Réflexions sur l'IA

L'histoire à reculons

Bacon et AM/Eurisko = 70/80. Depuis, rien.

Dans la ligne fixée en 56/57, Logic Theorist, GPS.

56/57 : il fallait le faire. C'étaient les idées les plus "naturelles", le "chemin le plus court". Un cadre était fixé (en fait avant par Turing), il fallait le « remplir ». Et "il fallait le faire", i.e. ce travail est remarquable.

Mais ensuite, il fallait examiner si les idées étaient bonnes. C'est aujourd'hui seulement qu'on le fait, après des années de "sommeil". Sommes-nous assez "adultes" pour nous réveiller ?

Origine de l'IA : en même temps que l'informatique. Schizophrénie apparente de l'informatique : logique appliquée (fonctions récursives générales, machine de Turing) et premier modèle cognitif de l'homme (cf. papier de Turing 36/37).

Papier de Turing 47/48 (probablement publications dès 1940 en Angleterre, mais je ne les connais pas, et c'était la guerre, Enigma, etc.). Robotique ? C'est ce qui serait de mieux, mais trop compliqué pour des raisons techniques. Deux domaines d'investigation pratiques : cryptographie et échecs. Déjà min-max, déjà "bidouille" de la fonction d'évaluation.

Remarque : Von Neuman = automates auto-reproducteurs, comparaison ordinateur/cerveau.

Donc : "double vue" logique et cognitif. Mais "paradigme" fondateur commun = machine de Turing, avec son modèle abstrait = théorie du calcul.

Remarque : Dès le début, "programmation automatique".

= FORTRAN (1957/58)

= Apprentissage

1958 : Gödel : "Une erreur philosophique de A.M. Turing". Augmentation du nombre d'états de la machine de Turing par des procédés "non mécaniques" et "parfaitement mécanisables".

"Thèse" de Church : Incroyable prétention. Dogme. ("Désir" puissant de la grande découverte non américaine, "enfin !" ?).

Nous sommes aux racines

Problème essentiel : comment un programme peut-il augmenter sa propre complexité ("apprentissage", ...) ?

Impossibilité dans le cadre "calcul" : Chaitin. Un programme ne peut pas faire plus que sa propre complexité.

Spéculations

1/ Différence machine de Turing/Calcul. Pour Gödel, machine de Turing universelle = découverte fondamentale, car "machine incarnée".

Mais calcul = seulement la moitié des machines de Turing : celles qui s'arrêtent !

Et puis, "états" dans la "tête" de la machine de Turing (i.e. instructions du programme)

invariables. Ou si l'on veut, système fermé. Or les systèmes fermés n'existent pas dans la nature - sinon de façon approximative et pendant un délai "court".

Donc deux "pistes" : machines qui ne s'arrêtent pas ("bouclent") et machines dont les instructions peuvent être changées.

2/ Changer les instructions : Si les instructions ne sont changées que par la machine elle-même, on en revient à la conception "fermée" (en ayant ajouté un niveau d'interprétation). Donc les instructions doivent être changées *par l'environnement*. Système *ouvert*.

Programme et données : Dans la machine de Turing, le contenu du "ruban" (i.e. les données) peut éventuellement provenir de l'extérieur (mais en général de façon "synchrone"). Autrement dit, monde extérieur = données. "Interagir" avec le monde = échanger des données.

En fait, actions du monde = instructions. C'est le contenu de la tête de la machine qui change, tout autant que le contenu du ruban. Pas de différence programme/données, jusqu'au bout.

Remarque : Il est courant de dire en informatique qu'il n'y a pas de différence programme/données. Mais cela signifie :

"Pour tout couple Prog/Data, on peut mettre une partie de Prog dans Data, i.e. il existe Prog'/Data'=f(Prog,Data) ayant le même comportement (i.e. quand les deux machines s'arrêtent, le contenu de leurs rubans sont isomorphes via un isomorphisme prévisible à partir de f). A la limite, Prog' = machine universelle."

