Plongement univoque du paradoxe de Richard.

Pour la compréhension de cet article, je suppose connu le concept de Machine de Turing ainsi que le procédé diagonal de Cantor.

Introduction

Voici un énoncé du paradoxe de Richard : si on numérote tous les nombres réels définissables en un nombre fini de mots, alors par le procédé diagonal de Cantor, on peut construire un nombre réel hors de cette liste. Et pourtant ce nombre a été défini en un nombre fini de mots.

Quand on regarde le paradoxe de Richard, on est gêné parce que la notion de définition est particulièrement délicate. Il faut reformuler cet énoncé à l'intérieur d'un cadre univoque. Le cadre des machines de Turing s'y prête bien. Certaines machines de Turing peuvent être assimilées à un algorithme de fabrication d'une suite de nombres. Pour contourner la difficulté de la définition des nombres réels, on va reformuler le problème en s'intéressant à la construction des suites infinies de nombres binaires. Cela produit le nouvel énoncé suivant :

Si on numérote toutes les suites infinies définissables par un programme de taille fini, alors par le procédé diagonal de Cantor, on peut construire une suite infinie hors de cette liste. Et pourtant cette suite peut être définie par un programme fini.

Une autre définition plus centrée sur la définition : Si on prend toutes les définitions des machines de Turing finies (donc numérotables) produisant des suites infinies en sortie, alors la machine de Turing qui les simule et fabrique comme sortie la diagonale-inversée de Cantor des sorties précédentes est-elle une nouvelle machine de Turing ou n'est-elle pas définissable ?

Concept et univocité

Le cadre des machines de Turing est un cadre de référence pour les manipulations univoques : une machine de Turing ne fait que des manipulations élémentaires mécaniques sans ambigüité. Tout y est défini de façon déterministe, par des graphes des schémas des successions d'actions élémentaires rigidement définies. C'est typiquement ce que j'appelle des réalités univoques. Maintenant, on peut assimiler une machine de Turing à un programme informatique. Un programme, c'est un plan qui exprime des concepts (des idées, de formes, des objets d'une certaine nature). Or un concept n'est plus fait uniquement de liens neutres indépendants de toute interprétation. Quand dans une machine de Turing, on lit des « concepts », je vois une lecture qui habille le mécanisme des machines d'idées qui dépassent la réalité univoque et se distance de cette neutralité sémantique propre à l'univocité élémentaire. Le concept produit une « interprétation ». Le concept donne un nom à certains assemblages de liens qui produisent des identités mentales. Ces identités pourront être visualisées et manipulées de façon globale. L'intérêt des machines de Turing et des programmes, c'est de pouvoir traduire les concepts en manipulation mécanique neutre, une traduction entre perception mentale et construction élémentaire univoque. Dans le cas d'un langage machine informatique en un programme fait d'une suite de chiffres binaires (digit), ou dans les cas des machine de Turging en une fonction de transition qu'on peut assimiler à un graphe orienté (neutre de toute interprétation en soi). Ainsi une machine de Turing est à la fois un cadre rigoureux pour l'univocité par ses mécanismes élémentaires, mais aussi l'accès à des concepts qui se lisent en observant des groupes fait de ces réalités élémentaires. Les concepts dans les machines de Turing (soit au niveau des fonctions de transition, soit au niveau du déroulement du programme), c'est tout ce qu'on ajoute à la neutralité des liens, tous les noms qu'on donne aux groupements de manipulations et/ou de digits qui permettent de reconnaitre « des idées » qui dépasse la mécanique élémentaire. Ainsi une machine de Turing binaire s'élève au dessus de la seule manipulation neutre de 0 et de 1, tout en ne réalisant que de telles manipulations univoques.

Plan, réalisation et possibilité de calculs

Un algorithme permet de produire une construction infinie par l'exécution d'un plan fini en suivant un ensemble d'instructions élémentaires pas à pas. La fonction de transition d'une machine de Turing peut-être considéré comme un plan de construction, que l'exécution va mettre en œuvre. Si la machine possède un ruban de sortie, on pourra considérer que le plan construit une suite de 0 et de 1 sur un ruban. Il existe des machines de Turing universelles qui permettent de simuler le fonctionnement de n'importe quelle machine de Turing (y compris elle-même d'ailleurs).

Puisque le cadre permet de formuler des constructions conceptuelles, en s'intéressant à toutes les machines de Turing, on a le sentiment d'accéder à un champ infini de « concepts ». La thèse de Church dit que tout ce qui est calculable peut l'être au travers d'une machine de Turing. Sous cette hypothèse, le cadre des machines de Turing peut même nous donner le sentiment d'atteindre le potentiel sémantique complet ! C'est-à-dire l'ensemble des définitions possibles (du moins celles qui produisent un calcul). Il y a peu de cadres qui permettent d'envisager une « totalité des définitions possibles » sans contradiction interne (du type du paradoxe de Russel). Les machines de Turing le permettent, sous l'hypothèse de Church, parce que les définitions d'une machine de Turing sont de nature mécanique univoque. C'est l'interprétation qui permettra (ou non) d'y lire une définition conceptuelle par dessus. Une machine de Turing est une réalité mécanique autonome qui ne dépend pas de l'interprétation ajoutée qu'on lui attribue. Par contre, elle peut modéliser toutes sortes de concepts c imaginables.

Un cadre qui permet la totalité des définitions possibles est typiquement celui où l'on peut mettre en œuvre le principe diagonal de Cantor. C'est pourquoi il est particulièrement intéressant de chercher à observer ce principe diagonal dans ce cadre qui peut plonger toute construction dans une mécanique univoque disjointe de toute interprétation.

Plongement linéaire

La contradiction diagonale repose sur une lecture extérieure de la totalité des définitions. Une des idées centrales qui conduit mon étude est la suivante : une lecture extérieure est une rupture qui permet l'apparition de subjectivité par une lecture polysémique d'un même objet. Aussi mon objectif est de pouvoir enfermer le processus constructif complet ainsi que l'observation à l'intérieur d'un unique processus de calcul.

Pour réaliser cela, il suffit que les éléments inverses de la diagonale soient construits au fur et à mesure de la fabrication de la diagonale. Puisqu'une tête de lecture unique doit réaliser ces différentes tâches, il faudra concevoir une gestion multitâche: un temps de calcul doit être réservé tour à tour pour chacun des concepts de calcul infini en jeu. Un temps pour chaque machine de Turing, et puis un temps pour tester si les valeurs diagonales sont écrites.

Choix du cadre

En informatique, on écrit les programmes en langage évolué et un compilateur les traduit automatiquement en langage machine. L'intérêt est de pouvoir parler du programme sous forme de concepts, sans quoi, en langage machine, il serait presque illisible. Nous procéderons naturellement de cette façon, par des descriptions conceptuelles, en supposant qu'on possède un compilateur pour traduire automatiquement notre programme sous forme d'une fonction de transition (qui est le « langage machine » d'une machine de Turing).

Pour parvenir à élaborer les concepts du paradoxe de Richard dans une machine de Turing sous forme d'un parcours linéaire unique qui enferme à la fois construction et observation, on va construire plusieurs cadres imbriqués l'un dans l'autre.

1er cadre : machine de base à sortie identifiable

Les machines de Turing peuvent être choisies parmi de nombreuses options. Voici les options qui nous paraissent les plus adaptées à notre recherche.

- On travaillera avec un alphabet binaire (0 et 1) et éventuellement un troisième symbole X qui servira juste à initialiser les rubans. En particulier, on ne prendra pas de symbole signifiant l'arrêt. Parce que dans notre recherche, on n'a aucun intérêt à voir les machines s'arrêter ; on construit des suites infinies. Seuls les symboles 0 et 1 seront écrits.
- Dans l'objectif de produire une suite binaire, on utilisera une machine à deux rubans :
    - le ruban de travail : il fonctionne avec un déplacement bilatéral, rempli initialement de X (ou éventuellement de 0). Il permettra de réaliser les calculs nécessaires.
    - le ruban de sortie : il fonctionne aussi avec un déplacement bilatéral qui écrit les sorties (un déplacement unilatéral aurait été préférable pour garantir les valeurs de sortie, seulement on aura besoin de le remplir dans un ordre qui n'est pas croissant). Le ruban est initialement rempli de X (ou éventuellement de 0).

      On ajoute en plus la possibilité de ne pas écrire sur ce ruban de sortie lors d'un déplacement (pour pouvoir traverser les parties du ruban non écrites sans les écrire).

      Pour facilité la conceptualisation et pour produire une interprétation stable du ruban de sortie, je propose en plus que chaque case du ruban de sortie ne puisse être écrite qu'une seule fois. On ajoute à la machine le mécanisme qu'une case déjà écrite ne se transforme plus. Cela n'a pas d'impact sur le déroulement du programme qui dépend uniquement du ruban de travail.
- On supposera que la fonction de transition de la machine ne comporte pas d'état d'arrêt. De sorte que toutes les machines vont tourner indéfiniment, ce qui est le cadre naturel d'une production de nombre infini.

Chaque fonction de transition produit une machine de Turing. On associera donc une telle machine M_i pour chaque fonction de transition. Ce qui suppose de pouvoir énumérer les fonctions de transition. Cela est accessible au travers de la machine de Turing universelle.

Machine universelle

Turing a montré la possibilité de construire des machines universelles capables de simuler n'importe quelle machine au travers d'un codage. Ce codage consiste à traduire sur le ruban les informations de transition de la machine à simuler. La machine universelle possède elle-même une fonction de transition qui peut être vue comme un « algorithme » permettant d'exécuter n'importe quel plan. Elle reproduit chaque action élémentaire de la machine à simuler au travers d'un déroulement bien défini : elle lit sur le ruban les informations de transition et se déplace en des lieux réservés du ruban pour lire et écrire les données simulées. Au milieu de plusieurs étapes intermédiaires, la lecture et l'écriture d'une case précise de ce déroulement correspondent au pas élémentaire à simuler. Ces étapes intermédiaires (de déplacements, de lectures et d'écritures) sont des « actions structurelles » qui permettent cette simulation. Pour « observer » la machine de Turing universelle simulant une machine, il faut donc « interpréter » le déroulement et ne voir que certaines positions comme signifiante pour la simulation. Ces positions sont clairement identifiées par le processus de codage.

C'est analogue à un programme d'émulation écrit dans un langage de programmation de haut niveau (C, Java, Pascal, Basic,…) qui saurait lire les textes des programmes et les exécuter. Ca ne pose pas de problème majeur. De tels principes de simulation seront utilisés à chaque étape d'imbrication des cadres.

On travaille avec des machines sans ruban d'entrée parce que la machine finale ne doit pas posséder de dépendance de la sortie par rapport à une variable mais construire simplement une suite de sortie. Aussi les programmes à simuler (les différentes machines de Turing codé en binaire) seront inscrits sur le début du ruban de travail de la machine universelle.

