Une fonction d'arrêt constructive…

Introduction

Le théorème d'arrêt nous dit que la fonction d'arrêt sur les machines de Turing est impossible. Mais par des raisonnements en « rupture conceptuelle », je propose de lever cette impossibilité. J'appelle rupture conceptuelle l'utilisation d'une structure mathématique univoque qu'on choisit de plaquer par une interprétation « naturelle » sur une situation mathématique sans pouvoir justifier de la liaison de cette structure ajoutée avec la structure initiale. Par exemple, quand on applique et manipule la structure booléenne de la vérité sur l'ensemble des propositions mathématiques en supposant que chaque proposition possède une valeur de vérité.

Je propose de construire ici une fonction d'arrêt qui contourne le théorème évoqué.
La notion d'arrêt est un concept relativement naturel. Un algorithme tourne une étape après l'autre dans un rythme univoque. Plongées dans une machine de Turing, les étapes présentent une apparence limpide d'univocité. Il semble donc évident de pouvoir statuer sur le fait qu'un algorithme s'arrête ou non. Seulement l'arrêt est un concept en rupture qui n'a rien à voir avec la structure mathématique qui définit un programme au travers d'une machine de Turing. Un calcul ne peut pas produire la réponse « s'arrête ou non » en dehors d'une interprétation. Eventuellement on peut ajouter un état qui signifie arrêt à la machine ; de sorte que parvenu sur cet état, on possède une notion effective d'arrêt. Mais c'est insuffisant pour définir la notion d'arrêt, car le non-arrêt ne possède aucune notion effective en dehors d'attendre si la machine s'arrête. Et si ce n'est pas le cas à cause d'une boucle ou d'un comportement complexe, comment avoir le signe effectif d'un non-arrêt. La question est et demeurera : jusqu'à quand on attend pour statuer sur la réponse ?
Une boucle qui se répète identifiée dans des conditions reconnaissables permet de statuer le non-arrêt, mais comment le faire face à des comportements complexes et non linéaires. Mais un des avantages des mathématiques sur le calcul est justement de pouvoir user de concept en rupture sur la structure.

On entend souvent dire que l'arrêt est facile à prouver, pour trouver si une fonction s'arrête, il suffit de la programmer et d'aller assez loin. Si elle s'arrête bien, ça finira par arriver. Le problème c'est de s'engager dans cette voie si elle ne s'arrête pas. On a tendance à croire que si elle ne s'arrête pas, on ne pourra jamais le savoir parce qu'on ne fera qu'attendre. Mais avec un peu d'imagination, on peut construire des concepts qui permettent de « savoir » si une fonction ne s'arrête pas (bien sûr, le sens du mot « savoir » est relativement subjectif). Pour cela je propose de définir le concept d'enveloppe maximale d'une machine de Turing.

Enveloppe maximale d'une machine de Turing.

On se place dans le cadre où la variable m parcours toutes les machines de Turing d'un type donné. Le choix de la définition de la machine possède relativement peu d'importance.

Définition de la fonction q (quantité d'étapes) :
Si on accepte le concept d'arrêt, pour chaque machine de Turing m, il existe un nombre d'étapes q(m) où elle s'arrête, sinon on pose q(n)=0.

Définition de la fonction enveloppe maximal M (nombre d'étapes jusque là):
Comme q est définie pour tout n, il existe parmi les machines les n premières machines un maximum pour la fonction q : on définit M(n) =max{q(m), m<n}. C'est une fonction croissante en n.

Définition de la suite s_n des enveloppes maximales :
La suite M(n) est croissante, mais bien souvent constante. La suite s_n est la suite ordonnée des n où M(n) a effectivement cru : $s_1=1$ et s_{n+1} est la plus petite valeur qui vérifie $M(s_{n+1})> M(s_n)$. C'est la suite des valeurs maximale locale.
Pour un besoin ultérieur définissons aussi la suite des « enveloppes interprétées » par les contraintes suivantes : $E_1=1$ et $E_{n+1}=\min\{s_n ,n>M(E_n)\}$. Vu la complexité des enveloppes, on obtient une suite d'une croissance gigantesque définie à l'aide de peu de concept (en fait c'est tellement mystérieux qu'on peut en dire peu de chose- voir l'article « paradoxe de Kolmogorov »).