Ici, on entend quelque chose de différent. Le programme n'est pas "coupé du monde", invariable, mais est susceptible de changements imprévisibles.

Remarque bis : Dans « notre » cadre, changer (de façon asynchrone) le programme ou les données, c'est pareil. On peut donc garder la machine universelle "invariable". Autrement dit, *la machine de Turing est sans doute "plus" que son modèle théorique, i.e. le calcul*. On peut aussi remarquer que les ordinateurs sont souvent utilisés pour des programmes qui ne s'arrêtent pas !!!

Remarque ter : La machine de Turing est de la "logique réifiée", translatée du monde platonicien des idées vers le monde physique. Mais le calcul arrête cette progression à mi-chemin : sa tête est "coupée du monde". Ici, on plonge la tête de la machine de Turing - le programme - dans le vaste monde.

3/ Changer les instructions ==> Abandonner l'arrêt.

Si l'on change au hasard une instruction dans une machine qui doit s'arrêter, il y a toutes les chances pour qu'elle ne s'arrête plus. Il faut donc *abandonner l'arrêt*. Et il faut donc *abandonner le calcul*.

4/ Que reste-t-il ? Nous sommes bien embêtés, car une théorie scientifique commence par considérer une classe (infinie) d'objets (ici, machines de Turing), puis doit distinguer une sous-classe stricte non vide (infinie) d'objets "intéressants". Comme les Indiens et les cow-boys : il y a les bons et les mauvais. Mais savoir qui sont les bons et qui sont les mauvais change selon les époques.

Les machines qui s'arrêtent constituent une telle sous-classe. Or, on veut regarder des machines qui

ne s'arrêtent (potentiellement) pas. Alors qui sont maintenant les « bons » ?

Remarque : En fait, on n'aime carrément plus les machines qui s'arrêtent, car comment feront-elles pour continuer à se modifier ? Heureusement, le "monde extérieur" est encore là, qui un jour ou l'autre modifiera quelque chose dans la machine et la "relancera".

5/ Autre point de vue : syntaxe et sémantique. La classe des machines de Turing, c'est de la "syntaxe". La classe des machines de Turing qui s'arrêtent, c'est de la "sémantique". Remarque : Très différent du sens du mot "sémantique" habituellement appliqué à un programme. Le "sens" d'un programme qui calcule le pgcd, ce n'est pas qu'il calcule le pgcd, c'est qu'il s'arrête. *On devrait se moquer de ce que fait un programme*.

Mais avec des machines qui s'arrêtent (obligatoirement) on *ne peut pas* ne pas se demander dans quel état est la machine à l'arrêt.

Changement radical de point de vue là aussi, dans la réponse à la question "que fait votre programme" ? *Fin du finalisme*.

6/ Quelle sémantique de substitution ? Les Indiens deviennent les "bons". Une machine intéressante *est justement celle qui ne s'arrête pas*.

Oui, mais un programme invariable qui ne s'arrête pas, c'est facile à faire. Mais *quid d'un programme changé de façon imprévisible par son environnement* ?

Il faut donc que le programme assure une sorte de "permanence" indépendamment, ou plutôt malgré les actions de l'environnement. Evidemment, cela n'est pas formel et a des limites : si l'environnement supprime d'un coup le programme tout entier, ok, c'est fini. Mais si l'environnement n'est pas aussi "destructeur" (ou "chaud"), alors le pg peut peut-être "tenir".

7/ Considérons un programme pg plongé d'une manière ou d'une autre dans le vaste monde, et un changement causé par ce vaste monde. Pour simplifier, deux cas : a/ quelque chose est *ôté* au programme; b/ quelque chose est *ajouté* au pg. Evidemment, il faudrait définir ce qu'est "quelque chose", "ôté", "ajouté", et considérer des cas généraux (et considérer aussi la "synchronisation" des actions du pg et de celles du monde, et les "lois physiques" du monde, i.e. dans quelles limites ces changements peuvent intervenir).

a/ Ce qui serait bien, c'est que le pg reconstitue *par son action* l'élément enlevé. Un bon moyen pour y arriver, *c'est qu'il reconstitue* ***en permanence tous*** *ses éléments*.