Pour simuler toutes les machines possibles, il importe juste que l'ensemble des codages de programmes soit surjectif et recouvre l'ensemble des fonctions de transitions possibles. Il n'est pas indispensable de trouver un codage bijectif, même si c'est un peu plus facile de raisonner en supposant qu'on a la bijectivité. Aussi nous ferons comme si l'énumération lexicographique des codages correspondait à un accès bijectif. En plaçant toute les combinaisons de codes possibles par ordre alphabétique, on sera certain de simuler toutes les machines de Turing. Voilà expliquée la notation indicée sur les entiers naturels $\{M_i , i \text{ dans } N\}$ pour l'ensemble des machines de Turing. Par défaut on dira que M_i est une machine de Turing.

2eme cadre : machine universelle à ruban unique

Le second cadre consiste en une machine universelle qui va simuler toutes les machines de Turing M_i. On va choisir une machine universelle avec une infinité de rubans et même deux infinités : R_i et R'_i pour i entier naturel. R_i sera le ruban de travail de la machine de Turing numéro i et R'_i sera son ruban de sortie. Chaque ruban possède sa tête de lecture-écriture, relié à la fonction de transition qui les pilote que j'appellerai souvent le « programme extérieur de gestion globale ».

Pour piloter cette infinité de ruban, on a besoin de pouvoir désigner quelle tête doit écrire. Or le numéro des têtes est une information de type non limitée. Elle ne peut donc pas venir de la fonction de transition finie. On supposera donc qu'un ruban supplémentaire gère cette infinité de numéro à manipuler.

Sur chaque ruban de travail R_i, on suppose qu'au départ sera écrit le nombre i, le reste du ruban étant initialisé à X. Alors que R'_i est complètement initialisé à X.

Notre machine universelle va exécuter en parallèle toutes les machines M_i. Pour cela il faut envisager un déplacement vertical en plus du déplacement horizontal de la machine de Turing traditionnelle. On ajoutera donc un ruban supplémentaire pour la mémorisation des informations techniques nécessaire à ce traitement multitâche. On peut supposer que l'algorithme de la machine universelle possède une gestion multitâche qui va exécuter alternativement 1 étape pour la machine 1, puis 2 étapes pour les machines 1 et 2, puis 3 étapes pour toutes les machines jusqu'à 3, etc… . On peut supposer dès lors que toutes les suites seront écrites.
Sur le ruban supplémentaire seront aussi inscrit l'état de chaque Machine M_i, les positions des machines simulées sur chaque ruban de travail R_i, et probablement toutes sortes d'informations nécessaires au déplacement et à la gestion de ce multitâche,… bien sûr tout cela devra être organisé de façon rigoureuse par des codages adaptés.
A partir de ce cadre, on peut « voir » la diagonale constituée de la liste des i-ièmes sorties de chaque machine M_i. Et la diagonale-inversée sera la liste de ces mêmes nombres inversés par échange des 0 et des 1. Jusqu'ici, c'est par un « regard d'observation extérieure » qu'on peut les voir.

3eme cadre : plongement dans une machine à ruban unique

Le troisième cadre consiste à concentrer le 2eme cadre et ses infinités de rubans en une seule machine de Turing avec un seul ruban.

Une bijection des positions : Pour réaliser cette concentration, il faut ajouter une couche de gestion des positions sur le ruban. Au départ, il s'agit d'une bijection de $N \cup N^2 \cup N^2$ dans N. Cela permet de simuler les positions du cadre 2 sur une seule ligne. Mais pour gérer tout cela, il faudra en plus réserver des places pour le traitement technique de cette gestion. On ne procédera pas à l'écriture de cet algorithme assez technique, supposant qu'il est réalisable et cette technicité inutile ici.

La Gestion technique : avec le concept de machine universelle, il apparait qu'il y a deux façons de construire des concepts.

- à l'extérieur : La fonction de transition peut être conçue pour exprimer des concepts de traitement
- à l'intérieur du ruban : le ruban de la machine universelle possède un programme qui contient des concepts (qui sont traités par l'exécution du programme).

Il y a d'un côté l'exécution de toutes les machines de Turing et de l'autre un programme de fonctionnement qui gère ces exécutions. Il doit gérer à la fois l'algorithme complexe des positions à l'aide de réservations et de valeurs mémorisées, et l'algorithme de la machine universelle qui permet la simulation en multitâche de toutes les machines (donc de tous les concepts). Qu'il soit intérieur ou extérieur, ce programme de fonctionnement aura besoin de beaucoup places réservées pour réaliser ses propres manipulations. A titre de remarque, on peut observer que la manipulation de nombres non bornés à partir d'une base binaire est déjà en soi toute une stratégie algorithmique élaborée.

Conformément au cadre 2, on suppose que chacune des sous-suites qui matérialisent le ruban R_i contient déjà le nombre i au début du « ruban ». Ce nombre i est le code de la machine à interpréter. La suite du ruban de travail (vu interprétativement) sera suivie de X (ou de 0). Les sous suites correspondant au ruban R'_i sont initialisé à X (ou à 0).

Chaque machine Mi devra donc être observée au « lieu éclaté » qui lui est dévolu. Il faut interpréter le ruban (le regarder en assemblant certains groupements) pour y lire les sorties ou pour suivre le travail exécuté.

4eme cadre : un programme constructif

Aux cadres 2 et 3, on a des objets qui sont initialisés avec des informations en quantité infinie. Ce n'est pas constructif. Pour compléter la démarche d'une approche linéaire constructive, on va devoir construire un programme qui réalise l'initialisation des machines de Turing M_i en écrivant leur numéro.
Pour réaliser cela, on le choix entre :

- Une solution extérieure : placer le programme de gestion globale dans la fonction de transition de la machine universelle avec un ruban vide au départ. Toute l'information nécessaire au programme serait contenue à l'extérieur du ruban. Il s'agirait donc d'une machine universelle enrichie de nombreux concepts.
- Une solution intérieur : ou bien au contraire choisir une fonction de transition « simple » dont le rôle se limite à être une machine universelle élémentaire. Dans ce cas, le programme de gestion doit être « écrit » au début du ruban.

Qu'on choisisse la première où la deuxième solution (qui possède une couche d'interprétation supplémentaire), je propose de nommer P ce programme de gestion globale. Et nommons R, son ruban de Travail. R est une sous partie du ruban ou bien tout le ruban selon le cas.
Pour chaque entier i successivement, le programme P va inscrire le nombre i à sa position définie, et dès que celui-ci est inscrit, il débutera la simulation de l'exécution du programme. Le tout en multitâche pour qu'aucune action ne soit bloquante.
On possède enfin notre cadre fini de base pour gérer la totalité des définitions de suite. Mais encore plusieurs concepts vont lui être ajoutés pour produire la diagonale inverse.

Construction de l'inverse de la diagonale

Jusqu'ici nous avons parlé de la construction et l'exécution de toutes les machines de Turing. Il s'agit d'observer l'inverse de la diagonale du tableau réalisé. Construire la diagonale-inversée pose à nouveau la question du cadre de cette construction : est-ce une opération extérieure ou intérieure à la construction. Si c'est extérieur, est-ce que la diagonale est imaginée conceptuellement sans se soucier d'aucune réalité élémentaire ou bien identifiée mécaniquement en certains lieux et temps bien précis. Dans notre contexte de recherche d'univocité nous préférerons au contraire une approche constructive de l'observation. Une construction qui atteste une observation effective. Il faudra un mécanisme pour lire et relier les valeurs des rubans de sortie (simulés) vers des lieux réservés au processus d'observation. La grande difficulté, c'est que la lecture après une quantité de travail infini n'est qu'un concept non constructif. Bien qu'il s'agisse d'un concept

général de la pratique mathématique, la lecture après une infinité ne peut être réalisée constructivement. Il faudra trouver des solutions palliatives.

Pour répondre au paradoxe de Richard, il faut chercher si parmi les programmes M_i un ruban R'_i correspond à cette même construction de réponse. A priori, il devrait exister une colonne qui fait la même chose : le programme M_i qui correspond exactement à la même construction conceptuelle que le programme P. Avec cependant la nécessité pour ce programme Mi d'écrire la diagonale-inversée sur son ruban de sortie (notons ce programme P^1 c'est le programme situé sur la machine M_P qui simule P).
En effet, nous avons dit que les machines de Turing M_i possèdent un ruban de sortie. Ce choix d'un ruban de sortie correspond on désir de matérialiser les résultats. Pour parler effectivement d'un concept de sortie, on construit sa réalité mécanique. Un écrit bien identifiable et stabilisé permet d'observer un concept. C'est le même problème qui nous a préoccupé quand on souhaite stabiliser les valeurs de sorties. On le retrouve donc pour la diagonale. Si une machine M_i construit le diagonale-inversée, on a choisi qu'elle doit le construire sur son ruban de sortie pour pouvoir l'observer. C'est aussi notre souhait originel : matérialiser l'observation de cette diagonale-inversée à l'intérieur d'un processus linéaire unique. On va donc attendre la même chose pour P : la production effective de la diagonale et pas seulement son observation.
C'est pourquoi, on ajoute un ruban de sortie R' à la machine du cadre 4 (en plus du ruban de travail que nous venons de décrire). Et l'on va inscrire la diagonale-inversée comme résultat du travail de P. On se retrouve, au final, dans la situation du cadre 1 : c'est une machine de Turing qui doit gérer un ruban de sortie.
Pour écrire les digits de la diagonale-inversée, il faut aussi ajouter en multitâche une nouvelle procédure de surveillance qui contrôle les écritures et qui reconnait l'écriture d'un digit-diagonal. En effet, la position précise des nombres diagonaux est connue, le système peut donc les reconnaître aussitôt qu'ils sont écrits sur le ruban R'_i.

Maintenant, on a vu que les machines M_i qui n'écrivent pas leur digit-diagonal sont à l'origine d'un problème : il faut donner du sens et interpréter cette non-écriture. Devant la difficulté, on pourrait choisir de n'attribuer de valeur diagonale qu'aux machines qui ne bloquent pas. Il suffirait de choisir de lire la diagonale «interprétativement » : la diagonal lue serait simplement restreinte aux valeurs qui existent. Seulement, il est impossible de savoir à l'avance quels digits-diagonaux seront écrits ou non (c'est une conséquence du théorème d'arrêt). De plus, à cause d'effets de mise en miroir dont on va parler bientôt, il y a de bonne chance pour que le nombre paradoxal à l'intersection de la diagonale et du ruban R'_P qui simule P ne soit jamais écrit. Ce qui réduirait à néant la possibilité faire porter une contradiction.

