Les mathématiques de la langue : l'approche formelle de Montague *

Yannis Haralambous

16 mai 2014

Résumé

Nous présentons une méthode de modélisation de la langue naturelle qui est fortement basée sur les mathématiques. Cette méthode, appelée «sémantique formelle», a été initiée par le linguiste américain Richard M. Montague dans les années 1970. Elle utilise des outils mathématiques tels que les langages et grammaires formels, la logique du 1er ordre, la théorie de types et le λ-calcul. Nous nous proposons de faire découvrir au lecteur tant la sémantique formelle de Montague que les outils mathématiques dont il s'est servi.

1 Introduction

La modélisation mathématique de la langue que nous présentons ici, date des années 70 du siècle dernier. Les langages artificiels (comme le langage de la logique ou les divers langages de programmation) existaient déjà mais les linguistes pensaient que les langues naturelles étaient bien trop chaotiques pour que l'on puisse leur appliquer les mêmes méthodes que pour les langages artificiels. Arrive alors un certain Richard M. Montague qui prétend que c'est tout à fait possible, à condition d'utiliser des outils mathématiques évolués, et il le démontre à travers trois articles (dont le premier, de 1970, s'intitule, de manière assez provocante, *L'anglais en tant que langage formel* [13]). Malheureusement Montague est mort assez jeune [1] mais son travail a néanmoins révolutionné la linguistique et l'a rapproché des mathématiques et de l'informatique.

Dans cet article nous allons décrire les outils mathématiques dont s'est servi Montague pour modéliser la langue naturelle : les langages formels, les grammaires formelles, le λ-calcul typé, la logique du 1er ordre, la théorie des ensembles. Ce texte s'inspire fortement (tout en le simplifiant) de l'ouvrage [4].

2 L'approche de Montague

La langue sert avant tout de parler du monde. Nos points de départ seront donc le «monde» et la «langue». Le «monde», que nous noterons $\mathfrak{Monde}$ peut être le monde réel, un monde imaginaire, ou simplement un ensemble d'objets ou d'idées abstraites (d'ailleurs, à la section 7.3 nous parlerons de l'ensemble des «mondes possibles»). La «langue (naturelle)», que nous noterons $\mathfrak{LangNat}$, sera un ensemble de phrases françaises, orthographiquement et syntaxiquement correctes, et qui se réfèrent aux objets de l'ensemble $\mathfrak{Monde}$.

Pour étudier la langue, Montague propose d'intercaler entre $\mathfrak{LangNat}$ et $\mathfrak{Monde}$, un ensemble de formules écrites dans un langage artificiel intermédiaire, basé sur la logique de 1er ordre et le λ-calcul typé. Nous noterons par $\mathfrak{LangLog}$ l'ensemble des traductions des phrases de $\mathfrak{LangNat}$ dans ce langage.

*Une version plus évoluée de ce texte va paraître dans la revue *Quadrature* (`http://www.quadrature.info`) en 2015.

1. Et de manière tragique, au point où sa mort a inspiré un roman policier, la *Sémantique du meurtre* d'Aifric Campbell [3].

On a donc la situation suivante :

$$\mathfrak{LangNat} \xrightarrow{\textbf{Trad}} \mathfrak{LangLog} \xrightarrow{\textbf{Inter}} \mathfrak{Monde},$$

où **Trad** est la *traduction* des phrases françaises en langage intermédiaire, et **Inter** le lien entre $\mathfrak{LangLog}$ et le $\mathfrak{Monde}$ que l'on appelle *interprétation*[2].

Dans la suite nous utilisons des *caractères bâton inclinés* pour représenter les mots de $\mathfrak{LangNat}$, et des **caractères gras** pour représenter les entités de $\mathfrak{Monde}$. Ainsi, *Gérard* $\in \mathfrak{LangNat}$ sera un nom (propre) masculin de six lettres, qui se prononce /ʒe.ʁaʁ/. Par contre, **gérard** $\in \mathfrak{Monde}$ sera un individu, en chair et en os, identifié par ce prénom. On va considérer que dans $\mathfrak{Monde}$ il n'y a qu'un seul **gérard**.

Dans cet article nous allons décrire, dans l'ordre, $\mathfrak{LangNat}$ (§ 3), $\mathfrak{LangLog}$ (§ 4), **Trad** (§ 5) et $\mathfrak{Monde}$/**Inter** (§ 6).

3 La structure de la langue

Parmi les nombreuses manières d'étudier la langue naturelle, celles qui vont nous intéresser dans cet article sont la *syntaxe* et la *sémantique*.

La syntaxe étudie *grosso modo* la composition des phrases, c'est-à-dire la manière dont les mots se combinent pour former des phrases, selon leurs catégories grammaticales (nom, adjectif, adverbe, verbe, etc.).

La sémantique est l'étude de la signification.

Il s'avère que la syntaxe est un outil indispensable pour l'étude de la sémantique grâce au principe suivant :

Principe de compositionnalité. *La sémantique d'une phrase s'obtient à partir des sémantiques de ses parties et de la manière dont elles ont été composées (= la syntaxe de la phrase).*

En effet, pour comprendre le sens de la phrase *Gérard aime Alice* il faut savoir ce que sont (ou peuvent être) Gérard, Alice et l'action d'aimer, et identifier par la syntaxe de la phrase le fait que c'est Gérard qui est en position de sujet et c'est donc lui qui aime Alice, et non pas l'inverse.

Pour commencer, voyons comment analyser syntaxiquement les phrases de $\mathfrak{LangNat}$. Les outils mathématiques qui vont nous servir sont les *langages formels* et les *grammaires formelles*.

3.1 Langages formels

Définissons d'abord la notion de *monoïde libre*. Soit Σ un ensemble quelconque que nous allons appeler *alphabet*.

Définition 1. *On appelle* ***monoïde libre*** $(\Sigma^*, \cdot, \varepsilon)$ *sur* Σ*, l'ensemble de tous les produits* $x_1 \cdots x_n$ $(n \geq 1)$ *d'éléments de* Σ*, auquel on ajoute l'élément* ε*, appelé* élément neutre. *La loi* $\cdot$ *doit être associative.*

On appelle *concaténation* la loi du monoïde, et on s'autorise de ne pas la noter. Les éléments du monoïde sont appelés «mots», et ε est le «mot vide».

Définition 2. *Un* ***langage formel*** L *sur un alphabet* Σ *est un sous-ensemble quelconque du monoïde libre* Σ^*.

Ce qui fait la force des langages formels, c'est que — malgré son appellation —, un «alphabet» n'est pas forcément un ensemble de lettres. Les éléments d'un «alphabet» peuvent être des mots-clé d'un langage de programmation (dans ce cas, un «mot» sera, par exemple, un programme dans ce langage), des symboles mathématiques (un «mot» sera alors une formule), des nucléotides (un

2. Attention, ici le mot «interprétation» a un sens technique strict (cf. § 6), qu'il ne faut pas confondre avec celui du langage courant.

«mot» sera alors un sous-brin d'ADN), etc. Dans notre cas, l'«alphabet» sera formé de mots de la langue française, et les éléments de $\mathfrak{LangNat}$ seront des *phrases* de la langue française.

Se pose alors la question : si un langage formel est un sous-ensemble quelconque du monoïde libre, alors comment le décrire ?