C'est la définition (Varella) d'un système autopoïétique, i.e. *qui se produit lui-même*.

Attention ! Il ne s'agit pas de duplication ou de reproduction. Le pg ne crée pas un *autre* individu qui lui ressemblerait. Simplement, ses constituants se produisent mutuellement (métabolisme). C'est une définition du vivant.

Remarque : Cela fait du pg justement un *individu*, en un sens beaucoup plus fort que dans le cadre du calcul. Dans ce dernier, c'est nous qui disons qu'il s'agit d'un programme. Il "est" programme dans le même sens qu'une pierre "est" (ou même qu'un verre d'eau). Ici, un programme "est" par lui-même, au même sens qu'une cellule ou tout autre *être* vivant. Pb de l'être (ontologie).

b/ En cas "d'ajout d'élément", deux solutions : le nouvel élément produit une chaîne de

“réactions métaboliques” qui ne “bouclent pas”. Après quelque temps, le pg redevient identique à ce qu’il était auparavant. Autrement dit, le nouveau “sous-programme” s’arrête ! Les pgs qui s’arrêtent s’excluent donc d’eux-mêmes. Ou bien on aboutit à une “boucle”, et donc à un métabolisme *augmenté*.

Evidemment, cette augmentation eut aussi conduire à des suppressions (inhibitions). Dans ce cas, c’est tout le métabolisme qui est modifié. On pourrait d’ailleurs voir le pg “mourir”.

Mais, intuitivement, la probabilité que le pg meure doit être très faible. *L’autopoïèse n’est pas l’exception, c’est la règle*.

Quelques “expériences” papier très limitées. Mais je ne sais plus où j’ai fourré mes notes.

8/ Retour sur la “sémantique”. Il est habituel de dire que tel programme (censé s’arrêter) calcule telle ou telle quantité (par exemple le pgcd). Autrement dit, on se contrefiche de la *forme* du programme, seule sa *fonction* (son *contenu*) compte. Que le pg soit écrit en assembleur, en Maciste ou en Descartes, c’est pareil.

Mais avec des pgs qui changent, la forme a, comme dans la chanson de Reggiani, *une certaine importance*.

On voit ce pb dans le cadre actuel comme *négatif* : c’est difficile de changer un pg C, par exemple, car on a toutes les chances d’obtenir un pg non compilable. Il faudrait de nouveaux langages, *asyntaxiques* (n’importe quoi est exécutable). Remarque : à l’opposé de la “tendance Descartes”, où les pgs sont de plus en plus “incompilables”, notamment en y cherchant beaucoup plus les “erreurs”. C’est volontaire dans Descartes : pousser le “paradigme” actuel aux limites.

Mais c’est en fait *positif*. Par exemple, on dit souvent que deux pgs font la même chose, et on en supprime un des deux. Mais s’ils doivent changer, les changements appliqués sur l’un ne donnera sûrement pas la même chose que les mêmes changements appliqués sur l’autre ! Voilà par exemple un effet positif de cette “redondance de la fonction”. Une même fonction incarnée simultanément dans plusieurs programmes distincts aura toutes les chances de ne pas être supprimée par un changement donné. L’autopoïèse est *robuste*. Cf. par exemple les codons du patrimoine génétique : un de leur “usage” ne serait-il pas de “résister” aux mutations ?

Cette conception de “la fonction d’abord” est ancrée dans toute notre vision “technique” ou scientifique” du monde. On considère une machine à vapeur ou un moteur électrique comme équivalents, mais on n’imagine pas une machine à vapeur se transformant en aspirateur (ou en réfrigérateur, i.e. en plus en quelque chose dont la fonction utile potentielle - produire du froid - est à l’opposé de “l’effet de bord” de la machine à vapeur - produire du chaud).

