International Journal of Engineering and Information Systems (IJEAIS)
ISSN: 2643-640X
Vol. 7 Issue 8, August - 2023, Pages: 24-33

From Conditional Syllogistic to Programmatic. Break or continuity?

Léonard BAKAJIKA NGALAMULUME

[1]*Licencié en Sciences Informatiques (orientation : Conception des systèmes Informatiques) et Assistant de Recherche à l'Université Notre Dame du Kasayi (U.KA.), Kasaï Central, Kananga (RDC).* **Leonardbakaj1@gmail.com**

Abstract: *Computer programming is one of the applications of Aristotle's bivalent logic. This had to be demonstrated by establishing the continuity between the so-called categorical syllogism and the so-called hypothetical or simply conditional one. In this sense, the same syllogism with three terms in three propositions can take an implicative configuration putting in correspondence an antecedent and a consequent, this because the reasoning according to Aristotle is only categorical and the hypothetical aspect is only a way of presenting the same reality. The "if...Then" applies both in the sequential succession of instructions and in the event-driven interdependence of actions leading to a programmed result. In the first case, the execution of an instruction is conditioned by the end of the previous instruction. In the second case, an action is triggered by a given event. It is therefore the setting in motion of conditionality, on a scale greater than the traditional structures of conditionality known to any experienced programmer. It is then clear to say that From the conditional syllogistic to the programmatic, there is continuity and not rupture. Or in other words, no unconditional programming, because any program is an explicit or implicit continuation of conditions. The inescapable conditional appears as well in explicit "if" structures, as in iterative functional loops ("while", "for" etc) and even through cause-effect relationships linking events to the execution of coded instructions, which are the brains of any computer application. This article is intended as an introduction for any programmer who must know the role of the essential conditional.*

Keywords – *Syllogism, Conditional, Programmatic, Break, Continuity, etc.*

De la Syllogistique conditionnelle à la Programmatique. Rupture ou continuité?

RESUME

La programmation informatique est l'une des applications de la logique bivalente d'Aristote. Il fallait le démontrer en établissant la continuité entre le syllogisme dit catégorique et celui dit hypothétique ou simplement conditionnel. En ce sens, le même syllogisme à trois termes dans trois propositions peut prendre une configuration implicative mettant en correspondance un antécédent et un conséquent, ceci parce que le raisonnement selon Aristote n'est que catégorique et l'aspect hypothétique n'est qu'une manière de présenter la même réalité. Le "si... Alors" s'applique à la fois dans la succession séquentielle d'instructions et dans l'interdépendance événementielle d'actions conduisant à un résultat programmé. Dans le premier cas, l'exécution d'une instruction est conditionnée par la fin de l'instruction précédente. Dans le second cas, une action est déclenchée par un événement donné. C'est donc la mise en mouvement de la conditionnalité, à une échelle supérieure aux structures traditionnelles de la conditionnalité connues de tout programmeur averti. Il est alors clair de dire que Du syllogistique conditionnel au programmatique, il y a continuité et non rupture. Ou en d'autres termes, pas de programmation inconditionnelle, car tout programme est une continuation explicite ou implicite de conditions. Le conditionnel inéluctable apparaît aussi bien dans les structures explicites "si", que dans les boucles fonctionnelles itératives ("while", "for" etc) et même à travers des relations de cause à effet liant les événements à l'exécution d'instructions codées, qui sont le cerveau de toute application pour ordinateur. Cet article se veut une introduction pour tout programmeur qui doit connaître le rôle du conditionnel essentiel.

Mots clés – *Syllogisme, Conditionnelle, Programmatique, Rupture, Continuité, etc.*

I. INTRODUCTION

Ignorer la continuité entre la syllogistique d'Aristote et la programmatique bloque parfois des esprits épris de la réalisation des systèmes d'information, en éclaboussant l'interdisciplinarité et même la descendance dans le domaine des sciences. Celles-ci pourtant sont nécessaires à la compréhension du raisonnement logique utilisé dans la conception des applications informatiques (10), qui sont, de par le mode de pensée du programmeur, les fruits d'une combinaison des instructions intellectuelles complexes, similaires à celles

qu'accomplissent les mathématiciens, les ingénieurs et les scientifiques (14). C'est dans ce sens que la programmation informatique est déductible de la logique formelle d'Aristote.

En effet, "la syllogistique d'Aristote est un système dont l'exactitude dépasse même celle d'une théorie mathématique, et c'est là son mérite le plus durable"(5), et la logique moderne est un niveau de développement supérieur de la logique formelle antique, dont la logistique d'aujourd'hui n'est ni plus ni moins qu'une suite et une extension (5). Appliqué à la programmation, ceci se découvre dans les implications tacites ordonnant l'enchainement des instructions codées et ensuite soumises à une exécution sous forme d'un programme. En d'autres termes, une certaine dépendance guide la réalisation de l'idée d'un algorithme rendu syntaxiquement par un langage spécifique, dans une séquence pouvant se lancer à la suite d'un événement déclencheur déterminé. Dans ce dernier cas, l'implication se joue à deux niveaux, à savoir : l'événement appelant la procédure séquentielle qui, à son tour, fait dépendre ses instructions les unes des autres, suivant une logique. Comment le démontrer ?

Rechercher d'abord le raisonnement hypothétique dans le catégorique, avant de tirer au clair la permanence du premier dans les algorithmes et les codes sources. Ainsi, nous pourrons éradiquer l'acception purement théorique de la logique d'Aristote, en ressortant la multi-conditionnalité dans la programmation, partant du postulat qu' "une théorie précise sur le fonctionnement de la pensée peut permettre d'exprimer celle-ci sous forme d'un programme d'ordinateur"(10). Il est possible qu'à la lecture de cet article, on soit capable "de rédiger des programmes en privilégiant l'aspect algorithmique, sans être pollué par la complexité et la technicité d'un langage donné"(3).

