AN EXTENSION OF THE MONTAGUE SEMANTICS

by PHILIPPE DELSARTE and ANDRÉ THAYSE

Philips Research Laboratory Belgium, B-1348, Louvain-la-Neuve, Belgium

Abstract

Several authors have pointed out some limitations inherent in Montague’s approach to natural language representation, and have attempted to remedy these limitations by working out suitable extensions of the Montague formalism. One of these limitations originates from the fact that logical connectives and modal operators can only be applied to formulae and not to arbitrary logical expressions, which stands in contradiction with the common practice of natural language where connectives and modalities can affect expressions of various syntactic categories (and not only complete sentences).

This paper introduces an extension of Montague’s intensional logic that precisely addresses the problem just alluded to. The proposed extension is equipped with a well-defined ‘Boolean semantics’, analogous with the Keenan–Faltz semantics.

Keywords: Boolean semantics, categorial grammar, intensional logic, Montague semantics, natural language representation

1. Introduction

The formal representation of natural language can be approached by means of the Montague semantics1,2). The main difficulties inherent in the Montague formalism have been pointed out by several authors3–6). Let us give an outline of these limitations.

The Montague semantics can deal only with isolated sentences and, except in some simple cases, cannot correctly interpret the anaphoras. The modalities can affect only sentences; they cannot affect arbitrary expressions as is normally the case in natural language. The logic used by Montague (i.e. the intensional logic) is only able to deal with ‘referentially opaque’ constructions; therefore, it cannot accommodate the perception verbs, since the logic of such verbs is not referentially opaque. (This means, for example, that in a construction such as ‘see + infinitive’, two logically equivalent infinitives may not be
interchangeable.) In view of the limitations of Montague's semantics, several researchers have attempted to develop some other formalisms, more closely adapted to the natural language specificities.

The formalism based on Kamp's discourse representation theory allows one to process a set of sentences (forming a discourse), where the interpretation of each individual sentence depends on the context in which this sentence is stated. The same formalism is also able to deal correctly with anaphoric constructions 7). The Boolean semantics introduced by Keenan and Faltz allows the modalities to affect not only the sentences but also many other types of expressions of natural language. Furthermore, this approach enables us to extend the notion of logical consequence from the syntactic category of sentences to a larger family of syntactic categories 6). Finally the situation semantics introduced by Barwise and Perry allows one to correctly deal with the contexts that are not referentially opaque, and thus to correctly treat the perception verbs 8).

Several authors have attempted to 'translate' these different formalisms into the language of Montague's intensional logic, and have worked out the extensions of this logic that are required for their purpose. The contributions by Hendriks, Groenendijk and Stokhof are especially worth being mentioned in that context 4,5). A partial overview can be found in ref. 9.

This paper is concerned with a simple extension of the intensional logic language, equipped with a well-defined 'Boolean semantics'. The proposed extension allows one to transfer some of the advantages of Keenan and Faltz's Boolean formalism to Montague's formalism. In particular, the logical connectives and the modal operators of the extended logic can affect expressions of any type (not only formulae as in the classical case). This enables us to obtain a straightforward translation for some natural language connectives and modalities, which can affect not only sentences but also expressions of various other syntactic categories.

Section 2 is devoted to recalling the basic concepts of Montague's intensional logic. Section 3 gives a detailed description of the proposed extension and of its 'Boolean semantics'. Section 4 contains some illustrative examples that aim at showing how the extended intensional logic can be applied for natural language representation.

Although this paper is relatively self-contained, the reader is assumed to have a certain acquaintance with the Montague grammar 2) and with the elementary aspects of Boolean semantics 6,10). This paper is taken from a chapter written by the authors in the book From Natural Language Processing to Logic for Expert Systems 9).
2. Logic formalization of natural language

2.1. Categorial grammar

Each natural language is endowed with a grammar, that is to say, a set of structures and rules by means of which all statements belonging to the language, and these statements exclusively, can be produced. In order to be able to process (i.e. to 'understand') the sentences of a natural language, in an automatic manner, it is necessary to formalize the description of its grammar. The categorial grammar formalism is one of the possible formalizations for the syntax of a natural language. The grammatical syntax of the fragment of English language introduced by Montague is an example of a categorial grammar. In this grammar, the notion of a syntactic category is defined recursively as follows:

- $S$ is a syntactic category, associated with the sentences;
- $IV$ is a syntactic category, associated with the intransitive verbs;
- $CN$ is a syntactic category, associated with the common nouns;
- if $A$ and $B$ are syntactic categories, then $A/B$ and $A//B$ are syntactic categories.

More precisely, this is Bennett's definition of the syntactic categories. It differs slightly from Montague's initial definition, in which the categories $IV$ (Intransitive Verb) and $CN$ (Common Noun) are defined in terms of the two basic categories $S$ (Sentence) and $e$ (referring to some extra-linguistic 'entities'), as follows: $IV = S/e$ and $CN = S//e$.

The Montague grammar uses some abbreviations that have become classical. For example, $T$ (Term) represents $S/IV$, $TV$ (Transitive Verb) represents $IV/T$ and $DET$ (Determiner) represents $T/CN$.