Toute la difficulté est là : comment choisir les mots qui forment un langage formel — ou, de manière équivalente, comment décider si un mot appartient ou non à un langage formel donné (éventuellement infini) ?

Une méthode pour décrire des langages formels est celle des *grammaires formelles*.

3.2 Grammaires formelles

Une grammaire formelle est un ensemble de *règles de production*. En partant d'un symbole appelé «axiome de départ», on applique ces règles à un ensemble de symboles auxiliaires appelés «non-terminaux» jusqu'à aboutir aux mots du langage que l'on souhaite définir (et qui donc s'appellent «terminaux», puisqu'on ne peut aller plus loin).

Définition 3. *Soit Σ un alphabet. Une* ***grammaire formelle*** *G est un quadruplet (S, V, Σ, P) où S est un élément appelé* ***axiome de départ****, V l'ensemble des* ***symboles non-terminaux****, Σ l'ensemble des* ***symboles terminaux****, et P l'ensemble des* ***productions****.*

Une ***production*** *p est un couple (X, u), où $X \in V \cup \{S\}$, et $u \in (\Sigma \cup V)^*$ (c'est-à-dire le monoïde libre sur les éléments de Σ et de V).*

Une suite d'applications successives de productions est appelée une ***dérivation****.*

On note une telle production $p : X \to u$.

Une production envoie donc un élément non-terminal (d'où son nom de «non-terminal») ou l'axiome de départ, à un mot comportant des terminaux et des non-terminaux.

On peut appliquer des productions aux mots : si $p : X \to u$ et si le mot w s'écrit $w_1 X w_2$ alors p produira le mot $w_1 u w_2$. Mais attention : si w s'écrit $w_1 X w_2 X w_3$ alors une application de p peut produire $w_1 u w_2 X w_3$ ou $w_1 X w_2 u w_3$ (mais pas les deux en même temps, pour que cela arrive il faudrait l'appliquer deux fois). Et si w ne contient pas X, alors p laisse w inchangé.

On se pose alors la question suivante : est-ce qu'un mot w peut être l'image d'une dérivation partant de S ? On dira alors que w est *G-dérivable*.

Définition 4. *Soit $G = (S, V, \Sigma, P)$ une grammaire formelle. Le* ***langage formel engendré par*** *G, noté L_G, est l'ensemble des éléments de Σ^* qui sont G-dérivables.*

Notons bien que les éléments de L_G sont tous des mots sur des *terminaux*. En effet, l'axiome de départ et les non-terminaux sont indispensables pour obtenir le langage formel engendré mais n'y apparaissent pas eux-mêmes. Dans notre cas, les non-terminaux vont être des catégories et fonctions grammaticales. Prenons un exemple. On peut décrire syntaxiquement la phrase *Gérard dort* par la grammaire

p_1 : S $\to$ GN GV
p_2 : GN $\to$ N
p_3 : GV $\to$ V
p_4 : N $\to$ *Gérard*
p_5 : V $\to$ *dort*

où S est l'axiome de départ ; les symboles GN (groupe nominal), GV (groupe verbal), N (nom) et V (verbe) sont des non-terminaux ; et, enfin, *Gérard* et *dort* sont des terminaux.

3.3 Arbre syntaxique d'une phrase

Comme on vient de le voir, pour décrire la structure syntaxique d'une phrase, il suffit de donner sa dérivation partant de S, c'est-à-dire l'ensemble de productions à travers lesquelles on obtient cette phrase.

Cette dérivation peut être admirablement bien illustrée par un *arbre syntaxique* : il s'agit d'un arbre (c'est-à-dire d'un graphe connexe sans cycle) orienté, dont la racine est l'axiome initial, les sommets intermédiaires sont les non-terminaux qui interviennent dans la production de la phrase, et les feuilles sont les mots de la phrase. Tout ensemble d'arêtes (orientées) sortant d'un sommet représente alors une production, et les enfants d'un sommet sont exactement les symboles de la partie droite de sa production.

L'arbre syntaxique de la fig. 1 représente la dérivation partant de S qui produit la phrase *Gérard aime Alice*.

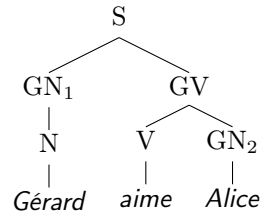

FIGURE 1 – Arbre syntaxique de la phrase *Gérard aime Alice*.

4 Le langage intermédiaire

La prochaine étape consiste à décrire LangLog, c'est-à-dire les formules du *langage intermédiaire*. Ce langage est basé sur deux outils mathématiques : la logique du 1er ordre et le λ-calcul typé.

4.1 Notions de logique du 1er ordre

Nous ne développerons ici de la logique du 1er ordre que ce qui est nécessaire pour notre exposé. Pour définir les *formules logiques* nous allons nous servir des symboles suivants :

1. symboles de constante $a, b, c, \ldots$;
2. symboles de variable $x, y, z, \ldots$;
3. symboles de prédicat $P, Q, R, \ldots$ qui prennent un certain nombre d'arguments (on appelle *arité* le nombre d'arguments d'un prédicat). Les arguments peuvent être des variables ou des constantes. On note, par exemple, $P(a, b, x)$ le prédicat ternaire dont les arguments sont a, b, x ;
4. symboles d'opérateur qui s'appliquent aux prédicats : l'opérateur unaire de négation $\neg$, et les opérateurs binaires de conjonction $\wedge$, de disjonction $\vee$, d'implication $\rightarrow$, de double implication $\leftrightarrow$;
5. symboles de quantificateur existentiel $\exists$ et universel $\forall$, qui s'appliquent aux variables ;
6. le symbole d'égalité $=$;
7. des parenthèses.

Définition 5. *On définit une* ***formule logique*** *de manière récursive :*

1. *Un prédicat d'arité n appliqué à n constantes et/ou variables est une formule.*
2. *Si a et b sont des constantes ou des variables, $a = b$ est une formule.*
3. *Si f est une formule, (f) et $\neg f$ sont des formules.*
4. *Si f est une formule et x une variable, $\exists x\, f$ et $\forall x\, f$ sont des formules.*
5. *Si f et g sont des formules, $f \wedge g$, $f \vee g$, $f \rightarrow g$ et $f \leftrightarrow g$ sont des formules.*

Le lecteur aura sans doute remarqué que nous venons d'énumérer une multitude de symboles sans donner leur signification.

Il y a une raison à cela : en fait, une formule logique n'est *a priori* rien d'autre qu'un assemblage de symboles abstraits (selon les règles syntaxiques données par la définition 5) et c'est grâce à ce degré élevé d'abstraction qu'elle peut être appliquée à une infinité de situations différentes.

Pour appliquer une formule à une situation et lui donner ainsi un «sens», on fait une *interprétation*, c'est-à-dire une correspondance entre $\mathfrak{LangLog}$ et $\mathfrak{Monde}$. Mais qu'est-ce qu'une interprétation au juste ?

Définition 6. *Soit Z_2 l'ensemble* $\{\mathbf{vrai}, \mathbf{faux}\}$. *Une* ***interprétation*** *d'une formule logique ϕ est une application* $\mathbf{Inter}\colon \mathfrak{LangLog} \to \mathfrak{Monde}$ *telle que pour chaque constante c de ϕ,* $\mathbf{Inter}(c) \in \mathfrak{Monde}$, *et pour chaque prédicat n-aire P de ϕ,* $\mathbf{Inter}(P)$ *est une fonction* $\mathfrak{Monde}^n \to Z_2$. *D'autre part, pour toute variable x de ϕ,* $\mathbf{Inter}(x)$ *devient une variable à valeurs dans* $\mathfrak{Monde}$.