Définition du non-arrêt
La Machine de Turing numéro n ne s'arrête pas si le nombre d'étapes effectuées par la machine a dépassé M(n).

Ce sont tous des concepts qui bien que non calculatoire permettent de raisonner sur les machines de Turing avec univocité (mais bien sûr en rupture). Et dans une certaine mesure, il est possible d'implémenter ces fonctions dans une machine de Turing de façon constructive. Bien sûr on ne va les définir ici que d'un point de vu algorithmique de haut-niveau, une implémentation réelle sur une machine de Turing serait excessivement complexe si l'on ne dispose pas d'un compilateur automatique. Il faut juste retenir qu'en principe rien n'empêche une machine de Turing de recevoir les calculs qui vont être décrits.

Fonction d'arrêt constructive
Partons d'une Machine de Turing universelle avec l'alphabet d'écriture {0,1} qui va simuler en multi-tâche tous les programmes possibles pris dans leur ordre lexicographique (en simulant un état d'arrêt par un codage quand c'est le cas). La simulation effectuera les étapes de chaque machine en diagonale : quand n étapes seront réalisées pour la machine 1, on aura n-1 pour la machine 2, … ,1 étape pour la machine n, puis 0 ensuite, jusqu'à ce que la nouvelle diagonale commence…

Dans la gestion du ruban, on prévoira aussi une place pour l'observation de ces simulations de machines, en particulier pour l'éventuel nombre totale d'étapes effectuée par chaque machine qui se termine.
On aura naturellement un suivi du nombre d'étape effectué par chaque machine en cours (comme les étapes progressent en diagonale par rapport au numéro de machines, 2 variables (emplacement ruban extensible) suffisent à en tenir le compte.

Dans cette gestion multi-tâche, on ajoutera aussi la fonction suivante : dès qu'une machine s'arrête (en identifiant l'état d'arrêt), le nombre de pas effectué est mis en mémoire pour le numéro de machine. Cela perme de construire le suivit des enveloppe maximale. Chaque nouvel arrêt supérieur à tout autre se propose à la candidature d'un M(n), on le nommera enveloppe maximale provisoire, parce qu'on est jamais à l'abri qu'un algorithme numéro k non encore arrêté de rang inférieur s'arrête et détruise ainsi toutes les enveloppes provisoire supérieur à k. Ces valeurs d'enveloppe provisoires seront inscrites en mémoire pour pouvoir effectuer la comparaison de maximalité à chaque terminaison : on vérifie si le nombre de pas effectués correspond à une enveloppe maximale

provisoire en testant les valeurs d'arrêts des machines en dessous jusqu'à découverte ou non d'un nombre supérieur. En cas de lieu d'enveloppe maximale provisoire avéré, on testera les machines inférieure qui auront réalisé au moins le même nombre d'étape, et on les marquera 'provisoirement' comme « machine ne s'arrêtant pas ». Bien sûr certaines d'entre elles pourront s'arrêter et devenir le lieu d'une enveloppe maximale (ou non).

Pour chaque machine de Turing n simulée, on va produire une valeur de sortie. Soit on réserve la place sur le ruban de travail, soit on l'inscrit sur un ruban de sortie. On se propose mettre 3 valeurs de sortie possibles :

- 0 pour l'initialisation : cette valeur est inscrite au moment de la mise en route de la simulation d'une machine. On devrait aussi l'inscrire lorsqu'il y a eu des changements de valeurs, on l'inscrit aussi à chaque réinitialisation, dès que le nombre d'étape se révèle insuffisant à dépasser la dernière enveloppe maximale calculé. En fait, cela n'arrive pas en vertu du choix de déploiement des étapes par machine qui sont décroissant en n.
- 1 pour les machines arrêtée comme enveloppe maximale provisoire locale.
- 2 pour les machines arrêtées qui ne sont pas des enveloppes maximales provisoires.

Ce nombre sera stocké pour chaque machine de Turing depuis l'origine de sa simulation. Il servira à lire la fonction d'arrêt provisoire : $H_p(n)$ la valeur inscrite pour la n-ième machine après p étapes de la machine universelle.