On ne peut pas faire autrement que de dire “à quoi ça sert”, même en biologie, même pour l’évolution des espèces. Le rapport entre forme et contenu est impossible à comprendre aujourd’hui (en particulier c’est un rapport discontinu : un petit changement de forme peut produire un grand changement de fonction).

9/ Faut-il un nouveau “modèle d’automate”, et si oui lequel ?

a/ Oui, les pgs doivent “s’appliquer à eux-mêmes”. Sinon, on retombe sur les pgs fixes, il y a quelque part un pg qui ne change pas.

b/ Les pgs ne sont pas “en dehors du monde”. Dans la machine de Turing, les données (le ruban) sont “dans le monde”, i.e. susceptibles de changement. Dans le modèle classique, seule la tête de lecture peut les changer, mais il est facile d’imaginer qu’il en soit autrement. Par contre, les pgs de la tête apparaissent “coupés du monde”. Ils sont “idéaux”, dans le sens de non plongés dans le monde matériel. Il faut “*plonger les pgs dans la matière*”, et les soumettre à ses lois.

c/ Cependant, paradoxalement, cela n’interdit pas pour l’étude d’utiliser une machine de Turing fixe, notamment une machine universelle (un ordinateur). En effet, sa propriété est de pouvoir interpréter n’importe quelle machine décrite de façon appropriée sur son ruban. “Laisser le monde changer les pgs” revient donc à rendre possible les changements sur le ruban d’une machine universelle. Ces changements pourront intervenir aussi bien dans la “zone du programme” que dans la “zone des données”.

d/ Cependant, une machine de Turing universelle donnée (i.e. un interpréteur) doit pouvoir s’y reconnaître dans le contenu du ruban : la machine interprétée doit être *codée* d’une certaine manière. Ou, si l’on veut, le contenu du ruban doit suivre un certain langage. Or, il est possible de définir des langages sans syntaxe : n’importe quelle chaîne de caractères sur le ruban est “interprétable”. C’est en fait évident, les restrictions de plus en plus fortes qu’apportent les langages de programmation sont des limitations visant à aider les humains à trouver les erreurs qu’ils commettent inévitablement en écrivant leurs programmes (qui doivent calculer une certaine fonction).

e/ Une autre idée est de définir des programmes (“programmes-molécules”) mis ensemble dans un bécher, et de laisser réagir selon un protocole à définir. Chaque programme est une chaîne de caractères, et travaille sur une chaîne de caractères pour la modifier. Chaque programme s’applique donc aux autres programmes vus comme chaînes. On laisse “tourner”, et il est intuitivement très probable que des cycles se forment. L’intervention du monde extérieur consiste en l’ajout et/ou le retrait de molécules du bécher.

f/ La machine universelle unique dont le ruban est soumis aux modifications du monde, ou les “programmes-molécules” sont nécessairement équivalents (du moins en faisant des hypothèses sur les changements extérieurs possibles). Mais une “vision” est peut-être plus parlante, et surtout plus propice à l’expérimentation que l’autre. Notamment l’autopoïèse est plus visible avec les programmes-molécules.

Remarque : Les réseaux de neurones, ou mieux, les algorithmes génétiques semblent répondre aux propriétés ci-dessus. Il y a cependant une différence majeure : dans les réseaux de neurones, les connexions ne changent pas, or elles constituent *in fine* le programme. En particulier, on ne peut probablement pas dépasser une complexité maximum avec un réseau fixe dont seuls les poids sont variables. Dans les algorithmes génétiques, les “gènes” sont en fait des morceaux de données, ayant souvent une taille maximum figée, et n’agissent pas comme programmes. Des recherches récentes ont changé cela (la structure du réseau peut changer, ou les “gènes” agissent comme pgs) avec quelquefois des résultats impressionnants. Un exemple célèbre (mais peut-être finalement moins impressionnant) est *Tierra*, une simulation de l’évolution. Les gènes sont des programmes auto-reproducteurs - au sens de Von Neuman - qui agissent sur les autres programmes pour les changer. Les programmes produits ne sont pas forcément auto-reproducteurs, mais s’ils ne le sont pas ils sont éliminés. Ce qui paraît incroyable à première vue, c’est que de nouveaux pgs auto-reproducteurs (et même plus compacts que les pgs initiaux) sont produits. Le nombre de caractères des programmes-chaîne de caractères en question oscille entre 21 et quelques centaines. Une critique approfondie de Tierra serait intéressante.