II. AUX SOURCES DE LA CONDITIONNALITE

La logique est une partie intégrante de la vie, comme l'entend Aristote, en considérant que le jugement requiert une multitude de procédures d'application à la réalité (9). Par ailleurs, "il est une actualité de la syllogistique aristotélicienne qui impose encore aujourd'hui les règles de l'argumentation logique dans la production du savoir scientifique"(9). Aristote n'a jamais conçu un appareillage logique ou syllogistique qui serait opposé à un ensemble de procédures opérant sur la relation intentionnelle entre les termes et les concepts représentant ainsi les éléments de la nature. D'où le caractère onto-logique de l'opération syllogistique, résultant de la correspondance langage-objet(9) ou lois de la matière avec celles de la pensée (9).

La postérité a semblé diviser l'indivisible, en parlant du raisonnement hypothétique comme opposé au catégorique, ce dernier tenant son nom d'Aristote, alors que l'autre serait la suite non imaginée au départ élaborée par l'école stoïcienne. Le professeur Yvan PELLETIER le déplore : "Il est tellement de mise, depuis toujours, de mentionner Aristote, à propos du syllogisme hypothétique, simplement pour dire qu'il ne l'a pas connu, n'en a pas parlé, ou a parlé, sous ce nom, de tout autre chose…"(12).

Cette idée prêterait flanc à l'admission d'une tendance opposée, si le même auteur n'avait pas apporté une précision en stipulant qu' "il serait d'autant plus inopportun d'exposer l'intelligence du lecteur à une démarche aussi fastidieuse que, dès le départ, on peut trouver chez Aristote les grandes lignes d'une doctrine adéquate capable non seulement de bien placer les fondements nécessaires à l'intelligence du syllogisme hypothétique, mais aussi les distinctions indispensables à la solution des extravagances multiples que l'histoire subséquente a développées sur le sujet"(12). C'est que dans la pensée d'Aristote l'affirmation catégorique peut se dire sous une apparence hypothétique, et l'on parlera d'un raisonnement hypothético-catégorique. Découvrons-le.

II.1. Le raisonnement n'est que catégorique

Pour Aristote, le raisonnement logique se compose des propositions qui sont les expressions verbales des jugements vus comme ce par quoi s'exprime la pensée catégorique (9). Dans cette optique, un raisonnement hypothétique, qui serait comme une forme essentiellement originale à opposer au raisonnement catégorique est inadmissible. Là-dessus, Pelletier est plus que clair quand il dit : "Raisonner, pour Aristote (…) se fait toujours catégoriquement. Aussi, entrer dans sa conception du syllogisme hypothétique requiert une vue très nette de sa conception du syllogisme catégorique" (12). Et Marcel NGUIMBI de surenchérir : "Nous sommes persuadés que le discours logique onto-logiquement vrai se fonde inéluctablement sur des propositions ou des jugements catégoriques qui savent rendre compte du vrai et du faux onto-logiques, parce que conformes aux faits sur lesquels on juge le vrai ou le faux"(9).

Pour fonder la catégoricité de la pensée, il n'est pas mal de lier l'objet pensé à l'esprit pensant. Ce qui peut aller jusqu'à l'admission d'une relativité appréhensive de la réalité. L'esprit reçoit alors des données qu'il traitera catégoriquement en vue d'une synthèse, la confrontation avec l'onto-logique étant un effort du second moment, pour les cas qui se révèlent en parfaite contradiction avec la pensée universelle. A priori, le jugement se rend par l'affirmation ou la négation de la copulation d'un prédicat face à un sujet, en scellant la relation de correspondance entre le dit et le vécu, la pensée et la réalité pensée. Les deux jugements –affirmatif et négatif– donnent deux propositions contraires, gravitant chacune autour d'un verbe, qui met en clair l'état du sujet portant toute la charge

ontologique du prédicat, laquelle charge "est une situation qui relève de la mise en œuvre du '*dictum de omni et nullo*' qu'Aristote définit parle fait de n'affirmer que lorsqu'il est nécessaire d'affirmer, et de ne nier que lorsqu'il est nécessaire de nier, le principe du '*dictum de omni*' admettant qu'un attribut affirmé à propos d'un concept donné peut être attribué à chacun des individus constituant la classe logique définie par ce concept, et le principe du '*dictum de nullo*' admettant également que ce qui est nié d'une classe logique l'est de chacun des termes qui la constituent"(9). Ceci ne relève pas d'une approche métaphysique ou naturelle, car le logicien se préoccupe des propriétés spéciales acquises par les choses du fait d'être connues. On dit alors qu'elles sont conçues, nommées, divisées, définies, affirmées, niées, proposées, déduites, induites, conclues…(12). C'est l'établissement des rapports pouvant être compris dans le sens du jugement qui importe et non la simple perception du milieu naturel par les sens.

La syllogistique se comprendra ainsi comme le lieu de la véritable projection mathématique, le cadre théorique du raisonnement rendu efficace par la cohérence de la relation de l'objet pensé à la pensée qu'on a de l'objet. Elle se constitue en un important outil de la science, de par l'efficacité de ses enchaînements déductifs, les principes régissant la matière étant aussi ceux régisseurs de la pensée, et parce que toute pensée a nécessairement quelque chose d'analogue dans la nature. Cette analogie se répercute ainsi sur la nature empirique de la vérité qui déborde logiquement son état analytique de départ (12). Ce qui justifie le caractère non-formel de la logique aristotélicienne, grâce surtout à la question onto-logique de la vérité-correspondance, qui nie toute superficialité à un savoir en lui enjoignant d'apporter des réponses aux problèmes réels de la vie, selon le vœu sacré de Gaston BACHELARD qui s'exprime en se questionnant : "Ne faut-il pas aussi que le théoricien se renseigne sur toutes les circonstances de l'expérimentation, faute de quoi ses synthèses peuvent demeurer partielles ou simplement abstraites ?" (1).