Interprétation de réhabilitation

Cherchons un moyen de perpétuer le paradoxe de Richard en donnant une valeur de sortie aux digits-diagonaux des machines M_i qui ne pourrons pas l'écrire : au lieu de remplir initialement les rubans avec des X (qui signifie absence d'écrit), on peut imaginer qu'ils seront remplis de 0. Ainsi, lorsqu'une machine n'aboutit pas sur sa diagonale, on peut « considérer » qu'un 0 est le résultat. C'est une interprétation assez naturelle ; comme pour les nombres fractionnaires dont les décimales terminent, ils sont assimilés à des nombres dont les décimales sont des 0 jusqu'à l'infini.
Mais en faisant le choix d'initialiser tout à 0, il se greffe des problèmes techniques supplémentaires qu'il faudra résoudre. Quand on initialisera le programme M_i en écrivant le nombre i au début de son ruban, pour pouvoir l'exécuter, il faudra connaître où se termine le programme et où commence les 0 du ruban qui ne font pas partie de cette définition. Il faut donc inventer un codage pour les terminaisons.
[Voici une possibilité à titre d'exemple : à chaque fois que 11 est rencontré sur un ruban de travail, on le suivra systématiquement d'un nombre 0 qui ne sera pas lu comme étant une donnée, mais comme une valeur structurelle. Et lorsqu'on rencontrera 111 (11 suivi de 1), cela sera la marque d'une position de fin et non d'une donnée. Pour cela le programme de gestion de P devra ajouter cette analyse à chaque lecture de digits où le codage est mis en action. Pas

d'inquiétude pour notre paradoxe, ce code n'intervient que sur le ruban de travail, et pas sur celui de sortie. De sorte qu'il n'altère pas l'écriture des digits-diagonaux en décalant les valeurs. Par contre, il faut prévoir pour eux de la place dans la structure globale qui gère les positions. Ce qui la complique encore potentiellement.]

Quand peut-on écrire les nombres diagonaux ? On a la nécessité d'attendre au moins qu'ils aient été calculés pour les observer. Après cela, le « quand » importe peu pourvu qu'il soit fait. Il est assez naturel pour simplifier les concepts que chaque fois qu'un digit-diagonal est écrit, il soit « observé » aussitôt. Ce qui produit un automatisme de la recopie des nombres diagonaux à l'étage supérieur qui l'« observe » pour construire sa propre sortie.

Maintenant un autre problème de taille s'ajoute. Si tous les rubans de la machine M_i sont initialisés avec des 0, les machines qui bloquent dans un calcul n'auront jamais l'occasion de « dire » leur valeur au niveau supérieur puisque cela se fait au moment de l'écriture. Pour avoir la bonne réponse inversée à ce niveau supérieur, il faudrait donc initialiser le ruban de sortie de la diagonale-inversée R' avec des '1'. Comme on ne peut pas prévoir s'il sera écrit ou non, il faut initialiser toutes les valeurs à 1 pour être le contraire des 0 de R'_i. Puis changer ce nombre au moment où une case diagonale R'_i est effectivement écrite.
Seulement un tel choix a pour conséquence de produire plusieurs écritures successives sur une seule case (l'initialisation puis l'écriture effective). Or on verra que R'_i le ruban de sortie de P^1 sera lui-même le résultat d'un autre ruban en dessous. Et donc qu'il se produira deux écritures sur R'_i et donc 3 écritures sur R', qui va se mettre à jour à chaque écriture. Et en plongeant dans la réalité en miroir des programmes P^i qui s'exécutent l'un l'autre, s'il faut plusieurs écritures à l'étage inférieur, chaque modification se transportera à l'inverse pour l'étage supérieur. Et à chaque étage, en plus des copies s'ajoute l'initialisation. Ce qui produit une écriture supplémentaire pour chaque étage supérieur. Dans le cas d'une mise en miroir infinie, cela signifierait que des digit-diagonaux pourrait être réécrit sans cesse. Mais si on ne peut garantir la valeur d'une case, on peut difficilement parler de « sa valeur ». C'est pourquoi, le souhait d'une interprétation stable nous invite à éviter de pouvoir écrire plusieurs fois dans une même case de sortie.
On peut ruser et tenter de résoudre ce problème par l'usage d'une interprétation. Il suffit de lire le ruban de sortie à l'envers, on transforme les 1 en 0 et réciproquement. Pour anticiper les non-terminaisons, l'initialisation sera faite de 0 (ce qui signifie 1) ; autrement dit, aucune écriture d'initialisation n'est à ajouter à la donnée du ruban de départ. Toute écriture ultérieure sera inversée au moment de l'écriture. (Ce qui signifie qu'on recopiera la diagonale au lieu de l'inverser). Par contre, au moment de l'interprétation, on dira bien je lis « 1 » lorsqu'on verra « 0 ». Etant donné qu'il s'agit du ruban de sortie qui n'a aucune relation avec le ruban de calcul, ça ne pose pas de problème de déroulement ultérieur du programme. Comme tout le ruban de sortie est concerné par cette information, on peut stocker une information-d'inversion précisant le sens de lecture de chaque ruban de sortie. Car ce concept ne concerne pas que P, il faut aussi l'intégrer à P^1 à l'étage « inférieur ». Face à toute lecture ultérieure d'un nombre, on interprétera à partir de l'information-d'inversion. On peut dès à présent entrevoir une simple alternance des inversions entre les niveaux d'exécution en miroir.
A priori, cette approche n'enlève rien au paradoxe de Richard : chaque valeur est écrite en des lieux différents et l'usage des nombres est fait avec l'interprétation qui lui convient. Il s'agissait juste d'initialiser automatiquement à l'envers sans avoir à tout écrire.

Par ce mécanisme, on a conceptualisé le blocage des machines de Turing par l'écriture de 0 en sortie, ainsi que la diagonale inversée avec 1 en sortie par défaut. Maintenant, on ne sait jamais quand une machine bloquera ou non. C'est le prix à payer, la diagonale « n'existe que vers la fin des calculs ». Pour tout dire, c'était déjà plus ou moins le cas avant cette remarque ; c'est le cas pour toute réalité infinie.

Remarques

Une autre solution à notre construction globale consisterait à ne pas prévoir de rubans de sortie pour les machines de Turing (ni R', ni les R'_i), mais à intégrer la diagonale-inversée dans une sous-suite de positions réservées sur le ruban unique. On utiliserait que des machines de Turing M_i à ruban unique pour simuler à la fois le travail et la sortie dans des positions bien interprétées. Pour cela il faut que les informations de positions et d'écriture des cases diagonales soient anticipées dans le programme d'exécution.

On ne cesse d'ajouter des « éléments structurels », mais chaque ajout d'information suggère un changement radical de la structure interprétative à cause des changements de positions notamment. Et tout doit être structuré avant de commencer pour que le programme puisse gérer chaque concept en lieu et place connus. Tout cela montrer qu'une lecture « extérieure après coup » qui semble simple dans une approche conceptuelle, est en réalité une intrusion complexe dès lors qu'elle doit être intégrée au processus linéaire. C'est une réalité multitâche non indépendante qui modifie profondément la structure d'exécution, le programme lui-même et l'écriture des rubans (ce qui nécessite de changer l'interprétation de la lecture).

La modélisation univoque du paradoxe de Richard nécessite une très forte dimension interprétative. De nombreuses alternatives auxquelles on n'a même pas pensé seraient envisageables. Donnant lieu à de nombreux programmes P possibles. Le but des 4 cadres précédents est l'écriture d'un programme fini qui concentre tous les concepts constructifs du paradoxe de Richard. A chaque nouveau cadre, il faut faire passer le programme extérieur global au niveau du ruban qui sera lu par une machine universelle. Si d'apparence les concepts sont simples, la conception technique, elle, peut s'avérer un peu moins évidente. Par contre, on peut l'envisager plus simplement dans un langage algorithmique de haut niveau et après l'adapter automatiquement à une machine de Turing par un « compilateur ». Sauf erreur de ma part avec un peu d'astuce et de réflexion, cela ne semble pas inaccessible.
De plus, il existe une foule de machines de Turing universelle qui fonctionnent avec des codages très différents. En pensant à toutes les façons de produire des machines universelles, on multiplie encore les organisations possibles.

A chaque nouveau cadre, nous empilons des étages interprétatifs imbriqués les uns dans les autres. C'est même exacerbé dans l'idée d'une imbrication infinie en miroir. Seulement dès que plusieurs réalités interprétatives sont imbriquées, cela suppose aussi plusieurs lectures différentes possibles pour cette même réalité. Si A exécute B qui exécute C … On a là, une première lecture. Mais on peut choisir de voir uniquement A qui exécute un programme simple. Ou même simplement voir A qui calcule (sans voir un programme exécuté). Chaque niveau interprétatif peut-être ignoré comme formant une identité avec le niveau supérieur qui l'englouti. Ou bien au contraire, le niveau inférieur peut être interprété comme étant une réalité propre. Avec un programme-tête, un programme-ruban et des données préinscrites sur le ruban qui vont être utilisées pour des calculs, on peut lire plusieurs choses :

- un programme-tête qui donne vie à un programme-ruban qui réalise des calculs sur des données.
- un simple programme-tête qui produit une gestion de données préinscrites.
- Avec un regard encore plus désengagé, on pourrait simplement voir une machine de Turing qui fonctionne élémentairement étape par étape sur des données (données dans la tête et dans le ruban).

C'est le regard qui donne le sens, en groupant et conceptualisant ces groupes temporels et spatiaux d'activité mécanique.
De façon générale, la polysémie interprétative est naturelle dès qu'on fait intervenir des mots d'action comme 'exécuter', 'lire', 'faire' : selon qu'on les prend pour du texte ou bien qu'on les mette en action et qu'on considère le résultat de cette action, le résultat n'est pas le même que le texte initial. Ici, la polysémie provient du fait de l'interprétation des concepts en présence. La manipulation mécanique élémentaire d'une machine

de Turing ne porte pas de concept en soi, c'est nous qui lisons, groupons, souhaitons voir des groupements, des dépendances ou non.
Le programme P, dont nous parlons, possède une multitude de lectures possibles parce que de nombreux étages interprétatifs sont imbriqués.

L'identité des cases diagonales

A cause du blocage possible des machines de Turing, on a vu la nécessité de définir une interprétation subjective de la diagonale en attribuant arbitrairement la valeur 0 aux machines bloquées. On a vu que le concept de sortie d'une machine de Turing n'est pas univoque : quand les valeurs peuvent être réécrites, il est difficile de parler de « la » valeur de sortie ; quand toutes les valeurs du ruban ne sont pas écrites, il est difficile de parler d'une suite des valeurs de sortie. Il existe bien sûr des choix qui permettent une interprétation plus naturelle comme n'écrire sur le ruban de sortie qu'avec des déplacements vers la droite et sans permission de sauter des cases. Mais là encore ce n'est pas suffisant, il faudrait pouvoir garantir que toutes les cases souhaitées seront écrites. Et pourquoi pas toutes les cases après tout ? Or le problème de l'arrêt nous donne une sorte de garantie d'impossibilité dès qu'on n'a pas des machines spécifiquement identifiées avec cette propriété.
Dans notre cas, où l'on ne peut pas imposer de déplacement unilatéral, les problèmes s'accumulent (bien sûr, on pourrait toujours trouver une interprétation qui permet un déplacement uniquement vers la droite. Par exemple écrire les digit-diagonaux sur le ruban de sortie dans l'ordre où ils sortent, accompagnés de leur numéro d'ordre. Puisque c'est une information suffisante pour reconstruire le ruban, on peut décider d'interpréter qu'il s'agit du ruban). Choisir d'inventer des valeurs de sortie aux machines qui n'en produisent pas, ainsi qu'une stratégie pour les produire, tout cela rend légitime la question : qu'est-ce qu'un nombre diagonal ? Le plongement univoque rend compte de problèmes passés sous silence dans une « lecture conceptuelle supérieure ». Cela montre que la posture conceptuelle crédite facilement, à des phrases, un sens qui n'est peut-être pas aussi évident qu'il parait.

