de //www.springer.com/fr/book/ 9781447124993

MCCAULEY, R., GRISSOM, S., FITZGERALD, S. & MURPHY, L. (2015, janvier 2). Teaching and learning recursive programming : a review of the research literature. *Computer Science Education*, 25(1), 37-66. doi :10.1080/ 08993408.2015.1033205

MEN. (2016). Exercices de mathématiques pour la classe terminale -2e partie. Ministère de l'Éducation nationale et de la Jeunesse. Récupérée 30 octobre 2018, à partir de http://cache.media.eduscol.education.fr/file/Mathematiques/64/8/Exercices_de_mathematiques_pour_la_classe_ terminale_-_2e_partie_536648.pdf

MICHAELSON, M. T. (2008). A Literature Review of Pedagogical Research on Mathematical Induction. *Australian Senior Mathematics Journal*, 22(2), 57.

Murnane, J. (1992). To iterate or to recurse? Computers & Education, 19(4), 387-394.

PAPERT, S. (1960). Problèmes épistémologiques et génétiques de la récurrence. In P. GRÉCO, J. GRIZE, S. PAPERT & J. PIAGET (Éd.), *Problèmes de la construction du nombre. Études d'Épistémologie Génétique* (T. XI). Paris, France : Presses Universitaires de France.

POINCARÉ, H. (2017). *La science et l'hypothèse*. Paris, France : Flammarion.

POLYCARPOU, I. (2006). Computer science students' difficulties with proofs by induction : an exploratory study. In *Proceedings of the 44th annual Southeast regional conference* (p. 601-606). ACM.

RINDERKNECHT, C. (2014). A Survey on Teaching and Learning Recursive Programming. *Informatics in Education*, 13(1), 87-120.

ROUCHIER, A. (1990). Objets de savoir de nature informatique dans l'enseignement secondaire. *ASTER*, (11), 29-44.

WALKER, D. (2019). COS 326 : Functional Programming. Récupérée 9 septembre 2019, à partir de https://www.cs.princeton.edu/~dpw/courses/ cos326-12/notes/reasoning-data.php

WOLPER, P. (2001). *Introduction à la calculabilité : cours et exercices corrigés*. Paris, France : Dunod.

YOUNG, D. J. W. A. (1908, août 1). On Mathematical Induction. *The American Mathematical Monthly*, 15(8), 145-153. doi :10.1080/00029890.1908. 11997444