La catégoricité du raisonnement est fonction de la définition aristotélicienne du syllogisme : Une phrase où, des choses une fois posées, autre chose qu'elle doit être aussi, du fait qu'elles soient (1). C'est une construction d'un discours logique faisant aboutir des prémisses à une conclusion qui leur est nécessairement liée, comme un ***conséquent*** par rapport aux ***antécédents***, ou comme une opération de synthèse découlant des analyses déployées dans les propositions précédentes. La conclusion rend explicite l'information implicite des prémisses. C'est le "principe de la pertinence déductive selon lequel, raisonner consiste à passer des prémisses vers la conclusion en déduisant, ou en explicitant dans la conclusion l'information des prémisses, de sorte que l'information contenue dans la conclusion ne déborde jamais celle contenue dans les prémisses"(9). Ceci, pour rester dans les limites de la règle affirmant que la conclusion ne peut être plus large que les prémisses. Pelletier parle d'une conclusion issue rigoureusement des propositions qu'on était légitimé d'admettre immédiatement (12). Mais il n'y a pas de proposition sans termes.

II.2. Le trio inaliénable et l'élimination du terme intermédiaire

Aborder la notion des termes du syllogisme est légitime, étant donné que le passage du catégorique à l'hypothétique (conditionnel) jouera, entre autres, sur l'élimination du moyen terme dans le trio inaliénable du raisonnement aristotélicien. En effet, le petit, le grand et le moyen terme orientent le raisonnement pur. Ce qui est logique, dans le sens où le raisonnement, dans sa nature, est une architecture constituée des arguments qui sont eux-mêmes exprimés dans des propositions, lesquelles mettent en évidence les différents jugements, fruits des rapports de convenance ou de disconvenance entre les concepts rendus par les termes. Pour dire qu' "il s'agit de connaitre, de progresser dans la connaissance, et cela consiste toujours à juger de ce qu'un attribut convient ou non à un sujet"(12). Ainsi, en démontrant à chaque étape qu'un attribut est ou n'est pas à un sujet, cela universellement ou particulièrement, on arrivera, grâce à un moyen terme à une conclusion contraignante issue des rapports exprimés par les deux propositions, de manière telle qu'on ne saurait ensuite nier la conclusion sans se contredire. C'est ça syllogiser (12). Le rôle du moyen terme est de rapprocher les deux extrêmes et conséquemment les deux propositions. Ainsi, son élimination, sans signifier l'éradication pure et simple d'un terme, dit plutôt le maintien de deux termes sur trois sans changement de raisonnement.

Cette idée d'éliminer le terme central hantait déjà l'esprit d'Aristote, qui l'a rendue à travers une règle stricte du syllogisme catégorique : la non-figuration du moyen terme dans la conclusion. Mais George BOOLE dit son intention de généraliser les procédures aussi bien médiates qu'immédiates du syllogisme catégorique en ces termes : "l'élimination du terme intermédiaire, du fait qu'elle ne s'effectue que dans la conclusion chez Aristote, est incapable de garantir l'infaillibilité du raisonnement devant rendre compte des phénomènes de la réalité aussi bien physique que logique"(12). Mais est-ce une voie vers l'aspect hypothétique?

II.3. Le raisonnement hypothétique est-il spécifique ?

Sans faire le contour du raisonnement hypothétique, visons le recoin conditionnel. En effet, l'épithète hypothétique dérive du nom Hypothèse qui dit l'idée d'une "supposition destinée à expliquer ou à prévoir des faits"(4). Pelletier définit l'hypothèse en termes d'un "Enoncé sûr, réciproquement, sur lequel, parce qu'il est évident en lui-même, ou démontrable, ou endoxal, il est légitime d'appuyer l'enquête de l'intelligence ; en somme, l'énoncé capable de servir de proposition dans un syllogisme"(12). Ainsi, sans être confondu avec le syllogisme pur, le procédé hypothétique dépend d'une hypothèse, qui signifie autre chose que l'hésitation, la supposition pure

ou l'incertitude. Nous allons le découvrir déjà dans cette mise en garde : "Qui veut être à même d'argumenter aisément sur une chose doit se munir de ses antécédents et conséquents, en étendant largement ces derniers aussi bien aux accidents nécessaires de la chose qu'aux éléments de son essence"(12). Le penseur catégorique s'en fait une préoccupation au même titre que le penseur hypothétique, à une différence près de morphologie dans l'expression de cette pensée. C'est dans cette optique qu'au raisonnement hypothétique, PELLETIER préfère l'expression "Stratégie hypothétique"(12) qui se rend naturellement à partir d'une proposition conditionnelle, corroborant la thèse selon laquelle "le syllogisme conditionnel est moins une espèce du syllogisme hypothétique que son incarnation la plus commune, sa présentation la plus formelle"(12).

