Journal des traducteurs
Translators' Journal

Langage naturel et langages artificiels de communication avec l'ordinateur

Antonio A. M. Querido

Volume 10, Number 3, 3e Trimestre 1965

Traduction automatique et informatique

URI: https://id.erudit.org/iderudit/1061159ar
DOI: https://doi.org/10.7202/1061159ar

See table of contents

Publisher(s)

Les Presses de l'Université de Montréal

ISSN

0316-3024 (print)
2562-2994 (digital)

Explore this journal

Cite this article

Querido, A. (1965). Langage naturel et langages artificiels de communication avec l'ordinateur. *Journal des traducteurs / Translators' Journal*, *10*(3), 95–100. https://doi.org/10.7202/1061159ar

LANGAGE NATUREL ET LANGAGES ARTIFICIELS DE COMMUNICATION AVEC L'ORDINATEUR

Antonio A. M. QUERIDO,
Département de Linguistique et
Centre de Calcul,
Université de Montréal

Un ordinateur électronique est conçu pour lire, traiter, mémoriser et imprimer des symboles de n'importe quel type ; il n'a aucune prédisposition pour le traitement des symboles numériques. L'interprétation à donner aux symboles qu'il reçoit de son milieu extérieur et le traitement à leur appliquer, sont déterminés par le *programme* [1].

Ce qui a provoqué le retard dans le traitement des symboles linguistiques par rapport à celui des symboles numériques, c'est la difficulté d'écrire des programmes adéquats. Cette difficulté réside dans la nature même des symboles linguistiques, qui possèdent des structures beaucoup plus variées et beaucoup plus complexes que les symboles numériques.

Ainsi, pour communiquer à l'ordinateur une phrase en français, par exemple, il faut pouvoir représenter non seulement une séquence de mots, mais aussi la structure de cette phrase.

Ensuite, si l'on veut programmer l'ordinateur pour transformer cette structure, c'est-à-dire pour permuter les syntagmes, pour insérer des syntagmes nouveaux, pour effacer des syntagmes ou pour en substituer d'autres, il faut disposer d'un *langage de programmation* puissant et maniable.

Toute phrase contient, en réalité, plusieurs types de structures. Au *niveau phonologique,* par exemple, chaque phonème peut être représenté par un ensemble de traits phonétiques. Que ces traits résultent d'une analyse articulatoire ou d'une analyse acoustique des phonèmes, ils se structurent autour d'un petit nombre d'oppositions, binaires ou non [2]. Au *niveau syntaxique,* plusieurs représentations structurales de la phrase ont été proposées. Rappelons seulement ici les « stemmas de dépendance » de Tesnière [3] et les expansions syntagmatiques arborescentes de Chomsky [4].

Au *niveau sémantique,* chaque élément du lexique peut être analysé, d'une façon analogue à ce qui se passe en phonologie. Une telle analyse

1 — Pour une explication du fonctionnement de l'ordinateur, voir le résumé du mémoire de E. Faubert, pp. 109-112.

2 — Jakobson, R. & Halle, M., *Fundamentals of Language.* La Haye, Mouton, 1956.

3 — Tesnière, Lucien, *Eléments de syntaxe structurale.* Paris, Klincksieck, 1959.

4 — Chomsky, Noam, *Syntactic Structures.* La Haye, Mouton, 2e tirage, 1962.

produit des traits sémantiques, qui sont également structurés autour d'un nombre réduit d'attributs [5]. Les traits sémantiques constituent la représentation sémantique de la phrase, de la même façon que les traits phonétiques en constituent la représentation phonétique. Par eux, la médiation se fait entre la phrase et le continu sonore, d'un côté, entre la phrase et le continu de l'observable, de l'autre côté. Nous touchons donc là au passage du continu au discontinu, à l'opération qui est à la base de toute symbolisation.

Les structures que nous venons d'énumérer, ainsi que beaucoup d'autres structures du langage naturel, peuvent être réduites à deux types fondamentaux que nous appellerons, respectivement, des « chaînes » et des « arbres ».