10/ Retour sur la complexité. Les définitions données ici ne sont pas exactes.

La complexité d'une chaîne selon Chaitin est définie comme étant le log de la taille du plus petit programme produisant cette chaîne. C'est une généralisation de l'*information* au sens de Shanon. La quantité d'information d'une chaîne selon Shanon consiste moralement en la taille de la plus petite chaîne la décrivant, avec le code approprié. Par exemple, la quantité d'information du *nombre* 100 est moralement 3, car il suffit de 3 caractères pour le décrire en base 3. Chaitin étend cette notion en considérant qu'au lieu de transmettre une chaîne décrivant directement l'information désirée, on peut transmettre un programme (i.e. une autre chaîne), le langage de programmation (ou si l'on veut son interpréteur) étant fixé à l'avance. Cela diminue la complexité pour beaucoup de chaînes. Par exemple, la chaîne constituée de 10^{100} fois le groupe "01" a une complexité au sens de Chaitin très faible : un tout petit programme permet de la reconstituer.

Outre des résultats liés à l'indécidabilité de la résolution des équations diophantiennes (démontrée par Matujevic en 1971), notamment en produisant (par programme !) une équation "presque" diophantienne "contenant" à elle seule cette indécidabilité, mais avec plusieurs milliers de variables, Chaitin a "réinterprété" les résultats de Gödel en terme de complexité. Moralement, un programme ne peut pas plus que sa propre complexité. (Plus précisément, un programme-démonstrateur de théorèmes d dont le but est de trouver la complexité d'une chaîne donnée ne peut pas le faire pour des chaînes de complexité "substantiellement" supérieure à sa propre complexité k. On utilise le mot "substantiellement", car ces théorèmes sont valables asymptotiquement pour k grand.)

Chaitin a aussi des résultats sur la nature aléatoire des chaînes de complexité maximale (par rapport à leur taille).

Il faut noter que l'on ne parle ici que de programmes *s'arrêtant*.

Néanmoins, si l'on a pour but de faire des programmes se modifiant, qu'ils s'arrêtent ou pas, on a dans la tête une notion informelle de complexité : on voudrait que les programmes résultants soient *plus* que les programmes initiaux. Mais on a en fait l'espoir "fou" qu'il n'y a pas de limite à cette augmentation, notamment que les programmes produits pourront à un moment donné dépasser la complexité des programmes initiaux.

Or, dans la théorie de Chaitin, qui se situe elle-même dans le cadre de la théorie du calcul, cela est impossible. Il n'y a pas à tourner autour du pot : c'est un fait établi. C'est pour cela qu'il faut sortir du cadre du calcul.

Mais alors finalement, d'où peut venir cette complexité supplémentaire ? *Du monde*. On peut avoir une vision "naïve", on l'on modéliserait le monde comme une gigantesque machine de Turing. (L'idée n'est peut-être pas aussi folklorique : des physiciens se sont posés - pour s'amuser ? - la question de la calculabilité des lois physiques connues. Je ne sais pas pourquoi, mais j'ai entendu dire qu'en définitive elles sont toutes calculables. A creuser. A noter aussi que certains - Penrose, par exemple - appellent à la constitution de nouvelles lois physiques "non calculables", seul moyen selon eux d'expliquer la nature "non calculable" de l'esprit humain !). Cette "machine-univers" a une complexité donnée, et il n'est donc pas possible (dans notre pauvre petit univers !) de la dépasser. Mais quelle doit être sa valeur ! Certainement plus que celle des petits systèmes naturels ou artificiels s'ébattant à la surface de notre planète.