Pour élaguer cette conceptualité additive, on pourrait ne s'intéresser qu'aux machines qui produisent un digit-diagonal bien défini. En effet, la machine produit un nombre ou elle ne le produit pas. De la même façon, quand il y a un état d'arrêt, il y a les machines de Turing qui s'arrêtent et celles qui ne s'arrêtent pas. En réalité, ces concepts très simples sont inaccessibles. La simplicité de leur structure binaire est en rupture face à la connaissance des objets eux-mêmes. Leur apparence très simple les rend très crédibles, mais rien ne permet de les définir dans les calculs en dehors d'un choix conceptuel. Pour n'utiliser que les machines qui écrivent bien leur digit-diagonaux, il faudrait une démonstration de l'effectivité de cette écriture. Et comment le pourrait-on? Le théorème d'arrêt nous parle d'une impossibilité, et si l'on a des doutes face à un tel théorème, la complexité illimitée des algorithmes nous montre une barrière infranchissable.

La case problématique

Puisqu'on a élaboré un programme P qui simule les concepts nécessaires à la contradiction de Cantor, ce programme écrit sur le ruban d'une machine universelle correspond aussi à un nombre. On l'appellera P à nouveau. Et voyons ce qu'il en est du lieu précis du paradoxe de Cantor. C'est au niveau de la P-ième case réservée au P-ième programme qu'il doit y avoir une contradiction. Quel est le contenu de cette case ?
On met pour l'instant de côté le « Quand », on fait confiance au « où » avec la gestion organisée des places et on s'intéresse surtout au « quoi » et au « comment ». Avec la gestion multitâche et le déroulement ininterrompu, on peut conceptualiser que tout ce qui est calculable le sera par le programme P. Le programme cadre en amont « attend ». Dès qu'un digit-diagonal est écrit, il recopie son inverse en lieu et place voulue. Ainsi pour que soit écrit le P-ième digit-diagonal-inversé du ruban de sortie, il faut attendre que la P-ième machine M_i (notée P^1) produise ce digit.

Or on attend justement de cette machine qu'elle soit une simulation de P qui va donc elle-même construire la liste de toutes les machines de Turing puis les exécuter à l'intérieur de la place qui lui est réservée. En particulier, elle va exécuter le programme P^2 d'où ce programme P^1 va tirer son P-ième digit-diagonal-inversé. Ainsi se construiront, en miroir, toutes sortes de structures P^j hiérarchiquement imbriquées à chaque fois dans la place qui leur sera réservée (écrivant ces concepts dans des zones d'écritures toujours plus éclatées et lointaines avec un temps d'exécution toujours plus lent).
Pour l'instant, il ne s'agit que d'une approche conceptuelle, elle rencontrera bien des difficultés pratiques ultérieurement. Mais cela étant perçu, centrons notre attention sur le fameux P-ième digit et son contenu. Selon l'organisation retenue pour le déroulement de ces concepts, on peut envisager de nombreuses possibilités. En voici trois caractéristiques :

P-ième Digit non défini

La solution qui correspond à la description la plus commune, c'est qu'à chaque étage pour écrire le P^j-ième nombre diagonal, on attend la réponse de la sortie numéro P du programme P^{j+1}. Et ainsi de suite en miroir selon une descente infinie. Cette attente infinie produit qu'aucun nombre diagonal P^j ne sera jamais écrit. Aucune case réservée pour ces nombres ne sera affectée. Nous sommes dans la situation d'une machine de Turing qui bloque et n'écrit pas son digit-diagonal.
Nous avons proposé deux stratégies d'interprétation : si les places sont initialisées avec des X, c'est pour signifier « non écrit ». L'interprétation naturelle est que ce nombre contradictoire n'existe pas. Si les places sont initialisées par des 0, c'est pour définir le « résultat ». Comme un 0, mais c'est un choix interprétatif.
Comme souvent dans ces plongements constructifs, le paradoxal se transforme en une suite de calculs en miroir à l'infini. Si l'on considère que la valeur n'est pas définie, la question du P-ième digit-diagonal n'a pas de sens. Il n'y a plus de paradoxe de Cantor, et le paradoxe de Richard vu sous cette interprétation se dégonfle par le fait que la diagonale-inversée n'existe pas.

P-ième Digit convergent ou non convergent

On peut construire un autre système, un peu moins naturel. Etant donné que toutes les positions sont connues à l'avance par la gestion des places, on peut demander à notre programme P d'initialiser la valeur de sa p-ième case diagonale et d'y écrire un digit choisi arbitrairement. Sachant que la case ne sera plus réécrite à cause de l'attente en miroir, cette initialisation respecte la contrainte de n'écrire qu'une seule fois sur le ruban. On doit alors imaginer que cela se reproduit à chaque strate P^j. On peut alors envisager plusieurs situations :

- Le programme inscrit le même digit à toutes les strates. On peut supposer que l'initialisation n'a pas besoin d'être recopiée au niveau supérieur puisqu'elle est un choix initial et non le résultat d'un calcul. On peut donc désactiver la recopie dans le cas d'une initialisation. Dans ce cas, le digit écrit en initialisation devient le digit-diagonal. Vu qu'il est toujours le même, on ne peut pas vraiment dire qu'il est l'inverse de la diagonale de la strate inférieure. Mais d'un autre côté comme il n'est qu'une initialisation, on peut le considérer comme une interprétation légitime. Cette valeur de sortie ne sera inverse de la diagonale que par attente, mais pas dans les faits. Vu que malgré l'attente, il n'y a jamais eu de contact avec la réalité inférieur sensé lui donné sens, il y a une rupture entre le concept d'inverse et les fait où il n'y aucun lien d'inversion. Même si cela semble artificiel, c'est symptomatique de la construction de la diagonal de Cantor dans son approche conceptuelle naïve qui ayant dit « on prend l'inverse » imagine son existence et son accès. Par contre, pour rendre la méthode un peu crédible, il fallut développer toute une dialectique scénaristique.
- On peut aussi désactiver l'écriture unique et activer le recopiage inverse supérieure, de sorte que le digit va se recopier sans cesse de chaque nouvel étage inférieur vers tous les niveaux supérieurs. Et comme le digit initialisé est toujours le même, l'inversion produit une alternance dans les niveaux supérieur. Et on obtiendra un digit qui varie sans cesse sur chaque niveau. Ce qui n'est donc pas un digit-diagonal « bien défini », il n'a pas la stabilité minimale requise.

- Maintenant, si on choisi d'initialiser les digit-diagonaux avec des digits alternés selon les niveaux, il y aura toujours incessante recopie alternée vers les niveaux supérieures, mais plus d'alternance sur chaque niveau donné qui sera stable. Par contre, il faut bien qu'il s'agisse du même programme P à chaque strate (pour cela il suffirait en initialisant de laisser une marque sur une place précise du niveau inférieur lui permettant d'inverser l'initialisation, étant 0 par défaut au premier niveau. Cette marque serait lue et alternée à chaque niveau pour produire le choix d'écriture). Cette fois les digits seraient stables et conformes à notre souhait. Comme il y inversion pour chaque strate, le paradoxe diagonal de Cantor est résolu sans contradiction et avec n'importe quelle valeur souhaité arbitrairement. Evidemment ce choix d'initialiser et en alternance est peu crédible interprétativement, mais en cachant cela derrière une gestion structurelle, on pourrait presque le rendre naturel...

Tout cela est bien sûr très artificiel. Mais permet de considérer la complexité conceptuelle potentielle qui réside derrière le paradoxe de Cantor.

Rappelons cependant que nous n'étudions pas spécifiquement Cantor et sa contradiction, mais le paradoxe de Richard qui pose une question précise : « a-t-on construit une nouvelle suite ou non ? ». Dans la dernière proposition, la plus stable, la notion de «même suite » est ambigüe. Selon qu'on la comprend conceptuellement ou qu'on regarde les valeurs. Conceptuellement, c'est construit pour être la « même suite», mais les P-ième digits-diagonaux sont différents. Donc P n'est pas la « même » suite. Ce qui n'exclue pas, d'ailleurs, qu'une autre colonne, qu'un autre programme, produisent le résultat diagonal à l'identique. En effet, on peut parler d'un autre programme P' qui ferait exactement la même chose que P à l'exception de l'initialisation du P-ième digit-diagonal du niveau zéro à 1 au lieu de 0. Alors P' produira au premier niveau P'[1] la même sortie que P au niveau zéro. Ces deux programmes sont en miroir pour leur résultat au niveau de leurs P-ième digit-diagonaux avec tous les autres identiques. Autrement dit, la construction de Richard n'a pas construit une nouvelle suite. (En fait, notre discours porte ici sur des concepts d'apparence simple, mais c'est beaucoup plus subtil qu'il n'y parait. Il faudrait définir l'action « d'initialisation » de façon fine pour avoir une réponse univoque, parce qu'il y a beaucoup de possibilités derrière ces seuls mots. Pour y parvenir, il faut que le programme de gestion globale observe les deux programmes P et P', et que chacun d'eux observent l'autre comme lui-même et qu'ils interagissent spécifiquement ensemble parce que chacun des programmes doit initialiser lui et l'autre à l'envers. De plus, seul l'un des programmes peut tourner au niveau zéro de la machine de Turing. L'autre devra juste être observé extérieurement au niveau inférieur. On peut dire que cette gestion simultanée n'est pas un procédé pour le moins très « naturel ». Ainsi la diagonal-inversée existera déjà, mais à une autre place et conceptuellement).

Avec un peu de recul, on comprend que le problème n'est pas celui des digit-diagonaux, mais d'abord celui des concepts de lecture.

Approche d'un programme P en langage évolué

Présentation

Afin de rendre un peu crédible le programme P (indépendamment des options possibles), proposons quelques éléments s'approcher de notre objectif. On se limitera à la structure générale de gestion sans entrer dans les détails très techniques.

Cherchons à entrer dans une programmation conceptuelle de P. Le programme P est sensé simuler toutes les machines de Turing. La tentative de contradiction repose sur «l'exécution de lui-même ». La conception de P nous invite donc à être attentifs à l'exécution de P par lui-même. Pour réaliser cela, on a plusieurs options :

Contrôle direct

Un programmeur chevronné aurait la tentation de réaliser un programme de gestion globale. Mais ce programme qui contrôle tous les autres, c'est P. Ainsi ce serait P qui gère, observe et manipule tous les P^j en dessous de lui. Mais il y a plusieurs façons de gérer les P^j ; en direct ou différée.

