maire, mais naturel par son vocabulaire) qui pût décrire les documents (et aussi les questions auxquelles ces documents devraient fournir une réponse). Dans une seconde phase ils voulaient construire un programme qui donne à un automate la possibilité, à la lecture du document (ou au moins de son résumé) de construire la phrase ou les phrases qui, dans le langage artificiel ainsi défini, en exprimaient le contenu.

Dans une première étape les langages artificiels furent totalement dépourvus de grammaire. Ils ne comportaient que des *mots clés. Puis* on introduisit des relations qui permirent la constitution de *phrases clés (1).*

Mais la vraie difficulté n'était pas dans la constitution d'une syntaxe artificielle, mais dans la tentative d'exprimer, d'une façon ou d'une autre, la signification et les rapports de signification.

De nombreux essais furent tentés dont plusieurs sont maintenant en application (2).

D'autres essais intéressants sont ceux qui se sont efforcés de manier des fragments très spécialisés du langage naturel, de façon à réduire autant que possible l'étendue du champ sémantique correspondant. On en a vu un exemple dans l'article de Slagle que nous avons cité, à titre d'exemple, dans notre introduction.

Dans tous les cas les travaux en cours sont loin d'être suffisants pour qu'on dispose dès à présent d'un réseau de relations sémantiques vraiment efficace. On peut cependant penser que si les recherches se poursuivent la situation ira en s'améliorant régulièrement.

(1) P. BRAFFORT et A. LETTOV, Des mots clés aux phrases clés, *Bulletin des bibliothèques de France, 1959, P. 383.*

(2) Voir l'article de C. WASTON, Information retrieval, dans *Advances in Computers, 5, 1965.*

CHAPITRE VI

La complexité

Simulation du joueur, du mathématicien (dans certaines de ses activités les plus simples), du sujet parlant (avec également des restrictions), au cours des trois chapitres précédents, nous avons montré comment les automates pouvaient prendre en charge la résolution de problèmes de plus en plus difficiles et que l'on situe ordinairement dans le domaine des activités intellectuelles.

Parvenus à ce point, avons-nous le droit de dire que nous avons effectivement démontré l'existence et l'efficacité d'une « intelligence artificielle » ? Pas exactement : car nous avons pris le soin, pour tous les cas traités, de donner une analyse assez détaillée des programmes que les automates devraient exécuter ; mais, dans la plupart des cas, nous n'avons pas discuté de ce que représentaient quantitativement ces programmes du point de vue de la capacité et de la vitesse dont ces automates devraient être pourvus. Dans plusieurs cas il s'agissait d'ailleurs de schémas de programmes, et non pas de programmes effectivement écrits.

Cela veut dire, certes, que l'intelligence artificielle est possible *en principe.* Et puisque, à plusieurs reprises, nous avons fait allusion à des expériences déjà effectuées (jeu de Go-Bang, démonstrations de propositions logiques, traduction du russe vers l'anglais), nous savons même que ce principe est devenu, en diverses occasions, un *fait.* Mais à aucun moment nous n'avons touché du doigt les *limites* de ce qui est effectivement possible (de ce qui est possible aujourd'hui comme de ce qu'il est raisonnable de prévoir pour demain).

Nous sommes évidemment arrivés au point où une telle estimation devient indispensable. Or, on va voir qu'en s'y attelant on met en valeur une notion qui est commune à toutes les rubriques que nous avons successivement abordées et qui explicite vraiment l'unité du domaine de l'intelligence artificielle : il s'agit de la notion de *complexité.*

A plusieurs reprises nous avons indiqué, soit explicitement, soit implicitement que, dès qu'un problème était complètement formulé - c'est-à-dire dès qu'on disposait d'un système formel convenable dans lequel le problème en question prenait l'aspect d'une formule à découvrir -, on pouvait définir un automate et un programme pour effectuer le travail.

En réalité cette affirmation demanderait à être précisée et complétée car certains énoncés mathématiques (qui appartiennent à ce que l'on appelle le domaine *non constructif)* posent ici des difficultés particulières.

Mais si nous nous bornons (ce qui est parfaitement licite pour les domaines que nous avons abordés) au domaine *constructif,* nous pouvons nous en tenir à ce point de vue qui consiste, en somme, à associer à chaque problème une machine de Turing particulière, pour reprendre le modèle d'automate que nous avons présenté dans le chapitre II.

Nous avions alors souligné que si la machine définie (dès 1936) par Turing n'était guère semblable aux calculatrices électroniques que nous connaissons, rien ne s'opposait à ce que l'on programme une calculatrice moderne pour la faire se comporter comme une machine de Turing si ce n'est le point fondamental suivant : la longueur de la bande utilisée par la machine de Turing pour effectuer un calcul est *illimitée* alors qu'il ne peut évidemment pas en être ainsi d'une machine réelle quelle que soit la nature de la mémoire utilisée pour simuler la « bande» de la machine de Turing.

Et cette remarque fondamentale, c'est au fond la rencontre initiale de ce *mur de la complexité* auquel nous avons fait allusion ici et là, au cours de cet ouvrage.

L'EXPLOSION COMBINATOIRE

« Complexe » se situe quelque part entre «possible » et «impossible», mais c'est « impossible» qui demande tout d'abord à être précisé. Prenons pour cela deux exemples

a) Soit un alphabet formé de lettres *a, b, c...* Nous voulons ranger des mots composés avec des lettres de cet alphabet, dans l'ordre « lexicographique ». Si la longueur des mots que l'on veut classer n'est pas limitée *a priori,* la capacité de l'automate qui effectuera le classement doit être également illimitée. Car si un mot de longueur *n* (très grand) est déjà classé et que survient un second mot de longueur au moins égale à *n,* il est possible que les lettres d'ordre r, *2, ..., n* -1 soient identiques et que la décision n'intervienne que pour la dernière lettre. Il faut donc stocker n lettres en mémoire et effectuer *n* comparaisons. Si *n* est illimité, il en va de même de la capacité qu'on exigera de l'automate, et du temps pendant lequel il devra être utilisé. Cette remarque vaut en ce qui concerne le classement respectif de deux mots. Si l'on veut maintenant insérer un mot dans une liste de mots déjà classés, il faudra répéter cette opération un certain nombre de fois (les mots déjà classés étant stockés en mémoire). S'il y a N mots à classer, de longueur maximum *n,* un calcul simple montre qu'il pourrait être nécessaire d'effectuer

$$n. \sum_{1}^{N-1} m = n \frac{(N-1)(N-2)}{2}$$

opérations élémentaires de comparaison.