Nous l'avons admis certes : "Argumenter, c'est composer deux extrêmes à l'aide d'un moyen"(12), en suivant les règles de la déduction syllogistique, dans sa tendance à se servir du concept du milieu trouvé incontournable selon certaines acceptions. "Mais ce passage médiat d'un terme à un autre se résout nécessairement dans un passage immédiat entre deux termes : sous peine que l'argument s'étire à l'infini, le passage d'un extrême au moyen doit être immédiat, et de même celui du moyen à l'autre extrême. Or le moyen terme auquel, du majeur, on passe immédiatement et duquel on passe immédiatement au mineur est nécessairement antécédent de l'un et conséquent de l'autre, directement ou suivant quelque équivalence"(4), du moins dans la première figure du syllogisme catégorique dénommée "Sub-Prae". Par ailleurs, la nécessité du lien d'antécédent à un conséquent entre les deux termes entraine la formulation conditionnelle comme syntaxe la plus naturelle pour la stratégie hypothétique, à telle enseigne que condition concorde avec hypothèse.

Pour reprendre Aristote, penser c'est toujours penser catégoriquement, et l'apparence conditionnelle ne diminue en rien la teneur de la conclusion d'un tel déploiement, car une alternative constitue la majeure faite de l'antécédent et du conséquent et la mineure détermine deux figures de ce syllogisme, soit en allant de l'antécédent au conséquent ou vice versa.

Il est donc de bon ton de faire une illustration argumentative revêtant les deux formes d'une même pensée, afin de prouver avec Lukasiewicz qu' "un syllogisme aristotélicien est une proposition implicative de la forme 'si α et β, alors γ' "(5) ; même en recourant à cette autre affirmation selon laquelle "chaque syllogisme véritablement aristotélicien est une implication et donc une proposition hypothétique"(5), comme il est possible de passer d'un catégorique en BARBARA, vers un hypothétique de nature $(axb)xc$

III. L'IMPLICATION DANS LA PROGRAMMATION

Généralement, le terme programme évoque une suite ou un ensemble d'activités destinées à produire un résultat, selon qu'elles s'exécutent globalement ou en partie. DELANNOY y voit un ensemble de directives nommées instructions, qui spécifient aussi bien les opérations élémentaires à exécuter, que la manière dont elles s'enchaînent (3). Il prolonge pourtant sa vision en relevant une apparente contradiction définitionnelle faisant que certains voient dans la programmationun jeu de construction enfantin visant l'enchaînement des instructions élémentaires pour parvenir à résoudre n'importe quel problème ou presque, alors que pour d'autres, il s'agit de produire des logiciels avec des exigences de qualité qu'on tente de mesurer suivant certains critères comme l'exactitude, l'utilisation, l'extensibilité, la réutilisabilité, la portabilité, l'efficience, et surtout la robustesse entendue comme aptitude à bien réagir lorsque l'on s'écarte des conditions normales d'utilisation,... (3).

Le nœud de notre réflexion porte la mise en lumière des implications concrètes d'une représentation mentale simpliste ou trop abstraite, comme c'est l'essentiel de l'activité de tout programmeur, à en croire Gérard SWINNEN (14). Nous le ferons par l'analyse des codes sous-jacents et de leur mode d'exécution : soit dans l'ordre défini par le programmeur entre le début du programme et la fin de celui-ci, soit en faisant suite à une action d'un élément déclencheur étranger au code mais susceptible de provoquer l'exécution d'une instruction ou d'une suite d'instructions. L'approche, aussi bien séquentielle qu'événementielle, prouvera que tous les programmes informatiques reposent sur les mécanismes de logique formelle et utilisent le raisonnement déductif avec la syllogistique hypothétique comme grande règle d'inférence (10).

III.1. La programmation séquentielle

Le qualificatif "séquentiel" dérive du substantif "séquence" évoquant "une suite ordonnée d'éléments, d'objets, d'opérations…"(4). Est donc séquentiel ce qui se fait selon une séquence, un ordre préétabli. L'on comprendrait alors un stockage à accès séquentiel (Sequential Access Storage) qui arrange les enregistrements les uns derrière les autres, imposant un parcours de tous les prédécesseurs avant d'atteindre un enregistrement nommé, à la manière d'une lecture sur une bande magnétique, un parcours d'une liste nominale alphabétique, une lecture d'une chaine numérique des valeurs placées en ordre décroissant ou croissant avec un pas d'incrémentation régulier, une transmission d'une information dans un réseau à topologie linéaire etc. Cette illustration non exhaustive témoigne de l'obligation de passer par un nœud ni avant d'atteindre un nj. Mais comment y lire une opération implicative ?

Nous nous sommes assigné dès le début la tâche de retrouver le rôle de la condition dans la programmation, en acceptant comme vraie l'idée selon laquelle "les structures de contrôle servent à préciser comment doivent s'enchaîner les instructions d'un programme"(3). Christophe HARO abonde dans ce sens : "à partir des données du problème (…), des règles opératoires permettent d'obtenir un résultat défini. Mais pour accéder à ce résultat, il faut passer par un ensemble d'étapes intermédiaires : récupérer les données, calculer une moyenne…"(7). C'est le sens qu'il donne à l'algorithme : une succession d'opérations élémentaires à exécuter dans un certain ordre, ou un ensemble de règles opératoires et de procédés définis en vue d'obtenir un résultat déterminé au moyen d'un nombre fini d'opérations séquentiellement ordonnées (7). Voyons un exemple purement programmatique pour une démonstration plus proche de la quintessence de notre sujet :

Soient cet algorithme et ce code en langage Javascript

Algorithme	Codes sources(JavaScript)

Début
Var nom comme texte
Afficher : « Saisir votre nom »
nom ← nom saisi
afficher : « Bonjour » + nom
Fin
	<script>
	var nom;
	nom=prompt("Saisissez votre nom");
	alert("Bonjour " + nom);
	</script>

Cette exécution séquentielle répond à l'esprit de la composition descendante par affinages successifs qui permet le développement d'un programme par niveaux d'abstraction en utilisant à chaque étape des éléments de niveau inférieur (8), mais en quoi est-elle conditionnelle ? Ou encore, comment trouver l'antécédent et le conséquent ?