L'interprétation de l'opérateur unaire $\neg$ est une application $Z_2 \to Z_2$ et celles des opérateurs binaires $\wedge$, $\vee$, $\to$ et $\leftrightarrow$ sont des applications $Z_2 \times Z_2 \to Z_2$, dont les valeurs sont données par la table suivante (poétiquement appelée *table de vérité*) :

ϕ	ψ	$\neg\phi$	$\phi \wedge \psi$	$\phi \vee \psi$	$\phi \to \psi$	$\phi \leftrightarrow \psi$
vrai	**vrai**	**faux**	**vrai**	**vrai**	**vrai**	**vrai**
vrai	**faux**	**faux**	**faux**	**vrai**	**faux**	**faux**
faux	**vrai**	**vrai**	**faux**	**vrai**	**vrai**	**faux**
faux	**faux**	**vrai**	**faux**	**faux**	**vrai**	**vrai**

Dans une interprétation donnée, une formule qui ne contient aucune variable est nécessairement vraie ou fausse. Il en est de même lorsqu'elle contient uniquement des variables quantifiées, c'est-à-dire telles que pour chacune il y ait un quantificateur qui lui soit appliqué. Une variable non quantifiée est appelée *libre*. L'interprétation d'une formule contenant n variables libres est une fonction $\mathfrak{Monde}^n \to Z_2$: sa valeur de vérité dépend des valeurs que prennent ses variables libres.

Après cette rapide introduction à (une partie de) la logique du 1er ordre, voyons comment traduire les phrases de $\mathfrak{LangNat}$ en des formules logiques.

5 Traduction $\mathfrak{LangNat} \to \mathfrak{LangLog}$

Nous allons noter **Trad** l'application qui traduit les phrases de $\mathfrak{LangNat}$ en des formules de $\mathfrak{LangLog}$.

Une phrase du type *Gérard dort* décrit une situation où il y a un agent identifié par le nom *Gérard* qui effectue l'action de dormir. Dans la formule logique il est naturel de prendre une constante g pour représenter *Gérard*.

À son tour, g doit être interprétée par l'entité du monde réel qui correspond à *Gérard*, notons cette entité **gérard**. On a donc $\mathbf{Trad}(\textit{Gérard}) = g$ et $\mathbf{Inter}(g) = \mathbf{gérard}$.

Le choix naturel pour traduire le verbe *dort* est un prédicat unaire dort. L'interprétation de dort va être une fonction $\mathfrak{Monde} \to Z_2$, et, en particulier, on aura $\mathbf{Inter}(\text{dort}(g)) = \mathbf{Inter}(\text{dort})(\mathbf{Inter}(g)) = \mathbf{Inter}(\text{dort})(\mathbf{gérard}) = \mathbf{vrai}$.

Mais comment déduire la formule dort(g) à partir de la grammaire formelle de la syntaxe de la phrase *Gérard dort* ? Cette grammaire nécessite les règles suivantes :

p_1 : S $\to$ GN GV
p_2 : GN $\to$ N
p_3 : GV $\to$ V
p_4 : N $\to$ *Gérard*
p_5 : V $\to$ *dort*

Tous les symboles de cette grammaire sont traduits dans $\mathfrak{LangLog}$: en effet, GV et V sont traduits par le prédicat dort, GN et N par la constante g et, par le principe de compositionnalité, S devient alors dort(g). On définit :

$\mathbf{Trad}(S) = \text{dort}(g)$
$\mathbf{Trad}(GN) = \mathbf{Trad}(N) = g$
$\mathbf{Trad}(GV) = \mathbf{Trad}(V) = \text{dort}.$

Notre traduction est complète puisque tous les sommets de l'arbre syntaxique de la phrase de $\mathfrak{LangNat}$ ont été traduits dans $\mathfrak{LangLog}$.

Prenons maintenant une phrase légèrement plus complexe : *Gérard aime Alice* (cf. fig. 1). On peut s'attendre à avoir $\mathbf{Trad}(S) = \text{aime}(g, a)$ où g et a sont des constantes logiques dont les interprétations sont le vilain **gérard** et la belle **alice**. De même, dans ce cas, $\mathbf{Trad}(\textit{aime}) = \text{aime}$.

Mais attention ! Avons-nous traduit tous les sommets de l'arbre de la fig. 1 en $\mathfrak{LangLog}$? Hélas, non ! Car, quelle est alors la traduction de GV ? On ne peut écrire aime$(_, a)$, cela n'est pas une formule logique valide...

Montague aurait pu s'arrêter là, en disant : «je sais traduire mes terminaux (*Gérard*, *aime*, *Alice*) et mon axiome de départ (S) en langage logique, peu me chaut le reste».

Que nenni ! Son génie a consisté — entre autres — à dire que si l'on veut être honnête avec soi-même, si l'on veut aller au fond des choses, alors *le principe de compositionnalité doit s'appliquer partout*, aussi bien dans $\mathfrak{LangNat}$, que dans $\mathfrak{LangLog}$, et la traduction d'une composition doit être la composition des traductions.

Autrement dit : si S produit GN et GV, alors la traduction de S doit s'obtenir à partir des traductions de GN et de GV. Mais avec les outils mathématiques décrits jusqu'à maintenant, cela n'est pas possible. Il faut donc se servir d'outils plus performants. Montague en a choisi deux : la théorie des types et le λ-calcul.

Voyons d'abord ce que sont les types et à quoi ils servent.

5.1 Théorie des types

Avant de traduire en $\mathfrak{LangLog}$ tous les sommets d'un arbre syntaxique, il faut déjà se demander de quelle manière ils se combinent entre eux.

Un exemple : dans la phrase *Gérard aime Alice*, l'interprétation du prédicat binaire *aime* peut être considérée comme une application $\mathfrak{Monde}^2 \to Z_2$ (elle envoie la paire d'entités (**gérard**, **alice**) vers la valeur **vrai** si **gérard** aime **alice** et vers **faux** sinon).

Autre exemple : dans la phrase *Gérard aime Alice et Paul déteste Virginie*, la particule de coordination *et* va combiner deux phrases pour en produire une nouvelle, son interprétation sera donc une application $Z_2 \times Z_2 \to Z_2$.

La situation se complique encore plus dans le cas des adverbes : dans *Gérard aime beaucoup Alice*, l'adverbe *beaucoup* agit sur le verbe, donc on peut considérer qu'il transforme une application $\mathfrak{Monde}^2 \to Z_2$ en une autre application $\mathfrak{Monde}^2 \to Z_2$.

Et que dire alors des modificateurs d'adverbe comme *vraiment beaucoup* : en effet, *vraiment* agit sur *beaucoup* et est donc un transformateur de transformateur d'application $\mathfrak{Monde}^2 \to Z_2$...

Comment gérer cette complexité qui semble croître inexorablement ?

