Thématisation et philosophie de l'informatique

Baptiste Mélès

Thématisation et philosophie de l'informatique

Baptiste Mélès

Journée d'hommage à Jean Cavaillès,
École Normale Supérieure,
17 février 2014

1 De la pratique à l'universel

1.1 Vertu logique et vertu éthique

Pour commencer, j'aimerais exprimer envers la Société des amis de Jean Cavaillès ma reconnaissance, mais surtout ma surprise, pour l'honneur écrasant qu'elle m'a fait en m'invitant à cette journée — honneur, ou plutôt quiproquo, que je sais apprécier en proportion inverse de mon mérite.

Il y a vingt ans jour pour jour, dans la salle Cavaillès de la Sorbonne, et en des circonstances analogues à celles qui nous réunissent aujourd'hui autour de Jean Cavaillès, philosophe et résistant, Élisabeth Schwartz disait de sa pensée qu'elle était, « autant qu'un legs, une promesse fusillée dont nous savions porter le deuil chaque fois que nous entrions dans cette salle Cavaillès, ou celle de l'École Normale ». Sachez que ces mots peuvent être repris à l'identique par la seconde génération qui n'a pas eu la chance de connaître Cavaillès, qu'elle ait ou non pu fréquenter les salles portant son nom sur la montagne parisienne où il se distingua comme philosophe, ou sur les monts d'Auvergne où il se révéla comme résistant. Peut-être est-ce ce sentiment de gravité qui explique le décalage entre le respect dont on témoigne envers la pensée de Cavaillès et la parcimonie avec laquelle on la relaie.

De Cavaillès, on peut énoncer sans exagération ce que, avant Spinoza, les Stoïciens disaient du sage et de sa vertu : « La vertu prise dans son unité et à son degré le plus élevé est utilité, et les vertus [...] sont trois — vertu physique, vertu éthique, vertu logique ». Canguilhem et d'autres grands commentateurs de Cavaillès ont décrit le soin qu'il prenait à montrer, en théorie comme en pratique, l'unité des vertus logique et éthique — cette même unité des vertus logique et éthique qu'il nous arrive, philosophes des sciences, d'oublier en théorie, et parfois en pratique.

1.2 Théorie et pratique

Or cette continuité entre théorie et pratique traverse l'ensemble de la vie comme de l'œuvre de Jean Cavaillès. À l'instar de son camarade Albert Lautman, Cavaillès semblait en effet tenir pour évident que l'on ne pouvait raisonner sur les mathématiques sans les avoir d'abord apprises, en s'étant humilié à calculer des intégrales.

Cette modestie de la pratique n'empêchait pour autant en rien les plus hautes ambitions intellectuelles. Devant la Société française de philosophie, Cavaillès proclamait en effet :

> « Je ne cherche pas à définir les Mathématiques, mais, au moyen des Mathématiques, à savoir ce que cela veut dire que connaître, penser ; c'est au fond, très modestement repris, le problème que posait Kant. La connaissance mathématique est centrale pour savoir ce qu'est la connaissance ».

Ce souci du pratique, associé aux ambitions les plus spéculatives, montrent à quel point Cavaillès était un penseur concret, au sens que Hegel a donné à ce terme. Non point un philosophe du concret dans le sens où il se serait défié du général, comme l'attestent de façon suffisamment éprouvante la difficulté de ses textes et la patience qu'ils exigent de son lecteur ; mais Cavaillès est un penseur concret dans le sens où l'universel, bien loin d'être un point de départ, est surtout le résultat d'un processus.

Ce processus apparaît sous sa première forme comme le devenir même des concepts mathématiques, qui tendent vers l'universel. Mais ce processus est également celui qu'a suivi Cavaillès au travail, s'appliquant patiemment à apprendre les mathématiques là où elles s'enseignent — en Sorbonne — puis là où elles se découvrent — à Göttingen — afin d'atteindre par lui-même le stade où est parvenue l'âme mathématique du monde. Ce processus est, enfin, le travail que le lecteur de Cavaillès doit accomplir pour son propre compte s'il espère approcher le résultat atteint par les deux précédents.

L'universel n'est donc pas un point de départ, mais d'aboutissement. Il n'est pas un moyen d'éviter le concret.