On a donc là un *gigantesque réservoir de complexité*. Un système plongé dans le monde, s'il n'est pas étanche, peut donc probablement en intégrer au moins une petite partie.

Mais évidemment, le problème, c'est de *discipliner* cet "emprunt". On en est donc revenu au point de départ.

11/ La mémoire. Si l'on essaie (très naïvement pour ma part) de catégoriser les propriétés de l'esprit humain, la mémoire est une "fonction" très importante.

L'ordinateur fournit un "modèle" élémentaire de la mémoire : c'est le contenu du ruban. Mais on ne peut s'empêcher de voir ce modèle comme très primitif. En effet, un bit de nos disques durs est censé resté là indéfiniment inchangé tant qu'aucune action d'un programme ne vient le faire. Peut-on raisonnablement penser qu'il en est ainsi dans la nature ?

Faisons un petit exercice de programmation. Supposons qu'un processus diabolique et inévitable efface périodiquement (disons toutes les secondes) la mémoire de notre ordinateur. L'effacement est progressif : c'est un processus ininterrompu qui "balaie" toute la mémoire puis revient au départ. Comment faire tourner un programme malgré ce petit démon ?

Ce problème me semble aujourd'hui insurmontable. Il faudrait en effet que le programme reconstitue en permanence l'état de sa mémoire, évidemment à partir de celui-ci. On est proche de l'autopoïèse. Evidemment, une grande partie de l'activité du programme va être dévolue à cette tâche. A la limite, il ne fait que ça.

Une solution simple serait que ses "informations utiles" soit copiées 10 fois, par exemple, dans 10 secteurs de la mémoire de l'ordinateur. Il suffirait alors qu'il recopie ces secteurs de façon synchrone avec le démon. Mais si le démon devient irrégulier, imprévisible (dans certaines limites) ?

Un autre moyen, meilleur, serait de reconstituer la mémoire, non pas en la copiant comme un ensemble de données, mais en la *synthétisant* à partir de programmes. Finalement, la mémoire ne contiendrait que ces programmes. Donc mémoire = programmes permettant de constituer la mémoire (eux-mêmes). Un "élément de mémoire" (si cela doit avoir un sens), disons une chaîne de caractères, serait donc *à la fois* un *morceau de données* mémorisant selon un code donné un "fait", par exemple que toutes les Anglaises sont rousses (sigh...), et *comme programme* une partie de la "mémoire en tant que programme" permettant justement de reconstituer la mémoire. Ce même élément peut avoir les deux "aspects", i.e. les deux "sens", car il est interprétable par des programmes distincts : l'un, un programme qui "analyse" les "faits" mémorisés (pour l'action, par exemple), et l'autre, qui est justement celui qui reconstitue la mémoire. On aurait donc une sorte de "point fixe" fabuleux où une même chaîne représente deux choses distinctes pour deux programmes différents (en fait, elle est elle-même programme, mais a des fonctions distinctes selon le cycle d'autopoïèse où elle impliquée).

Le mécanisme par lequel tout cela peut s'opérer n'est pas du tout clair. Le mécanisme essentiel est celui de la mémorisation : quand un "fait" a été affirmé suffisamment de fois (i.e. quand une chaîne le décrivant a été introduit dans le système mémoire), comment se fait-il que, par son mécanisme propre, un "fait" équivalent fera finalement partie des cycles d'autopoïèse de la mémoire ?
"Affirmé suffisamment de fois" ne doit probablement pas être pris au pied de la lettre, on doit aussi avoir quelque chose comme "donne de bons résultats". On peut aussi voir là la source du "réveil de souvenir" : un nouveau "fait" ("ma voiture est grise"), en tant que nouveau programme du système-mémoire, "réveille" par son action d'anciens cycles. Le problème est alors la rétroaction permettant des corrélations pertinentes entre ces "faits" liés par des cycles communs (pourquoi le fait que ma voiture soit grise me fait-il penser à la rousseur des Anglaises ?).