En direct, cela signifie que P construit lui-même la manipulation des sous-programmes P^J à tous les niveaux. Pour faire cela, P doit séparer l'exécution de son propre programme (le nombre P) des autres numéros de programme. Il va gérer la totalité de l'exécution des P^J de façon récursive et complète alors que les autres programmes seront exécutés sans autre forme de conceptualisation (en dehors d'être un nombre exécuté par une machine de Turing universelle). Mais il y a deux interprétations de cette pratique de contrôle complet:

- D'un côté, on peut dire que P aura bien exécuté « tous » les programmes. Mais tous ne seront pas simulés de la même façon; parce qu'on veut observer le déroulement de la partie sensible.
- De l'autre côté, on peut dire que P n'aura pas simulé tous les programme. A aucun moment, il n'aura simulé lui-même. Il aura juste exécuté une partie de son propre programme récursivement. C'est sensé donner l'illusion qu'il se simule. Mais c'est lui et non une simulation de lui qu'il aura exécutée. C'est d'autant plus problématique que dans une gestion globale de l'exécution des Pi, il y aura des variables globales, ou des remontées de résultats gérées globalement qui seront donc exécutées par P. Si P gère tout, peut-on vraiment dire que les Pi exécutés en dessous sont des reproductions de P. Les reproductions en dessous sont amputées de la gestion globale. S'agit-il encore de P ?

  L'approche des P^j par un programme unique avait l'avantage d'un contrôle interprétatif simple et total (de première main). Mais de l'autre côté, ce n'est pas sans difficultés. Pour prendre le contrôle de ses répliques à exécuter, P devra « se » reconnaitre dans ses répliques. Ce qui n'est pas facile, on en reparlera.

1er niveau répliquable

Le fait que les répliques de P ne sont pas vraiment la totalité de P, nous empêche des les identifier par leur numéros. Cela nous incite à produire une autre solution : une méthode où P ne gère que le premier niveau de la récursivité. Les niveaux inférieurs qui reproduisent P devront être autogérés (parce la conception de P produit cette gestion) et se répliquer eux-mêmes de proche en proche, pour que P soit réellement « identique » à chaque niveau. Cette solution impose que les P^J des niveaux inférieurs sont sensés agir selon tous les concepts utilisés par P. Aucun concept ne doit y échapper. Et réciproquement, rien de ce qui gère les P^i inférieurs ne doit être absent de P. Comme le programme P du niveau 0 est la seule machine de Turing non simulé et que lui-même ne transmet aucune information à un autre programme, cela signifie qu'aucune remontée d'informations ne peut franchir les niveaux du bas vers le haut. Seul P peut venir chercher de lui-même des informations dans ce qu'il exécute. Aucune lecture hors de P, aucune lecture globale d'information, aucune gestion commune ne peuvent avoir lieux. C'est P qui gère tout ce que fait P^1. On a une relation de « domination » : chaque P^J sera le seul à agir sur P^{J+1} et lui-même complètement actionné par P^{J-1}. Aucune exception, sous peine de ne pas se conformer au concept de « programme qui se réplique ».

Maintenant, on a deux espaces de liberté qui correspondent aux deux traductions nécessaires à une telle conception :

- C'est la façon de transformer les concepts de P décrit dans le langage évolué en fonction de transitions d'une machine de Turing, une foule de nuances et de choix interprétatifs sont possibles.
- P doit lui-même contenir une machine de Turing universelle pour exécuter les programmes. La liberté dans les choix de construction de cette machine est importante. Chaque concept évolué peut être traduit par une structure librement choisie. Et cette machine de Turing universelle est un module indépendant de la construction des programmes d'où la possibilité d'une construction assez libre.

La séparation entre ruban de travail et ruban de sortie est fondamentale. Le ruban de sortie permet d'isoler le résultat à distance des structures utilisées pour le produire. Or c'est sur la construction du résultat qu'on fait

porter le paradoxe de Richard, pas sur le programme qui permet de produire ces résultats. Un ruban unique, rendrait plus difficile l'observation du résultat de la machine de Turing. Pour distinguer l'écriture sur le ruban de sortie de la manipulation du ruban de travail, il faut être certain que la machine possède les moyens d'écrire et de se déplacer sur chaque ruban de façons indépendantes. On pourrait choisir des états spécifiques à l'écriture et au déplacement sur chaque ruban; mais c'est contraignant pour la programmation. Il est plus simple d'étoffer la fonction de transition qui produira pour chaque transition une écriture et un déplacement éventuels pour chaque ruban (mais une lecture sur le seul ruban de travail). Le programme P devant simuler son exécution devra intégrer ce fonctionnement dans son module de simulation (la machine universelle à exécuter les programmes). Parce dans les niveaux inférieurs la distinction des rubans repose sur des places réservées.

On va utiliser la récursivité dans de multiples sous branches. Elle doit être intégrée à une procédure multitâche pour ne pas être bloquante. La séparation de P, et de ses répliques inférieures, nécessite une gestion du multitâche qui se reproduit à l'étage inférieur (comme tous les autres concepts d'ailleurs).

Tout notre raisonnement est présenté ici à haut niveau structurel, mais la machine de Turing est un processus élémentaire au niveau de l'écriture, des choix de transition et des déplacements. Pour concevoir une machines de Turing universelle, il est nécessaire de prendre en compte cette dimension élémentaire. Toutes les fonctions simulatrices devront être exprimées par rapport à ces mécanismes élémentaires. Nous n'iront pas au bout de la construction technique mais tenterons de laisser un aperçu de l'organisation qui rend sa réalisation crédible.

Remarques

-Dans notre syntaxe l'indentation sert de marquage aux blocs d'exécution.
-Pour réaliser ce programme, il faudrait développer une gestion des entiers non limités que nous plaçons tacitement sous silence.
-Des variables structurelles sont nécessaires pour accompagner la gestion multitâche du programme pour pouvoir passer d'une machine à l'autre. Il faut conserver en mémoire l'état des machines (Etat_num, PositionTrav_num, PositionSor_num). Elles auront besoin d'être stockées globalement pour passer d'un déroulement à l'autre. Cela nécessite une gestion globale du multitâche pour chaque niveau donné. L'exécution sur les niveaux inférieurs sera gérée aux niveaux inférieurs.
-<Séparation> est le nom d'une constante formée d'une suite de digits réservés aux marques de séparations comme nous l'avons évoquée précédemment.
-Une foule de détails importants sont omis dans cette ébauche (calcul des positions, stockage des nombres sous contrainte des valeurs de séparation, accès à la fonction d'état par le multitâche,…). Cela signifie assurément qu'il faudrait beaucoup plus de techniques pour arriver à un résultat effectif.

Niveau en jeux

Il y a tellement de niveaux interprétatifs en jeu qu'il est un peu difficile de s'y retrouver face à notre objectif : concevoir un programme P permettant d'exécuter tous les programmes possible. Les niveaux en jeu sont multiples, je propose de leur donner un sigle :

- PE : Le programme en langage évolué (dont nous ne produisons qu'une ébauche ici).
- PF : Le même programme sous forme d'une fonction de transition. On peut la réaliser à l'aide d'un compilateur automatique. Elle correspond à la construction d'un schéma binaire.
- PB : Le même programme sous forme d'une suite binaire finie qu'on pourrait placer au début du ruban d'une machine de Turing universelle qui saurait l'exécuter. Cette suite est la traduction d'une fonction de transition en langage binaire. On peut imaginer un compilateur automatique pour la réaliser. Ce

plan binaire doit correspondre au concept de la machine de Turing qui le traite. Chaque machine de Turing universelle demande une traduction spécifique. Pour l'essentiel cette suite binaire est un codage de la fonction de transition de PF. PB est le « programme » qu'on mettrait au début du ruban, si on choisissait d'exécuter P par une machine universelle. En tant que suite binaire, PB peut aussi être vu comme un nombre.

- PN : PE possède un module réalisant une machine de Turing universelle élaborée qui simule les machines de Turing à partir de leur plan PB. Ces simulations ont lieu sur des places réservées (une infinité de rubans infinis sont simulés). PN est la suite des digits qui forment le plan (du même type que PB) exécutable par cette machine de Turing universelle élaborée en PE. PN « est » une suite de digit, donc aussi un nombre formé par cette suite, et en même temps c'est un programme lisible par la machine universelle.
- PN^{niv} : le même programme que PN, installé à un niveau récursif inférieur pour chaque valeur du niveau. C'est-à-dire sur des cases éclatées qui simule un ruban .
- R : le ruban de travail de la machine qui exécute P.
- R': le ruban de sortie de la machine qui exécute P.
- R_{num} : la place réservée sur R pour le ruban de Travail de la machine numéro num.
- R'_{num}: la place réservée sur R pour le ruban de Sortie de la machine numéro num.
- R^*_{num} : la place totale réservée sur R pour faire fonctionner le programme num. Dans cette place, il y aura les deux rubans intriqués (parce qu'infinis) R_{num}, R'_{num} et la place technique nécessaire à la gestion. Le programme num (le nombre num) sera placé au début du ruban de Travail R_{num}.
- On retrouve les équivalents de tous ces rubans (R^*_{num}, R_{num}, R'_{num}) inclus dans le ruban de travail R_P qui simule à nouveau tous ces mécanismes. Et la même chose à chaque niveau récursif par l'intermédiaires des rubans de travail d'où les notations : R^{*niv}_{num}, R^{niv}_{num}, R'^{niv}_{num}.

Bien sûr, dans notre programme, on ne trouvera aucune trace des concepts récursifs (PN^{niv} ,R^{*niv}_{num}, R^{niv}_{num}, R'^{niv}_{num}). Pour la simple raison que le programme doit suffire, lui-seul, à générer ces concepts. Seul le premier niveau est explicitement traité.

Liste des principales procédures utilisées

Fonction [type vide] P //La fonction principale
Fonction [type vide] Exécute_en_multitâche(boucle) abrégé en MT
//On ne précisera pas comment est réalisé ce multitâche forcément complexe.
Fonction [type vide] Ecrire_le_programme_num_en position (num)
Fonction [type vide] Exécute_le_programme_sur_les_positions(num)
//le programme et son exécution se dérouleront sur $R^*_{num.}$
Fonction [type digit] Lire_un_digit_en_position_k (k) abrégé LC
//k parcours le tout le ruban R (sous-entendu : tout le ruban R^{niv}_P du niveau en cours)
Fonction [type vide] Ecrire_un_digit_en_position_k (d,k) abrégé EC
//k parcours le tout le ruban R (sous entendu : …)
Fonction [type vide] Ecrire_un_digit_Sort_en_position_s (d,s,num) abrégé ES
//s parcours le ruban sortie R' (sous-entendu : tout le ruban R'^{niv}_P du niveau en cours)
Fonction [type vide] Ecrire_une_Sortie_SuP_en_position_k (d,k) abrégé ESS
//k parcours le tout le ruban R' (sous entendu : …)
Fonction [type entier] Position_k_du_ruban_de_sortie (k,num) abrégé PRS
//la sortie est exprimée par rapport à tout le ruban R (sous entendu : …)
Fonction [type entier] Positions_du_programme_num (num)
//la sortie est exprimée en position sur le ruban R (sous entendu : …)
Fonction [type entier] Position_k_du_choix_num (k,num) abrégé PKC
//k désignant une position sur R^*_{num} (sous-entendu R^{*niv}_{num})
//La sortie est exprimée en position sur le ruban R. (sous entendu : …)
Fonction [type entier] Position_k_du_ruban_de_travail (k,num) abrégé PRT
//la sortie est exprimée par rapport à tout le ruban R (sous entendu : …)
Fonction [type entier] Position_k_du_ruban_de_sortie (k,num) abrégé PRS
//la sortie est exprimée par rapport à tout le ruban R' (sous entendu : R'^{niv}_{num})
Fonction [type entier]Taille_du_nombre(n,cor)

```
        //cor prend les valeurs vrai ou faux pour signifié « corrigé » par le codage ou « tel_quel »
Fonction [type digit] i_eme_digit_du_nombre_n (i,n,cor)
Fonction [type vide] EtapeMachine(Position,état)
Fonction [modification des entrées] Cherche_nouvel_Etat(etat,lecture,num,Ecriture,décalage)
         // les valeurs d'entrée seront modifiées par la fonction et accessible après exécution
```