Supposons que *n* soit de l'ordre de 10^3 et N de l'ordre de 10^5, le nombre d'opérations sera de l'ordre de 10^{13}, ce qui est en dehors des possibilités des machines actuelles (ou envisageable dans un proche futur), car il faudrait faire travailler une telle machine pendant une année entière sur le même problème de classement.

b) Reprenons le jeu de Go-Bang étudié à la fin du chapitre III. Sur une grille composée de 19 lignes et 19 colonnes, il y a 361 positions. Le joueur qui commence peut donc choisir entre 361 possibilités ; le suivant entre 360. Le premier joue à nouveau et a 359 Possibilités, etc. Si la partie s'arrête au bout de n coups (en totalisant le

nombre des coups joués par chacun des adversaires), le nombre de combinaisons possibles est évidemment

$361.360 \ldots\ldots (361 - n)$ c'est-à-dire $\dfrac{361!}{(361-n-1)!}$

Une partie normale ne dépasse guère une cinquantaine de coups, mais cela donne déjà un nombre de combinaisons qui excède largement 10^{100}

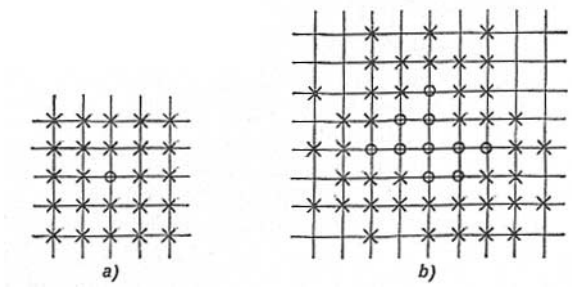


FIG. 31. — Coups envisagés
a) En début de partie ; b) au cours d'une partie

Bien entendu les joueurs sont loin d'exploiter ces possibilités. Le premier place son pion au voisinage du centre de la grille. Le second riposte en se plaçant non loin du premier, de façon à gêner d'éventuelles combinaisons. Supposons alors que chaque joueur n'ait à choisir que parmi les points de la grille qui ne sont pas distants de plus de deux noeuds d'un pion déjà posé (voir fig. *31).*
Il est facile de voir sur la figure (et on peut s'en convaincre plus complètement par le calcul) que le nombre des choix possibles reste supérieur à 20 après chaque coup (et augmente en général au fur et à mesure que la partie se développe). Dans ces conditions les combinaisons possibles, pour une partie d'une cinquantaine de coups, seraient certainement en nombre supérieur à 10^{50}, ce qui reste très au-delà des possibilités d'exploration par un automate.

c) A l'occasion de notre brève introduction à la théorie de la déductibilité, présentée au cours du chapitre IV, nous avons rencontré deux types de fonctions, celles qui appliquent un ensemble E dans

lui-même, et celles qui appliquent l'ensemble E dans l'ensemble B des « valeurs de vérité », ensemble qui comprend seulement deux éléments : { vrai } et { faux }. Si l'on suppose que l'ensemble E contient n éléments et que les fonctions sont à k arguments, on peut montrer que le nombre de fonctions de E dans E différents est égal à n^{nk}, celui des fonctions de E dans B étant 2^{n^k}.
Pour fixer les idées, on peut voir que pour E réduit à trois éléments et $k = 2$, le nombre des fonctions du premier type est déjà *19 683* et celui du second type 512. Or il est clair que pour les recherches pratiques de démonstration de formules tant soit peu intéressantes on doit utiliser des ensembles E beaucoup plus grands et des fonctions de plus de deux arguments.

Il est donc clair qu'ici encore on tombe très vite dans un domaine complètement inaccessible aux automates réels.

Il n'y aurait aucune difficulté, on s'en doute, à développer des exemples du même type pour les problèmes liés au traitement automatique du langage naturel : tous ces exemples sont des manifestations de ce que l'on appelle souvent *l'explosion combinatoire.* Cette expression rend, sous forme imagée, l'impression que l'on ressent lorsqu'on se livre, pour ces divers problèmes, à des calculs d'ordres de grandeur.

Ce qui caractérise, en effet, de tels ordres de grandeur, c'est la *vitesse* avec laquelle se développe l'ampleur du problème lorsque croît le nombre des éléments qui le caractérisent. Pour les problèmes traditionnels, la dépendance est linéaire ou quadratique, c'est-à-dire que lorsque le nombre des éléments croît comme n, la complexité du problème est proportionnelle à n *ou* au carré de n.

Mais dans les problèmes qui nous intéressent ici, la situation est bien différente : les ordres de croissances sont du type 2^n ou n^n ou pire encore ; la croissance est *exponentielle !*
Dans ces problèmes en effet, si on veut se livrer à un examen exhaustif des cas possibles, on doit utiliser, pour dénombrer ces cas, les formules de *l'analyse combinatoire* (élevée au rang de discipline mathématique par Leibniz), formules qui servent à évaluer le nombre de combinaisons de n objets soumis, dans leur arrangement, à des conditions de divers types.

C'est ainsi que si l'on se donne n objets différents et qu'on les dispose en file indienne, le nombre total des files distinctes que l'on peut ainsi constituer (que l'on appelle nombre des *permutations* de la collection) est égal *à n! (1). Or n!* est une fonction de n qui varie très rapidement lorsque n augmente. On en a une idée en utilisant la formule approchée duc à Stirling

$$n! \sim \sqrt{2\pi} n^{n+1/2}$$

pour $n > 10$ $n!$ devient très vite « astronomique ».

Les remarques qui précèdent permettent de comprendre cette expression « explosion combinatoire ». Associées aux exemples qui les ont précédées, elles donnent un aspect plus quantitatif à cette notion de « mur de la complexité » à laquelle il est si fréquent de voir les auteurs se référer.

Cependant il est possible, et utile, d'entrer un peu plus avant dans les détails de quelques problèmes afin de ne pas se contenter de voir naître et croître la complexité, mais de comprendre comment on peut se donner - au moins dans certains cas - les moyens de la *réduire.* C'est ce que nous allons faire à l'aide d'un problème mathématique particulier : le problème des mots.

LE PROBLEME DES "MOTS"

A la fin du chapitre II, nous avons rapidement indiqué comment la notion même d' « écriture », c'est-à-dire du rangement de lettres les unes à côté des autres, pouvait être associée à une structure algébrique particulière, celle de « monoïde ».

Un monoïde engendré par un alphabet, c'est l'ensemble de tous les mots que l'on peut écrire en prenant plusieurs lettres de cet alphabet (avec éventuellement des répétitions).

Soit l'alphabet

$$A=\{a,b,c,d,e\}.$$

(1) Rappelons que n ! (que l'on lit "factorielle n" désigne le produit 1.2.3...(n-1)n.

Les « mots » *aaabb, dbca, b, dbdbbdbd,* etc., font partie du monoïde engendré par A :

Si l'on ne se donne pas d'autre contrainte, le monoïde en question est le monoïde *libre* engendré par A.