Lecture de la fonction d'arrêt

Le raisonnement est alors le suivant. Comme pout tout n, il « existe » un maximum M(n) d'étapes après quoi, il n'y a plus d'arrêt possible, on peut affirmer que les machines qui ont franchit ce cap ne s'arrêteront plus. Dès lors que M(n)+1 étapes seront effectuées sur les machines m<n sans s'être arrêtée.

La vraie difficulté est que M(n) n'est pas calculable (voir le paradoxe de Klomogorov). Mais ce n'est pas rédhibitoire en « logique ». Car en allant assez loin en étapes, pour une valeur n donnée, les valeurs provisoires d'enveloppe maximale vont forcément atteindre ces valeurs M(n) qui, alors, ne bougerons plus. C'est l'essence même de la définition de ces enveloppes. Le ruban de sortie va donc « converger » pour tout n en un temps fini (mais inconnu). Par le raisonnement qui consiste à comparer chaque fonction aux valeurs d'enveloppe provisoire, la convergence aura lieu non seulement pour les machines qui s'arrêtent, mais aussi pour celles qui ne s'arrêtent pas.

On est face au problème de la définition en mathématique, et en particulier au problème de la définition en rupture conceptuelle. Même si on sait que ce nombre d'étape « existe » (mot qui reste à définir), on est incapable de dire lequel c'est. On est toujours dans l'attente qu'une autre machine, en dessous, s'arrête ultérieurement. Mais on « sait », selon un procédé logique de rupture, qu'il y a un temps pour chaque n, où cela sera terminé. Donc Oui, il y a bien un maximum et donc une rupture entre l'arrêt et le non arrêt. Toute la facilité du procédé de ruputure est de pouvoir se permettre d'en parler sans mettre aucunement les mains dans le cambouis pour rechercher sil y a une liaison de sens avec la structure décrite. Le véritable enjeu consisterait à savoir quand à lieu l'arrêt…

Je crois que c'est dans un tel lieu qu'apparait la nuance entre le « calcul » et les « mathématiques ». Ces dernières possèdent la liberté interprétative de la rupture. On sait que pour tout n, en allant assez loin en étape le ruban est stabilisé pour toutes les cases avant n. Même si ce n'est là qu'une vue de l'esprit.

On peut donc écrire la fonction d'arrêt sous un mode conforme aux mathématiques classiques :

Quelque soit n, il existe p tels que pour tout m<n , quelque soit t>p $H_p(m)=H_t(m)$ qui est la valeur d'arrêt qu'on nommera H(m).

On a satisfait au standard analytique des mathématiques classiques par un procédé purement constructif (pour le processus de fabrication, pas pour l'interprétation en lecture).

Et la diagonale inverse ?

Ce qu'on vient de faire avec les machines de Turing élémentaire, on peut facilement le transposer aux machines de Turing « fonctionnelle », c'est-à-dire avec un ruban qui prend un valeur en entrée (qui représente un programme). Au total la description change assez peu. On a besoin d'un tel cadre, plus riche, pour réfléchir à la question que pose le théorème d'arrêt classique qui est une machine « fonctionnelle » puisqu'elle renvoie une réponse.

Modifications structurelles

Etant donné qu'on a construit la fonction d'arrêt complète, qu'en est-il de la contradiction habituelle qui fait valoir l'impossibilité de la fonction d'arrêt ? En effet, dans le théorème d'arrêt on construit la diagonale inverse à partir d'une structure fonctionnelle avec une variable dentrée.

Les machines de Turing fonctionnelle ont un ruban en entrée qui ne se lit que vers la droite. Et donc toutes les fonctions décrites devront parcourir de façon « diagonale » chaque entrée dans une dimension supplémentaire. Par exemple on code le numéro de la machine et la valeur d'entrée sur une unique variable k, croissante et surjective. Chacune des fonctions décrites ci-dessus possède un paramètre supplémentaire à savoir la valeur d'entrée.
Il faut aussi étendre le concept d'enveloppe maximale qui portera sur la variable k (codant machine et entrée à la fois).
On modifie aussi la machine universelle qui simule tout cela en machine fonctionnelle ayant un ruban d'entrée. Le calcul complet reste identique pour toutes les machines. Par contre un test vérifiera si le calcul a lieu sur la valeur donnée en entrée.
Dans le cas où la valeur diagonale (la valeur d'entrée égale au numéro de la machine simulé) est modifiée, on écrira le résultat sur le ruban de sortie. Celui-ci est fait d'une case unique réinscriptible à volonté dans laquelle sera écrite la valeur d'arrêt provisoire qui recopie la case diagonale (modifiée en mémoire) a partir des 3 valeur de sortie, la case de sortie sera initialisée à 0, la réponse 1 pour oui et 2 pour non.
Bien sûr, la difficulté qu'on a passée sous silence est la gestion de la bande mémoire (l'essentiel étant traité dans le concept de machine universelle) et du codage de chacun des concepts cités. Mais malgré la technicité de l'ouvrage, je pense que cela n'est pas un motif à rejeter ces propositions.

