CHAPITRE 1

Langages logiques

La logique fournit un langage permettant d'écrire des assertions qui peuvent être « vraies » ou « fausses ». Les « phrases » de ce langage sont les *formules* logiques. Elles expriment des propriétés sur des objets d'un certain « univers du discours ». Par exemple, les trois énoncés :

« *Socrate suit en cours de logique* » (1.1)

« *Socrate lit ses messages sur son smartphone* » (1.2)

« *Socrate comprend la logique* » (1.3)

portent sur l'objet *Socrate* (les individus sont ici des objets) et spécifient trois propriétés qui peuvent être satisfaites, ou non, par *Socrate*. Ces propriétés sont *a priori* indépendantes : par exemple *Socrate* peut très bien suivre un cours de logique sans lire ses messages sur son smartphone, tout comme il peut comprendre la logique en lisant ses messages sur son smartphone. Toutes les situations sont ici envisageables. Dans la mesure où la *valeur de vérité* de chacun de ces trois énoncés est indépendante des lois de la logique, on pose qu'ils constituent des *formules atomiques* du langage.

Pour éliminer certaines situations, et donc pour introduire une certaine dépendance entre les valeurs de vérité de formules logiques, on les combine avec des *connecteurs* logiques. Les combinaisons logiques sont basées sur les conjonctions de coordination « et », « ou », de subordination « si alors » ou simplement la négation. Par exemple, la phrase :

« Si *Socrate* ne *lit* pas *ses messages sur son smartphone*
et *qu'il suit un cours de logique,* (1.4)
alors *il comprend la logique.* »

est une combinaison des trois formules atomiques (1.1), (1.2) et (1.3). C'est à son tour une formule et elle n'est pas atomique : sa valeur de vérité dépend non seulement de celle des formules atomiques qui la composent mais aussi du sens que l'on donne aux connecteurs qui relient ces formules. Par exemple, s'il s'avère que *Socrate* ne lit pas ses messages sur son smartphone et qu'il suit un de cours de logique mais qu'il ne comprend pas la logique, la formule (1.4) a la valeur de vérité « faux ». Toutefois si *Socrate* ne lit pas ses messages sur son smartphone, ne suit pas de cours de logique et comprend tout de même la logique, la formule (1.4) a la valeur de vérité « vrai ».

Certains énoncés ne portent pas sur un individu particulier (que l'on a désigné ici par son nom : *Socrate*), mais sur un ou des individu(s) indéterminé(s).

Par exemple, en français, on dirait :

« Si *on* ne *lit* pas *ses messages sur son smartphone*
et *que l'on suit un cours de logique,*
alors *on comprend la logique.* » (1.5)

L'indétermination est marquée ici par l'emploi du pronom indéfini « on ». En logique, comme en algèbre, on a recours à l'emploi d'un ensemble de symboles particuliers pour marquer l'indétermination : les *symboles de variable*. Par exemple, le symbole x :

« Si x ne *lit* pas *ses messages sur son smartphone*
et x *suit un cours de logique,*
alors x *comprend la logique.* » (1.6)

Enfin, on peut quantifier cette indétermination en construisant des énoncés qui portent sur tout ou partie (c-à-d au moins un) des objets de l'univers du discours :

« Pour tout x,
si x ne *lit* pas *ses messages sur son smartphone*
et x *suit un cours de logique,*
alors x *comprend la logique.* » (1.7)

« Il existe x tel que
si x ne *lit* pas *ses messages sur son smartphone*
et x *suit un cours de logique,*
alors x *comprend la logique.* » (1.8)

Les locutions « pour tout » et « il existe » correspondent aux constructions logiques appelées *quantificateurs*.

Il existe plusieurs langages logiques offrant un pouvoir d'expression plus ou moins grand selon les constructions qu'ils permettent. Dans cet ouvrage, nous étudions le langage des *formules logiques* obtenues à partir de *formules atomiques* que l'on peut combiner à l'aide de *connecteurs propositionnels* et moduler par les *quantificateurs* « pour tout » et « il existe » portant sur des variables désignant des objets de l'univers du discours. Ce langage est celui de *logique des prédicats du premier ordre*. Dans notre présentation, nous distinguons le fragment de langage sans variable, qui constitue le *fragment propositionnel* du langage et le langage complet avec variables et quantificateurs.