1.3 Paradigme et thématisation

Il n'y a donc pas de paradoxe à ce que Cavaillès cherche la généralité la plus élevée dans la pratique la plus concrète — la sienne comme celle des mathématiciens. C'est parce que, selon ses mots, « l'activité des mathématiciens est une activité expérimentale », c'est-à-dire que l'expérience des mathématiciens est un « système de gestes », de procédés, qui sont transversaux aux circonstances particulières de leur application.

Deux de ces procédés, qu'il appelle « polymorphes » dans l'ouvrage posthume *Sur la Logique et la théorie de la science*, revêtent une importance particulière : le paradigme et la thématisation. Ces deux concepts joueront

un rôle de premier plan pour l'objectif qui est le nôtre, de voir dans quelle mesure l'enseignement de Cavaillès peut être prolongé dans la philosophie de l'informatique.

Il y a, d'abord, passage au paradigme quand un concept est élargi de sorte que les constantes mobilisées dans sa définition se transforment en variables. Cavaillès donne à plusieurs reprises l'exemple des généralisations successives de la notion de nombre, dont à chaque fois les exemples précédemment connus s'avèrent toujours n'être rétrospectivement que des cas particuliers. Les objets ainsi obtenus se prêtent, comme les précédents, à des actes, et ces actes sont la généralisation, aussi naturelle que possible, des actes antérieurs : l'addition de nombres entiers est, par exemple, prolongée par l'addition de vecteurs. Ainsi, dans le passage au paradigme, le degré de généralité des concepts varie, mais les objets conservent leur nature d'objets, et les actes leur nature d'actes.

Il en va autrement de la thématisation, second grand procédé identifié par Cavaillès. Il y a thématisation quand un acte mathématique est soudain considéré comme n'étant qu'un objet, et devient, à ce titre, passible d'actes de niveau supérieur. Cavaillès mentionne souvent l'exemple de la « topologie des transformations topologiques » : les actes de la topologie deviennent eux-mêmes les objets d'une topologie. On peut également penser à la représentation matricielle des applications linéaires, qui permet de voir ces actes que sont les transformations de vecteurs comme s'ils étaient à leur tour des vecteurs, c'est-à-dire des objets, pendant que la composition des applications — acte sur des actes — devient de ce fait une simple multiplication de vecteurs — acte sur des objets. À la différence du paradigme, qui transforme les actes en actes, la thématisation transforme les actes en objets.

On voit donc que ces deux gestes fondamentaux de l'expérience mathématicienne peuvent eux-mêmes être décrits par les interactions de deux concepts que Cavaillès a repris, en les détachant, de la phénoménologie husserlienne : les concepts d'acte et d'objet. Cavaillès semble indiquer que l'histoire des mathématiques, dans sa « dialectique », progresse à sens unique dans le sens du paradigme et de la thématisation.

À dire vrai, Cavaillès envisage également une sorte de procédé inverse de la thématisation, mais simplement sur le mode du « principe de réductibilité » qu'il attribue à Husserl. Selon ce principe, les constructions abstraites sont toujours en dernière instance convertibles en constructions concrètes, apparemment sur le seul mode du retour à l'origine. Redescendre l'arbre de l'abstraction ne pourrait être que revenir à des connaissances du passé, modèles particuliers — et réducteurs — des théories présentes. De plus, ce principe conférerait à la conscience un lieu absolu, celui des connaissances portant sur le « sensible primitif [1] ». La critique du principe de réductibilité est ainsi essentielle à Cavaillès pour montrer au contraire la mobilité de la

1. MAF, p. 179.

conscience qui se transforme selon les « moments [2] » d'une dialectique.

L'enseignement méthodologique qu'il nous paraît fécond de retenir de Cavaillès en philosophie des sciences est ainsi le suivant : partant d'une expérience directe de la science, s'interroger sur les procédés généraux de la pensée en recherchant l'éventuelle nécessité sous l'ordre apparemment contingent de l'apparition des concepts. Cavaillès disait ainsi devant la Société française de philosophie :

> « [Le devenir des mathématiques] est autonome, c'est-à-dire que, s'il est impossible de se placer hors de lui, on peut, en étudiant le développement historique, contingent, des Mathématiques, tel qu'il se présente à nous, apercevoir des nécessités sous l'enchaînement des notions et des procédés. [...] J'ai essayé, pour ma part, de le faire pour la théorie des Ensembles »