Voici la modélisation mathématique qui nous délivre du cauchemar décrit ci-dessus : Montague considère qu'il n'existe que deux *types sémantiques primitifs* : celui de «formule» (dont l'interprétation dans $\mathfrak{Monde}$ sera **vrai** ou **faux**) et celui de «constante individuelle» (dont l'interprétation sera un élément de $\mathfrak{Monde}$). Il note t les formules et e les constantes. Ensuite il prend le monoïde libre $\{e, t\}^*$ dont il note la loi comme un produit scalaire $\langle , \rangle$. Attention : cette loi *n'est pas* associative, donc pas question de faire des «simplifications» : $\langle e, \langle e, t\rangle\rangle$ n'est pas la même chose que $\langle\langle e, e\rangle, t\rangle$!

Ensuite, il appelle les éléments de ce monoïde, des *types sémantiques complexes* et fait correspondre chaque sommet de l'arbre syntaxique à un type sémantique complexe, de la manière suivante : l'élément de gauche de $\langle , \rangle$ est la «donnée d'entrée» du type, son élément de droite est sa «donnée de sortie».

Exemple : un prédicat unaire, comme *dort*, s'applique à une constante, son interprétation fournit une valeur de Z_2. On dira donc qu'il est de type $\langle e, t\rangle$ (= «il prend un e et il nous rend un t»).

Mais attention : on ne prend qu'un seul élément d'entrée à la fois ! Un prédicat *binaire* ne sera donc pas de type $\langle\{e,e\},t\rangle$ (cette syntaxe n'est pas valide) mais sera décrit de *manière récursive* comme $\langle e,\langle e,t\rangle\rangle$.

Si on y réfléchit un peu, c'est parfaitement logique : un prédicat binaire auquel on fournit une valeur devient prédicat unaire, donc pour une entrée e, la sortie est $\langle e,t\rangle$. De même, le type d'un prédicat ternaire sera $\langle e,\langle e,\langle e,t\rangle\rangle\rangle$ et ainsi de suite...

De la même manière, la particule de conjonction ***et*** sera de type $\langle t,\langle t,t\rangle\rangle$, l'adverbe ***beaucoup*** de type $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$, et le modificateur d'adverbe ***vraiment***, de type $\langle\langle\langle e,t\rangle,\langle e,t\rangle\rangle,\langle\langle e,t\rangle,\langle e,t\rangle\rangle\rangle$. Arrivé à ce stade, le lecteur/la lectrice doit normalement se sentir ébloui(e) devant l'époustouflante beauté de ce modèle : en effet, *quelque soit* le type sémantique d'un mot, aussi complexe soit-il, on peut le décrire simplement par un élément du monoïde libre $\{e,t\}^*$... c'est simple et efficace.

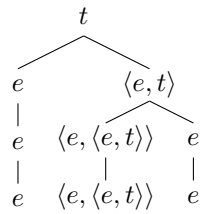

FIGURE 2 – Types sémantiques de l'arbre de la fig. 1.

Nous réécrivons sur la fig. 2 l'arbre syntaxique de la fig. 1 en remplaçant les symboles de la grammaire formelles par leurs types sémantiques. Le lecteur peut constater que (a) à chaque fois qu'un sommet n'a qu'un seul enfant, le type sémantique ne change pas, et (b) à chaque fois qu'un sommet a plusieurs enfants, leurs types sémantiques se *composent* : ainsi, $\langle e,t\rangle$ appliqué à e donne t, et $\langle e,\langle e,t\rangle\rangle$ appliqué à e donne $\langle e,t\rangle$. On note $\times$ cette composition : $\langle e,t\rangle \times e = t$. Pour qu'une composition $T_1 \times T_2$ puisse avoir lieu, il faut que T_1 soit un type complexe et que sa première composante soit égale à T_2.

La cohérence sémantique d'une phrase provient du fait que les types sémantiques des sommets de son arbre syntaxique se composent correctement, pour arriver au type de S qui est, invariablement, t (comme on peut le constater sur le graphe de la fig. 2).

Tout cela est bien pensé, mais on constate que l'on ne sait toujours pas comment écrire les formules logiques correspondant aux sommets intermédiaires de l'arbre syntaxique. C'est là que le λ-calcul vient à la rescousse.

5.2 Le λ-calcul

Sous ce nom exotique et mystérieux se cache tout simplement la notion de *fonction* : appliquer un «λ-opérateur» à une expression mathématique ou logique revient tout simplement à la transformer en fonction.

Ainsi, $\lambda x.f(x)$ est la même chose que $x \mapsto f(x)$, c'est-à-dire la fonction f. L'intérêt de la notation est qu'elle nous permet de définir toutes sortes de fonctions. Par exemple, $\lambda x.\lambda y.(x+y)$ est la fonction (de deux variables) qui à (x,y) associe $x+y$, alors que $\lambda x.(x+y)$ est la fonction (d'une variable) qui à x associe la somme $x+y$ (sans donner plus d'information sur y). De même, $\lambda P.P(x)$ est la fonction qui associe à un prédicat unaire sa valeur en x et $\lambda P.\lambda x.P(x)$ est la fonction qui à P et à x associe $P(x)$. Les amateurs de λ-calcul s'amusent même à noter $\lambda x.x$ la fonction identité et $\lambda x.c$ la fonction constante de valeur c.

Lorsqu'on a une fonction f, on peut l'appliquer à une valeur x, et on note le résultat $f(x)$. De même, on peut appliquer une λ-expression à une valeur. Ainsi $(\lambda x.\sin(x))(\frac{\pi}{2})$ est tout simplement $\sin(\frac{\pi}{2})$. On appelle cela, tout naturellement, une *application*.

L'utilisation que Montague fait de la notation λ est très judicieuse. Elle obéit au principe suivant :

Principe de «composition-application». *Toute* composition *de types sémantiques correspond à une* application *de λ-expressions.*

Ce principe va nous guider pour retrouver les λ-expressions correspondant aux sommets des arbres syntaxiques.

Prenons comme exemple la phrase *Gérard aime Alice* et les arbres des fig. 1 et 2. On sait déjà que $\mathbf{Trad}(\mathrm{GN}_1) = g$, $\mathbf{Trad}(\mathrm{GN}_2) = a$ et $\mathbf{Trad}(\mathrm{S}) = \mathrm{aime}(g, a)$. Appliquons le principe de composition-application pour trouver $\mathbf{Trad}(\mathrm{GV})$ et $\mathbf{Trad}(\mathrm{V})$.

Mais avant de le faire, un petit changement s'impose, afin de nous mettre en conformité avec la théorie des types : on n'écrira plus $\mathrm{aime}(g, a)$ pour *Gérard aime Alice*, comme on l'a fait jusqu'à maintenant, mais $\mathrm{aime}(a)(g)$. Cela ne change en rien l'amour indéfectible de Gérard pour Alice, c'est juste que maintenant «aime» n'est plus un «banal prédicat binaire», mais est fièrement devenu un type complexe $\langle e, \langle e, t\rangle\rangle$!

Raisonnons maintenant à reculons : on vient de décréter que $\mathbf{Trad}(\mathrm{S}) = \mathrm{aime}(a)(g)$; d'autre part, on sait que $\mathbf{Trad}(\mathrm{GN}_1) = g$; quel sera $\mathbf{Trad}(\mathrm{GV})$?

