Intellectica, 2004/2, **39**, pp. 125-137

La notion de récursivité, de la première cybernétique au connexionnisme

Pierre LIVET*

RÉSUMÉ. La première cybernétique (McCulloch), puis la seconde (Ashby et von Foerster) ont mis la récursivité au centre de leurs projets. Mais les deux ont sous estimé la puissance des fonctions récursives (trouver le code du programme qui calcule une fonction dépendant de ce code), tout comme les problèmes de ce qui n'est pas récursivement énumérable. Les théories liées au premier renouveau du connexionnisme ont prétendu elles aussi se situer à la fois en deçà et au-delà de la récursivité.

Mots clefs : récursivité, cybernétique, code, programme, connexionnisme.

ABSTRACT. The concept of recursivity, from the first cybernetics to connectionism. In the first cybernetics (McCulloch) and the second one (Ashby and von Foerster) as well recursivity was supposed to be a central concepts. But in both theories, the power of recursive functions (to find the code of the program which computes a function that depends on this very same code) as well as the troubles raised by non recursively enumerable functions have been overlooked. A similar position can be found in some theories related to the connectionism of the eighties, that have also been pretended to take place beyond the real power of recursivity.

Key words: recursivity, cybernetics, code, program, connectionism.

Le mot de « récursif » revient très souvent sous la plume aussi bien de von Foerster que de Maturana et Varela. Et quand on étudie l'analyse des comportements téléologiques dans l'article de Bigelow, Rosenblueth et Wiener, de 1943, on s'aperçoit que certaines de leurs distinctions correspondent à des catégories qu'ultérieurement von Foerster a empruntées explicitement à la théorie de la récursion. Enfin quand on étudie les tentatives des connexionnistes pour traiter des séquences de symboles tout en préservant leurs propriétés structurelles, d'une part on retrouve la notion de récurrence, et d'autre part on peut encore repérer un écho lointain des distinctions propres à la théorie des fonctions récursives. Il m'a donc semblé que ces distinctions pouvaient permettre des comparaisons utiles entre des travaux d'époques diverses, mais d'inspiration commune. Nous verrons que les cybernéticiens de la première période (Wiener, McCulloch, Shannon) faisaient de la récursion sans trop le savoir ; que ceux de la seconde (Ashby, von Foerster, puis Maturana et Varela) en faisaient en le sachant, mais en étant insensibles à la dualité d'aspects propre à la récursion, dualité entre son aspect de certitude et son aspect de limitation ; et enfin que les connexionnistes – nous n'étudierons que les 20 premières années du connexionnisme – se situent à la fois en deçà et en dehors

* Université d'Aix-Marseille 1. Département de Philosophie, 29 avenue Robert Schuman, 13621 Aix en Provence Cedex 1. E-mail : livet@newsup.univ-mrs.fr.

des fonctions récursives (les fonctions définies par composition d'autres fonctions simples) tout en en proposant des approximation

Rappel de la théorie de la récursion

Il nous faut d'abord disposer de quelques repères donnés par la théorie des fonctions récursives. On peut soutenir qu'elle est liée à trois perspectives : 1) déterminer ce qui est calculable ; 2) engendrer des fonctions complexes à partir de fonctions simples ; 3) présupposer l'inconnu pour produire du connu. La notion de récurrence chère à Poincaré, qui en faisait le moteur des mathématiques, nous donne un lien entre la seconde perspective et la troisième. Cette dernière perspective flirte avec le paradoxe. Elle l'évite, en s'appuyant sur les fondations proposées par les étagements de la deuxième perspective, mais elle permet d'introduire, avec la possibilité de points fixes, des procédés de diagonalisation qui, si on les généralise, nous font justement sortir du domaine du récursif patiemment construit selon la deuxième voie, et nous montrent ainsi les limites de ce qui est calculable de façon décidable.

Je n'ai pas besoin d'expliciter la première perspective. Elle renvoie à la thèse de Church ; « toute fonction algorithmique est une fonction récursive ». Elle nous donne par contraposition un moyen de suspecter une opération de n'être pas algorithmique : il nous suffit de démontrer que la fonction qui lui correspond n'est pas récursive. Inversement, si nous avons trouvé un algorithme qui calcule la fonction, nous savons qu'elle est récursive. Comme la notion d'algorithme est simplement décrite, mais non définie, le passage par la notion de fonction récursive est notre meilleur critère si nous voulons répondre à la question concernant la négative.

La seconde ne pose pas davantage de problèmes. La notion de fonction récursive recouvre des fonctions qui peuvent être définies à partir de fonctions précédemment construites. On fournit tous les pas d'opération nécessaires à la production des valeurs de la fonction. On peut ensuite la définir en composant d'autres fonctions, et enfin on peut faire appel à la récursion elle-même. On partira d'une valeur définie par stipulation pour 0, f(x,0) = une valeur donnée (qui pour l'addition, par exemple, est 0, et pour l'exponentiation est x). Puis on définira par récurrence f(x,n+1) comme une fonction de f(x,n). Ainsi pour l'addition, f(x,n+1) = f(x,n)+1 ; La définition de la fonction fait donc appel dans le *definiens* à la fonction à définir (la fonction f se retrouve aussi bien dans le definiens que dans le *definiandum*). Nous voyons donc déjà apparaître la troisième perspective : produire du connu à partir de l'inconnu. Car présupposer la fonction inconnue pour la définir provoque une circularité qui pourrait être gênante. Mais comme on sait comment amorcer l'entrée dans ce cercle, et comment procéder étapes par étapes, la récurrence, loin de nous bloquer dans un cercle, nous assure que nous pourrons toujours progresser de la même manière. Pour être plus exact, la définition de l'addition ne fait pas appel au " + ", mais seulement à la fonction successeur, qui est donnée comme fonction de départ. On se donne aussi les fonctions 0, d'identité, de composition, et la démarche récursive que l'on vient d'exemplifier. Une fonction qui peut être définie en utilisant ces procédés est une fonction récursive primitive. Si on veut pouvoir traiter le cas de récursions qui se font en parallèle sur deux variables (x+1 et y+1, par exemple) on passera à la notion de fonction récursive générale. On peut raisonner aussi sur des fonctionnelles (des fonctions de fonctions). Si on remplace une fonction f unaire totale par une séquence infinie