Le rôle d'antécédence et celui de conséquence étant déterminés par la position de part et d'autre de l'opérateur, nous saurons, tout en considérant l'implication, trouver s'il s'agit des propositions singulières ou des groupes de propositions mises en relation. Pourtant l'algorithme ou les codes ne révèlent pas cette physionomie de l'implication, alors que rien n'échappe à cette réalité, étant vrai que toute opération logique est un problème de sens et non seulement d'orthographe. Ici l'ordre d'agencement des idées instructives est logique à la manière d'une chaine numérique descendante dans un ordre décrit par le séquençage bien qu'il faille fournir au programme les données nécessaires à l'accomplissement de ses tâches (11). Nous le découvrons en analysant les lignes de l'algorithme que nous devrons d'abord nommer en vue d'une formalisation progressive :

a = var nom comme texte
b = afficher : « Saisir votre nom »
c = nom ← nom saisi
d = afficher : « Bonjour » + nom

III.1.1. Var nom comme texte

La déclaration de la variable est le point de départ de ce programme. En effet, il faudra une réservation avant une affectation. Ainsi le programme qui commencera trouvera un espace mémoire réservé à la conservation temporaire de la valeur ***nom*** qui sera utilisée par la suite. Michael HALVORSON parle de la variable en termes d'emplacement de stockage de données dans le programme ou de conteneur utilisé pour conserver toute sorte d'information (6). Claude DELANNOY explicite en disant qu'"une variable est donc un nom qui sert à repérer un emplacement donné de la mémoire centrale"(3). Dans le cas d'espèce, la declaration constitue, à elle seule, un antécédent, une proposition "***a"*** par rapport à la ligne suivante, comme il est inimaginable d'habiter une maison avant de l'avoir construite.

III.1.2. Afficher : « Saisir votre nom »

L'affichage du message demandant la saisie du nom, ne sera pas possible tant qu'il n'y aura pas d'espace mémoire réservé à accueillir cette valeur texte. La symbolisation serait : (***a → b***), le message demandant la saisie du nom étant un conséquent par rapport à l'antécédent qui est la déclaration de la variable. Nous avons deux propositions singulières mises en implication directe, la deuxième découlant logiquement de la première.

III.1.3. nom ← nom saisi

Il serait possible de penser à un cumul par conjonction de la ligne "***a et b"***, pour déclencher "***c***". Ce qui correspondrait à : "***(a ∧ b) → c"***. Nous entamons la série des propositions complexes engageant un autre opérateur comme proposition singulière par rapport à l'opérateur principal. C'est la matrice conjonctive de ***(a et b)*** qui implique "***c"***. Aucun nom ne serait affecté à la variable "***nom"***, si

cela n'était pas demandé clairement par le programme et le programme n'afficherait jamais un message demandant la saisie, si la variable destinataire n'avait pas été déclarée préalablement.

III.1.4. afficher : « Bonjour » + nom

L'affectation de la valeur saisie à la variable est aussi une conséquence du cumul des opérations précédentes. ***((a ∧ b)∧ c) →d***. la fin suppose la réalisation de tout ce qui précède. Alors seulement le programme peut s'arrêter. Fort de tout ce qui précède, la programmation séquentielle est une application conditionnelle basée sur l'exécution selon un ordre de préséance dans la succession. Cette logique peut être apparentée à celle d'un sorite aristotélicien, qui fait fi des impératifs de la catégoricité, pour enchainer des propositions, en faisant du prédicat de la précédente un sujet pour la suivante, jusqu'à la conclusion qui reprend le sujet de la première proposition et la conclusion de la dernière. La formulation conditionnelle peut répondre à un algorithme admettant des si imbriqués, c'est-à-dire un cas où l'une des parties d'une structure de choix contient à son tour une structure de choix (3). L'implémentation se ferait avec tout langage de programmation possible.

III.2. L'incontournable conditionnel dans la programmation événementielle

Bien que le raisonnement conditionnel soit inhérent à toute conception informatique, nous voulons le circonscrire ici dans le cadre d'une programmation pilotée par les événements. A ce propos Halvorson tranche : ″Pour traiter les informations d'une procédure événementielle, l'expression conditionnelle est l'un des outils les plus utiles″(6), prenant l'événement pour l'antécédent qui provoque d'autres opérations de l'application. Examinons d'abord quelques termes.

III.2.1. Quelques définitions essentielles

1. Un événement

Etymologiquement, nous découvrons que l'événementiel est lié à l'événement, comme le séquentiel l'est face à la séquence. Alors, événement qu'est-ce ? Le Larousse lie l'évocation de ce concept à la signification de ce qui se produit, ce qui arrive ou apparait (4). Il en résulte, pour la programmation, une acception allant dans le sens d'un stimulus étranger différent des propriétés intrinsèques d'un contrôle ou d'un objet. Il nous ramène à un comportement interne ou externe lié au contrôle, dans le sens où l'objet concerné peut être actif ou passif. On parlerait mieux des événements liés aux périphériques à côté de ceux liés au système et aux applications. Néanmoins, chaque objet possède un jeu prédéfini d'événements auxquels il peut répondre, et ces derniers sont listés lors de la sélection d'un d'objet (6). Le plus important est que le déclenchement d'un événement singulier entraine une action conséquente. Comme qui dirait: Si événement ***e*** alors code ***c***.