Cette recherche de nécessité est ce qui pousse Cavaillès, comme le fera plus tard son élève Jules Vuillemin, à mobiliser les pensées de Descartes, de Leibniz et de Kant pour analyser les conflits qui déchirent les mathématiciens de leur époque. L'auteur de la *Philosophie de l'algèbre* n'eût en effet pas désavoué la formule de son maître : « l'histoire montre la liaison étroite entre semblables conflits techniques et les systèmes édifiés par les philosophes [3] ». Les deux auteurs inclinaient à voir de l'ontologie dans les sciences pures comme dans les grands systèmes, les unes comme les autres étant des produits purs de la raison.

Cavaillès et Vuillemin partageaient ainsi l'idée désuète selon laquelle la pratique des sciences pures et l'histoire de la philosophie pouvaient avoir des choses à nous apprendre l'une sur l'autre — au mépris pur et simple de toute classification institutionnelle de la recherche et de l'impératif de lisibilité des profils universitaires.

2 Philosophie de l'informatique concrète

2.1 Qu'est-ce qu'un système d'exploitation ?

Nous essaierons de montrer que la méthode de Cavaillès peut être transposée à une certaine philosophie de l'informatique. Pour cela, nous analyserons les concepts de fichier et de processus tels qu'ils furent systématisés dans le système d'exploitation Unix, et nous montrerons à partir de là ce que ces concepts peuvent apporter à la réflexion sur le paradigme et la thématisation comme procédés fondamentaux de la pensée. En d'autres termes, nous allons lire Unix comme si c'était du Dedekind.

Les systèmes d'exploitation sont, de très loin, les programmes les plus ambitieux que l'on puisse écrire pour un ordinateur. Parmi les plus cé-

2. LTS, p. 78.
3. MAF, p. 21

lèbres, quoique de qualité diverse, figurent Unix, Linux, MacOS et Microsoft Windows. Le système d'exploitation est le programme chargé de manipuler toutes les ressources et de veiller à la sécurité du système ; il contrôle tous les calculs envoyés au processeur, détermine comment les données peuvent être rangées sur les disques, écoute toutes les informations provenant des périphériques, et permet l'exécution de tous les programmes lancés par l'utilisateur. Il est l'administrateur exclusif du matériel de notre ordinateur.

Le système d'exploitation doit en particulier connaître dans les moindres détails le contenu matériel de l'ordinateur. Il doit savoir que l'ordinateur est une machine de Von Neumann, c'est-à-dire qu'il est fondamentalement composé d'une mémoire où les données sont rangées à certaines adresses, d'un processeur qui contient lui-même des registres pour ranger les opérandes, une unité arithmétique et logique capable d'effectuer les plus élémentaires des opérations booléennes et arithmétiques, et une unité de contrôle capable de déterminer l'ordre d'exécution des instructions. Le système d'exploitation doit être capable de parler le langage du matériel.

Mais le système d'exploitation présente également un tout autre visage : celui, simple et parfois convivial, de l'interface qu'il offre à l'utilisateur. Tout en cachant l'ontologie matérielle qu'il manipule en réalité, il présente à l'utilisateur une ontologie fictive, généralement plus proche des modes humains de représentation, composée de fichiers, de répertoires, d'invites de commande, d'icônes, de fenêtres. L'utilisateur humain peut ainsi laisser au système d'exploitation le soin de traduire l'ontologie de second niveau qui lui est familière dans une ontologie de premier niveau dont il ignore tout.

Le système d'exploitation assure ainsi la médiation entre le matériel présent dans le boîtier et les intentions de l'utilisateur humain. Il tient lieu, pour le matériel comme pour l'utilisateur humain, d'interlocuteur exclusif, proscrivant au passage toute communication directe entre eux. Il a quelque chose du Dieu de Malebranche. Nous autres humains ne connaissons généralement l'intérieur du boîtier que par cette sorte de vision en Dieu. Fort heureusement, la théologie qui correspond à ce Dieu peut aisément devenir une science, pour peu que le code source du système d'exploitation soit public, et si possible libre, afin que l'on puisse lire le texte qui le décrit exhaustivement. Ce n'est pas le cas de la majorité des systèmes d'exploitation commerciaux, qui se condamnent ainsi à la théologie négative ou à la foi.