Valeur contradictoire ?

Jusqu'ici on a construit une machine qui extrait la diagonale, il faut la modifier une dernière fois en construisant l'inverse pour aboutir à la contradiction.
Pour cela la technique habituelle est de produire un fonction d'inversion qui encadre la précédente : « si la valeur de sortie est arrêt , on initie une boucle, si c'est non arrêt, on s'arrête. » En fait c'est la fonction diagonale qui sera entrée en entrée.
Et là de nombreux problèmes nous assaillent. A commencer par la possibilité de faire un choix à partir de l'arrêt. La valeur ne se lit qu'une fois l'enveloppe maximale dépassée dont on ignore la position. Ainsi, il n'est pas possible de terminer le programme.

Par ailleurs, avant cette inversion la valeur de sortie est parfaitement connue. Au vu du programme, il est facile de voir que lorsque le nombre d'étapes effectué sur la numéro d'entrée dépassera le nombre d'étapes d'une enveloppe maximale provisoire elle passera à 2 :non. (Selon le mode choisi, elle pourra éventuellement repasser ou non à 0 lorsqu'une valeur de terminaison plus grande que se manifesterait). Dès que l'enveloppe maximale sera atteinte, la réponse sera définitivement stabilisée. Ainsi au bout d'un temps fini on aura la valeur « non-arret ». Cette valeur d'arrêt n'est pas calculée par déduction d'une valeur calculée à l'intérieur du programme dans la strate inférieure (ou n'importe quelle strate plus profonde). Car chacune de ces valeurs est stockée en un lieu différent et aucune lecture de ces valeurs n'agit dans la modification d'une des valeurs des strates supérieures.

Cela dit la contradiction est porté par l'encapsulation dans ce petit programme d'inversion « si la réponse est « arrêt » alors poursuivre indéfiniment, sinon s'arrêter. On peut se dire que la réponse est acquise dans la valeur limite du résultat. Mais si la réponse n'existe que sous forme d'une limite infinie (comme pour la définition d'un réel), cela exclue de pouvoir constructivement lire la réponse et réagir en un temps fini constructif. Cela n'est pas possible sans rupture conceptuelle qui considère finie la réponse. D'où l'incompatibilité de ce rajout fonctionnel. On peut « savoir » la valeur, mais pas suffisamment pour la faire entrer dans un calcul constructif. (C'est le moment d'exprimer le parallèle qu'on peut faire avec la définition d'un réel, est-il suffisamment défini pour dire qu'on le connait ?)

Mais en réalité, on possède ici bien plus qu'une limite infinie. On peut affirmer qu'il existe une valeur finie où le résultat est atteint. Le problème est qu'on ne sait pas quand. On pourrait éventuellement essayer de l'estimer, de la majorer et dire : « Quand cette valeur est dépassée, lire le résultat et appliquer la valeur d'inversion ». Mais majorer constructivement une valeur qui est « maximale » pour une taille donnée signifie qu'il faut un énoncé plus grand, parce que par définition, il n'en existe aucun en dessous de cette taille. Pour coder cette valeur de majoration dans le programme, la taille du code source va augmenter et donc déplacer la valeur diagonale ; et ainsi augmenter la valeur d'estimation (à supposer même que cela soit possible). Il est donc impossible constructivement de majorer une telle valeur à l'intérieur du programme. Cela en dit long sur la nature de la complexité.