Ce chapitre présente formellement les aspects syntaxiques de ces deux parties du langage, successivement dans les sections 1.1 et 1.2, avant de décrire, dans la section 1.3, des outils de manipulation des variables utiles pour la suite. Dans les chapitres suivants les deux principales approches permettant d'établir qu'une formule est « vraie » ou « fausse » sont introduites.

1.1 Constantes, fonctions, prédicats et connecteurs

Cette section présente le langage logique des prédicats du premier ordre sans variable : après une illustration des principes dans la section 1.1.1, il décrit les éléments du langage, appelés *termes*, dans la section 1.1.2, puis les formules logiques qu'ils permettent de construire dans la section 1.1.3.

1.1.1 Principes

On distingue traditionnellement deux niveaux du langage logique sans variable : la logique des propositions et la logique des prédicats.

Logique des propositions La logique des propositions fournit un langage très rudimentaire dans lequel chaque énoncé atomique est représenté par un symbole appartenant à un ensemble noté $\mathcal{P}_0$.

Par exemple, en choisissant les symboles p_1, p_2 et p_3 pour représenter les énoncés (1.1), (1.2) et (1.3), l'énoncé négatif *« Socrate* ne *lit* pas *ses messages sur son smartphone »* s'écrit comme la négation de p_2, c-à-d *«* non p_2 *»*. On transcrit alors la phrase (1.4) par la formule :

« si p_1 et non p_2 alors p_3 *»*

Quoique que d'aspect assez pauvre, ce langage n'est pas sans contenir quelques propriétés logiques fondamentales. Par exemple, on sait y déterminer que la formule *«* p et non p *»* est « fausse » quelle que soit la valeur de vérité de la formule atomique p. Ce langage sera étudié plus en détail dans le chapitre 7. En revanche, le langage de la logique des propositions interdit toute finesse pour les formules atomiques qui y sont réduites à un seul symbole. Ainsi, la transcription de la phrase (1.4) par la formule *«* si p_1 et non p_2 alors p_3 *»* masque que c'est le même *Socrate* qui suit et comprend le cours de logique. Pour dégager cette dépendance, il faut décomposer l'énoncé « *Socrate suit un cours de logique* » en un sujet (*Socrate*) et un prédicat (*suit un cours de logique*).

Le langage de la logique des prédicats utilisé dans toute la suite ajoute cette possibilité de structuration plus riche des énoncés atomiques.

Logique des prédicats Le langage de la logique des prédicats ajoute aux symboles utilisés pour désigner les objets du discours (comme *Socrate*) des symboles utilisés pour désigner ce que l'on en dit, les propriétés qu'on leur attribue. Ce que l'on désigne sous le terme générique de *prédicat*. Par exemple on dit de *Socrate* qu'il *« suit un cours de logique »*. Pour composer objets et prédicats, on « applique » un *symbole de prédicat* au symbole de l'objet (comme on « applique » une fonction à son argument) :

$$suit\_un\_cours\_de\_logique(Socrate)$$

Dans le langage de la logique des prédicats, le terme de « prédicat » s'étend aux « relations » qui relient entre eux des objets. Par exemple, si l'univers du discours est celui des nombres, on pourra énoncer qu'un nombre est plus petit qu'un autre. On choisit un symbole, disons *le* (pour *less or equal*), pour désigner la relation d'ordre sur les nombres, et on écrit que *«* 6 *est inférieur à* 3 *»* (indépendamment du souci de savoir si c'est « vrai » ou « faux ») comme l'application du symbole de prédicat aux désignations symboliques des objets : $le(6, 3)$.

Ainsi, certains symboles de prédicat s'appliquent à un objet, d'autres à deux objets, d'autres à trois, etc. Le nombre d'objets auxquels s'applique un symbole est appelé son *arité*. On parle de prédicat *unaire* (arité 1), *binaire* (arité 2), etc.

Plus encore, le langage de la logique des prédicats permet de décomposer les objets eux-mêmes. En effet, pour rester dans le domaine numérique, un nombre peut être le résultat d'une *fonction* appliquée à un, deux, etc. autres nombres. Par exemple, en

choisissant le symbole *add* pour désigner l'addition, l'application $add(4, 2)$ désigne le nombre obtenu par somme de 4 et 2. On exprime la relation entre ce nombre et la constante 3 avec la formule atomique $le(add(4, 2), 3)$. Comme les symboles de prédicat, les symboles de fonction ont une arité.