Mais on peut s'imposer certaines relations qui *identifient* des mots du monoïde. Il est possible alors de ne considérer qu'un ensemble restreint où, lorsque des mots du monoïde sont identiques d'après l'application des relations d'identification, on les remplace par l'un d'entre eux, ou par un symbole de leur classe.

Par exemple si les relations du monoïde comprennent le groupe :

$$ab \rightleftarrows b \rightleftarrows ba$$
$$ac \rightleftarrows c \rightleftarrows ca$$
$$ad \rightleftarrows d \rightleftarrows da$$
$$ae \rightleftarrows e \rightleftarrows ea$$

tous les mots du monoïde qui contiennent un ou plusieurs a et d'autres lettres peuvent être remplacés par le mot obtenu en effaçant tous les a.

Le *problème des mots* est alors le suivant. Soit un monoïde défini par un alphabet et un groupe de relations. Soit deux mots quelconques dans cet alphabet. Définir une méthode qui, pour *tout* alphabet (fini), *tout* groupe (fini) de relation et *tout* couple de mots, permette de décider si les deux mots sont identiques, après utilisation des règles d'équivalence du monoïde.

Sous cette forme très générale, il est normal de penser que le problème ne peut être résolu.

Pourtant même si l'on *particularise* le problème en se bornant à l'alphabet A défini ci-dessus et en choisissant un groupe de relations bien défini, par exemple :

$$ac \rightleftarrows ca$$
$$ad \rightleftarrows da$$
$$bc \rightleftarrows cb$$
$$bd \rightleftarrows db$$
$$abac \rightleftarrows abacc$$
$$eca \rightleftarrows ae$$
$$edb \rightleftarrows be.$$

un jeune auteur soviétique, Teitsin (1), a pu montrer que le problème du mot est encore indécidable.

Remarquons cependant que le problème ainsi précisé est encore, en fait, une *classe de problèmes* puisque les mots à comparer peuvent être quelconques. En effet il est facile de montrer que le problème peut être résolu pour certains mots particuliers. Par exemple, en se reportant à la liste des substitutions permises, on s'aperçoit que le mot

aaabb

ne peut être obtenu à partir d'aucun autre et ne mène vers aucun autre. Il n'a donc pas d'équivalent.

Au contraire, si on choisit comme couple de mots à comparer l'un de ceux qui servent à définir les relations d'équivalence, par exemple *eca* et *ae,* le problème est résolu immédiatement, cette fois-ci, par l'affirmative.

Il est donc clair que la possibilité ou l'impossibilité de résoudre le problème est liée ici à la quantité d'information qui est contenue dans l'énoncé du problème particulier. C'est un point sur lequel nous allons bientôt revenir.

CALCULABILITES THEORIQUE ET PRATIQUE

Le résultat que nous venons d'évoquer, c'est-à-dire l'impossibilité de construire un algorithme général pour résoudre le problème des mots est un des nombreux *théorèmes de limitation* qui sont apparus, à partir de 1930, sur les frontières des mathématiques et de la logique (2). Les plus célèbres sont : le théorème de Gödel qui conclut à l'impossibilité de démontrer la non-contradiction de l'arithmétique dans un système qui ne soit pas plus fort que l'arithmétique elle-même ; le théorème de Church qui montre l'existence, dans tout système formel comprenant au moins le calcul des prédicats, de propositions indécidables. Les diverses démonstrations de ces théorèmes

(1) Cité par TRAHTENIIROT, *Algorithmes et machines à calculer,* Dunod, 1962.
(2) Sur tout ceci, on consultera avec profit le grand ouvrage de J. LADRIERE, *Les limitations internes des formalismes,* Gauthier-Villars, 1957.

de limitation se rattachent à quelques schèmes fondamentaux sur lesquels nous reviendrons un peu plus loin ; notons dès à présent que l'utilisation de la notion de machines de Turing permet d'énoncer ces faits de limitation sous la forme de l'impossibilité de construire certains programmes.

Dès lors il devient relativement facile de comparer les impossibilités «relatives» évoquées dans le paragraphe consacré à l'explosion combinatoire aux impossibilités « absolues » que nous venons d'évoquer :

- dans certains cas, il est impossible de construire un programme, pour des raisons liées à la nature même du problème : c'est le cas du problème général des mots que nous venons d'évoquer ;
- dans d'autres cas, il est possible de construire un programme, au moins en principe, mais il n'est pas *effectivement possible :*
  - soit de l'écrire complètement ;
  - soit de trouver un automate réel qui possède la capacité de mémoire nécessaire ou qui puisse fonctionner le temps nécessaire à la résolution complète du problème.

A première vue, la distinction semble très importante. En effet, pour ce que l'on sait être impossible *a priori,* rien de raisonnable ne peut être tenté. Par contre, s'il s'agit d'une limitation pratique, on peut s'efforcer de trouver, dans chaque cas, une procédure *ad hoc* qui permette de se tirer d'affaire.

Par exemple, si nous revenons à l'exemple *a)* de la page 119, lorsqu'on veut ranger un mot dans une liste déjà constituée ; on peut se proposer, au lieu d'attaquer les comparaisons par le premier mot qui se présente, commencer au milieu de la liste. La première comparaison élimine ainsi l'une des moitiés de la liste. On se porte alors au milieu de la moitié restante, et ainsi de suite. On peut alors voir que le nombre d'opérations n'augmente pas proportionnellement au *carré* de N, mais au *logarithme* de N. Avec les valeurs numériques que nous avions choisies plus haut, le nombre d'opérations n'est plus de l'ordre de 10^{13}, mais 10^7 et est parfaitement compatible avec les performances des calculatrices actuelles.

Par contre un résultat négatif comme celui relatif au problème des mots semble d'autant plus décourageant que nous nous sommes efforcés, tout au long de cet ouvrage, et en particulier au cours du

chapitre II, de montrer que les problèmes de l'intelligence artificielle pouvaient se mettre sous une forme qui les rende voisins à des problèmes d'identification de mots dans un certain alphabet.

On pourrait donc se demander si les « astuces » qui s'efforcent, dans certains cas particuliers de réduire l'explosion combinatoire ne soit pas un travail de Sisyphe, condamné en fin de compte à l'échec.

C'est donc la possibilité même de l'intelligence qui pourrait être mise en question si cette remarque devait être prise au sérieux.

D'ailleurs il s'est effectivement trouvé un certain nombre d'auteurs, en particulier au début des années 1960, pour contester tout l'effort entrepris et évoquer, à l'appui de leur thèse, les grands théorèmes de limitation de la logique mathématique, et en particulier le théorème de Post. Ce fut le cas notamment de Y. Bar-Hillel et de M. Taube.