de nombres x, alors les étapes F(x;x, 0) = G(x;x) et F(x;x,y+1) = H (x;x,y, F(x;x,y)) nous donneront la fonctionnelle F par récursion.

La troisième démarche permet des développements encore plus audacieux. Elle s'exprime dans le théorème de la récursion, et dans les extensions de l'opération de diagonalisation. Je vais sommairement en résumer quelques points saillants, car cela va me servir de grille d'analyse pour la suite. Disons tout de suite que ces théorèmes montrent à la fois le pouvoir impressionnant de la récursion, et montrent aussi que ce pouvoir qui nous permet de définir ce que nous pouvons maîtriser comme calculable est en un sens « juste trop grand », puisqu'il nous permet d'indiquer aussi les limites de ce qui est calculable.

On peut associer à tout programme un code. Ce code peut d'ailleurs être simplement la suite des instructions qui constituent le programme, une fois agencées en une séquence qui a elle-même un code, et qui est elle-même une séquence de nombres. On sait par exemple que la machine de Turing universelle est construite dans l'idéal pour lire une séquence d'instructions qui comporte les données à traiter par un programme et le code de ce programme. Une fois trouvé le code, elle applique le programme qui lui correspond à la suite de données et effectue les opérations qu'il contient. On peut évidemment se poser la question de savoir si un programme donné fournit bien les mêmes résultats de calcul, les mêmes valeurs de sortie, qu'une fonction donnée. Maintenant supposons que la fonction en question soit une fonction de plusieurs variables, dont le numéro de code du dit programme. Il semble y avoir là une difficulté. Si on ne dispose pas encore du programme, comment pourrait-on identifier ce que font des opérations dont le contenu n'est déterminé qu'une fois qu'on dispose des données, à savoir le code du programme. En particulier, comment pourrait-on trouver le code qui identifie le programme ? Mais c'est justement l'intérêt d'avoir mis à part les programmes et les données. Cela permet à la fois de ne pas les confondre, et de prendre si on le souhaite le code du programme pour donnée, donc de réaliser une sorte d'auto-référence.

On peut alors se proposer de trouver le code du programme qui calcule une fonction qui dépend elle-même, entre autres données, du code de ce programme. Si on écrit {e} le programme de code e, et {e}x l'application à la donnée x du programme de code e, on cherche donc le programme tel que {e}x = f(x,e). L'étape suivante est de considérer que e peut varier, et qu'on aura alors différents programmes selon les variations de la variable du code. Les différents codes sont maintenant donnés par une fonction f(z). On se demande alors si les programmes codés par f(z), les {f(z)}, donnent bien à chaque fois les mêmes résultats que la fonction f(x,f(z),z), qui en fonction du code z, calcule le code du programme qu'elle applique à la valeur x. La dépendance des valeurs de sortie calculées par rapport aux x varie donc avec les z selon la fonction f(z). On peut exprimer cela en termes de fonctionnelles, en remplaçant comme plus haut les fonctions par des séquences associées aux variables. Dans les deux cas, le théorème de la récursion (Kleene, 1938) nous dit que si la fonction (resp. fonctionnelle) prescrite est récursive, alors on peut trouver une telle séquence f. Autrement dit, dès que le programme correspond à une fonction récursive, nous savons que le problème consistant à trouver le code du programme qui calcule une fonction dépendant entre autre de son propre code a une solution (pour savoir laquelle, il faut que la fonction récursive soit décrite).

Ce théorème est la face positive de la troisième perspective. Même si nous compliquons les choses, et que nous voulons trouver par exemple le coût du programme, incluant le paiement de celui qui est en train de chercher le programme et de le construire, coût qui dépend du temps mis à construire le programme, coût qui sera calculé en fonction de ce temps par le programme en question (qui reste pour l'instant une inconnue, comme son temps de construction et sa procédure de calcul), il nous suffirait de savoir que ce programme correspond à une fonction récursive (construite selon les procédés de la deuxième démarche) pour que nous sachions que nous pouvons trouver une solution au problème. Nous pouvons donc présupposer une inconnue pour trouver du connu si le lien entre les deux nous est donné par une fonction récursive.