Les formules atomiques ainsi obtenues sont composables à l'aide des connecteurs propositionnels. Pour prendre un exemple issu de l'informatique, on considère le tableau suivant :

2	5	6	9

On énonce que les éléments de ce tableau sont rangés par ordre croissant par la formule :

$$le(2, 5) \text{ et } le(5, 6) \text{ et } le(6, 9)$$

Mais ici manque encore l'information que 2, 5, 6 et 9 sont les valeurs d'un certain tableau. Pour préciser cela on choisit un symbole de fonction binaire *pos* et l'on désigne le i-ème élément d'un tableau t par $pos(t, i)$ [1]. Si l'on appelle *tab* le tableau qui contient la suite $2, 5, 6, 9$, on énonce que ses éléments y sont rangés par ordre croissant avec la formule :

$$le(pos(tab, 0), pos(tab, 1)) \text{ et } le(pos(tab, 1), pos(tab, 2)) \text{ et } le(pos(tab, 2), pos(tab, 3))$$

Dans le paragraphe suivant nous présentons un schéma général de définition du fragment du langage de la logique des prédicats destiné à désigner les objets de l'univers du discours appelé *langage des termes*. Nous présentons ensuite un schéma général de définition des formules atomiques et de leur combinaison à l'aide des connecteurs propositionnels. Le langage des formules défini dans ces deux paragraphes est un langage sans symbole de variable. L'usage des variables est l'objet du paragraphe 1.2.

1.1.2 Langage de termes

Signature Les *termes* sont les expressions syntaxiques qui désignent les objets du discours. Pour définir l'ensemble des termes on se donne un ensemble $\mathcal{F}$ de symboles de constante et de fonction, appelé une *signature*. Ainsi, dans les exemples précédents, on a utilisé les constantes *Socrate*, *tab*, 2, 3 et les fonctions *add* et *pos*.

L'ensemble $\mathcal{F}$ est construit comme l'union des symboles de constante, des symboles de fonction d'arité 1, de fonction d'arité 2, etc. que l'on veut utiliser dans les formules. On note $\mathcal{F}_0$ l'ensemble des symboles de constante (on considère les constantes comme des fonctions d'arité 0) et, plus généralement, $\mathcal{F}_n$ l'ensemble des symboles de fonction d'arité n. On pose :

$$\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{F}_2 \cup \cdots = \bigcup_{n \geq 0} \mathcal{F}_n$$

Définition 1.1 (Ensemble $\mathcal{T}_0(\mathcal{F})$ de termes sans variable)
Étant donnée une signature $\mathcal{F}$, l'ensemble $\mathcal{T}_0(\mathcal{F})$ des termes est défini inductivement comme suit.

1. *Si $k \in \mathcal{F}_0$ est un symbole de constante, alors k est un terme.*

1. Nous adoptons la convention usuelle en informatique selon laquelle la première position dans un tableau est désignée par 0.

2. *Si $f \in \mathcal{F}_n$ est un symbole de fonction d'arité n, et si $t_1, \cdots, t_n$ sont des termes, alors $f(t_1, \cdots, t_n)$ est un terme.*

Notez que pour former des termes, outre les symboles d'une signature, on utilise les deux parenthèses, ouvrante et fermante, ainsi que la virgule.

Exemple 1.1 Pour prendre un exemple simple de langage de termes, considérons les ensembles $\mathcal{F}_0 = \{a, b\}$, $\mathcal{F}_1 = \{f\}$ et $\mathcal{F}_2 = \{h\}$ et posons $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{F}_2$. Parmi les termes de *cet ensemble* $\mathcal{T}_0(\mathcal{F})$ *particulier* on trouve a, $f(b)$, $h(b, a)$, $f(f(a))$, $f(h(a, b))$, $h(f(a), a)$, $h(f(b), h(a, b))$, etc.

Si l'on veut pouvoir utiliser correctement les symboles d'une signature pour définir un langage de termes, il faut faire deux hypothèses sur les ensembles $\mathcal{F}_0$, $\mathcal{F}_1$, etc.

— La première est que $\mathcal{F}_0$ n'est pas vide : il existe au moins un symbole de constante. Elle est d'une importance cruciale car elle garantit que l'ensemble de termes $\mathcal{T}_0(\mathcal{F})$ ainsi défini n'est pas vide, puisqu'il contient au moins les symboles de constante.