In the categorial grammar formalism, the syntactic categories always receive a recursive definition. Each basic expression (or word of the vocabulary) is assigned a syntactic category. Note that the $S$ category is assigned to no basic expression (since a sentence cannot reduce to a single word).

The compound expressions (usually called phrases) of the language are obtained from the basic expressions by means of the syntactic rules of the categorial grammar. Most of these syntactic rules are instances of two rule schemata: the functional application schema (sometimes called categorial cancellation schema), and the Boolean schema.
• Functional application schema

An expression $A$ of syntactic category $\mathcal{A}/\mathcal{B}$ or $\mathcal{A}///\mathcal{B}$ combines with an expression $B$ of syntactic category $\mathcal{B}$ to form an expression $h(A, B)$ of syntactic category $\mathcal{A}$.

• Boolean schema

If $A$ and $B$ are two expressions belonging to the same syntactic category $\mathcal{C}$, then 'not $A$', 'A and $B$', and 'A or $B$' are expressions which also belong to the syntactic category $\mathcal{C}$.

The Montague grammar includes syntactic rules that are defined from five instances of the functional application schema:

$$(\mathcal{A}, \mathcal{B}) = (S, IV), (IV, T), (T, eN), (IV, S), (S, S).$$

For example, the first case, i.e. $\mathcal{A} = S$ and $\mathcal{B} = IV$, yields the following instance of the functional application schema:

An expression of category $T=S/IV$ combines with an expression of category $IV$ to produce an expression of category $S$, i.e. a sentence.

2.2. Intensional logic

Two types of constraints bear strong influence upon the formal logic language devised by Montague: an expressiveness constraint and a translation constraint. The first constraint requires the formal language to be sufficiently 'expressive'; it should be able to account for the subtleties of the fragment of natural language that it is supposed to represent. As to the translation constraint, it requires the syntax of the formal language to be sufficiently 'close' to the syntax of the natural language, in such a way that an 'automatic translation' process can be obtained.

2.2.1. Expressiveness constraint

The logic language has to be able to represent a fragment of natural language comprising certain modalities.

In normal speech, we often talk about possibilities, about goals to be achieved, about forecasts regarding the future, and so on. Most of the sentences of natural language can be now true and now false, depending on the circumstances, the time instant, the point of view of somebody. If we want to be able to represent these modalities by means of a logic language, we have
An extension of the Montague semantics

to give it the structure of a multimodal language, comprising at least the following modal operators:

□: necessarily,
◇: possibly,
[F]: always in the future,
⟨F⟩: sometimes in the future,
[P]: always in the past,
⟨P⟩: sometimes in the past.

The modalities of belief or knowledge, such as 'the agent a believes (or knows) the event e', are represented in Montague's logic language by means of two-place predicate forms as follows:

\[\text{believe}(a, e): a \text{ believes that } e,\]
\[\text{know}(a, e): a \text{ knows that } e.\]

Furthermore, the logic formalism has to be able to express the fact that most sentences of natural language can be analysed according to two different moods: the \textit{de re} mood, and the \textit{de dicto} mood. Two additional operators are introduced to that end:

\[\hat{\text{\textasciicircum}}: \text{intension operator},\]
\[\hat{\sim}: \text{extension operator}.\]

These operators provide us with one of the main ingredients by means of which most expressions of natural language can be given at least two different translations into intensional logic; one of them formalizes the \textit{de re} meaning and the other one formalizes the \textit{de dicto} meaning.

The extension operator is the inverse of the intension operator, in the sense that any logical expression \(\alpha\) satisfies the semantic equivalence relation

\[\hat{\sim} \alpha \approx \alpha.\]

It should be noted that the modal operators can exclusively affect a logical formula, while the intension operator does not have such a limitation; it can affect any logical expression.
2.2.2. Translation constraint

The most essential part of the translation constraint imposed by Montague amounts to the existence of an isomorphism between the analysis of an expression in natural language and the analysis of its translation in intensional logic. This isomorphism is obtained through the introduction of a type-theoretic syntax for intensional logic, and of a translation function that maps each syntactic category of natural language to a type of intensional logic. Each natural expression $A$ of syntactic category $\mathcal{C}$ is given a logical translation, denoted by $A'$ (an expression belonging to intensional logic); the syntactic category of $A'$ is a type, denoted by $f(\mathcal{C})$, where $f$ is the translation function from natural syntactic categories to logical types. The essential observation is the following. If an expression $A$ of category $\mathcal{C}/\mathcal{B}$ (or $\mathcal{A}/\mathcal{B}$) combines with an expression $B$ of category $\mathcal{B}$ to form a compound expression $h(A, B)$, of category $\mathcal{A}$, then the translation $A'$, of type $f(\mathcal{A}/\mathcal{B})$, combines with the translation $B'$, of type $f(\mathcal{B})$, to form a translation $h'(A', B')$, of type $f(\mathcal{A})$. An additional requirement bearing on the logical language is the following: the function $h'$ should be 'as close as possible' to the function $h$. In particular, if the function $h$ simply acts as a concatenation of its arguments $A$ and $B$, then, as far as possible, the function $h'$ should also act as a concatenation of its arguments $A'$ and $B'$.