Les chaînes servent à représenter les structures plus simples, comme, par exemple, les séquences de mots et les ensembles de traits phonétiques. L'analyse et la transformation d'une chaîne sont plus faciles lorsque, au moment de la mémorisation, on attache à chaque élément un rappel de son successeur. La technique utilisées dans ce cas consiste à inscrire, dans la cellule de mémoire qui contient un élément de la chaîne, l'*adresse* de la cellule qui contient son successeur. Un langage de programmation apte à créer un tel système de rappels est appelé un « langage de listes ».

Les « arbres » servent à représenter les structures linguistiques plus complexes, comme, par exemple, les stemmas de dépendance ou les expansions syntagmatiques. Un « arbre » est un ensemble de symboles partiellement ordonné, dans lequel chaque élément a un ancêtre unique, à l'exception de la « racine » de l'arbre qui, elle, n'a pas d'ancêtre [6].

Pour représenter un « arbre » dans un langage de listes, il suffit de considérer que chaque élément d'une chaîne peut être lui-même une chaîne, que les éléments de cette sous-chaîne peuvent être aussi des chaînes et ainsi de suite. On obtient alors ce qu'on appelle en programmation une « structure de listes ».

La construction et la manipulation des structures de listes comporte des opérations élémentaires si nombreuses que sa programmation dans le langage propre à l'ordinateur devient très vite fastidieuse. En programmation, d'ailleurs, on ne se sert que très rarement de ce langage pour communiquer avec l'ordinateur.

Dans le langage naturel, il est possible de construire une phrase telle que: « *Un homme est arrivé hier* » et d'utiliser cette phrase comme élément constitutif d'une autre phrase. Elle peut y fonctionner comme adjectif, par exemple: « *L'homme qui est arrivé hier / est son cousin* ».

Ce procédé est appelé « subordination » ou « subjonction ». La grammaire du français permet d'utiliser la subjonction à plusieurs reprises dans la construction d'une phrase. Outre la subjonction adjective ou relative, on dispose en français de la subjonction nominale. Ainsi la phrase: « *tu*

5 — Katz, J. J. & Fodor, J. A., "The Structure of a Semantic Theory", *Language* 39 (April-June 1963): 170-210.

6 — Berge, Claude, *La Théorie des graphes et ses applications*. Paris, Dunod, 1958.

n'aimes pas Stendhal » est utilisée comme syntagme nominal dans la phrase: « *Il m'a dit que / tu n'aimes pas Stendhal* ».

En programmation, on fait grand usage de la subjonction. On écrit des programmes élémentaires dans le langage de l'ordinateur. Ces programmes sont mémorisés par l'ordinateur. Le programmeur connaît les noms de ces programmes élémentaires, et peut donc programmer à un niveau plus élevé: pour faire exécuter une opération, même complexe, il lui suffit d'écrire le nom du programme approprié.

Pour prendre un exemple de la vie quotidienne, supposons un programme qui servirait à traduire le commandement: « *Marchez le long du trottoir jusqu'au prochain coin de rue* ». Il se lirait sans doute comme suit:

¶ 1. *Avancez le pied gauche; allez à* 2.

2. *Avancez le pied droit; allez à* 3.

3. *Si vous êtes arrivé au coin de la rue, allez à* 4; *sinon, allez à* 1.

4. *Terminé.*

Si l'on désigne ce programme par le symbole M, et que l'on ait à écrire un programme pour traduire le commandement: « *Traversez la rue* » (symbolisé par T), il est dès lors aisé d'écrire des programmes du type: « *Allez jusqu'au coin de la rue X* ». On écrira simplement:

¶ 1. *Faites M; allez à* 2.

2. *Faites T; allez à* 3.

3. *Si vous êtes au coin de la rue X, allez à* 4; *sinon, allez à* 1.

4. *Terminé.*

Plusieurs langages conçus pour fonctionner à un niveau élevé ont été construits spécialement pour représenter et traiter des « chaînes » et des « arbres ».