Laissons-nous guider par les types : dans la fig. 2, le type de GV est $\langle e, t\rangle$. On peut en conclure que $\mathbf{Trad}(\mathrm{GV})$ nécessite un λ-opérateur, qui doit capter le e pour en faire un t. Et c'est ce λ-opérateur qui va recevoir le g quand on va appliquer $\mathbf{Trad}(\mathrm{GV})$ à g. Écrivons donc $\mathbf{Trad}(\mathrm{GV}) = \lambda y.\mathrm{aime}(a)(y)$.

Vérifions : $\mathbf{Trad}(\mathrm{GV})(g) = (\lambda y.\mathrm{aime}(a)(y))(g) = \mathrm{aime}(a)(g) = \mathbf{Trad}(\mathrm{S})$, donc tout va bien.

De la même manière, on définit $\mathbf{Trad}(\mathrm{V}) = \lambda x.\lambda y.\mathrm{aime}(x)(y)$, et on vérifie : $\mathbf{Trad}(\mathrm{V})(a) = (\lambda x.\lambda y.\mathrm{aime}(x)(y))(a) = \lambda y.\mathrm{aime}(a)(y) = \mathbf{Trad}(\mathrm{GV})$. CQFD.

Le lecteur trouvera sur la fig. 3 la traduction dans **LangLog** de l'arbre syntaxique de la phrase *Gérard aime Alice* :

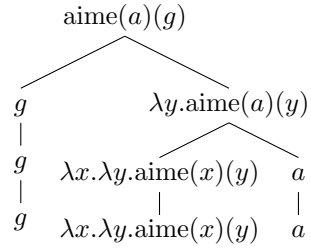

FIGURE 3 – Traduction en **LangLog** de l'arbre de la fig. 1.

On constate que la taille et la complexité des formules logiques sont assez variées : alors que $\mathbf{Trad}(\mathrm{GN}_1) = g$ et $\mathbf{Trad}(\mathrm{GN}_2) = a$ sont très simples, $\mathbf{Trad}(\mathrm{V})$ est bien plus complexe. On peut dire qu'intuitivement cela montre le potentiel d'action caché dans le verbe : lorsque on monte vers la racine, le verbe agit sur les autres constituants, jusqu'à fournir la phrase complète.

5.3 Un exemple : la coordination

Pour montrer la force de cette théorie et exercer un peu nos neurones, posons-nous un petit casse-tête : la coordination.

Le problème commence quand l'insatiable *Gérard* se met à préférer *Alexia* alors qu'il aime *Alice*. Prenons la phrase *Gérard aime Alice mais préfère Alexia*, et son arbre syntaxique (fig. 4). Les indices des GV et cGV servent uniquement à les distinguer dans la suite. On a noté cGV_3 la partie *mais préfère Alexia*, qui n'est pas un simple groupe verbal mais un «groupe verbal muni d'une conjonction» (d'où le «c» de cGV).

Que dire de cette phrase ? Le mot *mais* correspond logiquement à une conjonction et donc la traduction de S sera $\mathrm{aime}(a)(g) \wedge \mathrm{préfère}(a')(g)$ où $a = \mathbf{Trad}(\textit{Alice})$ et $a' = \mathbf{Trad}(\textit{Alexia})$. D'après la section précédente, $\mathbf{Trad}(\mathrm{GV}_2)$ et $\mathbf{Trad}(\mathrm{GV}_4)$ seront resp. $\lambda y.\mathrm{aime}(a)(y)$ et $\lambda y.\mathrm{préfère}(a')(y)$.

La grande question est : que seront $\mathbf{Trad}(\mathrm{GV}_1)$, $\mathbf{Trad}(\mathrm{cGV}_3)$, et surtout $\mathbf{Trad}(\mathrm{CONJ})$?

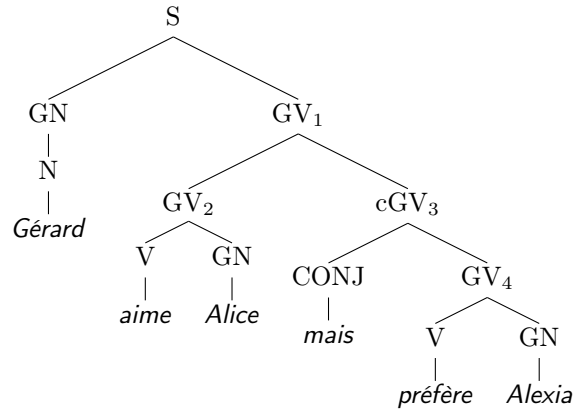

FIGURE 4 – Arbre syntaxique de la phrase *Gérard aime Alice mais préfère Alexia*.

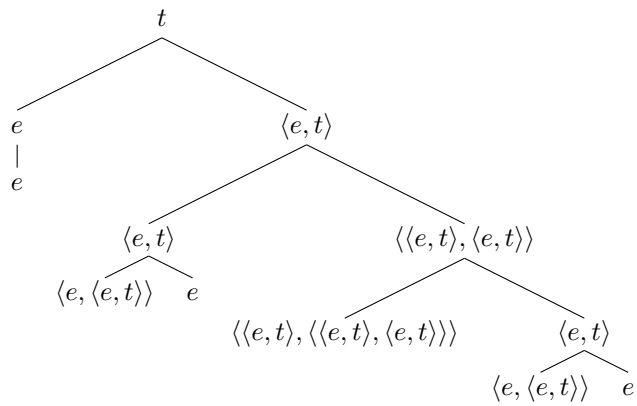

FIGURE 5 – Arbre des types de la phrase *Gérard aime Alice mais préfère Alexia*.

Traçons d'abord l'arbre des types (fig. 5). Pour GV_1, GV_2, GV_4, rien de nouveau, ce sont des $\langle e,t\rangle$, comme tout GV qui se respecte. Quid de cGV_3 ? Coincé entre GV_1 et GV_2, il ne peut être que $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$. Et donc, CONJ ne peut être que $\langle\langle e,t\rangle,\langle\langle e,t\rangle,\langle e,t\rangle\rangle\rangle$ (intuitivement : à partir d'un prédicat, et ensuite d'un deuxième prédicat, il fournit un nouveau prédicat).

Et maintenant, pour trouver $\mathbf{Trad}(\mathrm{GV}_1)$, $\mathbf{Trad}(\mathrm{cGV}_3)$ et $\mathbf{Trad}(\mathrm{CONJ})$, allons de nouveau à reculons :

1. Pour trouver $\mathbf{Trad}(\mathrm{GV}_1)$ il faut éliminer (le terme correct est «λ-*abstraire*») g de $\mathbf{Trad}(\mathrm{S}) = \mathrm{aime}(a)(g) \wedge \mathrm{préfère}(a')(g)$. On le fait en écrivant $\mathbf{Trad}(\mathrm{S}) = \lambda x.(\mathrm{aime}(a)(x) \wedge \mathrm{préfère}(a')(x))(g)$ et donc $\mathbf{Trad}(\mathrm{GV}_1) = \lambda x.(\mathrm{aime}(a)(x) \wedge \mathrm{préfère}(a')(x))$.

2. Essayons de λ-abstraire $\mathbf{Trad}(\mathrm{GV}_2)$ de $\mathbf{Trad}(\mathrm{GV}_1)$ pour obtenir $\mathbf{Trad}(\mathrm{cGV}_3)$: on a $\mathbf{Trad}(\mathrm{GV}_1) = \lambda x.(\mathrm{aime}(a)(x) \wedge \mathrm{préfère}(a')(x)) = \lambda x.\mathrm{aime}(a)(x) \wedge \lambda x.\mathrm{préfère}(a')(x)$. Le terme $\lambda x.\mathrm{aime}(a)(x)$ est un prédicat.