Les conséquences de cette offensive se firent surtout sentir dans les domaines de la documentation et de la traduction automatique, mais, d'une façon générale, on put constater un certain recul de l'activité de plusieurs équipes, et même la disparition de certaines d'entre elles.

Aussi nous semble-t-il important d'examiner ce genre d'objections, afin, si possible, d'éliminer les ambiguïtés qui se sont glissées ici et là lorsqu'on se cantonnait dans des généralités au lieu de se livrer à une analyse complète de chaque cas.

Mais nous verrons qu'en avançant dans cette direction il devient possible non seulement d'écarter les objections de principe dont l'importance a d'ailleurs diminué, mais surtout d'ouvrir des perspectives plus générales en ce qui concerne les tactiques à mener dans le combat anticombinatoire.

C'est donc à double titre que la comparaison « calculabilité (ou non-calculabilité) théorique-calculabilité pratique » est importante. Nous allons l'ébaucher en reprenant l'exemple du problème des mots. Pour cela reportons-nous à l'organigramme de la figure 32 qui illustre ce problème sous son aspect « machine ».

On voit que ce schéma ne soulève aucune difficulté dans sa conception. Deux sorties y sont indiquées, une réponse positive et une réponse négative. Mais il est possible également que l'on reste très longtemps dans la boucle sans atteindre une sortie, et ceci parce que les substitutions permises créent constamment des mots nouveaux qui ont

besoin, à leur tour, d'être testés. C'est en ceci que notre programme n'est pas complet : on n'est jamais sûr qu'il s'arrête une *fois* qu'on l'a mis en marche.

Si l'on modifie légèrement le programme, de telle façon qu'un compteur soit défini, qui sera augmenté d'une unité chaque *fois* que la boucle principale sera parcourue une fois, on peut prévoir un test qu'arrête l'automate lorsque le compteur atteint le chiffre K.

Dès lors, quand on se donne deux mots

- ou bien ils apparaissent comme identifiés avant que le compteur atteigne K ;
- ou bien ils apparaissent comme différents avant que le compteur atteigne K ;
- ou bien aucune décision n'est possible en K itérations.

Dans les exemples du paragraphe consacré à l'explosion combinatoire, des calculs élémentaires permettaient de calculer la borne supérieure du nombre des opérations à effectuer. Cette borne était très au-delà de la capacité des automates réels. Cependant on pouvait concevoir qu'une meilleure utilisation des propriétés et des particularités du problème permettrait d'abaisser considérablement la borne supérieure.

Par ailleurs le résultat négatif relatif au problème des mots nous indiquait qu'il n'existait pas de valeur finie de K pour laquelle le problème de l'identification de deux mots *quelconques* a reçu une réponse définitive (qu'elle soit positive ou négative). Mais rien n'empêche d'imaginer qu'en se limitant à la comparaison de mots appartenant à des *classes spécifiques,* correspondant à des propriétés explicites de ces mots, on ne pourrait pas modifier complètement la situation.

De plus, ce qui est important pour justifier les efforts des chercheurs, ce n'est pas tant la borne supérieure, mais la *borne inférieure.* Si nous trouvons que notre problème devra, *dans tous les cas,* nécessiter N opérations élémentaires et que N est trop grand, nous abandonnons. Mais si nous savons seulement que ce nombre *pourrait éventuellement* être voisin de N, nous nous contentons de chercher des procédures qui nous permettent d'éviter cette borne supérieure. Certes,

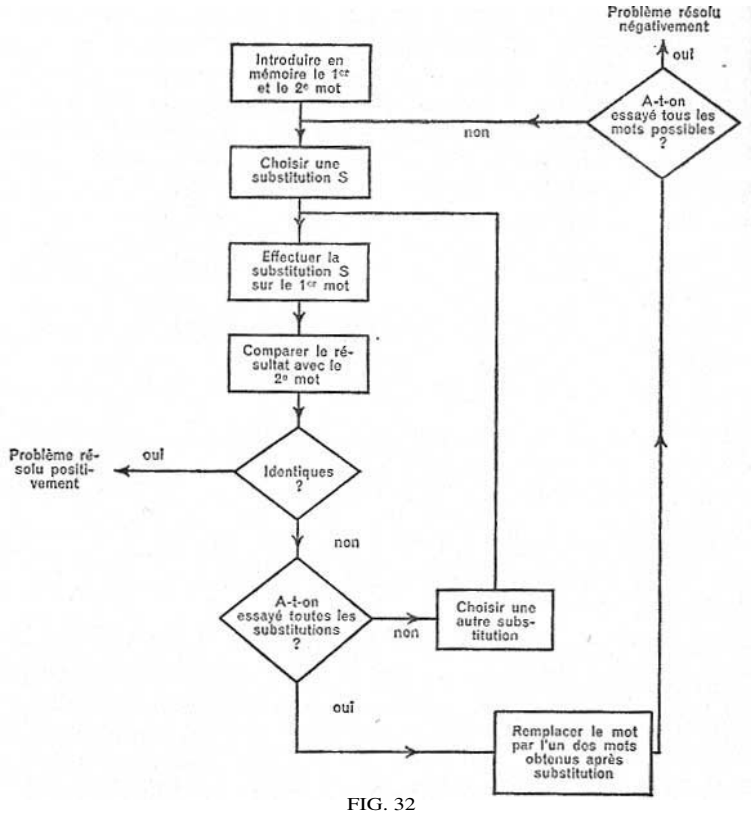


FIG. 32

rien ne nous permet d'être sûr que de telles procédures seront trouvées dans les différents cas où le problème se pose, mais nous sommes désormais dans une situation bien différente de celle qui était évoquée par des auteurs tels que Bar-Hillel ou Taube.

Nous allons illustrer ces réflexions à l'aide de quelques exemples tirés des chapitres précédents.

ANALYSE DE QUELQUES EXEMPLES

Notre premier exemple sera tiré à nouveau de l'ensemble des jeux de grille et nous choisirons, pour simplifier la discussion, le jeu où $a = b = 3$ (c'est-à-dire le Tic-Tac-Dou). Pour rendre les diagrammes plus lisibles, nous remplacerons les noeuds par des cases, d'où le jeu à 9 cases (fig. 33)

1	2	3
4	5	6
7	8	9

FIG. 33. - Le champ du jeu

Au premier coup, 9 choix sont possibles. Le second joueur n'a plus que 8 possibilités, etc.

Au total, on a 9 ! = 9.8.7.6.5.4.3.2 .
= 355 68o possibilités.

Bien entendu, un automate pourrait explorer toutes les possibilités. Mais pour a = 5, on aurait déjà 25! possibilités, ce qui est au-delà de toute possibilité pratique.