The base symbols occurring in the definition of the logical types are $t$ (for 'truth'), $e$ (for 'entity') and $s$ (for 'sense'). The logical types are defined recursively as follows:

- $t$ and $t$ are types;
- $e$ and $t$ are types;
- if $a$ and $b$ are types, then $\langle a, b \rangle$ is a type;
- If $a$ is a type, then $\langle s, a \rangle$ is a type.

The translation of the fragment of natural language into the intensional logic language is obtained by associating a logical type with each natural syntactic category. More precisely, a translation function $f$ is defined; it is a mapping from the set of syntactic categories of natural language into the set of types of intensional logic. The function $f$ is defined in a recursive manner as follows:

1. $f(S) = t$,
2. $f(IV) = f(CN) = \langle e, t \rangle$,
3. $f(\mathcal{A} / \mathcal{B}) = f(\mathcal{A} / / \mathcal{B}) = \langle s, f(\mathcal{B}) \rangle, f(\mathcal{A}) \rangle$ for all syntactic categories $\mathcal{A}$ and $\mathcal{B}$.

P. Delsarte and A. Thayse

28
In fact, the translation function $f$ given above corresponds to Bennett's version. In Montague's initial version, the type associated with the categories of intransitive verbs and of common nouns is defined as follows:

2'. $f(IV) = f(CN) = \langle \langle s, e \rangle, t \rangle$.

To the functional application schema:

An expression of category $\mathcal{A}/\mathcal{B}$ combines with an expression of category $\mathcal{B}$ to form an expression of category $\mathcal{A}$, there corresponds a combination rule over the types:

An expression of type $\langle b, a \rangle$ combines with an expression of type $b$ to form an expression of type $a$.

The isomorphism between the syntactic analysis of natural language and the syntactic analysis of intensional logic language results from the definition of a correspondence between syntactic rule schemata and translation rule schemata. Thus, to the functional application schema there corresponds a translation schema as given below.

- **Functional application schema:**

$\langle A \rangle_{\mathcal{A}/\mathcal{B}} + \langle B \rangle_{\mathcal{B}} = (h(A, B))_{\mathcal{A}}$.

In the particular case where the function $h$ acts as the concatenation of its arguments, this becomes:

$\langle A \rangle_{\mathcal{A}/\mathcal{B}} + \langle B \rangle_{\mathcal{B}} = (AB)_{\mathcal{A}}$.

- **Translation schema:**

$\langle A' \rangle_{\langle s, f(\mathcal{B}) \rangle, f(\mathcal{A})} + \langle B' \rangle_{f(\mathcal{B})} = (h'(A', B'))_{f(\mathcal{A})}$.

In the particular case where the function $h'$ is the translation of the concatenation, this becomes:

$\langle A' \rangle_{\langle s, f(\mathcal{B}) \rangle, f(\mathcal{A})} + \langle B' \rangle_{f(\mathcal{B})} = (A' \langle B' \rangle)_{f(\mathcal{A})}$.

(In the schemata above, each expression has an index which represents its syntactic category or its type. The symbols + and = occurring in the functional application schema are used to describe a syntactic rule of Montague's...
categorial grammar. The same symbols are used in the translation schema, in an informal manner.)

To the Boolean schema there also corresponds a translation schema. Let $\tau$ be a dyadic connective of natural language ($\tau = \text{‘and’}, \text{‘or’}$), and let $\tau'$ be its translation in formal logic ($\tau' = \land, \lor$). The correspondence is the following.

- **Boolean schema:**
  
  $$(A)\varphi \tau (B)\varphi = h_\varphi((A)\varphi, (B)\varphi, \tau) = (A\tau B)\varphi$$

- **Translation schema:**
  
  $$(A')_{f(\varphi)} \tau (B')_{f(\varphi)} = h'_\varphi((A')_{f(\varphi)}, (B')_{f(\varphi)}, \tau').$$

(Note that the latter writing is informal, since the symbol $\tau$ does not belong to the logic language.)

### 2.3. The Montague semantics

In addition to the requirement of an isomorphism between the syntactic analysis of the natural language expressions and the syntactic analysis of their formal logic translations, Montague sets the requirement of a homomorphism between the syntax of natural language and its semantics. Thus, the semantics of natural language has to be compositional, i.e. to comply with Frege’s compositionality principle, according to which the meaning of the whole is a function of the meaning of the parts.