Ebauche du Programme

```
Fonction [type vide] P
        MT(Pour num =0 , augmentant par pas de 1
                [Création de variables technique :]   //pour la gestion multitâche.
                        Etat_num ; PositionTrav_num ; PositionSor_num
                Ecrire_le_programme_num_en position(num) //lecture-écriture 'corrigée' de num
                Exécute_le_programme_sur_les_positions(num)
        )

 Fonction [type vide] Ecrire_le_programme_num_en position (num)
        Pour i=1 à Taille_du_nombre(num,corrigé)
                EC(i_eme_digit_ du_nombre_n (i,num), Positions_du_programme_num (num) +i-1)
        Pour i=1 à taille_du _nombre(Separation,non_corrigé)
                EC(i_eme_digit_ du_nombre_n (i,num),
                        Positions_du_programme_num (num)+ Taille_du_nombre(num,corrigé)+i-1)

Fonction [type vide] Exécute_le_programme_sur_les_positions(num)
        Répéter sans fin
                EtapeMachine(PositionTrav_num,etat_num,num)
        //pas de multitâche car les valeurs sont en dépendance chronologique

Fonction [type vide] EtapeMachine(PositionTrav,etat,num)
        Lecture= LC(PRT(PositionTrav,num))
        etat=Cherche_nouvel_Etat(etat,lecture,num,Ecriture, Décalage, EcritureSor,DécalageSor)
        Si EcritureSor <>vide //état specifique d'écriture sortie
        Alors ES(EcritureSor, PositionSor,num)
        Si PositionSor+DecalageSor>=0 alors PositionSor := PositionSor +DecalageSor
        DeplaceSor_vers(PositionSor)
        Si EcritureSor <>vide
        Alors EC(Ecriture,PRT(PositionTrav,num))
        Si PositionTrav+decalage>=0 Alors PositionTrav := PositionTrav +decalage
        Deplace_vers(PRT (PositionTrav,num))

Fonction [type vide] Ecrire_un_digit_en_position_k(d,k)  abrégé EC
        Deplace_vers(PKC(k,num))  //nombre de pas à effectuer
        Ecrirecase(d)  //procedure finale d'écriture

Fonction [type vide] Lire_un_digit_en_position_k (k)  abrégé LC
        Deplace_vers(PKC(k,num))  //nombre de pas à effectuer
        Lirecase -> positions de variables de retour

Fonction [type vide] Ecrire_un_digit_Sort_en_position_s (d,s,num)  abrégé ES
        Deplace_vers(PRS(k,num))  //nombre de pas à effectuer
        Ecrirecase(d)  //procedure finale d'écriture
        //surveillance de l'écriture diagonal à chaque écriture
         Si (s=PRT(num,num) )  //reconnaissance d'un digit-diagonal
        Alors ESS(1-d,num) )  //recopie la valeur inversée sur le ruban de sortie du niveau supérieur
```

Analyse

La fonction d'où démarre l'ensemble du programme est P, naturellement (Qu'il soit compilé pour une fonction de transition ou pour le ruban d'une machine de Turing universelle).

La question qui prime en regardant ce programme, qui est aussi la question à la base de sa conception est : que va donner P lorsqu'il se simulera lui-même au travers de son mécanisme d'exécution universelle interne ?

La fonction ESS est la seule à se dérouler en dehors du ruban de travail, parce qu'elle ne travaille pas sur la simulation des machines, mais qu'elle écrit sur le ruban de sortie un résultat. Cela ne pose pas de problème particulier, car la fonction de transition gère déplacements et écritures sur les deux rubans. Donc le module « machine de Turing universelle » contenu dans P doit être aussi capable de simuler une écriture sur les deux rubans. En particulier, il doit savoir coder l'écriture sur les rubans de sortie lorsqu'il simulera cette fonction ESS.

La procédure non explicitée <Déplace_vers(k)> est stratégique. Elle établit le nombre de pas à réaliser et les réalise un par un. Ces pas sont gérés uniquement pour le premier niveau. Pour les niveaux suivants, c'est la machine de Turing universelle qui lisant les déplacements devra les interpréter un par un et les adapter aux positions de la simulation. Pour que la machine de Turing universelle puisse simuler une machine complète sur un ruban qui n'et pas connexe, il faudra qu'elle gère des déplacements distants. C'est donc une attribution centrale de la machine universelle de transformer le déplacement d'un pas en déplacements par rapport à des positions réservées. Et puisque chaque programme tourne par la simulation d'une machine de Turing universelle au dessus de lui, cela se répercute de proche en proche. Les seuls « vrais pas mécaniques » sont ceux du niveau 0 qui donnent mouvement à tous les niveaux (qui devront être regardés interprétativement). A chaque niveau inférieur, une quantité de déplacements toujours plus importante sera vue comme mouvements fonctionnels. Et seul l'accès à la case réservée, à la fin de ce déplacement, sera vu comme le mouvement simulé. Pour une même exécution, chaque niveau interprétatif produit une lecture différente : plus le niveau est bas, plus rare sont les déplacements simulés par rapport aux mouvements réels.
Pour réaliser, ces déplacements conceptuellement imbriqués, la machine de Turing universelle doit être basée sur la notion de déplacement et de places réservées. Le concept de déplacements élémentaires doit être un fondement conceptuel de la machine universelle. Chaque déplacement élémentaire doit être interprété puis simulé par de nombreux déplacements structurels entre des positions réservées à la simulation du ruban concerné. Chacun des concepts qui forme la machine de Turing (fonction de transition, ruban de travail, de sortie) doit être transformé en des concepts sur des positions manipulable par la machine universelle. D'étage en étage, le résultat d'un « déplacement à droite de la tête» se transforme en une fonction récursive de déplacements imbriqués dans les niveaux supérieurs. C'est à la machine supérieure d'interpréter sa lecture à chaque niveau. Les concepts qui font tourner une machine universelle quelconque ne conviendrait pas nécessairement pour simuler ces déplacements récursivement interprétés. Il faut que la lecture des déplacements possède une interprétation transportable de niveau en niveau. Cela doit être réalisé avec les seuls concepts qu'utilise P. Il faut que se crée une imbrication capable de produire une interprétation qui s'imbrique dans un plan de lecture global.

On a vu que les nombres doivent couvrir surjectivement les machines de Turing. Mais que se passe-t-il si une machine (un nombre) n'est pas la traduction d'une machine de Turing. Il importe que la machine Universelle soit conçue pour parer à cette éventualité. Si un nombre ne correspond pas au codage attendu pour simuler une machine de Turing, il ne faut pas que son exécution en tant que programme perturbe le déroulement des autres programmes. Il faut que son comportement face à ce nombre-programme reste cantonné à son ruban réservé, sans rien altérer aux autres rubans, ni à l'organisation multitâche, ni aux valeurs structurelles des autres programmes. On doit veiller à ce que la machine universelle garantisse cette stabilité de fonctionnement. Pour cela, il faut produire des sécurités sur les zones de déplacement et des sécurités sur les actions à générer suite à la lecture d'un codage non conforme.

Comprenons biens que les différentes options fantaisistes que nous avons parcourues pourraient être programmées ici, en modifiant légèrement le programme P à différents endroits. La solution retenue est celle qui s'impose comme la plus naturelle : l'exécution imbriquée, sans ajout d'interprétations spécifiques. L'exécution de P par lui-même sera placée au même niveau que les autres exécutions. Aucun concept particulier d'initialisation ou de lecture spécifique pour qualifier le programme d'une interprétation supplémentaire.
En conséquence, l'interprétation de P est assez classique : le p-ième digit-diagonal ne devrait jamais exister parce qu'il ne sera jamais écrit, il dépend d'une attente qui se transporte à l'infini sans jamais avoir de retour.

Surveillance du P-ième nombre diagonal

Le programme que nous avons conçu possède un suivi des digit-diagonaux, mais pas un suivi spécifique du p-ième digit qui est sensé porter la contradiction. Aussi tout le commentaire que nous avons fait sur ce p-ième digit est issue d'une observation extérieure conceptuelle.
Si conformément à nos intentions, on souhaitait effectuer une surveillance objective de la construction de ce p-ième digit, il faudrait reconnaitre parmi tous les programmes qui tournent en multitâche lesquels sont le programme P (de même pour ses descendants). Une telle volonté nécessite de contrôler les numéros en cours d'exécution et de les comparer à P. Il faudrait inclure dans le programme une gestion qui effectue des actions effectives d'observation. On pourrait modifier P de la sorte:

```
Fonction [type vide] P
        MT(Pour num =0 , augmentant par pas de 1
                [Création de variables technique :]   //pour la gestion multitâche.
                        Etat_num ; PositionTrav_num ; PositionSor_num
                Ecrire_le_programme_num_en position(num) //lecture-écriture 'corrigée' de num
                Si num='P' //comparaison nécessaire, mais délicate à réaliser
                Alors Exécution_surveillé_du_programme_sur_ses_positions(P)
                Sinon Exécute_le_programme_sur_les_positions(num)
        )
```

Mais P est à la fois nombre et programme. Ici 'P' est un nombre binaire à comparer. Comment peut-on comparer le nombre-P à l'intérieur du programme P ? En effet, le nombre P est écrit avec un certain nombre de digit précis. Si, à l'intérieur de lui-même, il doit se reconnaître et se comparer pendant le défilement d'autres valeurs, il faut nécessairement que le nombre-P prennent moins de place que le programme P. Il faut que P puisse s'exprimer plus courtement que lui-même et à l'intérieur de lui-même.
Ce n'est pas nécessairement une équation impossible à résoudre. Gödel en a résolu une similaire. Mais un tel procédé devra passer par un codage pour être réduit (et un décodage pour être reconstruit), c'est-à-dire par une interprétation. Il faudra reconstruire P à partir d'un P condensé. L'équation semble assez difficile à résoudre : placer P à l'intérieure de lui-même parmi tout un programme complexe. On possède certaines libertés sur les formulations de P, on peut même ajouter des parties de programme « ne servant à rien » ou à rien d'autre que cela. Mais restreint à ces constats, c'est un peu maigre pour atteindre l'objectif. (P possède beaucoup de particularités comme « être égal au nombre inscrit sur le ruban de travail » à qui ont peu le comparer, mais ce n'est pas suffisant pour caractériser P, il faut une caractérisation unique du programme P spécifiquement choisi.)
Pour construire une observation de P, il faut un traitement spécifique à ce programme qui l'isole des autres programmes qui ne seront pas 'observés'. Il faut l'accompagner d'une structure d'exécution spécifique et de concepts structurés pour l'observation, qui seront traduite récursivement.
Même si l'on parvenait à résoudre l'équation, le fait de devoir isoler spécifiquement P dans une exécution personnalisée par rapport aux autres programmes le distingue complètement des autres. En attribuant un traitement particulier à P, ce traitement devient un concept central du programme entier. Il ne s'agit plus du

même programme. C'est un programme qui distingue lui-même et les autres machines de Turing. L'exécution de P par lui-même ne peut être vue « comme les autres » qu'au prix d'un effort interprétatif peu naturel. Car l'exécution du programme, le remplissage des cases, les concepts partageant les places seront centrés sur P qui ne s'organisent pas « librement », mais qui est encadré selon une forme imposée pour s'observer sous les critères désirés.