La mémoire est donc probablement un système autopoïétique particulier, avec certaines propriétés universelles.

Remarque : Quand on parle mémoire, on pense mémoire humaine ou mémoire d'ordinateur. On entend aussi souvent l'expression " mémoire de l'espèce" (ou "de l'individu") pour le code génétique. Mais - c'est ce que j'ai compris - on entend par-là ce qui permet à l'individu de croître à partir de l'ovule fécondé, ou à son métabolisme de se poursuivre. Par ailleurs, c'est le *patrimoine* transmis (partiellement) à sa descendance. Mais ce "patrimoine" a été légué par ses ancêtres, et constitue donc la réserve concentrée des "bons coups" antérieurs de l'évolution, qui lui ont justement permis d'exister. En ce sens, les gènes constituent une mémoire de l'évolution, et donc probablement pas seulement sujets passifs des mutations, mais aussi sélecteurs - limités - des "bonnes" mutations. Une mutation est comme un "fait" introduit dans le système génétique, qui va pouvoir "réveiller" d'anciens cycles, i.e. d'anciens bons coups potentiels. Autrement dit, les gènes ne sont passivement soumis à des mutations au hasard, mais constituent aussi une mémoire des bons coups évolutifs précédents.

12/ Quelles expériences tenter ?

Question très difficile, car on sait à peine de quoi on parle. Une chose est sûre, les "anciennes" idées (résolution de problèmes, démonstration de théorèmes, échecs, etc.) doivent être reconsidérées. Il faut une réflexion supplémentaire (abstraite) sur les nouvelles machines que l'on veut construire pour voir "ce qu'elles peuvent résoudre simplement".

Il y a un paradoxe apparent dans cette question. La nature des systèmes évoqués ici implique que poser la question "à quoi servent-ils" est absurde. Il servent à eux-mêmes. N'empêche, un système "raisonnable" *fait* quelque chose, même si c'est "malgré lui", ou "par effet de bord", ou "dans l'œil de l'observateur". Il faudrait plutôt se poser des problèmes très élémentaires exemplifiant les propriétés des systèmes qu'on veut construire. C'est d'ailleurs l'attitude du scientifique opposée à celle de l'ingénieur : le scientifique a des solutions et ne connaît pas les problèmes qu'il peut résoudre avec, l'ingénieur a des problèmes qu'il ne sait pas résoudre.

Aujourd'hui, on n'a ni problème ... ni solution.

13/ Changement de paradigme ?

Si l'on regarde l'épistémologie, on voit souvent d'une manière ou d'une autre la notion de "révolution scientifique". Pour Koyré, il s'agit d'un changement de "conception du monde", pour Kuhn d'un changement de "paradigme". En particulier, de nouveaux principes sont substitués aux anciens (même si les anciens peuvent subsister comme cas particulier ou approximation). Par ailleurs, toute science est fondée sur des principes.

Pas la peine de se le cacher : l'IA et l'informatique dans leur réalité (sinon dans leur intention) ne font qu'un, i.e. sont fondés sur les principes du calcul. Mais fort heureusement, cette théorie a produit un objet matériel, l'ordinateur, qui est peut-être plus que sa théorie.

Par ailleurs, le changement de principes en science n'est pas gratuit : on ne le fait que quand on ne peut plus faire autrement. Koyré montre comment le jeune Galilée a poussé à fond la théorie de l'impetus (pour la faire finalement imploser).

Alors, quid de l'IA aujourd'hui ? Personne ne peut répondre à cette question (les historiens futurs le feront, si du moins ils trouvent notre petite agitation intéressante). Mais il n'est pas du tout évident

que la situation soit mûre pour un changement de paradigme.

Tout ça est bien compliqué.