If $\alpha$ is an expression of intensional logic (which may represent the translation, $\alpha = A'$, of an expression $A$ of natural language), then

$$[[\alpha]]_{\mathcal{M}, w, g}$$

represents the semantic value of $\alpha$. This semantic value is determined with respect to a model $\mathcal{M}$, a possible world $w$, and an assignment function $g$. The model $\mathcal{M}$ is a fourtuple $(U, W, \mathcal{R}, V)$ having the following components:

- **$U$** is a set of individual objects; it is the domain of interpretation of the formal language.
- **$W$** is a set of possible worlds $w$.
- **$\mathcal{R}$** is a set of accessibility relations between the possible worlds; the different elements of $\mathcal{R}$ are associated with the different modalities of the language.
An extension of the Montague semantics

• $V$ is an interpretation function; it assigns elements of $U$ to the individual constants and it assigns sets of $n$-tuples of elements of $U$ to the $n$-place predicate constants, for each possible world $w \in W$.

The assignment function $g$ assigns elements of $U$ to the individual variables of the language; more generally, for each type $a$, the function $g$ assigns elements of a well-defined set $D_a$ to the variables of type $a$. (Details concerning the sets $D_a$ are given below.)

The semantic value of any expression $\alpha$ of the formal language is obtained, in a recursive manner, from the semantic values of the basic elements of the language (the non-logical constants and the variables) and from the rules that define the semantics of the logical connectives, the quantifiers, the modal operators, and the intension and extension operators.

The semantics of the basic elements is defined as follows:

• if $c$ is a non-logical constant, then
  $\llbracket c \rrbracket^{d,w,g} = V(w, c)$;

• if $x$ is a variable, then
  $\llbracket x \rrbracket^{d,w,g} = g(x)$.

If $\alpha$ is an element of a given type $a$, then the semantic value of $\alpha$, also called the denotation of $\alpha$, belongs to a well-determined set, which is noted $D_a$ and is referred to as the set of possible denotations of type $a$. The sets $D_a$ are recursively defined by the following rules:

$D_e = U$;
$D_i = B = \{1, 0\}$;
$D_{\langle a, a \rangle} = D_a^W = W \rightarrow D_a$;
$D_{\langle a, b \rangle} = D_b^a = D_a \rightarrow D_b$.

(The notations $Y^X$ and $X \rightarrow Y$ both represent the set of mappings from a set $X$ to a set $Y$.)

The ‘denotation function’ $D$ (mapping every type $a$ to the corresponding set of possible denotations $D_a$), recursively defined as above, allows the requirement of a homomorphism between the syntax and the semantics to be fulfilled.
Let us point out that the two homomorphisms

\[
\text{natural syntax} \rightarrow \text{formal syntax} \rightarrow \text{semantics}
\]

together with the requirement of associating a (unique) logical type with each syntactic category of natural language are at the origin of the introduction of the lambda calculus as an ingredient of intensional logic. For example, the formal translations of the noun phrases and of the determiners will be lambda expressions.

3. Boolean semantics of an extended intensional language

The general idea examined in this section has its origin at the contact point between the theory of Montague and that of Keenan and Faltz\(^1,6\). (A comparison between these two theories can be found in ref. 10.) The subject can be introduced as follows. For certain logical types \(a\), which will be referred to as Boolean types, the 'semantic category' \(D_a\) (of the Montague logic) has the structure of a complete atomic Boolean algebra; in concrete terms, \(D_a\) can be identified, in a natural manner, with the power set (i.e. the set of all subsets) of a well-defined set. This allows one to apply the logical connectives, the modal operators, and the quantifiers to the set of expressions of type \(a\), and to endow these connectives, operators, and quantifiers with a semantics that can legitimately be called a Boolean semantics.

The choice of this terminology, borrowed from Keenan and Faltz\(^6\), is motivated by the fact that the proposed semantics exactly reflects the Boolean algebra structure of the set \(D_a\), for every Boolean type \(a\). From a formal point of view, the main difference between the semantics described here and the semantics introduced by Keenan and Faltz is concerned with the ontology, i.e. with the semantic primitives. As it will be seen in what follows, one can preserve the ontology underlying Montague's intensional logic, while endowing this logic (after a rather immediate enrichment) with a Boolean semantics, in due form.

3.1. Structure of the set of possible denotations for a Boolean type

3.1.1. Preliminaries from intensional logic

The types of Montague's intensional language are defined from the three primitive symbols \(e, t\) and \(s\) by means of recursive rules\(^2,3\). The symbols \(e\)
An extension of the Montague semantics

and t represent the basic types; the symbol s, which is not a type, intervenes in the formation of intensional logic from extensional logic. It will prove convenient to give this symbol a status similar to that of a type; we will use the term 'pseudotype'.

Let us define \( E_a \) as the set of logical expressions of type \( a \) in the intensional language (extended as explained in Sec. 3.2 below), and let us define \( D_a \) as the set of possible denotations (semantic values) for the expressions of type \( a \). Thus, we can write

\[
\llbracket a \rrbracket^{\mathcal{M},w,g} \in D_a \quad \text{for all } \alpha \in E_a,
\]

for every model \( \mathcal{M} \), every possible world \( w \in W \), and every assignment function \( g \). (In what follows, we will explicitly write only that part of the triple \( (\mathcal{M}, w, g) \) which is really useful to the intelligibility of the semantic equalities under consideration.) The sets \( D_a \) are defined recursively, as explained in Sec. 2.3, from the primitives \( U, B \) and \( W \).