La puissance de la récursion nous mène encore plus loin. Le théorème de Matijasevic, Robinson, Davis et Putnam (démontré en deux temps, d'abord par les trois derniers, puis, pour la dernière étape, par le premier, 1961-1970) établit une propriété positive (l'équivalence entre la notion de relation récursivement énumérable et celle de relation diophantine), mais il mène directement à la démonstration de l'impossibilité de décider à tous les coups si une équation diophantine a une solution ou non. Il donne donc une réponse négative à une des problèmes du programme de Hilbert (une équation diophantine est une équation du type $f(X) = 0$, où X est un ensemble de variables et f est une fonction polynomiale à coefficients entiers rationnels et dont on cherche la solution en nombres entiers rationnels).

On part, en un sens, de la possibilité que nous donne le théorème de la récursion de connaître un programme par son code. On suppose ainsi qu'une relation P nous est donnée par le moyen d'un index e. Les variables reliées par P (les variables de l'ensemble X) appartiennent donc au domaine dont part le programme de code e, à savoir {e}. Le théorème de nos quatre auteurs donne alors une méthode pour trouver à partir de ce programme la fonction polynomiale f, fonction de m variables, pour laquelle la relation $R(X) = P(X,y_1,\ldots,y_m) = 0$. Sachant le code du programme, nous pouvons donc trouver des solutions, s'il y en a. Mais cette même puissance, utilisée de manière auto-référente, va nous amener à démontrer l'impossibilité du problème de la décision concernant une équation diophantine quelconque. Et ce parce qu'on va trouver une propriété (une relation unaire) pourtant reliée à une fonction polynomiale, mais qui va se révéler identique à une propriété récursivement énumérable mais non récursive (ce qui veut dire que l'on peut énumérer par une procédure effective les termes tombant dans l'extension de la propriété, mais qu'imaginer une procédure effective pour décider de ceux qui ne sont pas porteurs de cette propriété nous mènerait à une contradiction).

On suppose d'abord une propriété P(z) qui vaut pour z si et seulement si z est le nombre de code du programme réalisant une fonction f, polynomiale, dont le calcul des valeurs a une solution. Si on avait ce programme, on saurait que le calcul de la solution de l'équation f=0 de cette fonction est décidable.

On envisage ensuite une propriété R qui, elle, est non récursive, et on va montrer que pour un argument (f(z)), $R(z) = P(f(z))$, si bien qu'en fait notre propriété P est non récursive. Évidemment, on va choisir pour R une propriété d'auto-référence ou de « diagonalisation ». On sait par exemple que la fonction ou relation unaire R(z) telle que sa valeur soit égale à un z qui appartient au domaine du programme de code z, {z}, est bien récursivement énumérable –

elle correspond à une fonction f(z,z,y) – mais que sa négation ne l'est pas, si bien que la propriété R n'est pas récursive. Sa négation exige en effet que la valeur de la fonction f(z) n'appartienne pas au domaine du programme {z}. Or dans ce cas, z, la valeur de la fonction, appartient cependant au domaine d'un programme, disons de code {e}. Mais il suffit de faire {e} = {z}, donc d'appliquer une diagonalisation, pour voir que la récursivité de la négation de R conduit à une contradiction, la valeur appartenant et n'appartenant pas au domaine en question.

Cependant, le théorème « des quatre » nous permet de trouver, connaissant l'index (z), la fonction polynomiale pour laquelle f(X,z)=0, et dans chaque cas, f(X,z)=R(X). Pour chaque valeur de (z) nous allons trouver une telle fonction polynomiale fz telle que fz (X)=f(X,z). Supposons que f (z) soit la fonction réductible à une séquence de nombres qui nous donne justement par cette séquence le code du programme réalisant la fonction fz. La fonction f (z) est une fonction récursive. Mais alors, f (z)=fz, et fz (X)=f(X,z). Or que fait R(z) ? Elle nous dit comment fz varie avec chaque nouvelle valeur de (z). Mais f(z) nous assure que z est le code du programme qui réalise pour une valeur de (z) la fonction fz, laquelle permet de trouver une solution, en remontant à l'équation f(X,z)=0, pour notre fonction. Or cet ensemble de propriétés que nous venons d'énoncer est justement celui qui satisfait, pour un z donné, la propriété P initiale. Si donc nous prenons pour argument de P la fonction f(z) elle-même, en tant que séquence de nombres, nous lui donnons pour argument la fonction qui nous dit comment fz varie avec chaque nouvelle valeur de z, et nous retrouvons ainsi R(z), si bien que R(z)=P(f(z)). Mais on a montré que R(z) n'est pas récursive, donc P(z) ne peut pas l'être non plus[1]. Nous n'arriverons pas pour tous les z à décider si asserter la propriété P est valide.

Autrement dit les opérations qu'on peut dire au sens large d'auto-référence ne sont pas nocives tant qu'elles portent simplement sur des identités entre index du code et variable traitée. Mais elles peuvent nous faire sortir des limites du calculable dès que nous les étendons pour parler de l'existence de solutions, ou de définition ou non définition des valeurs de la fonction, bref, dès que nous les utilisons pour traduire des propriétés méta-linguistiques, dès que l'auto-référence devient une sorte de représentation des propriétés opératives du langage.

La leçon de ces théorèmes semble être la suivante : la puissance de la récursion est immense, puisqu'elle permet d'exprimer certains types d'auto-référence, et donc de formaliser des questions que le mathématicien se pose sur ses propres démarches. Mais nous voulions une puissance qui corresponde très exactement à ce que nous pouvons maîtriser. Notre maîtrise a cependant des limites. Or cette puissance nous permet de les atteindre. On peut aussi proposer une version optimiste : la puissance de ce que nous pouvons maîtriser va jusqu'à démontrer, de façon maîtrisée, que nous ne pouvons pas maîtriser certaines propriétés (c'est la version de Judson Webb). Nous pouvons donc maîtriser le fait de ne pas maîtriser certaines opérations. Évidemment, cela nous permet seulement de savoir que ces opérations sont au delà des limites, pas de maîtriser ces opérations.