Nous prendrons comme objet privilégié la version 6 du système d'exploitation Unix, qui date de 1975. Cette version est à la fois complète et minimale : son code source, aujourd'hui public, tient en seulement 9000 lignes, ce qui lui permet d'être appréhendé dans sa totalité par un lecteur unique. Nous verrons, de plus, qu'Unix a incarné l'un de ces tournants historiques qui ont irréversiblement renouvelé les concepts fondamentaux des systèmes d'exploitation.

2.2 Passage au paradigme : fichier et processus

Toute étude rigoureuse des systèmes d'exploitation, telle qu'on la trouve par exemple dans l'excellent *Operating Systems* d'Andrew Tanenbaum, commence par exposer les concepts de fichier et de processus : « Les appels système [...] se répartissent en gros en deux grandes catégories : ceux qui se rapportent à des processus et ceux qui se rapportent au système de fichiers ».

Dans le cadre d'un exposé technique, il suffit que les notions semblent ainsi tomber du ciel, sans que l'auteur ait à justifer d'où elles viennent. Mais un tel empirisme ne saurait satisfaire le philosophe. Il aimerait savoir pourquoi l'exposition des systèmes d'exploitation passe par ces notions précises, et si cela a toujours été le cas ; il aimerait savoir si quelque nécessité se cache sous ces airs de contingence.

Pourquoi le processus et le fichier sont-ils donc les notions fondamentales de ces programmes eux-mêmes fondamentaux que sont les systèmes d'exploitation ?

2.2.1 Fichier

Commençons par la notion de fichier. Elle n'est pas une invention d'Unix, puisqu'elle apparaissait déjà au moins dans le système d'exploitation CTSS en 1961. Un fichier désignait alors une suite de données commençant à une certaine adresse d'un périphérique de stockage donné, et occupant une longueur déterminée. Au premier abord, le code source de la version 6 d'Unix ne semble guère innover, puisque la structure de données associée aux fichiers contient toujours, parmi ses principaux champs, le périphérique, l'adresse, la longueur. Pour Unix comme pour les systèmes d'exploitation des années 1960, les fichiers sont ainsi des objets, éventuellement longs, que les programmes peuvent manipuler.

Mais ce qui va changer avec Unix est l'extension de ce concept. Les concepteurs d'Unix, Ken Thompson et Dennis Ritchie, ont fait un pas conceptuel radicalement nouveau en posant non seulement que tout fichier était un objet, mais surtout que *tout objet, c'est-à-dire tout ce que peut manipuler le système d'exploitation, aurait dorénavant le statut de fichier*. Pour Unix et pour toute sa descendance — Linux, BSD, MacOS, etc. — non seulement vos documents de texte, vos vidéos et vos images sont des fichiers, mais votre imprimante, votre scanner, votre processeur, votre écran et vos enceintes sont aussi des fichiers.

Cette décision relève d'un authentique choix conceptuel et non d'un simple choix technique. Elle a pour conséquence que, pour le système d'exploitation, « imprimer un texte » ne sera plus rien d'autre qu'écrire dans un fichier, en l'occurrence le fichier spécial de l'imprimante. Seul le pilote de l'imprimante, petit bout de programme écrit par son fabricant, saura à quoi renvoie ce fichier spécial et comment il convient de traduire les instructions

reçues. Numériser un texte, ce sera lire un certain fichier. Unix a ainsi supprimé tout ce qu'il y avait d'*ad hoc*, de contingent et de particulier dans la gestion du matériel informatique. Il n'y a plus pour lui que des fichiers, que l'on peut lire et écrire au moyen d'une seule et même syntaxe absolument générale.

Les concepteurs d'Unix ont ainsi généralisé le concept de fichier, par un passage au paradigme au sens de Cavaillès. Non seulement tout fichier est objet, ce qui allait de soi, mais surtout tout objet est fichier, ce qui relève d'une décision.

2.2.2 Processus

Qu'est-ce maintenant qu'un processus ?

En guise d'approximation, nous pourrons définir le processus comme une occurrence d'exécution d'un programme. Par exemple, si j'ouvre un navigateur pour consulter une page, je crée un processus, c'est-à-dire une occurrence d'exécution, et si je l'ouvre une deuxième fois en parallèle, je crée un second processus à partir du même programme.