The symbol \( s \) will henceforth be referred to as the pseudotype of intensional logic. Let us define the set \( D_s \), associated with the pseudotype \( s \), as follows:

\[
D_s = W.
\]

The definition \( D_{\langle s, a \rangle} = D_a^W \) can then be obtained by specializing the general definition \( D_{\langle b, a \rangle} = D_a^{D_b} \) to the case \( b = s \).

For each expression \( \alpha \in E_a \), there is a companion expression \( \check{\alpha} \in E_{\langle s, a \rangle} \), which denotes the intension of \( \alpha \). The semantic value of \( \check{\alpha} \) is a well-defined function from \( W \) to \( D_a \); its values \( \llbracket \check{\alpha} \rrbracket (w) \) are defined by

\[
\llbracket \check{\alpha} \rrbracket (w) = \llbracket \alpha \rrbracket^w, \quad \text{for } w \in W.
\]

Conversely, if \( \alpha \) is an intensional expression, i.e. an element of the set \( E_{\langle s, a \rangle} \) for a certain type \( a \), then it has a companion expression \( \check{\alpha} \in E_a \), semantically characterized by the fact that its intension equals the extension of \( \alpha \). In precise terms, the semantic value of the expression \( \check{\alpha} \) is defined as follows:

\[
\llbracket \check{\alpha} \rrbracket^w = \llbracket \alpha \rrbracket(w), \quad \text{for } w \in W.
\]

From these definitions we immediately deduce the rule of cancellation of intension by extension, that is \( \llbracket \check{\alpha} \rrbracket = \llbracket \alpha \rrbracket \), for all expressions \( \alpha \).

Recall that from each pair \( (\alpha, \beta) \), consisting of expressions \( \alpha \in E_{\langle b, a \rangle} \) and \( \beta \in E_b \), one can construct a compound expression \( \alpha(\beta) \in E_a \), the semantic value
of which is obtained by functional composition:

$$\mathcal{L}(\alpha(\beta)) = \mathcal{L}(\alpha)(\mathcal{L}(\beta)).$$

In addition, the Montague logic is equipped with the lambda calculus formalism, which plays an essential role in the translation of natural language into intensional language. Recall that for each pair \((x, \alpha)\) consisting of a variable \(x\), of type \(b\), and of an expression \(\alpha\), of type \(a\), one can construct an expression \(\lambda x[\alpha]\), of type \(\langle b, a \rangle\), called a lambda expression. (It is seen that the lambda operator ‘complexifies’ the types; it plays a role which somehow is the converse of the role played by the composition operator, which ‘simplifies’ the types.)

The semantic value of the expression \(\lambda x[\alpha]\) is a well-defined function from \(D_b\) to \(D_a\); its values are given by

$$\mathcal{L}(\lambda x[\alpha])^\theta(r) = \mathcal{L}(\alpha)^\varphi, \quad \text{for } r \in D_b,$$

where \(g\) is the considered assignment function. In this writing, \(g'\) represents the \(x\)-variant of \(g\), defined by \(g'(x) = r\) and \(g'(y) = g(y)\) for any variable \(y\) distinct from \(x\). From the definitions above one can deduce the lambda conversion rule, namely

$$\mathcal{L}(\lambda x[\alpha](\beta)) = \mathcal{L}(\alpha_x^\theta),$$

where \(\alpha_x^\theta\) represents the expression obtained from \(\alpha\) by replacing each occurrence of the variable \(x\) by the expression \(\beta\) (of the same type \(b\) as \(x\)). It is important to notice that this rule does not have full generality. The delicate cases are those where \(x\) is in the scope of a modal operator or of an intension operator (see ref. 2, p. 167 and p. 177). It will not be necessary for us to enter into a general discussion of the validity of the lambda conversion rule. We can be content with making the required verification in the cases that are of real interest for us; there is no difficulty about it.

### 3.1.2. Boolean types and associated structures

A type \(a\) will be said to be a Boolean type if the rightmost symbol in the explicit writing of \(a\) is the truth symbol \(t\) (and not the entity symbol \(e\)). The simplest case is \(a = t\). The general form of a Boolean type is

\[ a = \langle b_1, \langle b_2, \ldots, \langle b_n, t \rangle \ldots \rangle \rangle, \]
An extension of the Montague semantics

with \( n \geq 0 \), where \( b_i \) is either a type, or the pseudotype \( s \), for \( i = 1, \ldots, n \). In more precise terms, the Boolean types exactly are the types that are obtained by means of the recursive construction given in Sec. 2.2.2 under the restriction that the initial type is \( t \).

The logical expressions of Boolean type are those which, in the final analysis, give rise to an expression of type \( t \), i.e. to a logical formula. (It should be noted that the translation of any expression of natural language into intensional logic always gives a logical expression of Boolean type.) It will be seen in Sec. 3.2 how one can perform a semantic analysis of certain compound expressions, of Boolean type, which occur as 'meaningful pieces' of complete formulae. (These expressions belong to the extended intensional language that will be introduced on the way.)