Si effectivement, il y a fort à penser que les enveloppes maximales sont d'une complexité irréductible qui ne pourra pas être rendu constructivement avec des concepts simples, la singularité de chaque machine de Turing me fait penser de façon analogique à la singularité de chaque nombre entier qui rend difficile le positionnement anticipatif des nombres premiers. Sauf qu' ici, il n'y a pas de régularité dans la succession des diviseurs, il s'agit d'une interprétation complexe et singulière à déployer pour chaque machine. On ne maîtrise pas cette complexité parce qu'…elle est complexe. Toute cette construction artificielle de fonction d'arrêt prend le problème à l'envers : pour pouvoir dire qu'un numéro est un enveloppe maximale, il faudrait montrer qu'elle s'arrête bien et que toutes celles qui sont avant ne s'arrête pas sur une valeur supérieure ainsi que la preuve de toutes celles qui ne s'arrêtent pas. Il faudrait « tout savoir sur les valeur d'avant ». C'est une somme de « vrai » complexité à dominer. Pour poursuivre l'analogie avec les nombres premiers, les enveloppes maximales sont des informations en creux : « il n'y a aucun des diviseurs présents, à cette position donc… », « il n'y a personne qui s'arrête plus grand avant donc …». Ce qui les rend si difficiles d'accès.

Mais on peut procéder de façon radicalement différente : la valeur en réalité, on la connait, puisqu'il faut une infinité d'étapes pour être certain de converger, la valeur c'est le « non-arrêt ». Il n'y a aucune hésitation à partir de la conception du programme. On peut donc décider directement de

dire au programme d'inversion « arrêt direct ». Et là c'est fini et on peut donc réalisé l'opération de façon fini. Seulement à la différence d'une case où l'on inscrit une valeur révisable, il s'agit là d'une opération extérieure « un arrêt des calculs » qui empêche tout déroulement du processus jusqu'à son terme. Le <calcul de l'arrêt> ne sera jamais effectué si l'on fait cela ; et la décision devient purement extérieure et subjective, non fondé sur un « calcul ». Or une telle pratique change complètement le sens donné à « si la valeur calculée est 'arrêt' alors… ». Or il se trouve qu'en changeant la nature des calculs, on change aussi la nature de ce qu'on observe sur les calculs. Et on se retrouve bien là devant une définition contradictoire : comme elle s'arrête elle devrait dire « non arrêt ». Impossible de se définir comme étant différent de soi (A moins d'avoir assez de place et interprétation dynamique pour le faire comme une alternance infini).

Eventuellement, on pourrait aussi donner la réponse dans une case spécifique inscriptible une seule fois et puis continuer les calculs, ce qui ne serait plus contradictoire, mais qui ne correspondrait pas non plus <aux calculs> . En effet si on choisi un arrêt de la machine où qu'il soit, on ne peut pas calculer la fonction d'arrêt qui est infini par construction (ou plutôt par besoin face à l'ignorance d'apparence irréductible). Au lieu d'exécuter la réponse, on peut plutôt l'écrire et accepter de lire le résultat sur une bande.

S'il semble plus raisonnable d'écrire la réponse plutôt que l'exécuter, on pourra même produire une valeur en réponse au calcul : il suffit d'écrire « arrêt » ou « non-arrêt » par une valeur, au fur et à mesure de l'évolution de la valeur diagonale. La réponse certaine s'obtiendra alors elle aussi comme toute les autres à l'infini. C'est une limite qui se doit de « considérer la fin ». Comme on connait la convergence fini de la réponse que l'on exprimera avec notre alphabet à 3 symboles, la réponse est garantie correcte.

Ainsi, il n'y a plus de « problème de contradiction ». La réponse qui sera affichée sera sans ambigüité dès la première valeur provisoire : « contraire de non arrêt » soit « arrêt » (éventuellement, selon la programmation, elle repassera par des valeurs « indéfinie » et convergera de façons certaines vers cette valeur).

Il n'y a plus de contradiction lorsque la réponse est donnée par « une écriture » au lieu de « l'action » de l'arrêt. Si on omet de s'arrêter, cela n'empêche plus d'effectuer les calculs nécessaires à l'établissement du résultat de la fonction. On peut d'ailleurs à loisir ajouter un état final où elle s'arrêterait quand elle aurait « fini » (reste à définir le mot « fini » pour brancher le programme sur cette condition. Cela dit en passant toute condition jamais réalisée permet ce branchement, ce qui laisse une grande liberté pour l'interprétation du mot « fini » et donc « l'arrêt effectif une fois le calcul assurément effectué »..)