On peut le λ-abstraire en l'écrivant sous la forme $\lambda P.(P)(\lambda x.\mathrm{aime}(a)(x))$, ce qui donne $\mathbf{Trad}(\mathrm{GV}_1) = \lambda P.(P \wedge \lambda x.\mathrm{préfère}(a')(x))(\lambda x.\mathrm{aime}(a)(x))$ et donc, par le principe de composition-application, $\mathbf{Trad}(\mathrm{cGV}_3) = \lambda P.(P \wedge \lambda x.\mathrm{préfère}(a')(x))$.

3. De la même manière, nous λ-abstrayons $\mathbf{Trad}(\mathrm{GV}_4)$ de $\mathbf{Trad}(\mathrm{cGV}_3)$ et ce qui reste sera $\mathbf{Trad}(\mathrm{CONJ})$. On trouve $\mathbf{Trad}(\mathrm{cGV}_3) = \lambda Q.\lambda P.(P \wedge Q)(\lambda x.\mathrm{préfère}(a')(x))$ et donc, enfin, $\mathbf{Trad}(\mathrm{CONJ}) = \lambda Q.\lambda P.(P \wedge Q)$.

À y réfléchir, ce résultat n'a rien d'étonnant : après tout, un opérateur binaire comme $\wedge$ n'associe-t-il pas deux prédicats P et Q au prédicat $P \wedge Q$?

5.4 La quantification

Prenons maintenant la phrase *tout le monde aime Alice*. La différence avec *Gérard aime Alice* est que si *Gérard* (grammaticalement, un nom) peut être traduit par une constante g et puis interprété par une entité **gérard**, on ne peut faire de même pour *tout le monde* (un pronom indéfini), que l'on sera logiquement obligé d'interpréter par la totalité des objets de $\mathfrak{Monde}$ (puisque c'est «tout le monde» qui aime *Alice*).

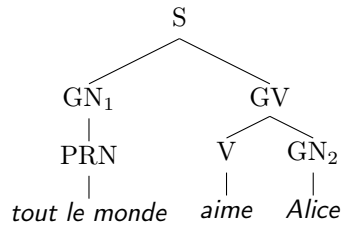

FIGURE 6 – Arbre syntaxique de la phrase *tout le monde aime Alice*.

Cela ne va pas changer outre mesure l'arbre syntaxique de la phrase (fig. 6), mais on voit la différence au niveau de l'arbre des types (fig. 7b). Là où *Gérard* était traduit par un type e, *tout le monde* est traduit par un $\langle\langle e,t\rangle,t\rangle$, c'est-à-dire qu'il prend le groupe verbal $\lambda x.\text{aime}(a)(x)$ en entrée et retourne une valeur de vérité (la réponse à la question : «tout le monde aime-t-il *Alice* ?»).

FIGURE 7 – Arbres des types des phrases (a) *Gérard aime Alice* et (b) *tout le monde aime Alice*.

Quelle va être la traduction de S ? Il est naturel de se servir du quantificateur universel pour écrire $\mathbf{Trad}(S) = \forall x\,(\text{aime}(a)(x))$.

Sachant que GV se traduit par $\lambda x.\text{aime}(a)(x)$, quelle va être la traduction de PRN ? Voici comment s'y prendre : posons $P = \lambda x.\text{aime}(a)(x)$, P est donc un prédicat unaire. S devient alors $\forall x\,(P(x))$, que l'on peut λ-abstraire en $(\lambda Q.(\forall x\,(Q(x)))(P)$, et on reconnaît ici une fonction de prédicat appliquée au prédicat P. Mais ce prédicat n'est autre que la traduction de GV, et on a donc trouvé une fonction qui, appliquée à GV, nous donne S : d'après le principe de compositionnalité, cela n'est rien d'autre que la traduction de PRN.

On remplace donc P par sa valeur et on a $\mathbf{Trad}(\textit{tout le monde}) = \lambda P.(\forall x\,(P(x)))$, qui est bien de type $\langle\langle e,t\rangle,t\rangle$.

5.5 L'article défini

Que de plus simple dans la langue française que l'article défini «le, la, les» ? Et pourtant, sa traduction en $\mathfrak{LangLog}$ sera pour nous un petit challenge !

Prenons la phrase *le philosophe aime Alice* (fig. 8). Notons tout de suite que *philosophe* ne peut être traduit par une constante, comme, par exemple, *Gérard*, puisque «être philosophe» est une propriété, et les propriétés sont traduites par des prédicats unaires. Ainsi, on écrira philosophe(g) pour dire que g est philosophe. De même, $\exists x\,\text{philosophe}(x)$ signifie qu'il existe un philosophe. Et donc, si la phrase de départ était ***un*** *philosophe aime Alice* (avec un article indéfini), sa traduction en $\mathfrak{LangLog}$ serait $\exists x\,(\text{philosophe}(x) \wedge \text{aime}(a)(x))$.

D'où l'arbre de types de la phrase *le philosophe aime Alice* (fig. 9), où l'on affecte à *philosophe* le type $\langle e,t\rangle$ (le même que celui du verbe *aime*) et donc GN_1 devient $\langle\langle e,t\rangle,t\rangle$. Il ne nous reste

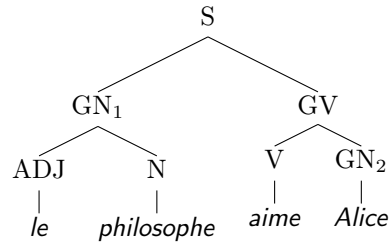

FIGURE 8 – Arbre syntaxique de la phrase *le philosophe aime Alice*.

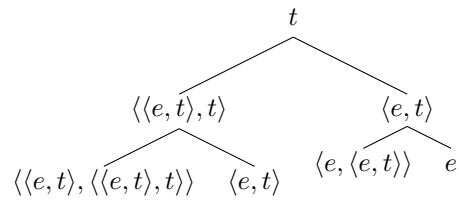

FIGURE 9 – Arbre des types de la phrase *le philosophe aime Alice*.

d'autre choix pour *le* que de lui affecter le type $\langle\langle e,t\rangle, \langle\langle e,t\rangle, t\rangle\rangle$, autrement dit : il prend un $\langle e,t\rangle$ (*philosophe*) et ensuite un autre $\langle e,t\rangle$ (*aime*) et retourne une valeur de vérité.

Mais comment traduire alors l'article défini *le* ?

Formulons la question autrement : comment indiquer qu'il n'y a qu'*un seul* philosophe, et que quand on dit *le philosophe* on parle justement de lui ?

Pour répondre à cette question, rappelons-nous que dans la définition de la formule logique de 1er ordre (déf. 5, p. 4) nous avons mentionné la relation binaire $=$ («égalité»). Mais qu'est-ce donc l'«égalité» ?

C'est une vaste question philosophique dont on ne parle guère en cours de mathématiques... et pourtant, en logique du 1er ordre, l'«égalité» a un sens bien particulier, que voici : *écrire $a = b$ signifie que l'on demande que dans toute interprétation, les constantes (ou variables) a et b soient interprétées par le même élément de* $\mathfrak{Monde}$.