---

[1] Pour tout ceci, cf. Bell et Machover, 1975, *A course in mathematical logic*, North Holland, p. 259, 269, 273-5, 282, 313., et Delong H., 1970, *A profile of mathematical logic*, Addison-Wesley Reading, Massachusetts.

Tout cela n'est que rappel de choses ressassées, et il nous faut maintenant montrer quels éclairages ces distinctions peuvent jeter sur les débats de la cybernétique et de sa descendance.

En deçà de la récursion.

La récursion amenant à raisonner sur des fonctions dont une variable au moins est la valeur fournie par une application précédente de la fonction, on pense immédiatement à relier la notion de récursion et celles de rétroaction ou de feed-back.

Or si nous revenons à l'article princeps de la cybernétique, celui de Bigelow, Rosenblueth et Wiener sur les comportements "purposive" (1943), nous noterons qu'ils faisaient une différence entre le comportement "purposeful" et celui régulé par "feed-back", puis entre celui qui est régulé par "feed-back" et enfin celui qui est régulé par anticipation. Le premier est représenté par les servo-mécanismes qui se bornent à réorienter la trajectoire d'un missile quand arrivé à un certain point sa cible n'est pas dans la zone reconnue. Il en est de même pour des robots qui se déplacent et repartent dans une autre direction quand ils ont heurté un mur. En revanche le feed-back consiste en une régulation en continu pendant toute l'exécution du déplacement. Autrement dit, la valeur de la sortie précédente de la fonction, dans le comportement "purposeful", n'est prise en compte que de manière discrète, à certaines étapes, alors qu'elle est influente de manière continue, ou en tous les points du temps, dans le feed-back. Et enfin quand on anticipe, c'est qu'on peut changer de fonction par rapport à celle dans laquelle on est présentement engagé. On retrouve donc ici des étapes de la construction récursive. On va voir que von Foerster reprendra cette idée d'une fonction qui consisterait à changer de fonction.

Éliminons cependant une difficulté qui pourrait surgir. Les différents auteurs que nous analyserons utilisent à la fois la notion de feed-back et celle de récursivité, sans s'appesantir sur le fait que prendre vraiment au sérieux la boucle d'un feed-back interdit justement toute récursivité. On ne peut pas considérer comme une fonction récursive une fonction qui serait fonction à la fois de son entrée x et de sa sortie y au même instant, si bien que x dépend de y qui dépend de x. Pour la retrouver, il faut évidemment décaler dans le temps les entrées et les sorties, et admettre un t0 où x est donné sans que y ait encore une influence, qu'il n'aura qu'au temps t+1 sur l'entrée x en t+1 (à supposer que le feed-back se fasse sans délai).

Passons maintenant à l'article d'Ashby sur « Les principes du système auto-organisé » (1962)[2] Il s'agit en quelque sorte d'un essai pour miner à l'avance toute théorie de l'auto-organisation qui prétende faire « émerger du nouveau » par des processus de rétroaction et de réentrée. Il part d'une machine, conçue comme une fonction. Il soutient que faire de f une fonction de l'état interne de la machine est un non sens (p. 268). En effet, si je tente de démultiplier la fonction en une fonction f_a quand l'état interne est a, f_b quand il est b, etc., quand j'aurais a en entrée, je ne vais appliquer que fa, si b est l'entrée, je ne vais appliquer que f_b, etc., si bien que tout cela se réduit à avoir découpé f en tranches, puisque f est une relation qui relie l'état a à une sortie, l'état b à une autre, etc. L'auto-organisation se ramène donc à changer de fonction en fonc-

[2] Von Foerster a été l'éditeur du volume qui comprend ce texte, et il est suprenant de voir qu'il a utilisé des notions similaires non plus dans un but de réduction de l'auto-organisation, mais dans un but, si l'on peut dire, de magnification.

tion de l'environnement. Si l'ensemble, si la nouvelle machine formée par l'environnement et la première machine a un riche potentiel (si sa fonction a des comportements diversifiés) alors nous pourrons avoir la possibilité de varier nos conduites en fonction des changements de l'environnement, c'est-à-dire, selon Ashby, la possibilité d'être intelligents, et il sera même inévitable que nous le soyons !

Ashby n'a pas tort sur le premier point, s'il refuse de faire de l'état interne s qu'une fonction de s et de x prend pour variable une source de modification de la fonction elle-même, qui régit justement ces variations. Mais il a tort s'il pense exclure par là la possibilité qu'une fonction dépende d'une variable qui est le numéro de code du programme que réalise la fonction. Il a sous-estimé la puissance du théorème de la récursion, qui nous dit précisément que nous pouvons trouver le code du programme calculant une fonction qui dépend entre autres variables de son propre code. De même, Ashby tient, semble-t-il, à traduire la relation entre environnement et machine comme une composition de fonctions, et non pas comme une fonctionnelle, ce que fera ensuite von Foerster. Bref on trouve chez Ashby à la fois une proximité avec la théorie de la récursion et une sous-estimation de sa complexité, traits que l'on va retrouver à un moindre degré chez von Foerster.