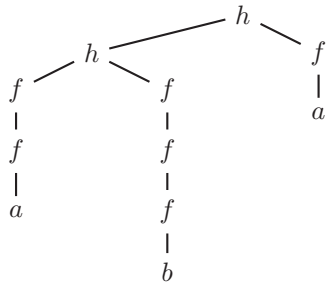
— La seconde est que les ensembles $\mathcal{F}_i$ sont deux à deux disjoints : un même symbole ne peut pas avoir deux arités différentes. Cette hypothèse a pour but d'éviter toute incertitude lors de l'analyse syntaxique des termes. En effet, comment dire que le terme $a(a)$ est « bien formé » ou non, si a est à la fois un symbole de constante et un symbole de fonction ?

Les règles de formation des termes peuvent être lues comme des règles de construction d'arbres dont les étiquettes sont les symboles de $\mathcal{F}$. Selon l'arité des symboles, un nœud de l'arbre a un, deux, etc. sous-arbres. Les feuilles de l'arbre sont les symboles de constantes : d'arité 0, on ne leur attache pas de sous-arbre.

Par exemple, avec la signature de l'exemple 1.1 donné ci-dessus, le terme :

$$h(h(f(f(a)), f(f(f(b)))), f(a))$$

peut être vu comme l'arbre :



La définition 1.1 donne les règles d'assemblage de symboles d'une signature, des parenthèses et de la virgule pour construire un ensemble de termes. Ces symboles constituent le *lexique* du langage des termes. On peut lire la définition 1.1 comme une *grammaire* définissant les écritures acceptables des termes. Dans ce cas, la représentation arborescente d'un terme correspond à son *arbre de syntaxe abstraite*.

Définition inductive, induction structurelle La définition 1.1 qui donne les règles de construction des termes indique que tout assemblage de symboles qui n'obéit pas aux règles (symbole de fonction inconnu, arité non respectée, parenthèse mal fermée, etc.) n'est pas un élément de cet ensemble. Il faut souligner que l'expression « *défini inductivement* » indique aussi que *tous les termes* de $\mathcal{T}_0(\mathcal{F})$ sont obtenus en appliquant un nombre de fois fini les règles de construction données.

Ainsi, on peut parler de tous les termes en considérant simplement les cas où un terme est un symbole de constante k et les cas où il est obtenu par application d'un symbole de fonction à d'autres termes $f(t_1, \ldots, t_n)$. Cette possibilité ouvre celle de poser sur l'ensemble des termes un schéma de raisonnement par *induction structurelle* qui permet de démontrer que tous les termes d'un ensemble $\mathcal{T}_0(\mathcal{F})$ vérifient une certaine propriété. Si P est une propriété portant sur les éléments de $\mathcal{T}_0(\mathcal{F})$, on a :

$$\begin{array}{ll} \text{si} & \text{tout } k \in \mathcal{F}_0 \text{ vérifie } P \\ \text{et} & \text{pour tout } t_1, \cdots, t_n \in \mathcal{T}_0(\mathcal{F}) \text{ et tout } f \in \mathcal{F}_n, \\ & \text{supposer que } t_1, \cdots, t_n \text{ vérifient } P \text{ entraîne que } f(t_1, \ldots, t_n) \text{ vérifie } P \\ \text{alors} & \text{tout } t \in \mathcal{T}_0(\mathcal{F}) \text{ vérifie } P \end{array} \tag{1.9}$$

La notion d'induction structurelle est d'une importance primordiale dans le domaine de la logique mais également de l'informatique. Nous nous y réfèrerons et l'utiliserons à de multiples reprises dans cet ouvrage.

Exemple 1.2 (Entiers de Peano)
Le langage de la logique des prédicats a été mis au point entre la fin du XIXe et le début du XX$^{\text{e}}$ siècle dans le cadre de recherches sur les fondements des mathématiques. En particulier G. Peano (1858-1932) s'intéressa aux fondements de l'arithmétique et posa la définition d'un langage de termes permettant de désigner tous les nombres entiers positifs, que l'on appelle depuis les *entiers de Peano*.