Et c'est ainsi qu'on traduit l'*unicité* : il n'y a qu'un seul philosophe x si et seulement si pour tout individu y tel que y soit philosophe, on ait $x = y$.

La phrase *le philosophe aime Alice* se traduira donc par $\exists x\,(\forall y\,(\text{philosophe}(y) \leftrightarrow x = y) \wedge \text{aime}(a)(x))$.

En appliquant les mêmes méthodes de λ-abstraction que dans la section précédente, on trouve que la traduction de l'article défini *le* ne peut être que

$$\mathbf{Trad}(\textit{le}) = \lambda Q.(\lambda P.(\exists x\,(\forall y\,(Q(y) \leftrightarrow (x = y)) \wedge P(x)))).$$

Le lecteur peut imaginer le désarroi de l'étudiant en linguistique qui, ayant raté le premier cours, se retrouve devant une formalisation de la langue naturelle qui traduit un des mots les plus simples de sa langue par cette horrible formule... Est-ce bien raisonnable ? En fait, nous n'avons fait que rendre visible l'important potentiel sémantique de ce petit mot grammatical — à première vue, insignifiant — qu'est l'article défini.

6 L'ensemble $\mathfrak{Monde}$ et la fonction d'interprétation Inter

Jusqu'ici nous avons surtout parlé de $\mathbf{Trad} : \mathfrak{LangNat} \to \mathfrak{LangLog}$. Il ne reste plus qu'à décrire plus précisément $\mathfrak{Monde}$, ainsi que la fonction $\mathbf{Inter} : \mathfrak{LangLog} \to \mathfrak{Monde}$. Pour cela, nous allons nous servir d'un autre outil mathématique, bien plus connu cette fois-ci : la *théorie des ensembles*.

Ainsi, si $a = \mathbf{Trad}(\textit{Alice})$ et $g = \mathbf{Trad}(\textit{Gérard})$ deviennent dans $\mathfrak{Monde}$ la belle **alice** et le vilain **gérard**, que dire alors des autres sommets des arbres syntaxiques que nous avons étudiés ?

Là aussi Montague a eu les bonnes idées !

D'après la définition de l'interprétation d'une formule (déf. 6), un prédicat unaire est interprété par une fonction $\mathfrak{Monde} \to Z_2$, où Z_2 est l'ensemble $\{\mathbf{vrai}, \mathbf{faux}\}$. Notons par $Z_2^{\mathfrak{Monde}}$ les fonctions de $\mathfrak{Monde}$ dans Z_2. Alors $\mathbf{Inter}(\lambda x.P(x)) \in Z_2^{\mathfrak{Monde}}$, c'est-à-dire que l'interprétation d'un prédicat unaire de $\mathfrak{LangLog}$ est un élément de $Z_2^{\mathfrak{Monde}}$.

Cette propriété se généralise aux prédicats n-aires quelconques. En effet, il suffit de constater qu'un prédicat binaire devient une fonction qui à chaque élément de $\mathfrak{Monde}$ associe un prédicat unaire. Donc $\mathbf{Inter}(\lambda x.\lambda y.P(x)(y)) \in Z_2^{\mathfrak{Monde}^{\mathfrak{Monde}}}$, et ainsi de suite...

Cela semble abstrait, mais en réalité on ne fait que manipuler de simples relations binaires. Prenons un exemple : supposons que $\mathfrak{Monde} = \{\mathbf{alice}, \mathbf{gérard}, \mathbf{billy}\}$ (notés dans $\mathfrak{LangLog}$ par a, g, b) et que l'on ait $\mathbf{Inter}(\text{aime}(a)(g)) = \mathbf{vrai}$ et $\mathbf{Inter}(\text{aime}(a)(b)) = \mathbf{vrai}$. Que va être $\mathbf{Inter}(\lambda y.\text{aime}(a)(y))$? Ce sera un élément de $Z_2^{\mathfrak{Monde}}$: l'ensemble de relations binaires $\phi_a = \{(\mathbf{alice}, \mathbf{faux}), (\mathbf{gérard}, \mathbf{vrai}), (\mathbf{billy}, \mathbf{vrai})\}$ (puisque **gérard** et **billy** aiment **alice**, mais **alice** ne s'aime pas elle-même). Nous avons appelé ϕ_a cet ensemble de relations binaires puisqu'il s'agit de savoir qui aime **alice**. De la même manière, il existe ϕ_g et ϕ_b qui concernent **gérard** et **billy**.

Que sera alors $\mathbf{Inter}(\lambda x.\lambda y.\text{aime}(x)(y))$? Ce sera un élément de $Z_2^{\mathfrak{Monde}^{\mathfrak{Monde}}}$: un ensemble de relations binaires entre des éléments de $\mathfrak{Monde}$ et des fonctions $\mathfrak{Monde} \to Z_2$. Dans notre cas, ce sera tout simplement $\Phi = \{(\mathbf{alice}, \phi_a), (\mathbf{gérard}, \phi_g), (\mathbf{billy}, \phi_b)\}$.

Supposons que nous souhaitions savoir si $\mathbf{Inter}(\text{aime}(g)(a))$ est **vrai**. Écrivons

$$\begin{aligned}
&\mathbf{Inter}(\text{aime}(g)(a)) \\
&\quad = \mathbf{Inter}((\lambda x.\lambda y.\text{aime}(x)(y))(g)(a)) \\
&\quad = \mathbf{Inter}(\lambda x.\lambda y.\text{aime}(x)(y))(\mathbf{Inter}(g))(\mathbf{Inter}(a)) \\
&\quad = \Phi(\mathbf{Inter}(g))(\mathbf{Inter}(a)) \\
&\quad = \phi_g(\mathbf{Inter}(a)) = \mathbf{faux},
\end{aligned}$$

donc, hélas, **alice** n'aime (toujours) pas **gérard**.

On voit donc de quelle manière il est possible d'interpréter n'importe quelle formule de $\mathfrak{LangLog}$.

7 Et le reste...

Cet article est déjà assez long. Jusqu'ici, nous avons parcouru et partiellement illustré une partie de la théorie de Montague. Le restant de sa théorie est tout aussi utile et intéressant, mais n'implique pas de nouvel outil mathématique ; nous allons donc nous contenter d'en décrire rapidement les grandes lignes.

7.1 Inférence

Ce qui fait la force de la logique mathématique est le fait qu'à partir d'un ensemble de formules (que l'on considère vraies pour une interprétation donnée), on a des mécanismes (appelés *règles d'inférence*) pour obtenir de nouvelles formules (également vraies dans la même interprétation). On admet donc certaines formules en tant qu'*axiomes* et on en déduit d'autres, appelées *théorèmes*. C'est ainsi que fonctionnent les mathématiques : que ce soit en géométrie, en algèbre ou en analyse, on admet certains axiomes et on construit des *théories* en démontrant des *théorèmes*.

Un exemple de règle d'inférence très utile est le *modus ponens* : si on a $\phi \to \psi$ et ϕ alors on peut en déduire ψ (exemple : si on admet que *tous les hommes sont mortels* et que *Socrate est un homme*, alors on peut en déduire que *Socrate est mortel*).

Montague introduit le mécanisme des règles d'inférence dans $\mathfrak{LangLog}$.