Le parallèle avec la récursion

Dans "What is memory" (1969), von Foerster utilise très systématiquement la notion de fonction récursive. La valeur y de la fonction f au temps t, nous dit-il, dépend de la fonction f appliquée à l'entrée x_t, et de la valeur y en t-1, qui elle-même dépend de f appliquée à x_{t-1}, de y_{t-2}, etc. jusqu'à ce qu'on arrive en y_0. Von Foerster ne fait là qu'appliquer à l'envers la démarche de définition des valeurs d'une fonction récursive, celle qui construit les valeurs de f en partant de x=0. Il ajoute seulement une indiciation par le temps. La récursivité permet d'utiliser la récurrence pour emboîter la fonction dans elle-même. L'interprétation qu'en donne von Foerster est plus audacieuse. Puisque la valeur y de f au temps t dépend non seulement de son entrée x au temps t, mais des précédentes valeurs de f au temps t-1, etc., c'est que cette valeur dépend de toute l'histoire de f. Une simple fonction joue ainsi le rôle de « mémoire », à ce que prétend von Foerster, mais de mémoire sans stockage de l'information, puisque tout se trouve à chaque instant dans l'opération f. Quant à la référence au passé, propriété nécessaire d'une mémoire qui se respecte, elle est donnée simplement, selon von Foerster, par l'emboîtement de la fonction en elle-même. Il serait plus correct de simplement parler d'un comportement dépendant du passé (par exemple une chaîne de Markov à n chaînons), plutôt que d'une mémoire.

Von Foerster passe ensuite à un autre niveau, et il développe son point de vue dans "Molecular Ethology" (1970). Il oppose ce qu'il appelle les machines triviales et les machines non triviales. Une machine triviale est une fonction. Une machine non triviale est tout simplement une machine qui change de mode d'opération en fonction d'une variable. C'est donc une fonctionnelle qui change de fonction, en fonction d'une variable (p. 235). Ashby utilisait une notion similaire, on l'a dit, mais au lieu de vouloir l'utiliser pour passer du trivial au non trivial, il y voyait au contraire un moyen de réduire la richesse de l'auto-organisation à une fonction de la richesse de l'environnement. Son raisonnement avait cependant une faille, puisqu'il sous-estimait la puissance de la récursion. Cette puissance, c'est celle que veut utiliser von Foerster. On

retrouve ainsi notre fonctionnelle qui permettait de définir une liste infinie de programmes, et d'en changer en fonction de la variable qui calculait le numéro de code du programme. Ou encore, on retrouve le rapport entre R(z) et fz. Le numéro de code du programme est choisi en fonction de l'environnement. Selon von Foerster, la machine non triviale peut ainsi faire de l'environnement une machine triviale (une fonction de base), puisque pour tout changement pertinent de l'environnement elle déclenche le programme qui y correspond. En fait, cela revient tout aussi bien à faire de l'environnement le facteur exogène qui permet à la machine non triviale de changer de fonction. Tout au plus peut-on dire, si on voit là une application du théorème de la récursion, que l'environnement est une des valeurs d'entrée de la fonction qui nous donne le numéro de code du programme à choisir.

Sur ce point, von Foerster est curieusement moins optimiste que Kleene. Il nous dit en effet, dans "What is memory", que partir du comportement pour retrouver la fonction est en général difficile (sauf si le comportement se répète pour des séquences répétées d'entrées) et que retrouver les changements de synapses qui implémentent les changements de fonctions (retrouver le f (z)) est impossible, ce qui revient à orienter son épistémologie vers la production d'artefacts au lieu de lui permettre d'expliquer le vivant. Certes, il a raison s'il entend par là soit trouver les opérations effectives qui réalisent la fonction (car il y en a une infinité de possibles) soit trouver ce programme si la fonction n'est pas récursive. Mais le théorème nous dit que si elle l'est, et von Foerster a commencé par le supposer, alors nous pouvons par elle trouver le numéro de code du programme. Reste bien sûr à le faire, et à savoir comment s'y prendre – le plus simple serait que ce code nous donne le programme lui-même – mais ce n'est pas impossible.

Par ailleurs von Foerster propose une différence entre l'expérience dans le passé, qu'il pense donnée par l'emboîtement des étapes de la fonction, et l'expérience présente de cette expérience, qui serait donnée par la fonctionnelle qui pose cette fonction comme la fonction en cours. Il semble qu'il vise par là, sans l'expliciter, une notion de représentation, et de représentation de la représentation. Et précisément, on retrouve cette idée d'une représentation qui n'a pour contenu que sa référence à la relation de représentation même, quand dans "On constructing a reality" (1973), von Foerster utilise la notion de récursion pour aller encore plus loin. Supposons que la cognition consiste en la "computation" d'une réalité. On ne peut faire de computation que sur une description de la réalité (disons, un codage). Mais une description est le résultat d'une computation. Si bien que finalement, on aurait une boucle, la cognition se réduisant à une computation de la computation. On sait quel sort a donné Edgar Morin à ce cercle auto-absorbant dans lequel il voit sa conception de la "récursivité", mais que ses lecteurs ont quelque mal à identifier clairement (*La Nature de la Méthode*, Seuil, 1977).