Mais ce que nous voulons examiner, dans le cas du Tic-Tac-Dou, c'est que toutes les possibilités ne sont pas également intéressantes.

En effet, un coup d'oeil sur le diagramme ci-dessus montre que, tant qu'il s'agit du 1er coup

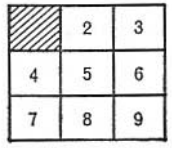
1, 3, 7 et 9 sont équivalents
2, 4, 6, 8 sont équivalents.

Il n'y a donc pas 9 coups différents possibles, mais 3

- un coup du type 1 ;
- un coup du type 2;
- le coup 5.

De même, si le 1er coup joué est 1, pour le second coup

2 et 4 sont équivalents
3 et 7 sont équivalents
8 et 6 sont équivalents (cf. fig. 34).



Fic. 34
Le champ du jeu
après le premier coup

Il y a donc 5 possibilités distinctes

- un coup du type 2 ;
- un coup du type 3 ;
- un coup du type 8 ;
- le coup 5 ;
- le coup 9, etc.

On constate qu'après 2 coups il y a 12 combinaisons réellement distinctes sur les 72 théoriquement possibles. Cela veut dire que les *propriétés caractéristiques du problème que l'on a à résoudre - ici* les propriétés de symétrie de la grille - *réduisent* la multiplication des opérations à effectuer lors du dépouillement de la situation par un automate.
Ceci ne suffit pas, bien entendu, pour assurer la réalisabilité pratique du programme, mais c'est déjà une indication intéressante en ce qui concerne la possibilité de trouver des estimations plus raisonnables de la borne supérieure N.

* * *

Examinons maintenant le problème d'algèbre élémentaire du chapitre IV.
Il s'agit de vérifier l'identité

$$2m + 2n = 2(m+n)$$

en opérant sur la première formule un certain nombre de substitutions choisies dans la liste : A, S, G, D, C, R, définie page 85.

Il est assez simple ici de vérifier qu'on se trouve dans un cas particulier du problème des mots, tel qu'il a été défini un peu plus haut dans ce chapitre. On exprime en effet l'action de deux substitutions successives en écrivant l'un après l'autre les symboles qui leur correspondent.
Une transformation est donc un mot dans l'alphabet : 0.

$$0 = \{A, S, G, D, C, R\}.$$

C'est ainsi qu'une transformation couronnée de succès, pour démontrer la formule recherchée, est donnée par le mot

SRGCGDAA.

Par ailleurs, il est clair que certains groupements de substitutions sont identiques. Si l'on désigne l'absence de substitution par le signe « 1 », on a

$$GD = 1$$
$$AS = 1$$
$$C^2 = 1$$

etc.

De plus, si 1 [ ] dénote la longueur de la formule entre crochets il est visible que l'on a

1 [G (formule)] = Z [D (formule)] =1 [C (formule)] =1 [formule]
1 [A (formule)] = Z [formule] -;- t
Z [S (formule)] =1 [R (formule)] =1 [formule] - i.

Toutes ces propriétés peuvent évidemment servir pour éliminer, parmi les mots construits sur l'alphabet O, ceux qui correspondent à des transformations de la formule $(2m + 2n)$ qui ne font pas *avancer* vers la formule $2(m + n)$.
Cela veut dire que même si notre problème peut être transcrit sous la forme d'un problème d'identification de mots, il s'agit d'un problème relatif à une structure de monoïde très *particulière,* concernant l'identification d'un couple très *particulier* de mots. Le théorème de Post-Markov ne s'applique donc pas. Cela n'entraîne pas qu'il n'existe pas de limitations, mais que ces limitations doivent être étudiées et estimées en fonction des données du problème *particulier* devant lequel on se trouve.

Mais de telles limitations, ainsi particularisées, n'ont plus rien de mystérieux et, à première vue, n'introduisent aucune barrière de principe entre l'intelligence naturelle et l'intelligence artificielle. Cependant, trop d'affirmations rapides ont été émises et publiées à ce sujet pour que nous ne nous attardions pas un peu. Nous nous servirons à nouveau d'un exemple.

Il s'agit maintenant d'un jeu assez subtil puisqu'il est lui-même basé sur la notion d'automate, et plus précisément sur la notion de machine de Turing. Une tâche confiée à une telle machine est équivalente à un jeu de cartes de la forme représentée sur la figure 35.

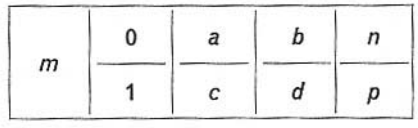


FIG. 35. — Schéma d'une carte dans le jeu de Rado

a, b, c, d sont o ou *1, m, n, p* sont des nombres entiers.

Une telle carte correspond au sous-programme suivant

m : Si la case sous inspection est marquée o, , écrire *a,* effectuer le mouvement correspondant à *b* (avancer si *b = 1,* reculer si b = 0) puis aller à l'instruction numéro *n.*
Si la case sous inspection est *1,* écrire c, effectuer le mouvement correspondant à d, aller à l'instruction numéro p.

Le jeu (inventé par T. Rado) est alors le suivant

1) On choisit $n > o$ et on se donne un ensemble de *n* cartes du type ci-dessus.

2) On constate que la machine de Turing, alimentée par ce jeu de *n* cartes, stoppe après s déplacements.

Le *champion d'ordre n* est celui qui obtient le score a (c'est-à-dire le nombre de *« 1 »* écrit sur la bande le plus élevé), à l'aide de son jeu de cartes M.

Un couple $(M, s)_n$ formé d'un jeu de cartes M et d'un entiers est dit *valide* si M comporte *n* cartes et si la machine s'arrête effectivement après s déplacements.

Le nombre de couples valides pour *n* donné, $N_e(n)$ est toujours fini. On a même

$$Ne(n) < [4(n+1)]^{2n}$$

Considérons alors la fonction

$$\Sigma(n) = \max_{(M, S)_n \text{ valide}} [\sigma]$$

Rado a montré que $E(n)$ (le score maximum que pourrait obtenir un champion d'ordre *n)* est une fonction *non calculable.* En effet, quelle que soit la fonction calculable $f(n)$, on peut démontrer qu'à partir d'un certain *n,* $E(n) > f(n)$.

Ce résultat entraîne le suivant :

Si $S(n)$ est le nombre maximum de déplacements pour une machine appartenant à un couple valide (M, s),,, $S(n) > E(n)$, donc $S(n)$ est également *non calculable.*

Enfin, résultat encore plus remarquable, la fonction $N_e(n)$, définie plus haut, bien que bornée par la fonction relativement simple $[4(n+t)]^{2n}$ est également *non calculable.*

Cet exemple est particulièrement intéressant parce qu'il montre l'impuissance, dans ce cas précis, de la méthode exhaustive » : en effet si on parcourt l'ensemble des nombres entiers de 1 à $[4(n+1)]^{2n}$ on est sûr de rencontrer à un certain instant la valeur de $N_e(n)$. Mais pour savoir quel est cet instant, il faut disposer d'un *raisonnement* dont le nombre d'étapes, lorsque *n* est quelconque, n'a pas de borne supérieure.