Pour aller plus loin, on comprend bien qu'en écrivant « arrêt » sur la bande de sortie, cela possède un sens subjectif. Du moins dans la portée de ce résultat : il ne dit pas que la fonction va s'arrêter (du moins pas dans le sens habituel), mais qu'il définit la valeur de sortie de la fonction. Or n'est-ce pas ce qu'on voulait. En mathématique tout est histoire d'interprétation, le support de l'interprétation peut-il être imposé ? L'interprétation utilise le mécanisme du codage qu'on utilise en permanence dans les machines de Turing. A priori, on est libre sur le choix du support qui n'a pas être imposé. Ainsi « l'arrêt » n'impose pas nécessairement l'arrêt effectif de la machine. C'est notre esprit qui attend de façon plus ou moins hégémonique de donner un sens maximal à ce qu'on dit, à ce qu'on veut. Si un lien univoque constructif a été crée entre le calcul et une valeur dont on connait la convergence, c'est suffisant en terme mathématique. On peut donc dire qu'au sens des mathématiques classique, nous possédons notre réponse. Du moins, avec beaucoup plus de cohérence que dans les autres tentatives. On voulait un résultat fonctionnel, or arrêter une fonction n'est pas spécifiquement un résultat fonctionnel, de plus elle est incompatible avec les calculs nécessaires. Donc une interprétation fonctionnelle et non « actionnelle » semble indispensable. On

remarquera que toute contradiction a disparu. C'est la formulation qui devient juste gênante : « si le programme ne s'arrête pas arrêter la machine. ». Ce n'est pas une valeur de retour qui est envisagée. Avec une phrase plus conforme comme « si le programme ne s'arrête pas renvoyer la valeur arrêt et arrêter la machine », on se doit de retirer le morceau de phrase « arrêter la machine » pour le besoin des calculs rendu impossibles dans le cas contraire. C'est donc face aux machines dont on peut programmer l'arrêt qu'on aura impossibilité de donner une fonction d'arrêt. Si la machine universelle qui simule devait s'arrêter, il est évident que qu'il n'y aurait pas de résultat. Car n'oublions pas qu'on simule, la fonction diagonale aussi est simulée. Il m'apparait donc trois alternatives :

- exécuter l'arrêt à un moment alors qu'on ne sait pas posséder la réponse ce qui ne permet pas donner une réponse valide.
- Transformer l'application pratique de l'arrêt en un calcul qui interprete car après tout, en simulant il y a nécessité d'interpréter. Sauf qu'il ne s'agit pas de l'interprétation attendu par la fonction arrêt. A moins au contraire d'y voir l'ordre d'un arrêt en construction qui ne sera jamais exécuté (pour cela, il faudra conserver la trace de la valeur dans la ruban de travail , pour pouvoir être lu).
- Considérer qu'il y a plusieurs sorte de réponses, et qu'une réponse infini convergente comme on les manipule en mathématique classique est recevable comme réponse, mais non calculable de sorte que la formulation « si la fonction diagonale appliqué au numéro diagonale est 1 n'est pas une formulation limpide dans le sens des interprétations possible » et que dans tous les cas, elle ne peut pas être interprétée constructivement à partir d'une valeur acquise de façon fini au cours du calcul.