2. Objet ou contrôle

Le mode graphique de conception insinue la présence des formes permettant d'accéder aux fonctionnalités qui constituent l'application résultante. Elles sont connues sous le nom d'objets, outils ou encore sous l'anglicisme contrôles. Par ces formes, l'utilisateur interagit avec le programme, en lui passant des commandes (Exemple : par une zone de saisie, un inputbox…), en recevant de lui des propositions en vue d'une sélection ou d'une validation (Exemple : par une listBox, un comboBox, une arborescence, une boite de dialogue, un messageBox…), etc. Halvorson dit que ″ les contrôles (…) sont des outils graphiques utilisés pour construire l'interface utilisateur d'un programme…″(6). Ils constituent le contenu de la boite dite à outils dans l'environnement de développement.

3. Les propriétés des contrôles

Chaque contrôle a des caractéristiques appelées propriétés. C'est le cas de la dénomination ***(name)***, les dimensions ***(size)***, l'affichage ***(text ou caption)***, la couleur de fond ***(backcolor)*** etc. La sélection d'un objet affiche, en même temps que les événements, les propriétés qui lui sont applicables. Les propriétés sont sujettes aux modifications voulues par le concepteur, qui le fait dans la fenêtre appropriée ou dans le code source. Il s'agira de l'affectation d'une valeur à la propriété nommée. Revenons à l'événement comme antécédent d'une proposition conditionnelle.

III.2.2. L'événement comme antécédent

L'expression "The world is event-driven" du Professeur David Luckham confirmait le fait que la nature est truffée de preuves d'existence des événements. Nous pouvons à peine évoquer la réaction d'un lion face à l'événement de l'arrivée d'un zèbre, comme celle du zèbre apercevant un lion (15). L'événement constitue un stimulus attendant une réaction, qui, dans notre cas vient d'un

programme informatique. Le concepteur établit les différentes relations de dépendance événement-codes, pendant qu'il soumet les blocs d'instructions aux commandes spécifiques de l'utilisateur. Les événements sont ainsi des éléments du langage entre le programme et la personne. L'ordre d'exécution des blocs de codes est dicté par l'ordre d'arrivée des commandes qui depend de l'utilisateur, et non du concepteur, comme c'est le cas en mode console. En gros, l'antécédent est le déclenchement de l'événement et le conséquent est l'exécution du code y afférent. En d'autres termes, les instructions à exécuter sont regroupées dans une procédure événementielle que l'on associe au composant concerné par l'événement. C'est la logique du stimulus → réaction spécifique. Sachant que la véracité de l'implication tient à toutes les possibilités excepté le ***1 → 0***, nous excluons le fait que l'événement se produise et que l'exécution du code qui lui est lié soit nulle.

Il y a lieu de relever à ce niveau le fait que l'événementiel fait appel au séquentiel, en termes du contenu des procédures soumises à des événements sur les contrôles ou objets. Le syllogisme explicatif présentera un antécédent d'une seule phrase, l'événement et un conséquent composé qui est une séquence d'instructions exécutables selon un ordre défini. Du moins, si l'événementiel peut inclure le séquentiel, l'inverse est inadmissible, restant vraie la présence d'autres opérations qui facilite la concaténation de ces propositions en exécution descendante.

III.3. L'implication entre des formules non atomiques

La notion de l'atomicité des formules tient au fait que celles-ci sont indécomposables alors que la non-atomicité est obtenue à partir d'autres formules plus petites, en appliquant des opérations logiques qui sont généralement la négation, la conjonction, la disjonction, voire le conditionnel (13). Nous soulevons ici la notion des conditions complexes qui sont "formées de plusieurs conditions simples reliées entre elles par des opérations logiques : et, ou et non"(3).

III.3.1. Le raisonnement conjonctif dans la programmation

Prenons appui sur une double antécédence conjonctive comme condition pour le lancement d'un programme, pour comprendre la procédure séquentielle soumise à l'événement ***buttonvalider_Click***, laquelle procédure inclut deux méthodes liées entre elles par une autre conditionnalité chronologique (***successive***), qui soumet le début de la méthode identification à la fin de la méthode authentification. Quel rôle joue alors la conjonction ? La première méthode sélectionne les identifiants conforme aux valeurs saisies dans des zones de saisie réservées (***mal1 et mal2***), selon ce texte de commande : ***"select * from adminitrateur where auteur='"+ mal1.Text +"' and mot_passe = '"+ mal2.Text + "'"***. Le résultat de la sélection s'affiche sur des étiquettes (***util1 et util2***) ayant la propriété visible fixée à ***false***. Chronologiquement, la fin de cette procédure amorce le lancement de la seconde. Comme qui dirait : ***Si authentification, alors identification*** (a → b). La procédure de l'identification se traduit à travers ce code essentiel :

```
if (util1.Text == mal1.Text && util2.Text == mal2.Text) { Form f1 = new Form1();f1.Show(); this.Hide(); }
```

Passons par une formalisation, afin de vérifier la véracité.

a = util1.Text == mal1.Text : la valeur saisie dans la zone appelée util1 égale à la valeur affichée dans l'étiquette mal1
b = util2.Text == mal2.Text : la valeur saisie dans la zone appelée util2 égale à la valeur affichée dans l'étiquette mal2
c = f1.Show() : f1 s'affiche
d = this.Hide() : le formulaire courent disparait

La comparaison entre les valeurs sélectionnées de la table et celles saisies dans les zones réservées exige la correspondance entre les valeurs Text de util1 et mal1.Text et entre util2 et mal2. La double conformité traduite par "&&", équivalant à "and", lance le programme, quand les deux propositions sont évaluées à Trueet la forme visible devrait être (a Λ b) → (cΛd) ; Ce qui passerait de la majeure à la conclusion, sans poser ou nier une partie de l'alternative. Evidemment la sortie donnerait une contradiction démontrable avec la méthode indirecte que nous avons choisie pour des raisons d'économie (11). La valeur "Faux" de l'implication, vous le constaterez, n'est pas contredite par les valeurs des variables. Ce qui prouve l'insuffisance de notre argumentaire, qui devait nier ou poser une partie de l'alternative, avant d'aller à la conclusion.