LIMITATION DES LIMITATIONS

Peu à peu les menaces que les théorèmes de limitation faisaient peser sur les perspectives de développement d'une intelligence artificielle perdent de leur acuité, mais il semble possible d'aller beaucoup plus loin et de renverser la situation. Auparavant il faut rappeler brièvement la signification de ce que nous venons d'établir

- Tout d'abord, les problèmes de l'intelligence artificielle sont des *problèmes combinatoires.* Une tâche - complètement spécifiée -

peut toujours être accomplie par une recherche exhaustive, c'est-à-dire en examinant successivement tous les éléments d'un ensemble. Mais lorsque la tâche est augmentée, l'ensemble à examiner croît dans des proportions qui font de ce processus une véritable *explosion combinatoire.*

– Les programmes qui *résolvent* les problèmes de l'intelligence artificielle *ne sont pas* des programmes de recherche exhaustive. Au contraire, ils se fondent sur les particularités, les propriétés spécifiques de chaque problème pour développer des techniques de *réduction* de l'expression combinatoire.

- On appelle souvent la méthode exhaustive : « méthode de la force brute ». Il semble normal, par contraste, de caractériser l'intelligence comme *l'aptitude à engendrer ou à utiliser des techniques anti-combinatoires.*

Le lecteur pourra vérifier l'aspect anticombinatoire de la notion d'ouverture dans un jeu comme les échecs, de la propriété de commutativité en algèbre, des règles d'accord en genre et en nombre en linguistique, etc.

Nous allons nous contenter ici d'énumérer quelques cas particuliers qui nous permettront peut-être de rendre plus « parlantes » des notions abstraites comme celles de schéma, de programme, de procédure, etc., dans la mesure où les rapports qu'elles possèdent entre elles seront plus transparents.

« Construire le milieu d'un segment » est un problème particulier dans la classe des problèmes de constructions par la règle et le compas, elle-même famille privilégiée dans l'ensemble des constructions géométriques, etc.

De même, un *schéma de programme* du type

- algorithme :
- documentation de vase :
- données :

devient un *programme* dès qu'on se donne les documents et les données explicitement. Ce programme devient une *procédure si* l'on peut montrer qu'il s'arrête au bout d'un temps déterminé pour produire un résultat. A la limite, le résultat peut être obtenu en introduisant

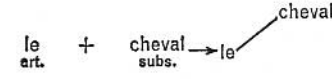
les données dans une simple *formule,* qui devient donc un cas particulier de programme.

Cette remarque permet sans doute d'apporter quelque lumière sur la distinction couramment évoquée - mais mal définie - entre les applications « numériques » et « non numériques».

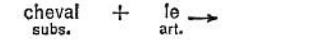
Dans les problèmes numériques, l'algorithme contient de nombreuses formules, arithmétiques ou logiques, et l'on peut tirer parti de l'existence, dans l'automate, de registres liés aux structures arithmétiques et logiques (accumulateurs, registres d'index). Du coup, il devient commode de disposer de langages de programmation qui donnent droit de cité aux formules arithmétiques et logiques dans un format très voisin du format usuel (c'est le cas du Fortran, de l'Algol et de bien d'autres langages).

Dans les problèmes non numériques, les opérations à effectuer sur les données, à l'aide des informations puisées dans la documentation de base, expriment des propriétés du problème, mais des propriétés qui ne sont pas directement liées à celles (commutativité, associativité, etc.) de l'arithmétique et de la logique algébrique.

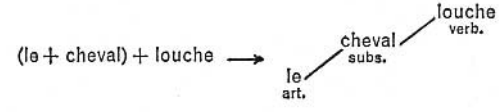
Par exemple, les algèbres que définissent des grammaires de dépendance du type de celles étudiées au chapitre V ne sont en général ni commutatives, ni associatives : on a



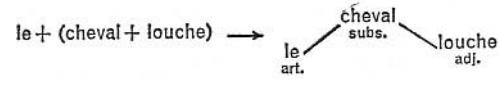
mais



de même



mais :



Or, des *propriétés* comme la commutativité et l'associativité, propriétés qui s'expriment par des formules

$$x + y = y + x$$
$$x + (y + 2) = (x + y) + 2$$

sont, nous venons de le souligner, des programmes d'une nature particulière, des programmes *sous forme concentrée :* elles possèdent un maximum *d'efficacité anticombinatoire.* On peut s'en rendre compte sur l'exemple de G. Teitsin analysé plus haut. Si, aux relations d'équivalence de cet exemple, on ajoute les relations

$$ab \rightleftarrows ba \qquad ae \rightleftarrows ea$$
$$bc \rightleftarrows cb \qquad cd \rightleftarrows dc$$
$$ce \rightleftarrows ec \qquad de \rightleftarrows ed$$

le monoïde est commutatif (on dit aussi « abélien »). Mais maintenant, au lieu, lors des transformations de mots obtenues en utilisant les relations d'équivalences, de se trouver en présence d'un foisonnement incontrôlable de mots nouveaux, la propriété de commutativité permet de réduire tous les mots à un équivalent dans lequel les lettres sont rangées par ordre alphabétique. Dès lors, l'identité des mots se vérifie aisément : le passage du non-commutatif au commutatif a transformé l'indécidable en décidable.

C'est l'absence de symétries, de relations simples dans une structure qui font dire qu'elle est *complexe.* D'après ce qui précède, il est manifeste que la « résolubilité » d'un problème est inversement proportionnelle à sa complexité. Par contre, la capacité « intellectuelle » d'un automate va de pair avec sa propre complexité.

S'attaquer au bilan « problème-automate » du point de vue de la complexité, c'est donc finalement tenter d'établir, sur une base scientifique, le diagnostic des possibilités de l'intelligence artificielle.

Pour que le bilan puisse être précis, il faut lui fournir une base quantitative, d'où l'importance que l'on doit attacher à une mesure quantitative de la complexité (d'un problème ou d'un automate). Ici, nous atteignons les frontières de la science actuelle. Il existe de nombreux travaux en cours dans ce domaine (principalement en U.R.S.S.). Mais avant de les examiner, nous allons revenir une dernière fois sur les « théorèmes de limitation » car nous pensons que ces théorèmes,

loin d'apporter des constats d'échecs, sont un point de départ indispensable pour le développement d'une conception positive de la complexité.