7.2 La temporalité

Pour analyser des phrases comme *Gérard n'aime plus Alice*, Montague introduit la temporalité dans $\mathfrak{LangLog}$. Le temps y est représenté de deux manières : par des *instants* et par des *intervalles*

temporels. Il propose des opérateurs entre les intervalles : deux intervalles $[t_1, t_2]$ et $[t_3, t_4]$ peuvent se chevaucher (quand, par exemple, $t_1 < t_3 < t_2 < t_4$), être totalement disjoints ($t_2 < t_3$) ou s'imbriquer l'un dans l'autre ($t_1 < t_2 < t_3 < t_4$).

Chaque formule de $\mathfrak{LangLog}$ est indexée temporellement : sa valeur de vérité dépend (outre les variables libres qu'elle contient) de ses propriétés temporelles.

7.3 Les mondes possibles/accessibles

Pour analyser des phrases comme *Gérard aime peut-être Alice, mais certainement pas Alexia*, Montague introduit les notions de *modalité* et de *mondes accessibles*. Il propose deux opérateurs modaux : $\Box\phi$ qui signifie «ϕ est nécessairement vraie», et $\diamond\phi$ qui signifie «ϕ est peut-être vraie».

Pour formaliser ces notions, il parle de «mondes accessibles» : en effet, on a toujours dit que $\mathfrak{Monde}$ pouvait être un monde hypothétique quelconque, alors pourquoi ne pas en imaginer plusieurs, voire *tous* les mondes possibles et imaginables ? Mais comme cela est un peu éloigné de la réalité des problèmes que l'on peut se poser, il est plus raisonnable de parler de mondes «accessibles» : ce sont ceux qui constituent des alternatives plausibles à un monde donné. Si $\mathfrak{Monde}$ et $\mathfrak{Monde}'$ sont des mondes, on peut imaginer une relation binaire Accessible($\mathfrak{Monde}, \mathfrak{Monde}'$) qui signifie que $\mathfrak{Monde}'$ est accessible à partir de $\mathfrak{Monde}$.

Alors la formule ϕ est «nécessairement vraie» ($\Box\phi$) si elle est vraie dans *tous* les mondes accessibles à partir du monde courant, et elle est «peut-être» vraie ($\diamond\phi$), si elle est vraie dans *certains* de ces mondes.

7.4 Intensionnalité

Définir un ensemble *extensionnellement* consiste à énumérer ses éléments, le définir *intensionnellement* consiste à en donner les propriétés. Ainsi, $\{p \mid p \leq 10, p \text{ premier}\}$ est une définition intensionnelle et $\{2, 3, 5, 7\}$ la définition sensationnelle du même ensemble.

Montague se sert de l'intensionnalité pour englober en un seul objet mathématique les valeurs d'une expression dans tous les mondes accessibles. Ainsi, si aime$(a)(g)$ est vrai dans les mondes $\mathfrak{Monde}_1$, $\mathfrak{Monde}_2$ et $\mathfrak{Monde}_4$, et aime$(a')(g)$ est vrai dans les mondes $\mathfrak{Monde}_1$, $\mathfrak{Monde}_2$ et $\mathfrak{Monde}_3$, alors ces deux formules ont la même valeur de vérité dans *certains* mondes mais pas dans *tous*. On dira que leurs *intensions* sont différentes.

L'*intension d'une formule* est donc une fonction qui envoie différents mondes vers les valeurs de vérité correspondantes des formules. En comparant les intensions de deux formules, on compare leur «comportement» dans tous les mondes accessibles, c'est bien plus puissant que de les comparer dans un seul monde. Cette notion est tellement importante pour Montague que sa théorie est souvent appelée *sémantique intensionnelle*.

8 Conclusion

Nous avons exploré les outils mathématiques qui ont servi à Montague et à ceux qui l'ont suivi pour modéliser la langue naturelle en tant que langage formel. Ce parcours nous a permis d'évoquer et de décrire brièvement les langages et grammaires formels, une version légèrement restreinte de la logique de 1$^{\text{er}}$ ordre, la théorie de types, le λ-calcul. Dans chaque cas nous avons tenté de justifier l'utilité de l'outil pour l'analyse de la langue naturelle, et de l'illustrer par des exemples.

Notre but a été de faire découvrir au lecteur comment les mathématiques permettent d'étudier les liens entre la langue naturelle et le monde — liens qui nous affectent tous profondément puisque, comme disait Wittgestein :

> «*die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt*»
> (= les limites de ma langue sont les limites de mon propre monde) [14, § 5.6]

Conseils de lecture

Les langages et grammaires formels sont décrits avec beaucoup de rigueur dans [6]. La logique du 1er ordre est admirablement bien présentée par J.-P. Delahaye dans [7].

En ce qui concerne la sémantique de Montague, une présentation très accessible est donnée dans [9]. L'ouvrage [5] est un peu plus technique mais tout aussi abordable. Enfin, [10], tiré du Séminaire de philosophie et mathématiques de l'ENS (dirigé, excusez du peu, par J. Dieudonné et R. Thom), en donne une synthèse très efficace. Dans son exposé filmé «La modélisation mathématique des langues naturelles» de l'Université de tous les savoirs, S. Kahane parle de la sémantique de Montague [12, 30′24″–34′45″] avant d'enchaîner sur une de ses propres contributions : les grammaires à bulles [11].

Enfin, en ce qui concerne la vie (et la mort) de Richard M. Montague, voir l'essai *That's just Semantics !* [1], ainsi que les romans [2], [3] et [8].

Références

[1] Sacha Arnold. That's just Semantics ! en ligne : `http://shar.es/StiAm`

[2] David Berlinski. *Less than meets the eye*. St. Martin's Press, 1994.

[3] Aifric Campbell. *The Semantics of Murder*. Serpent's Tail, 2009.

[4] Ronnie Cann. *Formal Semantics, an Introduction*. Cambridge Univ. Press, 1993.

[5] Michel Chambreuil and Jean-Claude Pariente. *Langue naturelle et logique*. Peter Lang, 1990.

[6] Patrick Dehornoy. *Mathématiques de l'informatique*. Dunod, 2000.

[7] Jean-Paul Delahaye. *Outils logiques pour l'intelligence artificielle*. Eyrolles, 1988.

[8] Samuel Delany. *The mad man*. Masquerade Books, 1996.

[9] Michel Galmiche. *Sémantique linguistique et logique*. puf, 1991.

[10] Paul Gochet. La sémantique récursive de Davidson et de Montague. In *Penser les mathématiques (Séminaire de philosophie et mathématiques de l'École normale supérieure)*, pages 73–87. Éditions du Seuil, 1982.

[11] Sylvain Kahane. Extractions dans une grammaire de dépendance à bulles. *TAL*, 41 :187–216, 2000.

[12] Sylvain Kahane. La modélisation mathématique des langues naturelles (vidéo). `http://www.canal-u.tv/video/universite_de_tous_les_savoirs/la_modelisation_mathematique_des_langues_naturelles.1315` 2002.

[13] Richard M. Montague. English as a formal language. In B. Visentini et al., editors, *Linguaggi nella Società et nella Tecnica*, pages 188–211. Edizioni di Comunità, 1970.

[14] Ludwig Wittgenstein. *Tractatus logico-philosophicus*. Tel. Gallimard, 2001.