On ne peut pas jouer impunément sur les interprétations. A force de faire varier le sens sous multiples concepts, certains sens deviennent plus contraignants. En restreignant le contexte à une interprétation constructive finie, on arrive bien à une contradiction. Dans la tentative d'inversion, calculer l'arrêt de fonction sans fin en s'arrêtant est contradictoire. Et ce n'est pas une contradiction directement due à l'énoncée. La contradiction n'est pas complètement dans la définition : comme toute machine de Turing est finie, on pourrait espérer en lisant la fonction lire sa finitude ou non par un calcul. Il y a donc bien sous jacent, le problème de la complexité des algorithmes. Sauf qu'avec cette procédure d'inversion, on n'a pas mis le nez dans le moteur. Il s'agit d'une lecture extérieure en rupture de nature interprétative. En programmant la question, il reste beaucoup d'interprétation à trancher, comme dans le cas du paradoxe de Richard. Il est claire qu'en plongeant extérieurement « Tout concept constructif (y compris l'arret)» dans la structure binaire de l'arret , il y a contradiction en la demeure. Mais vu au niveau de la fonction diagonale, il s'agit bien d'une contradiction de définition.
Maintenant, on a vu qu'en élargissant le concept de « réponse », la fonction diagonale inverse n'est pas compatible avec certains types de réponse. Cette contradiction n'est donc pas suffisante pour rejeter toute fonction d'arrêt. L'approche la plus raisonnable consisterait à produire une analyse de la complexité de chaque programme pour y trouver des raisons structurelles d'arrêt ou non. C'est là qu'on se trouve confronté et très vite arrêté par… la complexité.

La lecture d' « existence d'une fonction d'arrêt » est finalement bien subjective, on l'aura compris. La fonction que nous avons fournie s'accorde plutôt bien avec les mathématiques classiques, certainement beaucoup moins avec les mathématiques constructive. La conception a beau être de nature constructive, l'interprétation finale ne l'est pas. Cela s'accorde d'autant moins bien qu'elle laisse entendre une impossibilité constructive, soit au travers d'un éventuel programme d'inversion, soit au travers de l'incompressibilité de l'enveloppe maximale.

Constat final

On dispose là d'une situation assez rare en mathématique (classique) : un procédé itératif constructif qui produit une valeur univoque (oui ou non) qui arrive surement, mais on ne sait pas quand (pour peu qu'on apporte quelque crédit au théorème d'arrêt classique qui bien qu'étant un raisonnement en rupture reste assez « naturel ». On dira même « on ne peut pas savoir quand » constructivement).

Une telle occasion permet de produire une foule de situation non intuitive face aux mathématiques classiques : on possède l'assurance d'existence avec l'impossibilité d'accès. Ca promet de construire bien des existences cocasses (à ce sujet on pourra voir des exemples sur le Théorème des valeurs intermédiaires).

Bien sur, tout cela ne repose que sur des regards conceptuels en rupture. Et l'on voit comment les concepts nous plongent entièrement dans la subjectivité parce qu'ils affirment l'existence de choses essentiellement inaccessible (on se croirait presque dans un cours de théologie). Maintenant ils ont beau être subjectifs, ces concepts restent pertinents parce qu'ils permettent de raisonner avec le regard humain sur des réalités qui nous sont inaccessibles en elle-même. Par contre, on peut estimer et se donner le droit de les rendre approchables avec un certain réalisme.

Tout cela nous impose un recul, une prise de distance sur les affirmations péremptoire issue de choix conceptuels. En particulier, une affirmation négative n'est pas forcément verrouillée de toute part. En créant de nouveau concepts plus fin, d'autres interprétations sont possibles... C'est une remarque qui transforme en passoire toute philosophie issue des théorèmes mathématiques. Mais est-ce si surprenant quand on voit les mathématiques comme de l'univocité interprétée. L'univocité est neutre, que pourrait-elle dire de la vie. L'interprétation est plus ou moins libre… avec un peu d'imagination, on recolle beaucoup de choses étrangères, les cardinaux infinis en sont un bon exemple. Eux qui ont pour définition quelque chose comme « A=B avec A différent de B » (identifiable avec des sous parties strictes de lui-même).

Mais une fois de plus, il est amusant et rassurant de voir que ces concepts subjectifs produisent des conclusions relativement pertinentes sur un mode d'affirmation impertinent : « la fonction d'arrêt est impossible car… ». En effet, même si on peut subjectivement rendre possible une telle fonction, elle semble manifestement impossible. Non à cause d'un raisonnement extérieur en rupture, mais à cause de la complexité illimité qu'elle est censé mesurer alors qu'on a aucune idée de qui est et de ce que peut être une grande complexité... Maitriser la complexité par un concept simple semble manifestement dérisoire. Ce qui révèle la grande subjectivité du concept d'arrêt, et de bon nombre de ses congénères… on découvre là la dimension typiquement humaine et interprétative d'un concept aussi univoque est simple que la structure binaire de l'arrêt.

Navenne - 2014

Michaël Klopfenstein