(a	Λ	b)	→	(c	Λ	d)
1	1	1	0	1	0	0

Pour être plus complet, et remédier à cette tare susceptible d'engendrer une erreur logique, entendue comme une erreur humaine ou une erreur de programmation qui fait que le code génère un résultat erroné (6), nous avons tiré cette déduction en posant l'antécédent pour circonscrire la conclusion, comme on peut le lire ci-dessous.

(((a	Λ	b)	→	(c	Λ	d))	Λ	(a	Λ	b))	→	(c	Λ	d)
1	1	1	1	1	1	1	1	1	1	1	0	1	0	0

Antécédent — Conséquent

Clairement cette expression n'admet pas de contradiction, à remarquer la valeur de la proposition singulière d qui doit varier (de 1 dans l'antécédent à 0 dans le conséquent), pour asseoir la fausseté de l'implication centrale. Or en sachant que le rejet de la contradiction revient à l'admission de son contraire, nous allons conclure à une tautologie ou une loi logique qui confirme la validité du raisonnement ; chose aussi démontrable grâce à la méthode des tableaux sémantiques qui est similaire à l'indirecte, en recherchant les facteurs de contre-exemple.

V		F	
		1.(((a Λb) → (c Λd)) Λ (a Λb)) → (c Λd)	
2.1.(((a Λb) → (c Λd)) Λ (a Λb))		3.1.(c Λd)	
		4.3.c	
		5.3. d	
6.2. (((a Λb) → (c Λd))			
7.2. (a Λb))			
8.7.a			
9.7.b			
I.V	II.V	I. F	II.F
	10'/6 (c Λd))	10/6 (a Λb)	
11 /10' c			
12/10' d			
		13/10 a	14/10 b

La contradiction de la contradiction est prouvée par la présence de toutes les quatre variables (a, b, c et d) du côté Vrai et du côté Faux, en même temps. Utilisons à présent la table de vérité comme preuve de visibilité de la loi logique véhiculée dans l'expression mise en cause.

(((a	Λ	b)	→	(c	Λ	d))	Λ	(a	Λ	b))	→	(c	Λ	d)
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	1	1	1	1	1	0	0
1	1	1	0	0	0	1	0	1	1	1	1	0	0	1
1	1	1	0	0	0	0	0	1	1	1	1	0	0	0
1	0	0	1	1	1	1	0	1	0	0	1	1	1	1
1	0	0	1	1	0	0	0	1	0	0	1	1	0	0
1	0	0	1	0	0	1	0	1	0	0	1	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1	0	0	0
0	0	1	1	1	1	1	0	0	0	1	1	1	1	1
0	0	1	1	1	0	0	0	0	0	1	1	1	0	0
0	0	1	1	0	0	1	0	0	0	1	1	0	0	1
0	0	1	1	0	0	0	0	0	0	1	1	0	0	0
0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
0	0	0	1	1	0	0	0	0	0	0	1	1	0	0
0	0	0	1	0	0	1	0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	0	0	0	0	1	0	0	0

III.3.2. Le raisonnement disjonctif dans la programmation

Certaines circonstances exigeraient que l'une des clauses possibles soit respectée pour que la suite soit possible. Dans ce cas le programme se lance à condition que le mot de passe ou le nom soit authentique. Suivons ce code :

```
if (util1.Text == mal1.Text || util2.Text == mal2.Text)
        { Form f1 = new Form1();f1.Show(); this.Hide(); }
```

Il est possible que l'un soit vrai ou que les deux soient vrais. C'est la raison d'être de "||" qui équivaut à "Or".

III.3.3. Le conditionnement par la négative

Restant dans la même logique que celle qui a piloté le lancement du programme moyennant le mot de passe et le nom utilisateur, l'ouverture peut être possible sur les valeurs par défaut admise par le concepteur. Il s'agit de la valeur vide dans les zones réservées et les zones de saisie (mal1 et mal2 ; util1 et util2). Il se pourrait que le clic de validation trouve les conditions réunies en comparant les zones vides en présence.Ce que la programmation préventive évite en préconisant la réduction des possibilités d'erreur, entre autres, par la soumission de la suite du programme à la modification de la valeur vide de la propriété Text, comme on peut le

lire à travers lecode suivant : if (mal1.Text != ""){authentification(); identification()}disant que si l'étiquette mal1 n'est pas vide, alors…A la rigueur, on reflèterait la non atomicité de la formule par la négative dans l'expression comme : si (non("la zone nommée mal1 est vide")) alors… La combinaison de plusieurs conditions réunies aussi bien par la conjonction que par la disjonction introduit une nouvelle vision dans la vérification de la véracité ou de la fausseté. Il s'agit de la notion de court-circuit.

III.4. Le conditionnel court-circuitant

Ceci tient à la gestion du temps d'exécution du programme. Au fait, les possibilités de True ou dufalse n'étant toujours pas en nombre égal pour tous les foncteurs, il y a possibilité d'écourter le raisonnement. Par conséquent, sachant que le vrai pour la conjonction s'élève à $\frac{3}{4}$, nous pouvons les réduire à deux sans gâcher l'issue de l'analyse : le constat de la fausseté d'une seule proposition suffisant pour conclure à la fausseté de toute l'opération. Dans le même ordre d'idées, une seule proposition vraie conduirait directement à la véracité de la disjonction, comme pour l'implication, la fausseté de l'antécédent ou la véracité du conséquent donnerait la valeur "Vrai" à toute l'opération (11). Appliquons-le aux opérateurs multiples.