It is easy to show that, for any Boolean type \( a \), the set of possible denotations, \( D_a \), possesses the structure of a complete atomic Boolean algebra. In more explicit terms, \( D_a \) can be viewed as the set, represented by \( 2^{G_a} \), consisting of all subsets of a certain set \( G_a \); the meet and join operations over \( D_a \) can then be identified with the intersection and union operations over the power set \( 2^{G_a} \).

To begin with, let us examine an example that illustrates the machinery. Consider the Boolean type

\[
a = \langle s, \langle s, e \rangle, t \rangle,
\]

the denotations of which represent properties of individual concepts. (If \( \alpha \) is an expression of type \( a \) and \( \beta \) is an expression of type \( \langle s, e \rangle \), denoting an individual concept, then \( (\check{\alpha})(\beta) \) is an expression of type \( t \), denoting a truth value.) In this case, the set of possible denotations \( D_a \) is the following:

\[
D_a = (W \rightarrow ((W \rightarrow U) \rightarrow B)) = (B^{U^W})^W = B^{(U^W \times W)},
\]

where \( \times \) represents the Cartesian product. This clearly reveals the property mentioned above, i.e. \( D_a = 2^{G_a} \) (identified with \( B^{G_a} \)), where \( G_a = U^W \times W \) in our example.

To obtain this result, we make use of the following standard identification (with an abuse of notation):

\[
(X^Y)^Z = X^{(Y \times Z)}.
\]

More precisely, we identify any function \( f \) from \( Z \) to \( (Y \rightarrow X) \) with a function
from \( Y \times Z \) to \( X \) by setting
\[
f(y, z) = f(z)(y),
\]
for all \( y \in Y \) and all \( z \in Z \). (The example above corresponds to the sets \( X = B \), \( Y = U^W \) and \( Z = W \).

In general, for \( a = \langle b_1, \ldots, b_n, t \rangle \ldots \rangle \), i.e. for a Boolean type of 'depth' \( n \), we thus obtain, by a straightforward induction argument, the following representation:
\[
D_a = 2^{G_a}, \quad \text{with} \quad G_a = D_{b_n} \times \cdots \times D_{b_2} \times D_{b_1}.
\]

Here we identify the power set \( 2^{G_a} \) with the set \( B^{G_a} \) consisting of the functions from \( G_a \) to \( B = \{1, 0\} \). More precisely, we identify a subset of \( G_a \) with its characteristic function. (For the type \( a = t \), i.e. for \( n = 0 \), we simply have \( G_a = \{0\} \).

Let us finally give an elementary result which will be needed in the next subsection. Consider a subset \( F \) of the Cartesian product \( G_a \times D_b \), i.e. a relation between the sets \( G_a \) and \( D_b \). (The symbol \( b \) here represents either an arbitrary type, or the pseudotype \( s \).) By definition, \( F \) is an element of the set
\[
D_{\langle b, a \rangle} = 2^{G_{\langle b, a \rangle}} = 2^{G_a} \times D_b.
\]
If we identify \( F \) with a function from \( D_b \) to \( 2^{G_a} \), then we can write the equality
\[
F(r) = \{ q \in G_a : (q, r) \in F \}, \quad \text{for each} \ r \in D_b.
\]
From this we deduce the following two Boolean identities:
\[
F^*(r) = (F(r))^*,
\]
\[
\left( \bigcap_{i \in I} F_i \right)(r) = \bigcap_{i \in I} F_i(r),
\]
where \( (F_i)_{i \in I} \) is any family of subsets \( F_i \) of \( G_a \times D_b \). (In the first identity, \( F^* \) represents the complement of \( F \) with respect to \( G_a \times D_b \), while \( (F(r))^* \) represents the complement of \( F(r) \) with respect to \( G_a \).) In view of de Morgan’s law, there exists a dual version of the second Boolean identity, in which the intersection operator is replaced by the union operator.
3.2. Enrichment of Montague's logic

3.2.1. Logical connectives

The first enrichment idea (the simplest one) consists in allowing the logical connectives ($\neg$, $\land$, $\lor$, $\rightarrow$, $\equiv$) to act on the logical expressions of any Boolean type $a$, and not only on the formulae (which are the expressions of type $t$) as in Montague's logic. If $\alpha$, $\alpha_1$ and $\alpha_2$ are expressions belonging to the set $E_a$, then the expressions $\neg \alpha$, $\alpha_1 \land \alpha_2$, $\alpha_1 \lor \alpha_2$, $\alpha_1 \rightarrow \alpha_2$ and $\alpha_1 \equiv \alpha_2$ are accepted as well formed and belonging to the same set $E_a$.

For example, if $\alpha_1$ and $\alpha_2$ represent two proper names, such as 'John' and 'Mary', then $\alpha_1 \land \alpha_2$ will be an acceptable representation of the noun phrase 'John and Mary'. Similarly, if $\alpha_1$ and $\alpha_2$ represent two intransitive verbs, such as 'sing' and 'dance', then $\alpha_1 \land \alpha_2$ will be a suitable representation of the verb phrase 'sing and dance'. (These naive examples aim only at intuitively explaining the main motivation of the approach. It will be seen in Sec. 4 how the enrichments proposed here can actually be used in the framework of natural language processing.)