* * *

On sait que l'apparition de paradoxes dans la théorie « naïve » des ensembles est à l'origine de *l'effort* de formalisation de la mathématique et de la logique modernes. Il est moins connu, mais au moins aussi important, que les mêmes paradoxes sont en réalité la racine des théorèmes de limitations. C'est ainsi que le fameux paradoxe de l'Épiménide (ou du « menteur ») : « Épiménide le Crétois dit que tous les Crétois sont menteurs», énoncé dont il est évidemment impossible d'apprécier la vérité ou la fausseté, sert de modèle à la démonstration du théorème de Gödel. Le paradoxe de Richard (que nous n'analyserons pas ici) sert de schéma aux démonstrations des théorèmes de Church, Kleene et Turing.

Pour comprendre la signification de ces paradoxes, à l'aide des images et des concepts auxquels nous nous sommes habitués, il est commode d'imaginer le problème : «quelle est la valeur logique de l'affirmation de l'Epiménide ? » comme une succession de réponses contradictoires, chaque conclusion inversant la prémisse sur laquelle elle se fonde. Cela évoque un programme pour automate qui contiendrait la suite d'instruction

n Exécuter l'instruction n° $n + 1$
$n + 1$ Exécuter l'instruction n° n

Un tel programme ferait évidemment « cycler » l'automate indéfiniment (1).

Considérons maintenant le paradoxe de Berry qui s'énonce comme suit : « le nombre de Berry est le plus petit nombre naturel non définissable au moyen d'une phrase contenant 50 mots au maximum pris dans un vocabulaire ». Comme la phrase précédente ne comprend que 37 mots le nombre de Berry est contradictoire.

(1) On trouverait des exemples détaillés de ce genre dans l'article de S. Gorn « The treatment of ambiguity and paradox in mechanical languages » dans *Proceeding of Symp. in pure Math., V,* 1962, p. 201.

Bien entendu la clef du paradoxe réside dans la constatation de ce que la notion de « définissabilité » utilisée dans la phrase en question n'est pas elle-même clairement définie. On utilise en effet le langage ordinaire pour construire un objet mathématique à qui l'on voudrait ensuite imposer des contraintes de rigueur qui ne sont applicables qu'aux définitions formelles.

Le paradoxe disparaît, en effet, dès qu'on se donne un système formel bien défini A qui contienne les symboles de l'arithmétique. Car dès lors le nombre de Berry correspondant au système A (c'est-à-dire le plus petit nombre naturel non définissable au moyen d'une *expression du système A* contenant 50 symboles élémentaires au maximum) n'est pas contradictoire.

Mais on peut aller plus loin et se servir de l'idée de Berry pour construire une notion mathématique nouvelle : c'est ce que nous proposons d'appeler la fonction de « Berry-Beth » (1) : « La fonction de Berry-Beth qui correspond au système formel A est la fonction f qui associe à un nombre naturel m, le plus petit nombre naturel n non définissable au moyen d'une expression du système A contenant m symboles élémentaires au maximum. » On peut montrer que la fonction f *ne peut pas* être définie formellement à l'aide du seul outillage symbolique contenu dans le système A. Pour définit f de façon complète, il faudrait disposer d'un système formel B plus puissant que A, etc. L'impossibilité de définir f dans A est un résultat *négatif,* caractéristique des résultats négatifs que sont les théorèmes de limitation. Mais on voit apparaître ici la possibilité de *renverser* le sens des applications de ce genre de théorèmes.

En effet les autres faits de limitation comme le théorème de Gödel mettent en oeuvre des énoncés mathématiques «monstrueux », expressions qui expriment leur propre non-déductibilité, etc. Bien que monstrueux ces énoncés appartiennent à un corps d'énoncés licites et les conséquences, négatives, de leur mise en oeuvre n'en sont pas moins suffisantes pour fonder les limitations. Par contre les êtres mathématiques comme la fonction de Berry-Beth se situent dans

(1) Cf. E. W. BETH, Les fondements logiques des mathématiques, Gauthier-Villars, 1955, p. 194

une une classe beaucoup plus attirante : celle des mesures de construction d'autres êtres (normaux, ceux-là) de la mathématique usuelle.

On peut alors s'attendre, dans cette direction, à l'apparition de résultats *positifs,* c'est-à-dire qui expriment une *limitation des faits de limitation : s'il* existe bien des problèmes trop complexes pour un système combinatoire donné, il existe *toujours* une réduction de la complexité permettant de résoudre le problème (démontrer - ou réfuter - un théorème), en augmentant la puissance du système, ou au contraire en diminuant le degré de généralité du problème posé, par exemple en remplaçant le problème général des mots par celui d'une classe particulière de mots.

LES THÉORIES DE LA COMPLEXITÉ

La notion générale de complexité est évidemment fort ancienne. Mais elle est restée, jusqu'à une période très récente, une notion qualitative, permettant d'exprimer un sentiment de difficulté, voire d'impuissance.

Mais à partir du moment où l'on s'est aperçu que beaucoup de notions qui appartenaient au langage quotidien étaient susceptibles d'une analyse logique, et, moyennant quelques conventions restrictives, pouvaient s'intégrer dans un système formel, il était naturel d'examiner l'un après l'autre les différents concepts qui se situaient dans le « voisinage sémantique » des concepts déjà formalisés.

C'est ainsi que la logique des propositions ayant exploité des conjonctions comme « et » et « ou », le calcul des prédicats ayant intégré des expressions telles que « quelle que soit» etc., le formalisme s'attaqua à des concepts plus larges comme celui de « déductibilité » et de « démonstrabilité ».

Il restait cependant un grand pas à franchir entre la formalisation de tels concepts susceptibles d'aboutir à des prédicats qui ne peuvent avoir que les valeurs « oui » ou « non » et celui de la complexité qui, s'il était convenablement formalisé devrait déboucher sur une échelle *quantitative* de valuations.

Le premier pas dans cette direction a été - du moins à notre connaissance - accompli par le polonais A. Lindenbaum dans une commu-

nication au Congrès international de Philosophie tenu à Paris en 1935 (1).

La disparition de Lindenbaum au cours de la seconde guerre mondiale l'empêche de développer son projet qui fut repris indépendamment par N. Goodman, à partir de 1943 (2).

Pourtant Goodman ne put mener très loin son projet (limité à la complexité de ce qu'il appelait les « bases extra-logiques », mais qui correspond, dans le langage Bourbakiste, aux espèces de structures algébriques).

Malgré cet échec, la notion de complexité était étudiée implicitement par les logiciens qui, après Gödel étudiaient les «hiérarchies » constructives (fonctions récursives primitives, générales, arithmétiques, analytiques, etc.). D'ailleurs l'idée d'utiliser la notion de « longueur d'une démonstration » était présente chez Gödel dès 1930.