Mais la définition du concept de processus pose de vrais problèmes, y compris aux informaticiens les mieux informés, et ce pour la simple et bonne raison qu'elle ne relève pas de la technique. John Lions — auteur en 1976 d'un commentaire du code source de la version 6 d'Unix aussitôt reconnu comme classique, et qui a longtemps circulé sous le manteau avant d'être enfin distribué officiellement — a bien vu que le problème de définition du processus dépassait la simple technique et relevait de la philosophie :

> « Fournir une définition généralement acceptable de la notion de “processus” soulève de nombreuses et de sérieuses difficultés. Elles s'apparentent à celles qu'affronte le philosophe voulant répondre à la question “qu'est-ce que la vie ?” ».

Nous avons déjà vu que les fichiers étaient invisibles dans le boîtier. Mais pour les processus, le mystère se redouble : ils sont non seulement invisibles dans le boîtier, mais ils le sont même dans le code source d'Unix ! On y trouve tout au plus une table des descripteurs de processus, qui contient leurs propriétés, mais ne les contient pas eux-mêmes — tout simplement parce que ce ne sont en rien des objets que l'on pourrait stocker.

Essayons toutefois de deviner leur nature d'après les quelques propriétés que nous pouvons en connaître. Nous voyons que les processus possèdent notamment un certain identificant numérique, se trouvent à chaque fois dans un certain état (en cours de création, d'exécution, de sommeil, de terminaison), et gardent la trace sous forme d'identifiant de leur processus père, c'est-à-dire du processus qui l'a créé. On peut ainsi savoir que le processus est à chaque fois un acte engendré par un acte antérieur (un autre processus), selon un arbre d'engendrements. Cet arbre a une racine, autrement

dit un premier moteur : le processus 0, occurrence d'exécution du système d'exploitation.

Il est important de souligner que le processus est une abstraction, un simple concept. Au sens propre, les processus n'« existent » pas. On ne les trouverait nulle part en scrutant au microscope l'intérieur du boîtier. On n'y verrait jamais que des opérations logiques et arithmétiques élémentaires effectuées sur des données à chaque fois élémentaires. Les processus sont une fiction du système d'exploitation pour rendre intelligibles, et manipulables en bloc, de grands paquets d'opérations et de données. Pour la machine ils n'ont pas d'existence concrète, et surtout pas spatiale.

Et pourtant, du point de vue du système d'exploitation, non seulement on voit des processus agir, mais, comme l'écrit Andrew Tanenbaum, « les seules entités actives d'un système Unix sont les processus ». Il n'existe aucune opération élémentaire du processeur qui ne soit adoubée par le système d'exploitation comme faisant partie d'un processus. Les processus sont ainsi très exactement la notion d'acte inscrite dans les systèmes d'exploitation depuis Unix. Aussi rigoureusement qu'Unix avait décrété l'équivalence entre objet et fichier, il a posé l'équivalence entre acte et processus.

John Lions n'était donc pas si mal avisé, laissant la question aux philosophes, d'apparenter la question « qu'est-ce qu'un processus ? » à « qu'est-ce que la vie ? ».

Si l'on entend maintenant par ontologie la *théorie de ce qui est*, le seul et unique engagement ontologique du système d'exploitation Unix et de tous ses descendants est donc que *que tout ce qui existe est fichier, et que tout ce qui agit est processus.* Le couple que forment fichiers et processus, et que l'on apprend dans les manuels, ne tombe pas du ciel, mais d'une ontologie strictement composée d'actes et d'objets. Nous insistons sur le fait que cette ontologie simple et élégante relève véritablement de concepts, et non de choix purement techniques faits par des ingénieurs forcément bornés. C'est en effet par un choix conscient d'universalité que les concepteurs d'Unix ont généralisé les concepts déjà existants de fichier et de processus.

Cette ontologie est une façon de penser ce qui se passe dans un ordinateur ; et cette façon de penser, une fois formalisée de façon détaillée dans un code source que l'on compile ensuite en un programme exécutable, influence en retour la façon dont les choses se déroulent effectivement dans l'ordinateur. L'ordinateur ne peut le faire que parce qu'une conscience l'a pensé avant lui. C'est en ceci que l'expérience de la pensée informatique est, comme l'expérience mathématique, créatrice de concepts. L'ordinateur n'est que le corollaire matériel de la pensée informatique.