C'est le problème de l'observation. Intégrer une observation au programme change le programme. Quand l'observation devient un centre de gravité du programme, reste-il pertinent de le voir toujours comme le programme initial ? Ces remarques ont des relents importants de « mécanique quantique » où l'observation est indissociable de l'objet à observer. L'observation perturbe la réalité de façon non négligeable. Ce n'est pas un concept habituel en mathématiques, mais on constate que l'observation transforme profondément la réalité en jeu. L'observation d'une réalité infinie demande d'ajouter une opération en multitâche à l'intérieur du programme, changeant par là-même sa nature. C'est la volonté de plongement linéaire univoque comme accès à la réalité univoque est à l'origine de ce constat. Soit on réalise une observation extérieure conceptuellement sans garantie d'univocité, soit on l'intègre à l'objet et il devient différent.

Après toutes ces remarques, quand on parle du programme P, duquel s'agit-il ? Il y en a tellement d'options à cause de la foule des concepts en jeu, et de leur subjectivité.

Fabrication de plan de construction ou de valeur?

Il existe aussi un problème de polysémie : il y a plusieurs façons de voir ce que construit le programme P.

- Pour chaque valeur de num, on peut voir la construction de la machine de Turing comme la manipulation d'une suite de digit sans concept (chaque programme num correspond à un traitement sur le ruban R^*_{num}), et on n'a pas de concept plus profond à observer : P produit en multitâche des manipulations groupées sur des rubans.
- Ou bien au contraire, on voit dans ce ruban R^*_{num} un nouveau programme de construction. Et l'on distinguera plusieurs concepts différents sur ce ruban : un programme de construction, des calculs et le résultat produit par ce programme de construction. Ce n'est plus le même chose. A partir du moment où on considère que le ruban R^*_P est un « programme de construction de beaucoup de suites », un changement d'observation s'impose : on distingue naturellement <ce qui construit>(le programme) et <ce que construit ce qui construit>(le résultat), mais plus <ce qui est construit> de façon uniforme (les digits de R^*_p qui cache le tout). C'est la polysémie résultante « d'une construction de <ce qui construit un résultat> ».

On est face au même problème interprétatif que dans de nombreux problèmes diagonaux. Comment faut-il interpréter les nombres, comme des résultats ou des constructeurs de résultats ? C'est à cause de cette difficulté que nous avons choisi au départ de séparer le ruban de travail et de sortie, ce qui impose une interprétation séparée entre le résultat (le ruban de sortie) et sa construction (le ruban de calcul). Mais l'interprétation récursive ne s'impose pas. A chaque étape constructive, on peut choisir de dissocier ou non les concepts construits.

Dans le paradoxe de Cantor, présenté de façon naturelle par un tableau, on est face à des cases où l'on identifie colonnes et diagonale. L'interprétation nous invite à lire en colonne des suites de digits sans autre approche conceptuelle. Derrière ces suites, on ne lit pas de programme de construction. Mais comment une suite infinie de digits peut-elle avoir du sens à elle seule ? Comment pourrait-on construire un objet infini univoque sans le mettre sous forme d'un programme initial fini par un procédé de construction élémentaire explicite ?

Si on veut voir les définitions, il faut les exposer comme telles. C'est tout l'intérêt du cadre des machines de Turing, elles peuvent gérer à la fois programmes et résultats. Face à cette machine qui définit toutes les suites, on voit la nécessité d'y accéder par leur programme fini. Le plongement dans un cadre univoque rend incontournable le plan de construction (le programme).

Selon qu'on s'intéresse à la machine R^*_{num} ou au résultat de la machine (R'_{num} inclus dans R^*_{num}), le P-ième digit du ruban n'est pas le même. Les informations définitionnelles de R^*_{num} décalent la position des résultats vers la droite, le p-ième digit de R^*_{num} n'a plus aucun rapport avec le concept récursif attendu.

Notre esprit raisonne par concepts ; c'est une pratique naturelle que « construire jusqu'à l'infini» puis lire la diagonale après coup. Mais les concepts infinis imbriqués dépendent d'un multitâche quand on veut atteindre une réalité univoque. En définissant conceptuellement une suite infinie sans autre forme d'accès à cette suite, comment pourrait-on la reconnaitre uniquement par ses digits ? Il faut plonger le paradoxe de Richard dans un cadre qui accède aux définitions pour comprendre que le problème d'une suite est d'abord celui de la définition de cette suite.

La contradiction de Richard porte sur une construction conceptuelle autant que sur des digits égaux ou non. L'importance de la contradiction diagonale pâlit du moment qu'on s'intéresse aux définitions. Bien sur toute lecture des concepts dans une machine de Turing est interprétative. L'identification de P à son alter-égo inférieur peut être contestée pour de nombreuses raisons. (Avec quoi identifier deux programmes : par comparaison des concepts de construction, (comprenant l'ordre, les manipulations intermédiaire ou non), par comparaison des écritures sur le ruban de sortie (comprenant l'ordre d'écriture, la possibilité d'écriture intermédiaire ou non)… Face à notre construction où les concepts sont interprétables pas à pas et l'écriture des sorties aussi, on peut raisonnablement parler d'une identité de fonctionnement entre P et sa simulation à l'étage inférieur. Et dans cette optique le digit contradictoire n'a pas de sens puisqu'il n'est jamais écrit.

Et même avec un peu de fantaisie, certes moins pertinente, il n'est pas impossible de construire le digit-diagonal problématique. Pour « voir », il faut accepter un regard conceptuel et donc interpréter les nombres avec un sens ajouté aux procédés constructifs.

La construction des « toutes les suites » n'est pas une construction de digits par essence, mais d'abord une construction de « concepts pour construire des digits» augmenté de concepts si on le souhaite et si on le peut... Qu'est-ce qu'une suite binaire : « une suite infini de 0 et 1 » ou bien un « algorithme » qui construit une suite de 0 et 1 ?

Une nouveauté ?

Le paradoxe de Richard plongé dans un cadre univoque nous éclaire sur le processus diagonal de Cantor. Quand Cantor nous dit : « face à une énumération de suites infinis, je peux construire une autre suite », je réponds : non, pas systématiquement. Notre construction atteste de la subjectivité de l'affirmation. Si aucune précision sur le processus de construction des suites n'est donnée, si le problème n'est posé que par la contrainte libre d'être des suites, il y a tellement d'ambigüité dans le discours, dans les possibilités de construire qu'il est difficile d'y souscrire. En posant le raisonnement diagonal face à une forme contrainte sans autre précision, le raisonnement s'apparente plus à un choix péremptoire d'être « conceptuellement autre ». Il n'y a pas de sens général à une telle proposition. Certes c'est pertinent dans certains contextes, mais pas assez précis pour être recevable dans un cadre vague.

A quoi correspond la construction d'une diagonale-inversée face à l'infinité des machines de Turing ? Revenons en à la question de Richard : est-on capable de construire une diagonale-inversée qui soit autre chose que du « déjà construit » ? Face à notre cadre complet des machines de Turing, si on se permet une lecture interprétative des calculs (et comment pourrait-on faire autrement) il semble clair que non. Il n'y a pas de contradiction sur le P-ième digit de la diagonale-inverse, les deux concepts qu'on cherche à mettre en contradiction ne permettent pas une construction effective. La définition montre une inadéquation de principe à fournir un résultat pertinent.

Quand on prend du recul, c'est la définition qui est contradictoire : dans un cas il est nécessaire de construire d'abord le nombre diagonal pour pouvoir ensuite l'inverser (le nom de diagonale-inversée en atteste). De l'autre côté, la notion de diagonale parle aussi d'une construction postérieure aux colonnes. Cela signifie qu'on lit les valeurs construites par une observation réalise après la construction des colonnes (du moins après leur

valeur-diagonale). Comment peut-on définir un tel monstre en une seule réalité univoque (non dynamiquement) : un digit qui se construit systématiquement après lui-même et différent de lui-même. La définition pose problème si on considère l'objet comme préalablement existant à son observation. Autrement si on construit une définition à partir de toutes les définitions. On peut bien sûr trouver une foule d'interprétation qui rend la définition constructible. On retrouve ici l'analyse que nous avons faite de la diagonale de Cantor. Les concepts servant à la contradiction posent plus de problèmes que la construction elle-même.
Voici donc résolu le paradoxe de Richard : on peut réellement construire toutes les définitions de calcul (du moins sous le regard de Church) et ne pas avoir de contradiction sur la rencontre de la diagonale inversée et de la diagonale. Cette couche définitionnelle, si souvent rendue invisible, nous en montre le déroulement.
Par contre la notion de définition n'est pas forcément clarifiée car l'interprétation des machines de Turing conçue lexicographiquement n'est certainement pas facile d'accès, voire peut-être impossible. Sans ajouter que les machines qui concluent par un nombre diagonal « bien défini » ne sont pas visibles, d'autant moins que « bien défini » ne l'est certainement pas lui-même.

Tout de même de la nouveauté

Dans notre approche constructive du problème, on voit là nécessité d'accéder aux suites par leur plan, c'est particulièrement incontournable quand on cherche une totalité. Mais dans bien des circonstances, face à des situations manifestement « bien définies », on pourra remplir un tableau uniquement de « résultats » sans subjectivité. Dans ce cas, il est clair que la méthode anti-diagonale construira bien une « suite de résultats différente des résultats écrit en colonne». Même s'il ne s'agit là encore que d'un regard interprétatif, il serait regrettable d'exclure ce type de pratique bénéfique a bien égard, à l'instar de toutes nos mathématiques conceptuelles. Par contre, il est important d'en considérer les limites, en particulier face au « tout imaginable ». Il est utile d'en percevoir la dimension fortement conceptuelle : observer une colonne infinie est un acte conceptuel.
Le concept de diagonale-inversée n'est pas forcément clair. Dans les situations non univoques, il ne reste qu'à raisonner à vue avec toute la subjectivité qui s'attache à certains concepts. Cette remarque nous invite à produire une pression univoque : chercher à observer la subjectivité sous un cadre toujours plus univoque, jusqu'à l'accès à une certaine pertinence.