Pour cela, il se donna le symbole Z de constante pour désigner 0 et un symbole de fonction unaire S pour désigner le *successeur* d'un entier (la fonction qui associe $n+1$ à n). En posant $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1$ avec $\mathcal{F}_0 = \{Z\}$ et $\mathcal{F}_1 = \{S\}$, l'ensemble des entiers de Peano est défini comme une « particularisation » de la définition 1.1 de l'ensemble $\mathcal{T}_0(\mathcal{F})$:

— $Z \in \mathcal{T}_0(\mathcal{F})$ (puisque $Z \in \mathcal{F}_0$)
— si $t \in \mathcal{T}_0(\mathcal{F})$ alors $S(t) \in \mathcal{T}_0(\mathcal{F})$ (puisque $S \in \mathcal{F}_1$)

On s'est ainsi donné un moyen pour écrire les entiers 0, 1, 2, 3, etc. sous la forme Z, $S(Z)$, $S(S(Z))$, $S(S(S(Z)))$, etc. Si cette écriture n'est pas utile en pratique, elle est suffisante en théorie.

Induction structurelle et récurrence sur les entiers Avec la définition des entiers de Peano, le principe usuel de récurrence sur les entiers [2] s'exprime comme le principe d'induction structurelle sur les termes de $\mathcal{T}_0(\mathcal{F})$. En effet, pour toute propriété P,

si $P(Z)$,
et si pour tout $t \in \mathcal{T}_0(\mathcal{F})$, $P(t)$ entraîne $P(S(t))$
alors, pour tout $t \in \mathcal{T}_0(\mathcal{F})$, $P(t)$

2. En considérant une propriété P, la récurrence usuelle sur les entiers s'exprime de la façon suivante : si $P(0)$ et si pour tout entier m, $P(m)$ entraîne $P(m+1)$, alors pour tout entier n, $P(n)$.

En fait, les ensembles $\mathbb{N}$ et $\mathcal{T}_0(\mathcal{F})$ sont ici isomorphes : à chaque entier n on peut associer un terme et réciproquement. Intuitivement, le passage de $\mathbb{N}$ vers $\mathcal{T}_0(\mathcal{F})$ s'effectue de la façon suivante : (i) à 0 on associe Z ; (ii) si à n on associe t, alors, à $n+1$ on associe $S(t)$. Réciproquement, le passage de $\mathcal{T}_0(\mathcal{F})$ vers $\mathbb{N}$ est défini de la façon suivante : à Z on associe 0 ; à $S(t)$, on associe $n+1$ où n est l'entier associé à t. Formellement, on définit deux fonctions $i2t : \mathbb{N} \to \mathcal{T}_0(\mathcal{F})$ et $t2i : \mathcal{T}_0(\mathcal{F}) \to \mathbb{N}$, la première par récurrence sur les entiers et la seconde par induction sur la construction des termes :

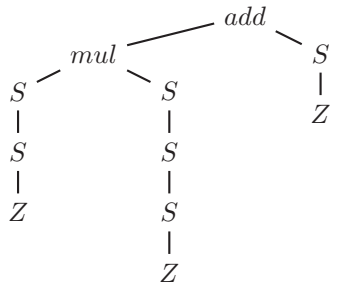
$$\begin{cases} i2t(0) & = & Z \\ i2t(n+1) & = & S(i2t(n)) \end{cases} \qquad \begin{cases} t2i(Z) & = & 0 \\ t2i(S(t)) & = & t2i(t)+1 \end{cases}$$

Ces deux fonctions sont des bijections entre $\mathbb{N}$ et $\mathcal{T}_0(\mathcal{F})$ et elles sont réciproques l'une de l'autre : $i2t(t2i(t)) = t$ et $t2i(i2t(n)) = n$.

Exemple 1.3 (Expressions arithmétiques sans variable)
Pour définir sa théorie de l'arithmétique, Peano va un peu plus loin en ajoutant deux symboles de fonction d'arité 2 : l'un pour l'addition, l'autre pour la multiplication. Posons $\mathcal{F}_2 = \{add, mul\}$ et $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{F}_2$. L'ensemble $\mathcal{T}_0(\mathcal{F})$ des expressions désignant des entiers est maintenant défini par :

- $Z \in \mathcal{T}_0(\mathcal{F})$ (puisque $Z \in \mathcal{F}_0$)
- si $t \in \mathcal{T}_0(\mathcal{F})$ alors $S(t) \in \mathcal{T}_0(\mathcal{F})$ (puisque $S \in \mathcal{F}_1$)
- si $t_1 \in \mathcal{T}_0(\mathcal{F})$ et $t_2 \in \mathcal{T}_0(\mathcal{F})$ alors $add(t_1, t_2) \in \mathcal{T}_0(\mathcal{F})$ et $mul(t_1, t_2) \in \mathcal{T}_0(\mathcal{F})$ (puisque $add \in \mathcal{F}_2$ et $mul \in \mathcal{F}_2$).