2.3 Thématisation

La pensée informatique, telle qu'elle s'est exprimée par les œuvres des concepteurs d'Unix et de leurs successeurs, sait donc pratiquer le passage

au paradigme, comme l'ont montré les concepts de fichier et de processus.

Une fois posés les fichiers comme objets et les processus comme actes de l'ontologie d'Unix, nous allons voir par différents exemples que la pensée informatique connaît également la thématisation. Il y a déjà thématisation dans l'idée même d'*écriture d'un programme*, c'est-à-dire un groupe d'instructions à exécuter, dans la mémoire. Par l'acte ou la pratique de la programmation, l'informaticien transforme une série d'actes à accomplir en un objet textuel.

Cette idée fut même pour ainsi dire l'acte de naissance de l'informatique. On sait qu'une machine de Turing est une machine contenant un programme qui à une certaine situation — décrite par un état et par la lecture d'un symbole sur un ruban suffisamment long — associe un certain comportement — l'écriture d'un symbole, un déplacement sur le ruban, et le passage à un nouvel état. Dans une machine de Turing en général, le programme de la machine même, que l'on peut noter par exemple sous forme d'un tableau, est toujours extérieur aux objets qu'il se donne. Il est pour ainsi dire « codé en dur », transcendant les données qu'il est autorisé à lire. Il ne peut pas se lire lui-même, ni se prendre pour objet. Mais un programme n'en peut pas moins devenir objet, typiquement pour une machine de Turing universelle ; il suffit de noter le tableau qui le décrit sous une forme linéaire, et de l'inscrire sur le ruban d'une machine que l'on programme précisément pour exécuter ce type de programmes linéarisés. L'idée d'inscrire les programmes en mémoire constitua ensuite le cœur des architectures dites de Von Neumann, qui ont donné naissance à nos ordinateurs.

Il y a thématisation également quand un processus, par un procédé appelé « sérialisation », est figé en un objet, ce qui permet par exemple sa transmission sur un autre support. Il en résulte une ambiguïté dans la notion même de processus, selon qu'elle désigne l'acte en cours d'exécution ou l'objet qui l'incarne. John Lions montrait ainsi que le pseudo-parallélisme d'Unix 6 — c'est-à-dire le fait que l'absence de véritable parallélisme au niveau matériel soit compensée par une alternance dans l'exécution des processus, l'ordonnancement de leurs exécutions respectives étant laissée aux bons soins du système d'exploitation — nous contraignait à distinguer dans la notion de processus un niveau supérieur et un niveau inférieur :

> « Au niveau supérieur, le "processus" est un concept important d'organisation pour décrire l'activité d'un système informatique pris dans son ensemble. [...] À ce niveau, les processus eux-mêmes sont considérés comme étant les entités actives du système [...] : les processus naissent, vivent et meurent ; ils existent en diverses quantités ; [...] ils peuvent interagir, coopérer, entrer en conflit, partager des ressources ; etc.
>
> Au niveau inférieur, les "processus" sont des entités inactives qui sont activées par des entités actives telles que le processeur.

> En permettant au processeur de passer fréquemment de l'exécution d'une image de processus à une autre, on peut créer l'impression que chacune des images de processus se développe continûment, ce qui mène à l'interprétation du niveau supérieur. »

Mais on accordera qu'il ne s'agit là encore que d'un niveau assez rudimentaire de thématisation, dans la mesure où il n'y est question que d'interrompre et de reprendre l'exécution de processus. L'acte a beau devenir objet, cet objet n'est pas véritablement soumis à des actes de niveau supérieur. On n'obtiendrait donc guère plus que ce que Hegel appelait un « mauvais infini » : une répétition indéfinie à l'identique.

Mais le processus une fois thématisé peut devenir objet pour des actes de niveau supérieur, donnant cette fois naissance au « bon infini » d'une réelle progression qualitative de la connaissance. Il est courant de voir des processus dont la fonction est d'écrire des programmes — et, plus encore, des processus écrivant des programmes dont l'exécution écrira des programmes.

De même, les débogueurs sont des programmes capables de figer l'exécution d'un processus pour analyser, étape après étape, chacun de ses états instantanés, donc objectifiés, et par là son comportement général en tant qu'acte.