Jusque là, von Foerster s'était borné à donner des interprétations généreuses de la notion de fonction récursive, puis de celle de fonctionnelle récursive. Ici, il s'engage sans le dire dans le domaine dangereux des extensions métalangagières de la diagonalisation. Comme on l'a vu, celle-ci consiste, par exemple, à appliquer une fonction sur une valeur qui code la fonction elle-même, ou son numéro d'ordre dans une liste. Tant qu'on dispose d'un langage formel, on peut toujours relier à cette idée de codage l'interprétation qu'on donne de cette opération dans des termes non formels, du genre : « la fonction qui opère sur elle-même ». Mais ce pronom réfléchi devient bien plus dange-

reux à manier quand on parle de représentation de la représentation, ou de pensée de la pensée (ce que n'a pourtant pas hésité à faire Dedekind). Cependant, parlant de computation de la computation, von Foerster pourrait s'appuyer sur l'idée du théorème de la récursion. Si calculer, c'est effectuer une fonction f(x), si une description est simplement le résultat de f(x) (voire une fonction, si on est passé au niveau des fonctionnelles), alors il y a bien une fonction f(x) qui dépend du numéro de code du programme, ou un programme qui dépend du calcul de la fonction portant sur son propre numéro de code. On a bien un bouclage. Mais tout d'abord il faut noter que ce bouclage n'est pas total, puisqu'à côté de la variable z, il y a au moins une variable x (voire un ensemble X de variables), si bien qu'on ne peut éliminer l'entrée externe (la « réalité », en suivant la métaphore) comme le fait von Foerster. Ensuite et surtout, si on accepte l'idée que la cognition est une représentation de représentation (une computation de computation), alors on s'ouvre la possibilité de représenter dans le langage des propriétés du calcul lui-même, en particulier le fait qu'il ait des solutions et qu'il se termine. Mais un langage dans lequel ces propriétés sont représentables est un langage qui comporte des fonctions qui sont récursivement énumérables mais non pas récursives, et donc qui conduit à des indécidabilités.

Ce qui est curieux, c'est que von Foerster ait suivi pas à pas la construction de la théorie de la récursion pour développer ses différents étages (machine triviale, machine non triviale, représentation de représentation) mais qu'il situe la difficulté là où la théorie de la récursion nous donne quelque assurance (trouver le numéro de code du programme qui calcule en fonction de son propre numéro de code est possible) et non pas là où cette même théorie la situe (une représentation des propriétés du calcul lui-même nous ouvre le domaine de l'indécidable). Il semble donc que von Foerster, qui était un physicien d'origine, ait bien eu connaissance des travaux de Kleene (de 1938), mais pas du théorème des quatre (de 1961-70), si bien qu'il considérait que la théorie de la récursion offrait des perspectives théoriques positives (il existe un code du programme qui calcule une fonction dépendant entre autres de ce code), la difficulté tenant pour lui seulement à trouver une procédure qui permette de calculer effectivement ce code.

On pourrait tenter de justifier von Foerster en adoptant l'interprétation de Webb, pour qui les théorèmes de limitation montrent justement que les fonctions récursives permettent d'exprimer l'auto-référence, donc l'indécidabilité, si bien que celle ci est mécanisable en un sens. Mais il faudrait aussi reconnaître avec Webb que cette mécanisation nous conduit à l'indécidabilité justement quand les fonctions en cause ne sont que récursivement énumérables et non pas récursives. L'interprétation de Webb voit donc dans les résultats d'indécidabilité simplement l'envers du théorème de la récursion. On pourrait admettre ce parallèle avec Webb, si justement von Foerster avait reconnu la positivité de ce théorème, ce qu'il n'a jamais fait explicitement.

Au delà de la récursion ?

Si on en vient à Maturana et Varela, on va rencontrer un usage un peu différent du terme « récursif ». Dans l'ouvrage de Francisco Varela, *Principles of biological autonomy*, par exemple p. 86, « récursif » est d'abord simplement un terme qui désigne l'existence, dans un système, de relations internes entre sous-systèmes, au lieu de se borner à des relations entre le système et l'environnement. Ces relations sont en boucles, si bien qu'on retrouve la notion

de feed-back. Mais « récursif » est utilisé aussi de façon plus précise et ambitieuse. Dans le chapitre 12, Varela propose de considérer des « réentrées » d'une forme (dans le calcul des « formes » proposé par Spencer Brown). Il reprend à Spencer Brown l'idée que d'un calcul des formes. Une forme a ici pour symbole un gnomon, et on peut emboîter un gnomon dans un autre, ce qui revient à une sorte de négation de la négation, ou bien les faire se suivre, mais alors leur réitération revient à un seul gnomon. On peut encore emboîter plusieurs gnomons à la suite dans un autre, etc. Or, ajoute Varela, si une forme consiste en un emboîtement de symboles, "abab", elle peut réentrer en elle-même si cet emboîtement est répété à l'infini. On considérera alors que la valeur de "ab" est moins déterminée que celle de "abab", qui est moins déterminée que celle de "abababab", etc. à l'infini. Si bien que la valeur d'une réentrée est la limite de cette forme de réentrée quand son emboîtement tend vers l'infini. Le problème est que dès qu'on admet que toute forme peut réentrer en elle-même, et que l'on considère cette valeur limite comme le point fixe de la réentrée (puisqu'alors, la réentrée ne modifie plus la valeur de la forme), toute forme est à elle-même (en tant que réentrante) son propre point fixe. Dès lors la distinction entre le décidable et l'indécidable n'est plus si tranchée, puisque les deux sont des points fixes, le décidable étant le point fixe du prédicat « démontrable » et l'indécidable le point fixe du prédicat « non démontrable ». La théorie de la récursion était sous estimée par la première et la seconde cybernétique, mais ces troisièmes cybernéticiens font un saut au delà d'elle.