Ces expressions sont également représentables comme des structures d'arbre dont les nœuds ont 2, 1 ou aucun fils. Par exemple l'expression arithmétique usuelle $2\times3+1$ est transcrite par le terme $add(mul(S(S(Z)), S(S(S(Z)))), S(Z))$ qui peut se représenter par l'arbre :



Termes et calculs Dans le langage de termes de Peano, $add(S(S(Z)), S(S(Z)))$ est un terme qui désigne un entier, mais lequel ? Si l'on sait que add représente la fonction d'addition et que $S(S(Z))$ représente l'entier 2 alors on peut deviner que $add(S(S(Z)), S(S(Z)))$ représente l'entier 4 car on sait calculer $2+2$. Mais sait-on réellement *calculer* $2+2$ ou avons nous simplement appris par cœur que « 2 et 2 égalent 4 » ? Avec son langage de termes, G. Peano fournit un moyen de calculer effectivement la somme de deux entiers. Pour cela, il pose les égalités élémentaires

suivantes pour tous termes t_1 et t_2 :

$$\begin{cases} add(Z, t_2) & = & t_2 \\ add(S(t_1), t_2) & = & S(add(t_1, t_2)) \end{cases}$$

Avec la donnée de ces équations on peut par exemple calculer :

$$\begin{array}{rcl} add(S(S(Z)), S(S(Z))) & = & S(add(S(Z), S(S(Z)))) \\ & = & S(S(add(Z, S(S(Z))))) \\ & = & S(S(S(S(Z)))) \end{array}$$

Pour aller plus loin : langage de termes avec sortes Originellement consacré à la formalisation des mathématiques, le langage de la logique des prédicats n'avait qu'un seul univers du discours : les nombres pour l'arithmétique ou les ensembles pour la théorie des ensembles. Mais, et en particulier avec l'apparition de l'informatique, s'est fait sentir le besoin de langages désignant des valeurs de natures différentes. Par exemple, des nombres et des listes de nombres. Le principe de définition 1.1 des ensembles de termes n'est pas adaptée à ce besoin.

Reprenons les éléments de signature donnés dans les deux exemples précédents pour formaliser l'arithmétique : Z, S, add et mul. Ajoutons-y les symboles suivants pour représenter les listes [3] :

— nil pour désigner la liste vide ;
— $cons$ pour désigner l'opération binaire d'ajout d'une valeur en tête de liste ;
— $append$ pour désigner l'opération binaire de concaténation de deux listes.

En nous basant sur le seul principe de définition 1.1, nous aurions :

$$\mathcal{F}_0 = \{Z, nil\},\ \mathcal{F}_1 = \{S\} \text{ et } \mathcal{F}_2 = \{add, mul, cons, append\}$$

Les termes $S(nil)$, $add(nil, Z)$, $cons(S(Z), Z)$, $append(Z, nil)$ seraient alors des éléments de notre langage. Or, nous ne souhaitons pas avoir à donner une valeur par exemple à l'addition de 0 et de la liste vide ou à l'ajout de 1 en tête de la liste 0, etc.

Notez que la notion d'arité permet déjà d'exclure du langage des termes des combinaisons non pertinentes ; par exemple, $add(Z) \notin \mathcal{T}_0(\mathcal{F})$ puisque add est un symbole d'arité 2. Pour exclure les constructions non pertinentes d'un langage où cohabitent plusieurs sortes d'objet, on étend la notion d'arité en une notion de *sortes*. Dans notre exemple, nous avons deux sortes de base, disons : nat pour les entiers et $list$ pour les listes d'entiers. À chaque symbole, on associe, selon son arité n, un $(n+1)$-uplet de sortes indiquant, si besoin, la sorte des arguments et la sorte du terme obtenu. Dans notre exemple, nous aurions la correspondance résumée dans le tableau suivant :

symbole		n-uplet de sortes
Z	$\mapsto$	(nat)
S	$\mapsto$	(nat, nat)
add	$\mapsto$	(nat, nat, nat)
mul	$\mapsto$	(nat, nat, nat)
nil	$\mapsto$	$(list)$
$cons$	$\mapsto$	$(nat, list, list)$
$append$	$\mapsto$	$(list, list, list)$

3. Nous reprenons ici les noms de fonctions utilisés par le langage de programmation LISP et ses successeurs.