L'utilisation d'un compilateur n'est autre qu'un processus analysant les propriétés d'un autre processus dont il scrute la forme objectifiée, à savoir le code source.

La hiérarchie peut même être encore enrichie d'un niveau : lorsque Xavier Leroy a certifié en Coq le compilateur C CompCert, il a exécuté un programme dont la fonction était d'analyser à sa façon le texte d'un programme qui lui-même analysait d'une autre façon des programmes. Lorsqu'un assistant de preuve thématise ainsi un compilateur qui lui-même thématise des programmes, la connaissance des actes de niveau inférieur — ici la compilation — progresse véritablement grâce à des niveaux supérieurs qui n'en sont pas le simple décalque.

La thématisation informatique n'est donc pas moins dialectique que celle que Cavaillès a vue dans les mathématiques. La pensée informatique, comme la pensée mathématique, s'enrichit par paradigme et thématisation. Son incarnation matérielle dans des machines ne l'empêche pas d'être une pensée pure.

2.4 L'antithématisation

Mais elle est également capable d'effectuer l'opération inverse de la thématisation, et de transformer ses objets en des actes.

Telle est en effet l'*exécution d'un programme*, lorsqu'un texte enregistré en mémoire est soudain considéré par le système — via un appel système particulier, ou une fonction particulière présente dans la quasi totalité des

langages de programmation— non plus comme un objet existant pour lui-même et passible de certains actes tels que la lecture et l'écriture, mais comme une suite d'instructions que le système lui-même doit exécuter : un texte qui n'est pour lui pas tant à lire qu'à assumer, à interpréter. La pensée informatique ne permet donc pas seulement de transformer des actes en objets par thématisation, mais également de transformer des objets en actes par une sorte d'« antithématisation ».

Ce procédé n'est d'ailleurs guère propre aux informaticiens. Les théoriciens des catégories ne font pas autre chose en considérant tous nos singletons comme autant d'applications de l'élément terminal dans un ensemble donné. Nous croyions que le singleton n'était un objet, et le voici devenu acte. *On peut pour ainsi dire exécuter le singleton.* De même, Ehresmann éliminait de sa présentation de la théorie des catégories le concept d'objet, puisque l'on peut tout aussi bien le représenter par un morphisme identité, c'est-à-dire le transformer en un certain acte. *L'objet en général s'exécute comme un acte.*

Contrairement à la thématisation, qui traite le complexe comme s'il était simple, ce dernier procédé met au jour la complexité cachée dans le simple. Loin d'être condamnée au retour à des objets anciens et bien connus, la « thématisation à l'envers » est parfois l'occasion de découvrir du nouveau dans ce que l'on croyait statique et bien connu. On peut ainsi transposer en mathématiques le constat de François Jacob : « même l'atome, le vieil irréductible, est devenu un système ». Les mathématiciens aussi ont leur microscope.

Si l'on en croit ces exemples récents, ces pratiques de mathématiciens sont apparues dans des décennies que Cavaillès a courageusement accepté de ne pas connaître. Si ces pratiques peuvent au premier abord sembler suivre parfois un cours inverse à celui de la « dialectique » de Cavaillès, elles restent fidèles à son esprit en accréditant la mobilité de la conscience par rapport aux objets qu'elle se donne, par localisation tout autant que par globalisation.

3 Conclusion

Nous espérons avoir ainsi contribué à montrer que l'informatique, science formelle à part entière, n'est pas moins créatrice de concepts purs que les mathématiques. L'examen détaillé des concepts techniques permet parfois de mettre au jour des formes de nécessité conceptuelle qui nous instruisent en toute pureté sur les potentialités de la pensée.

La question de savoir si les ordinateurs peuvent penser sert souvent de prétexte à dégrader l'idée de pensée en la réduisant à ce qu'un boîtier peut réaliser. Il nous semble au contraire que dans l'ordinateur, comme déjà dans la machine de Pascal, il ne faut pas voir un tas de matière brute mais une pensée au sens fort, grosse d'universalité et de nécessité, capable de se ma-

térialiser, mais une pensée d'abord : une théorie pure qui, comme nous y invitent la vie et la pensée de Cavaillès, n'a pas craint d'affronter la pratique.