COMIT [7] est un des langages de chaînes les plus connus. Une instruction-type, écrite en langage COMIT, propose à l'ordinateur une chaîne de symboles. En lisant cette chaîne, l'ordinateur l'indexe: le chiffre 1 représentera le premier symbole, le chiffre 2 le deuxième, et ainsi de suite. La deuxième partie de l'instruction utilise cet index pour « indiquer » les transformations voulues: des insertions, des ellipses, des substitutions ou des permutations de symboles. En COMIT, il est possible aussi d'associer à chaque symbole un ensemble d'attributs accompagnés d'une valeur. Ces valeurs peuvent être retrouvées, substituées, effacées. On peut aussi insérer des attributs nouveaux avec leurs valeurs respectives.

Cette façon de procéder est utilisée dans tous les langages de listes.

Parmi les langages appropriés au traitement des structures arborescentes, nous mentionnerons seulement « Information Processing Language

7 — *COMIT Programmer's Reference Manual.* Cambridge, Mass, The M. I. T. Press, 1962.

V » (IPL-V) [8], l'ancêtre de tous les langages de listes, ainsi que « List Processor » (LISP) [9] et « Symmetric List Processor » (SLIP) [10]. Les lecteurs pourront se reporter à l'article de Bobrow [11] pour y trouver des détails que nous ne pouvons aborder ici.

Ce qui donne à SLIP son caractère particulier, parmi les autres langages de listes, c'est que, dans la mémorisation d'une « chaîne » ou d'un « arbre », chaque symbole est accompagné, non d'un rappel, mais de deux,

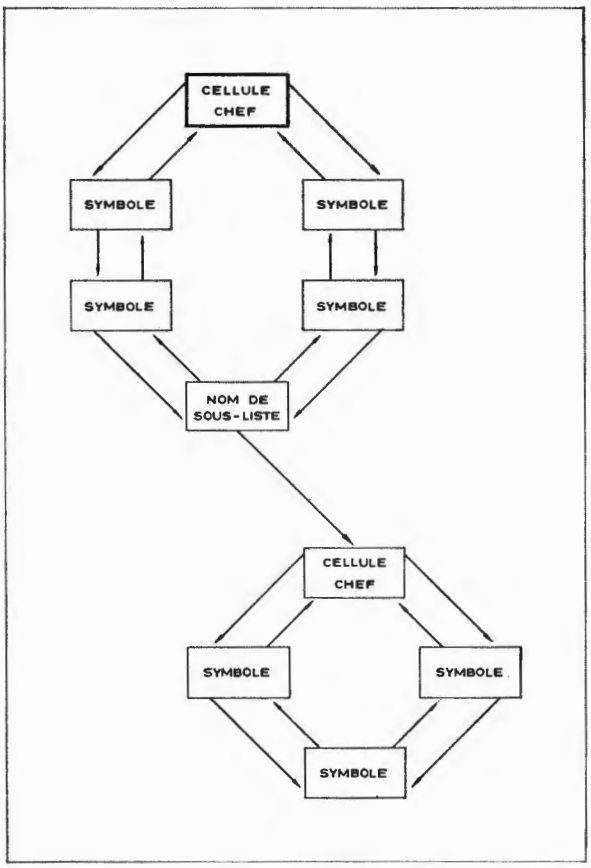

Fig. 1

8 — Newell, AL., (Ed.), *Information Processing Manual.* Englewood Cliffs, N. J., Prentice-Hall, 1961.

9 — McCarthy, John & al., *LISP Programmer's Manual.* Cambridge, Mass, the M. I. T. Press, 1962.

10 — Weizenbaum, Joseph, "Symmetric List Processor", *Communications of the ACM* 6 (a), September 1963 : 524-536.