Qu'en est-il enfin des connexionnistes ? On sait qu'ils achoppent sur le problème d'apprendre à un réseau à traiter des structures symboliques en conservant leurs propriétés structurelles. Le réseau associe, mais il mélange, et perd ainsi la trace des différences structurelles. Plusieurs dispositifs avaient été imaginés dans les années 90-95 pour permettre aux réseaux de retrouver par décomposition ce qu'ils avaient commencé par composer. En particulier avait été proposée la notion de mémoire auto-associative récursive. Elle consistait à partir d'une couche d'inputs, à les compresser dans une couche d'unités cachées, puis à les décompresser dans une couche de sortie. L'algorithme de rétropropagation consistant à répercuter les corrections d'erreur de la couche de sortie sur la couche cachée, on apprenait ainsi à cette couche cachée à opérer des compressions qui puissent redonner la forme décompressée, conservant ainsi sa structure.

On a ensuite perfectionné la procédure, d'abord en introduisant des boucles de récurrence dans un tel réseau. Soit la couche d'unités cachées renvoie ses états vers une partie des unités inputs, soit c'est la couche de sortie qui opère le même renvoi. Le but d'une telle récurrence est bien entendu de faire dépendre l'opération en cours du résultat de l'opération précédente (on retrouve notre emboîtement récursif à la von Foerster). Mais il est aussi et surtout de tenir compte des effets de contexte entre les symboles donnés en inputs, c'est-à-dire des relations pertinentes entre tel symbole qui apparaît sur la fenêtre de traitement de ce que le réseau prend comme entrée, et tel symbole voisin, mais hors de la fenêtre de traitement, et qui ne sera pris en compte que dans l'étape suivante, alors que le premier symbole ne figurera plus dans la fenêtre d'entrée.

Puis on a mis en l'oeuvre l'idée suivante. Analysons par des méthodes statistiques les distinctions actives dans les unités cachées. Construisons à partir de cette analyse les graphes d'automates à états finis. Plus l'analyse (par clusters) sera fine, plus le graphe sera complexe. Puis nous faisons tourner ces automates à états finis sur les nouveaux exemples à traiter. Ou encore, nous

pouvons extraire les règles propres à ces automates à états finis, et les encoder de façon approximative dans le réseau (en abaissant systématiquement certains poids de connexions, et en en remontant systématiquement certains autres, ce de manière compatible avec le traitement de telle entrée par l'automate). Les performances sont meilleures qu'avant dans les deux cas.

Qu'en était-il de ces manoeuvres par rapport à la théorie de la récursion ? Certes, comme l'ont montré Bochereau, Bourgine et Deffuant, il suffit de mettre en sortie d'un réseau une fonction de décision qui envoie le résultat soit sur 1 soit sur 0 pour avoir transformé le réseau en un classifieur, qui est une machine logique. On a ainsi une équivalence entre réseau et machine logique. Le problème, c'est qu'on ne sait pas quelle est la machine logique dont le réseau est l'émulation. Or c'est nécessaire de le savoir, pour s'assurer si c'est bien une fonction récursive, et non pas seulement récursivement énumérable, et pouvoir appliquer le théorème de la récursion. Mais, comme le rappellent les mêmes auteurs, le problème de l'extraction des règles (le problème que traitent de manière approchée ceux qui extraient des automates à états finis des réseaux) est NP-complet - il n'existe pas d'algorithmes qui puisse résoudre ce genre de problème en un temps borné par un polynôme de degré donné. Sans doute, on sait que les réseaux à couches (ou même à forme « buissonnante ») approximent des fonctions qui ne présentent pas des ruptures de pentes trop importantes (les fonctions « lipzitschiennes »). Mais justement ils n'en sont que des approximations. Hornik a montré que si la fonction d'activation du réseau est continue, bornée et non constante, la distance entre les valeurs de la fonction calculée par le réseau et les valeurs de la fonction réelle peut être rendue arbitrairement petite si on peut toujours ajouter de nouvelles unités cachées au réseau[3] . Mais pour cela, il faut que les fonctions d'activation soient non polynomiales[4]. La stratégie d'extraction des automates se heurte aussi à un problème : la performance réelle (en fait virtuelle, sur d'autres bases d'exemples) n'est celle des automates à états finis qu'à la limite, et elle comprend celle de tous les automates, en incluant ceux qui ont des résultats inconsistants avec la base d'apprentissage, c'est-à-dire qui ne donnent pas les sorties souhaitées. Bien entendu, on va se restreindre aux automates consistants avec les sorties souhaitées, et même on va préférer le plus petit de ces automates. Mais c'est là une décision pragmatique, dont on ne connaît pas les effets.