The semantics of the logical connectives thus extended (to each class $E_a$ with a Boolean type $a$) is defined by means of the following equalities:

\[
\begin{align*}
\llbracket \neg \alpha \rrbracket &= \llbracket \alpha \rrbracket^* = D_a \setminus \llbracket \alpha \rrbracket, \\
\llbracket \alpha_1 \land \alpha_2 \rrbracket &= \llbracket \alpha_1 \rrbracket \cap \llbracket \alpha_2 \rrbracket, \\
\llbracket \alpha_1 \lor \alpha_2 \rrbracket &= \llbracket \alpha_1 \rrbracket \cup \llbracket \alpha_2 \rrbracket, \\
\llbracket \alpha_1 \rightarrow \alpha_2 \rrbracket &= \llbracket \alpha_1 \rrbracket^* \cup \llbracket \alpha_2 \rrbracket, \\
\llbracket \alpha_1 \equiv \alpha_2 \rrbracket &= \llbracket \alpha_1 \rrbracket \oplus \llbracket \alpha_2 \rrbracket^*,
\end{align*}
\]

where the symbols $^*$, $\cap$, $\cup$ and $\oplus$ represent respectively the set-theoretic complementation, intersection, union, and symmetric difference over the power set $D_a = 2^{G_a}$ formed with the subsets of $G_a$. In the classical case, $a = t$, this exactly corresponds to the usual definition of the semantics of the logical connectives.

The main usefulness of the proposed approach lies in the fact that it allows one to perform some 'partial analyses' of a formula involving connectives. It is interesting to examine the question of verifying whether different possible analyses produce the same result. For example, it seems to be desirable that the expression '(Mary and Lucy) sing' always receives the same semantic value as the expression '(Mary sings) and (Lucy sings)'. Similarly, it seems to be desirable that 'Mary (sings and dances)' is semantically equivalent to 'Mary
sings) and (Mary dances)'. As explained in Sec. 4 below, the equivalences in question are actually satisfied. An important difference between these two examples should, however, be emphasized. In the first case, the equivalence is preserved if the proper names 'Mary' and 'Lucy' are replaced by arbitrary noun phrases. In the second case, if the proper name 'Mary' is replaced by a noun phrase such as 'a woman', then the equivalence ceases to be satisfied.

These informal comments aim at pointing out that the results obtained through the proposed approach often agree with the intuitive requirements. In fact, from a semantic point of view, our approach is analogous with the Keenan and Faltz approach. However, there are some noticeable differences. Consider, for example, the two expressions 'loves (John and Paul)' and '(loves John) and (loves Paul)', analysed as intransitive verb phrases (whose subject can be a proper name, e.g. 'Mary'). Keenan and Faltz's approach allows one to consider these expressions as semantically equivalent, while the approach introduced here does not allow it. (This is due to the specific structure of the translation of transitive verbs, such as 'love', in Montague's theory.)

In what follows, we will closely examine the above-mentioned questions of semantic equivalence (in the framework of our extended intensional logic). Let $\alpha, \alpha_1$ and $\alpha_2$ be expressions of type $\langle b, a \rangle$, and let $\beta$ be an expression of type $b$. Assuming that $a$ is Boolean, we can prove (in full generality), the following semantic equivalences:

$$(-\alpha)(\beta) \approx -(-\alpha(\beta)),$$

$$(\alpha_1 \land \alpha_2)(\beta) \approx \alpha_1(\beta) \land \alpha_2(\beta).$$

(There are similar results for the other connectives, and they require no additional proof.) The equivalence relation $\approx$ used here is defined as follows. For two logical expressions $\gamma_1$ and $\gamma_2$ of the same type, writing $\gamma_1 \approx \gamma_2$ means that the semantic values $\llbracket \gamma_1 \rrbracket$ and $\llbracket \gamma_2 \rrbracket$ are equal, for any model, any possible world, and any assignment function.

The semantic equivalences written above can be deduced immediately from the definitions, via the Boolean identities given at the end of Sec. 3.1. More precisely, the first identity, applied to the function $F = \llbracket \alpha \rrbracket$ and to the point $r = \llbracket \beta \rrbracket$, yields the equivalence involving the logical negation; the second identity, applied to the functions $F_i = \llbracket \alpha_i \rrbracket$ for $i = 1, 2$, and to the point $r = \llbracket \beta \rrbracket$, yields the equivalence involving the logical conjunction.