Mais c'est tout récemment que l'étude de la complexité a pris un nouvel essor, notamment en U.R.S.S. et ceci sur deux plans relativement indépendants : étude de la complexité des réseaux de relais avec Iablonski, Lupanov et bien d'autres ; étude des sous-classes de machine de Turing définies par des contraintes particulières sur la longueur de bande utilisée, ou le temps de fonctionnement autorisé (travaux de Ritchie, Yamada, Schutzenberger, Trahtenbrot, Rabin, etc.) (3).

Après ces développements - dont nous souhaitons qu'ils n'aient pas, à leur tour, semblé trop complexes ! - nous croyons être en mesure de répondre au problème posé dans ce chapitre, justifiant ainsi l'ouvrage tout entier.

Car dès le chapitre II, et à chaque fois qu'un exemple nouveau était présenté, nous avons tenu à faire remarquer l'identité foncière de tous les problèmes posés. Tous peuvent, en fin de compte, être présentés comme des problèmes d'identification de mots dans un certain alphabet, en conjonction avec l'existence de relation de substitution, ou encore la recherche d'un chemin dans un graphe.

(1) *Cf. Actualités scientifiques et industrielles, n° 394, Hermann, 1936, p. 29.*

(2) On peut trouver une analyse succincte des travaux de Goodman et de ses continuateurs (Kemeny, Svenonius) dans un article de R. RUDNER *paru dans Philosophy of Science, 28, 1961, p. 109.*

(3) Sur tout ceci voir notre exposé au Congrès d'Amsterdam, août 1967.

Suivant l'exemple choisi, le système de relations était énoncé plus ou moins complètement.

De même, le mot à comparer pouvait être donné explicitement ou seulement par un certain nombre de ses propriétés (de même pour le chemin). On se trouvait donc bien plus souvent en face d'une *classe* de problèmes que d'un problème *unique.*

Cette situation a un équivalent, bien entendu, dans le langage des automates et de leurs programmes, et nous avons décrit la situation avec quelques détails à la fin du chapitre II, en présentant - pour les distinguer - les notions de tâche, de programme, de moniteur et de système.

Lorsqu'une tâche est complètement définie, on peut écrire le programme et introduire les données initiales. Le seul obstacle qui peut se dresser est alors un obstacle matériel (encombrement possible de la mémoire, etc.), comme dans l'exemple a) ci-dessus. Lorsque la tâche n'est pas complètement définie, que le problème posé est en fait, non *d'accomplir une tâche,* mais de trouver une *méthode pour accomplir toute une classe de tâches, il* n'est pas possible, bien souvent, de spécifier les contraintes de temps ou de mémoire qui vont peser sur l'automate.

Dans certains cas heureux, il est possible de caractériser chaque tâche de la famille par un paramètre numérique *n.* La « taille » de l'automate sera alors une fonction de ce paramètre numérique. Il sera peut-être alors possible d'examiner comment la taille de l'automate varie (c'est-à-dire, en général, grandit) en fonction de *n. Si* la fonction croît démesurément avec n, on peut être assuré qu'à partir d'une certaine valeur, il n'y aura plus d'automate suffisant à accomplir la tâche attendue.

Le problème, qui est donc une classe de «tâches», est alors *indécidable.*

Mais s'il est extrêmement agréable de pouvoir disposer d'une méthode générale pour résoudre une classe immense de tâches, on peut bien accepter la solution moins élégante, mais bonne à prendre s'il n'y en a pas d'autres, qui consiste à diviser la classe en sous-classes : et même en autant de sous-classes qu'il est nécessaire pour que chacune d'entre elles puisse être pourvue d'une méthode générale de solution.

A ce point de l'exposé le lecteur aura probablement formé de lui-même l'idée que la clé du problème de réalisabilité en intelligence artificielle

réside dans la mise en correspondance de deux hiérarchies : celle des problèmes et celle des solutions.
Nous formulerons cette thèse sous la forme plus explicite que voici

A tout problème posé dans un système formel complètement défini, on peut associer un automate et un programme pour cet automate. L'automate atteindra effectivement la solution s'il dispose d'une structure dont la complexité soit au moins égale à celle du problème posé. S'il n'en est pas ainsi, il est cependant toujours possible de revenir à des conditions de résolubilité

- soit en augmentant la complexité de l'automate ;
- soit en diminuant le degré de généralité du problème.

Le lecteur pourra vérifier la validité de cet énoncé (que nous avons laissé un peu vague, puisqu'en tout cas, nous n'en proposons pas ici une démonstration), sur un exemple tel que celui de l'analyse grammaticale (on augmente la complexité de l'automate en augmentant le nombre des règles qui constituent sa « documentation de base », on diminue la généralité du problème en imposant une borne supérieure à la longueur des syntagmes).
Finalement, nous pensons que la notion de complexité donne la clé d'une autre notion que nous avons évoquée plusieurs fois, notion d'ailleurs fort controversée, qui est celle *d'heuristique.*
Par définition, la méthode heuristique est celle qui n'est pas fondée sur la mise en oeuvre mécanique d'une jeu de formules, mais sur l'exploration guidée des possibilités du problème. Comment guider cette exploration (car il ne s'agit évidemment pas de l'exploration exhaustive), autrement que par une intuition, une analogie, etc. ? L'heuristique n'est-elle pas, par conséquent, typique des capacités de l'homme, par contraste avec celle des automates ?
Et, de fait, les méthodes dites heuristiques introduites dans certaines recherches relevant de l'intelligence artificielle, n'ont pas entraîné la conviction (il s'agit notamment des points de vue développés par Newell et Simon pour le jeu d'échecs, et par Gelernter pour la démonstration des théorèmes de géométrie). Les critiques ont pu montrer que le guidage non formalisé dans la recherche des solutions était, soit inopérant soit informé par avance des solutions à obtenir.
Il nous semble que les remarques présentées au cours de ce chapitre

permettent de présenter une heuristique automatique parfaitement fondée.
Pour le voir, il est commode de représenter la résolution du problème comme la recherche d'un chemin dans un graphe. Si l'on disposait d'un jeu de formules donnant la solution, le graphe serait linéaire ou au moins aurait l'allure d'un treillis.
Dans la méthode exhaustive, tous les chemins, sans exception sont explorés jusqu'à ce qu'un premier chemin victorieux soit découvert.
Une méthode heuristique élimine, à chaque carrefour, un nombre suffisant de chemins possibles pour que l'exploration ne conduise pas à une explosion combinatoire du nombre d'opérations à effectuer.
Ce filtrage sera aisé si l'on dispose d'une mesure adéquate de la complexité : on éliminera les chemins qui accroissent la complexité du chemin restant.
On vérifiera que l'application de cette procédure redonne bien les limitations relatives que l'on avait rencontrées un peu plus haut.