Dès qu'on face à des « formes libre » (des situations où les possibilités de définitions sont établie sous forme de contrainte avec une potentialité interprétative sans limite), on raisonne comme si les objets était bien définis dès qu'ils semblent « interprétativement compatibles avec cette forme». Sans pouvoir accéder à la notion de compatibilité autrement que par l'évidence qui ne possède pas de frontière nette avec l'impertinence. Cette remarque n'a rien d'anecdotique car la plupart de nos ensembles (entier, réel, espace,…) sont des formes libre définies par contrainte sous « compatibilité ». Tout nos discours débutant par « soit x un réel… », « soit n un entier … » sans autre forme de définition ont de quoi interroger sur leur subjectivité …

Conclusion

Essayons de rassembler quelques idées saillantes des difficultés interprétatives que nous avons rencontrées au cours de cette démarche :

Concepts mentaux ou constructifs

Dans toutes les pratiques mathématiques, on a l'habitude d'accepter des concepts mentaux dès lors qu'ils supposent un support univoque (en particulier des schémas, des valeurs binaire, des tableaux,… qui semblent exempts de polysémie). C'est le cas de la diagonale de Cantor vu sous la forme d'un tableau. Ces concepts

mentaux, traduits dans un cadre linéaire, nécessitent une matérialisation. En observant comment le paradoxe de Richard porte son estocade, on constate que les opérations mentales trouvent bien leur appui sur de opérations localement constructives. Par contre, c'est rarement un programme constructif linéaire complet. Face à des difficultés interprétatives, il n'est pas souhaitable de raisonner sur des concepts abstraits en rupture les uns avec les autres. L'effort de plonger l'ensemble du problème dans un programme linéaire unique impose la mise relief des liens entres les concepts en rupture. Et ces liens constructifs qui unissent les concepts a priori séparés nous indiquent les lieux de subjectivité.

Intrication des concepts matérialisé

Pour matérialiser l'observation de concepts infinis à l'intérieur d'un programme linéaire unique, il est nécessaire de produire des intrications « multitâches » entre la construction et l'observation. L'intrication modifie l'apparence de l'objet construit parce qu'elle a besoin de place pour être écrite. Mais surtout parce qu'elle pose des concepts de lecture qui modifie la structure en lui imposant une forme. Tout étant mélangé, l'observation de l'objet devra donc être réalisée interprétativement (« mentalement » par la connaissance du plongement) en distinguant ce qui revient à l'objet et ce qui revient à l'observation. Cette pratique nous permet de prendre la mesure d'univocité qui réside dans nos concepts.

La conceptualisation permet d'enfermer l'infini en une fois de façon mentalement indépendante, elle a le sentiment de considérer l'objet dans sa totalité et s'intéresser au résultat « après toute la construction ». Cela donne une dimension au raisonnement que le calcul ne peut pas atteindre. C'est un mode de raisonnement mental très classique. A moins que ce ne soit, en même temps, une perte d'univocité qui donne accès à la manipulation d'une réalité subjective et attribue une perception instinctive à des réalités non précisément définies.

Définition face à une non-terminaison

La question de la définition se pose. On a vu qu'il peut exister plusieurs modes de bonne définition possibles : la finitude des calculs, l'écriture unique garantie par un déplacement unilatéral, l'écriture finale garantie d'une valeur (pas d'écriture postérieure par-dessus), l'écriture stable (pas de modification de la valeur même en cas d'écriture), etc… et l'on a déjà abordé la difficulté de concevoir ce genre de problème face à une infinité de valeur.

La notion de « bonne définition », possède une dimension subjective. L'infini pose problème. Car un processus infini est toujours en action. Sans connaissance spécifique sur le processus, le cadre de construction linéaire ne peut pas garantir une « bonne définition ». Conceptuellement, on peut s'imaginer mentalement des cases, des colonnes, des liens de reconnaissances, etc, mais dans toutes ces pratiques, on ne peut garantir la bonne définition des contenus. Un programme possède une dimension dynamique irréductible qui ne peut garantir la stabilité du contenu de la construction.

Parler de toutes les suites de $\{0,1\}^N$ pose le problème du bon sens de cet objet :

- En général, on l'utilise à chaque fois que peut être évoqué une compatible avec une suite de valeur binaire. Mais l'idée de compatibilité n'est pas bien définie.
- Est-ce que cela évoque toutes les constructions possibles. Mais alors lesquels, les effectives ? Qu'est-ce que cela signifie en dehors de « je vois bien que ça marche ». Mais alors par Cantor, on trouve la possibilité de reconstruire par l'extérieur différent de tout ce qui a déjà été construit. Il y a une tension contradictoire dans l'air dans la totalité.
- Est-ce que cela évoque les possibilités constructives. Mais les machines de Turing, qui en sont un cadre général, ne peuvent garantir que le programme d'une suite est bien défini. Et même plus encore : une machine de Turing peut ne pas fournir les moyens de lire un nombre. Détaillons cela.

La possibilité de donner du sens.

Nous avons présenté l'interprétation comme un mécanisme de base des mathématiques humaines : lire autre chose que ce qui est strictement écrit. Pour lire des concepts, il faut interpréter des groupements qui ne sont pas forcément marqué dans une machine. Pour que notre programme P processus linéaire unique produise « tous les concepts », il est nécessaire d'interpréter l'écriture de la machine de Turing à partir de groupement qui repose sur la connaissance des positions d'écriture pour y lire les concepts en jeu. Mais quand on y réfléchit, il s'agit là d'informations ajoutées. Les groupements ne sont pas dans la machine. Rien n'empêche que la machine de Turing puisse être lue sous une autre interprétation différente.

On constate l'existence d'une tension permanente entre l'univocité et la conceptualité ; entre la notion d'objet conceptuel ayant un contenu sémantique mental et la notion de lien univoque se limitant aux liens univoques symbolisés neutre d'interprétation ajoutée. La notion même d'objet flirte avec la question d'univocité. La réalité univoque linéaire d'un objet ne contient pas toute l'information de l'objet. Un objet plongé dans une dimension linéaire a besoin de concepts de lecture pour être perçu et suivi. Ce phénomène est caractéristique dans la notion de « codage ».

Les mathématiques pour créer des structures ont besoin de reconnaitre conceptuellement des parties d'objets. Or cette notion semble subjective. Lorsqu'un objet complexe est programmé sur un ordinateur, tous les liens entre les « parties conceptuelle» sont fixés de façon univoque dans un processus linéaire qui existe indépendamment de son interprétation. Le lien entre la perception humaine et le résultat des calculs rend cette subjectivité quasi-réelle. Lorsqu'on accède conceptuellement aux parties isolées (sans qu'elles soient reliées par des liens univoques à l'ensemble de la structure), on produit un groupement sans univocité, soit une subjectivité.

La notion d'isolement d'une partie est pourtant fondamentale à nos pratiques mathématiques. On a de cesse d'isoler des parties d'objets, soit mentalement, soit par des procédés localement constructifs. Mais c'est en même temps le mécanisme humain de l'accès à l'univocité qui en fait sa force. Car c'est sur ces constats d'isolement de parties que naissent les possibilités d'interprétation entre objets et donc que prend forme tout l'édifice mathématique. Autant les manipulations que les structures statiques.

Totalité et rupture

« Si on pouvait énumérer toutes les suites bien définies, on pourrait en construire une nouvelle ». Finalement Richard n'avait pas vraiment tord. Intuitivement, ça tient presque la route après notre analyse. On pourrait cependant modifier légèrement la phrase : « si on énumère des suites assez bien définies, on peut en construire une nouvelle ». Par contre, on a gagné sur la notion de définition et de totalité. La totalité mécanique (l'ensemble des machines de Turing) n'est pas « bien définie » en tant que résultat, ou alors par des interprétations plutôt « scabreuses». Et si on voulait des suites pertinemment « bien définies », il faudrait se limiter. On trouve la frontière du problème, elle possède deux natures : l'interprétation et la complexité qui se conjugue d'ailleurs. La « totalité des définitions » est probablement un concept un peu présomptueux. Bien pratique pour parler de compatibilité, mais très subjectif, car en rupture avec les constructions effective.

Cantor et richard : même combat.

Cantor et Richard, vu sous notre approche, forme l'étude du même phénomène, la recherche de la totalité des suites infinies binaires. Cantor propose un cadre mental conceptuel, alors que le cadre des machines de Turing, présenté ici, propose un cadre constructif linéaire. Après cette étude dans le cadre de Turing, beaucoup de commentaires sur Cantor prennent sens. Ce cadre qui découpe les mécanismes en pratiques élémentaires, permet d'observer à distance les pratiques effectué dans le cadre de Cantor. Et l'on y retrouve les problèmes d'interprétation, le problème d'ordre de construction non défini.

Dans Cantor, le concept subjectif de « cases bien définies » est sous-entendu. Ce concept apparait plus précisément dans le cadre de Turing dans la question de « l'écriture stable des cases », ce que j'ai appelée

aussi la « bonnes définition ». Il faudrait disposer de la sous-suite des « bonnes définitions des suites possible » parmi les machines de Turing. Ce serait une condition nécessaire pour établir le projet de Richard dans de bonne condition. Seulement, on a compris qu'elle n'est pas accessible, ni en complexité (par l'analyse de ce qui produit des blocages), ni en sémantique (par l'immensité des possibilités d'interprétés comme satisfaisant) et qu'il serait en plus contradictoire (par la diagonal de Cantor qui en produirait un de plus…). Et c'est sans parler du problème de multiples interprétations d'une même construction.
Ce cadre nous montre plein de lieux de fuites sémantiques qu'une approche conceptuelle ne pouvait pas détecter, par essence. Notamment, la temporalité constructive d'une définition présentée par deux contraintes incompatibles : il n'y a pas forcément contradiction mais écrasement temporel successif des valeurs. Pour une définition atemporelle des mêmes contraintes, on obtient une contradiction, une absurdité. Ce qui semblait conceptuellement univoque dans Cantor, ne l'est plus forcément dans Turing. La notion de colonne, de suite binaire, de diagonale,… tout cela possède une dimension subjective.

La conclusion que j'ai tirée de Cantor est que l'on peut toujours reconstruire au-dessus ce qui est construit. Quand on plonge ce concept dans un cadre Turing-linéaire, on en voit :

- la difficulté : elle ne concerne que les cadre bien définit et compatible. Ce qui malgré l'aspect relativement subjectif est suffisant pour nous guider dans les cas pratiques usuels.
- le prix : à chaque observation, il faut redéfinir tout le cadre. L'objet, comme l'observation, prennent une dimension interprétative (parce qu'au total les deux forment un unique objet matérialisé sur le ruban de Turing).
- Si l'on face à un ensemble des possibles, construire un « différent de chacun » est soit contradictoire soit l'identification à un objet dynamique qui s'éloigne sans cesse de lui-même.

C'est le constat qu'« une observation univoque coûte en construction ». C'est le constat qu'il n'est pas possible de dire « j'ai fini et je n'ai pas fini » de façon univoque. C'est ce que disait à sa façon Poincarré : on ne peut pas établir une liste d'étiquette et ne pas les fixer pour manipuler.

Quand on connait l'importance du mécanisme de la diagonale de Cantor dans la culture mathématique classique, où par lui sont construit tant de concepts structurants de nos conceptions habituelles, nonobstant le manque de rigueur de mon discours et les lieux techniques laissés dans l'obscurité, cette question reste une position stratégique.