Au total, on ne peut pas dire si un réseau calcule telle fonction. Il l'approche. Évidemment cela implique (comme en principe les variations des poids des connexions peuvent être continues) qu'il existe une infinité de réseaux qui approchent cette fonction (mais certains d'eux pourraient aussi être dits en approcher d'autres). Or une fonction « approchée », dans sa singularité, n'est justement pas, en général, une fonction récursive. A fortiori, on ne peut rien dire de la séquence de fonctions ou de la fonctionnelle que pourrait approcher un réseau sous différents apprentissages, sous différentes évolutions. Les réseaux connexionnistes vont donc au delà des fonctions récursives, alors même qu'ils sont aussi en deçà, puisqu'ils ne garantissent jamais totalement le respect de la structuration des symboles nécessaire à la récursivité.

---

[3] Hornik Kurt, "Approximation Capabilities of Multilayer Feedforward Networks", *Neural Network*, Vol. 4 pp. 251-257, 1991, p. 252.

[4] Hornik Kurt, "Some new results on Neural network Approximation", in *Neural Networks*, Vol. 6, pp. 1069-1072, 1993, p. 1070.

Yves-Marie Visetti a cependant fait remarquer que le but des connexionnistes, pas plus que ne l'était celui de von Foerster, n'est de garantir que les réseaux retrouveront des fonctions que nous pouvons identifier par d'autres moyens, ni de servir eux-mêmes comme moyens pour identifier des classes de fonctions (celles qu'ils permettent d'approcher et les autres, celles que les réseaux à trois couches et sans récurrence permettent d'approcher, celles que les réseaux à trois couches et avec récurrence approximent, etc.). Le but des connexionnistes est simplement de produire de la complexité, donc de ne pas se borner à des fonctions qui décrivent un espace borné, mais de déplacer les bornes tout en développant la fonction, pour découvrir de nouveaux espaces. De même von Foerster n'avait pas pour ambition ni pour souci de relier les comportements de ses fonctions récursives à une théorie de la récursion qui permettait de savoir d'avance dans quel espace de fonctions il se déplaçait, mais il voulait tout simplement produire des comportements complexes.

C'est bien une tendance du connexionnisme, mais il ne pourra devenir autre chose qu'une technique de simulation de la complexité que si l'on peut utiliser les réseaux pour mettre à l'épreuve des classifications mathématiques, donc si on se repose la question de savoir quelle classe de fonctions tel type de réseau peut approximer (et non pas vraiment calculer). Bien des travaux vont depuis 10 ans dans ce sens, mais bien d'autres se bornent simplement à combiner des réseaux avec diverses formes de systèmes symboliques, de manière à pallier les difficultés de stabilité catégorielle des réseaux.

Finalement, les premiers cybernéticiens aussi bien que leurs descendants successifs ont toujours tourné autour de la théorie de la récursion, que ce soit en deçà ou au delà, tout en étant toujours en déphasage avec elle. Il devient maintenant nécessaire de ne plus désigner la complexité comme un horizon ou comme des lendemains qui chantent. Nous ne pouvons pas l'étudier directement, puisqu'elle se définit par ce qui échappe à une analyse effective. Mais les travaux d'analyse des outils artificiels comme les systèmes connexionnistes ont pour but de relier certaines formes complexes à des catégorisations mathématiques, pour nous donner des repères d'exploration de ce domaine que nous savons ne pas pouvoir épuiser.

Références

Ashby, W.R. (1962). Principles of the self-organizing system. in H. von Foerster et G.W. Zopf (eds.), *Principles of self-organization*. Pergamon Press: 255-278.

Bell J. L. et Machover M., 1975, *A course in mathematical logic*, Amsterdam, New York, North Holland : 259, 269, 273-5, 282, 313.

Bochereau, L., Bourgine, P., Defuant, G. (1991). Équivalence entre classificateurs connexionnistes et classificateurs logiques. *Intellectica*, 1991/2, 12 : 139-158.

Elman, J.L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7: 195-225.

Giles C. Lee, Omlin Christian. W. (1993). Extraction, Insertion and Refinement of Symbolic rules in dynamically driven recurrent neural networks ", *Connection Science*, Vol. 5, n° 3 et 4 : 307-334.

Hornik Kurt, 1991, " Approximation Capabilites of Multilayer Feedforward Networks ", *Neural Network*, vol 4 : 251-257.

Hornik Kurt " Some new results on Neural network Approximation ", in *Neural Networks*, Vol. 6, pp. 1069-1072, 1993, p. 1070.

Kwasny, Stan C., Kalman Barry L. (1995). Tail-recursive Distributed representations and Simple recurrent networks. *Connection Science*, vol 7, n°1: 61-80.

Pollack, J. (1990). Recursive distributed representations. *Artificial Intelligence*, 46: 77-105.

Rosenblueth Arturo, Wiener Norbert, Bigelow Julian, (1943). Behavior, Purpose and Teleology. *Philosophy of science*, vol. 10, January: 18-24.

Varela, Francisco (1979). *Principles of biological organization*. Amsterdam, New York: North Holland.

von Foerster, Heinz (1969). What is memory that it may have hindsight and foresight as well. *The Future of brain sciences*, Bogoch ed, Plenum Press: 19-64

von Foerster, Heinz (1970). Molecular ethology, an immodest proposal for semantic clarification. in *Molecular mechanism in memory and learning*, G. Hungar ed., Plenum Press: 213-248,

von Foerster, Heinz (1973). On constructing a reality. in *Environmental design research*, vol. 2., F.E. Preiser ed. Dowden, Hutchinson and Ross, Strousburg : 35-46.

Webb, Judson C., *Mechanism, Mentalism, and Metamathematics: an essay on Finitism*, Reidel, Dordrechte, 1980.