11 — Bobrow, D. & Raphael, B., "A Comparison of List-Processing Computer Languages", *Communications of the ACM* 7 (4), April 1964 : 231-240.

l'un vers son antécesseur (le rappel de gauche), l'autre vers son successeur (le rappel de droite). C'est pourquoi on appelle ces structures des *listes symétriques;* elles n'ont pas d'orientation préférentielle et peuvent être parcourues aussi bien de gauche à droite que de droite à gauche. La dernière cellule de la structure contient un rappel de droite vers la première cellule qui est la « cellule-chef »; à son tour, la cellule-chef contient un rappel de gauche vers la dernière cellule (Fig. 1).

Les branchements de l'arbre sont représentés en SLIP par une cellule contenant l'adresse de la cellule-chef d'un sous-arbre. La cellule de branchement est le « nom » de ce sous-arbre.

De la même façon, la cellule qui contient l'adresse de la cellule-chef de l'arbre principal est appelée « nom » de cet arbre. Le nom de l'arbre principal est mémorisé quelque part dans la mémoire en dehors de la structure elle-même. Quand on possède l'adresse de la cellule, on peut commander la lecture et la transformation de la structure arborescente, sa comparaison avec d'autres structures, etc.

SLIP dispose d'un sous-programme particulier, destiné à créer ce qu'on appelle un « lecteur ». Le lecteur se déplace le long de la structure et lit le symbole qui se trouve au point indiqué par le programmeur. Il peut lire les sous-arbres ou les sauter le cas échéant, et n'a pas de préférence pour le mouvement direct ou rétrograde. Ainsi, il peut utiliser le mouvement direct pour atteindre un symbole qui se trouve plus près du début de la structure, et utiliser le mouvement rétrograde si ce symbole se trouve plus près de la fin.

Le langage SLIP fonctionne à un haut niveau de subjonction. Il est construit à partir de quelques programmes élémentaires dits « primitifs », écrits dans le langage de l'ordinateur. Ces programmes élémentaires s'emboîtent dans d'autres programmes destinés à exécuter des opérations plus complexes. A leur tour, ces programmes s'emboîtent dans d'autres plus complexes, et ainsi de suite.

Pour communiquer en SLIP avec l'ordinateur, il suffit de savoir ce que peuvent faire les programmes que ce langage met à notre disposition et de les combiner dans un programme correspondant au problème étudié. Les règles de combinaison ou la syntaxe de SLIP sont les mêmes que celles de FORTRAN (abréviation de « Formula Translator »), autre langage de niveau élevé et se prêtant particulièrement bien au traitement des symboles numériques. FORTRAN étant l'un des langages de programmation les plus répandus, les utilisateurs de SLIP ont très rarement à apprendre une nouvelle syntaxe.

Le langage SLIP et les autres langages de listes ont des applications non seulement en linguistique, mais aussi dans les autres Sciences de l'homme. Ils se prêtent au traitement de l'information qualitative et aux structures qualitatives. L'anthropologue-programmeur doit recourir à l'introspection afin de reconstituer minutieusement le processus d'analyse structurale qu'il veut programmer [12]. Les développements de cette analyse

12 — Lévi-Strauss, Claude, *Anthropologie structurale.* Paris, Plon, 1958.

fourniront les bases théoriques nécessaires pour le traitement de l'information qualitative.

Certains anthropologues estiment que l'analyse structurale peut jouer dans l'avancement des Sciences de l'Homme un rôle analogue à celui des mathématiques dans le progrès de la Physique moderne.

Espérons que de la confluence d'une analyse structurale exacte et profonde et de langages de programmation puissants et faciles à manier résultera une technique de traitement des symboles qualitatifs que l'on pourrait appeler la « sémiologie automatique ».

NDLR

Nous regrettons que l'abondance des matières nous oblige à reporter au numéro suivant les rubriques habituelles sur l'**Actualité** et la **Vie des Sociétés.**

Notre numéro 10/4 sera un numéro anniversaire spécial : les lecteurs et les Sociétés qui voudraient en commander des exemplaires supplémentaires voudront bien le signaler au plus tôt à la Rédaction, pour que l'on en tienne compte au tirage.