En effet : Soient a et b deux propositions et Et l'opérateur conjonctif qui les relie ; sachant que la véracité de Et est fonction de la véracité simultanée de a et de b, le fait que a soit déjà fausse annule toute possibilité du vrai pour Et. Halvorson dit à ce sujet : "Imaginons une instruction If avec deux conditions connectées par un opérateur ***AndAlso*** (…) Si la première condition s'évalue à False, Visual Basic passe à la ligne suivante ou immédiatement à l'instruction Else, sans tester la deuxième condition. Cette condition partielle, ou court-circuit d'une instruction If, est logique"(6). C'est le cas de l'authentification de l'utilisateur moyennant conjointement un nom et un mot de passe.

Dans le même ordre d'idées, la disjonction permet un court-circuit positivant le raisonnement, en cas d'une première proposition évaluée vraie. Ceci tient au fait que dans l'argument a ou b, ou n'est faux que si acomme best faux.Visual Basic rend cet opérateur par l'expression ***OrElse*** (6), réduisant la complexité, tout en donnant une sortie logiquement cohérente par une voie efficiente.

CONCLUSION

Le retour aux sources nous a prouvé que le syllogistique conditionnel n'est qu'une autre façon de raisonner catégoriquement. Alors, on relèvera la non-discontinuité entre ce raisonnement d'Aristote et la programmation informatique, pour faire de ce travail un vade me cum élémentaire du programmeur, qui n'ignore pas que la logique de programmation entre dans cette même vision, en se voulant une application particulière de la logique antique revue à la moderne. Raison qui nous a poussé à aller aux origines, pour déduire l'hypothétique du catégorique, sans établir une grande dichotomie entre ces deux facettes d'un même raisonnement, et cela, en passant par certaines voies comme l'annulation du moyen terme.

La programmation est une application de l'implication, par la succession des instructions selon un ordre du concepteur, et aussi la mise en action des procédures même séquentielles à partir des événements spécifiques vus comme ***stimuli*** des comportements appariés aux objets donnés. Nous avons montré ainsi que la programmation événementielle peut inclure celle séquentielle, sans permettre l'inverse, aussi à travers les conditions complexes qui entrainent la multiplication des foncteurs dans les deux parties principales obligatoires à la réflexion conditionnelle. La programmation peut envisager alors le court-circuit rationnel dans le traitement de ces conditions multiples, sans biaiser la sortie.

Faut-il conclure que toute conception programmatique est tributaire de l'implication logique qui ne va pas sans évoquer d'autres foncteurs ? La réponse est oui : L'incontournable conditionnel apparaît aussi bien dans des structures explicites du "if", comme dans des boucles fonctionnelles itératives ("while", "for" etc) et même à travers les relations causes-effets reliant les événements à l'exécution des instructions codées, cervelle de toute application informatique. Nous avons fait nôtre la préoccupation de l'ingénieur Christophe Haro qui présentait son livre sur l'algorithmique en refusant de bloquer la pensée du lecteur dans les carcans d'un langage particulier, mais plutôt en le poussant à raisonner pour concevoir (7).

BIBLIOGRAPHIE

1. BACHELARD G., *Le Rationalisme appliqué*, Paris, P.U.F., 3è Edition, 1966.
2. DELANNOY C., *S'initier à la programmation. Avec des exemples en C, C++, C#, Java et PHP*, Paris, Eyrolles, 2008.
3. DELANNOY *C., Apprendre le C++*, Paris, Eyrolles, 2007.

4. GARNIER Y. et KAROUBI *L., Larousse Maxipoche 2008*, Paris, Ed. Larousse, 2007.
5. GOURINAT J.B., "Aristote et la 'logique moderne' : sur quelques paradoxes de l'interprétation de Lukasiewicz", dans *Philosophia Scientiae*, 15-2(2911), pp.69-101.
6. HALVORSON M., *Visual basic 2008. Etape par étape*, Paris, Donod, 2010.
7. HARO Ch., *Algorithmique. Raisonner pour concevoir*, sl. ENI Editions,
8. MEYER B., "Quelques concepts importants des langages de programmation modernes et leur expression en Simula 67", dans *Bulletin de la direction des études et recherches Série C– Mathématiques, Informatique* n°1, 1930, 1980, pp.89-150.
9. NGUIMBI M., Boole, critique d'Aristote : la logique de l'élimination du moyen terme, dans *Philosophia Scientiae*, 14-1(2912), p.84-125.
10. NTUMBA BETU KOLAMOYO D., Approche du "Système Expert de prise en charge des épidémies", dans *Revue de l'U.KA.*, Vol.3, n°5(2015), pp.103-117.
11. NTUMBA S. et alii, Programmation orientée-objet et classique. Evaluation comparative et application avec C++, Java et C#, dans *Revue de l'U.KA., Vol.1*, n°2(2013), pp.101-131.
12. PELLETIER Y., *Le Syllogisme hypothétique (Sa conception aristotélicienne)*, l'Université Laval, Société d'Études Aristotéliciennes, 2005.
13. ROEGEL D., *Logique formelle et modélisation du raisonnement. Notions de base*, 1999, Centre de Recherche en Informatique de Nancy.
14. SWINNEN G., *Apprendre à programmer avec Python 3*, Paris, Eyrolles, 2012.
15. David Luckham cité par SMAIL DJERIR, *Etude et implémentation d'un moteur d'inférence dans une architecture orientée événements, Mémoire présenté en vue de l'obtention du titre d'ingénieur civil informaticien à finalité ingénierie informatique.* Université Libre de Bruxelles, 2009-2010, p.13.