A very similar argument allows us to obtain analogous results about the extension operator (instead of the composition operator). If $\alpha, \alpha_1$ and $\alpha_2$ are expressions of type $\langle s, a \rangle$, where $a$ is Boolean, then we have the semantic
An extension of the Montague semantics

equivalences

\[ \tilde{(-\alpha)} \approx \tilde{(-\alpha)}, \]
\[ (\alpha \land \alpha_2) \approx (\tilde{\alpha}) \land (\tilde{\alpha}_2). \]

It is also useful to see how the lambda operator (the ‘converse’ of the composition operator) interacts with the extended connectives. Let us mention the following semantic equivalences:

\[ \lambda x[\tilde{\alpha}] \approx \tilde{(-\alpha)}, \]
\[ \lambda x[\alpha_1 \land \alpha_2] \approx \lambda x[\tilde{\alpha}_1] \land \lambda x[\tilde{\alpha}_2], \]

where \( \alpha, \alpha_1 \) and \( \alpha_2 \) are expressions of type \( a \) (Boolean) and \( x \) is a variable of type \( b \) (arbitrary). The proof is obtained, from the definition of the semantics of the lambda operator and of the extended logical connectives, by a simple application of the Boolean identities. Let us give analogous results about the intension operator. If \( \alpha, \alpha_1 \) and \( \alpha_2 \) are expressions of type \( a \) (assumed to be Boolean), then we have the semantic equivalences

\[ \tilde{(-\alpha)} \approx \tilde{(-\alpha)}, \)
\[ \tilde{(\alpha_1 \land \alpha_2)} \approx \tilde{(-\alpha_1)} \land \tilde{(-\alpha_2)}. \]

Let us now go back to the question of the ‘distributivity of composition with respect to conjunction (or disjunction)’. In analyses like ‘Mary (sings and dances)’, for which the connectives occur on the right, the semantic equivalences given above cannot be applied as such; the desired results are only obtainable through the commutation effect of the lambda operator. This question deserves to be examined in detail. In Montague’s translation system, a proper name, e.g. ‘Mary’, is represented by a logical expression \( \beta \) of the following form:

\[ \beta = \lambda x[\tilde{x}(m)], \]

where \( m \) is a constant of type \( \langle s, e \rangle \), denoting an individual concept, and where \( x \) is a variable of type \( b = \langle s, \langle s, e \rangle, t \rangle \), denoting a property of individual concepts. By definition, this expression \( \beta \), of type \( \langle b, t \rangle \), denotes a set of properties of individual concepts.

If \( \alpha_1 \) and \( \alpha_2 \) are two constants, of type \( \langle \langle s, e \rangle, t \rangle \), which denote sets of individual concepts, and which are the formal translations of intransitive verbs
such as 'sing' and 'dance', then the expression $\beta(\langle x_1 \land x_2 \rangle)$, of type $t$, can be considered as a translation of the sentence 'Mary (sings and dances)' in the extended intensional language. By successively applying the lambda conversion rule, the rule of cancellation of intension by extension, and the distributivity of composition with respect to conjunction, we obtain the semantic equivalence

$$\beta(\langle x_1 \land x_2 \rangle) \approx x_1(m) \land x_2(m)$$

$$\approx \beta(\langle x_1 \rangle) \land \beta(\langle x_2 \rangle).$$

As a result, the analysis 'Mary (sings and dances)' is equivalent to the analysis '(Mary sings) and (Mary dances)'.

We can also obtain this conclusion, in a more direct manner, by examining the semantics of Montague's formal representation of the proper names. For the expression $\beta$ defined from the constant $m$ as above, the semantic value $\llbracket \beta \rrbracket^w$ can be identified with the principal filter generated by the atom $\{\llbracket m \rrbracket, w\}$, in the lattice $D_b = 2^{G_b}$, with $G_b = U^w \times W$. In other words, we have the equality

$$\llbracket \beta \rrbracket^w = \{r \subset U^w \times W : (\llbracket m \rrbracket, w) \in r\}.$$  

The equivalence $\beta(\langle x_1 \land x_2 \rangle) \approx \beta(\langle x_1 \rangle) \land \beta(\langle x_2 \rangle)$ immediately follows from this property. Note that the filter structure for the formal semantics of the proper names also occurs in the Keenan–Faltz approach.

**Remark:** The semantic equivalence $\beta(\langle x_1 \land x_2 \rangle) \approx \beta(\langle x_1 \rangle) \land \beta(\langle x_2 \rangle)$ is valid for any Boolean type $a$, under the following general assumption. The expression $\beta$ is a lambda expression of the form $\lambda x[ x(y) ]$, where $x$ is a variable of type $b = \langle s, \langle c, a \rangle \rangle$, and $y$ is an expression, of type $c$ (arbitrary), containing no free occurrence of $x$; and $\alpha_i$ is an expression of type $\langle c, a \rangle$, for $i = 1, 2$.

### 3.2.2. Modal Operators

By a similar method, we can extend Montague's intensional logic for what regards the modalities. Let us first recall some classical notions belonging to this area. Consider a pair $\langle [M], \langle M \rangle \rangle$ consisting of a universal modal operator $[M]$ and of the corresponding ('dual') existential modal operator $\langle M \rangle$, together with an associated accessibility relation $R$, defined over a set $W$ of possible worlds. In the framework of natural language modelling, one especially needs the following three (universal) modal operators:

1. The necessity operator, $\Box$, characterized by $wRw'$ for all $w$ and $w'$ in $W$;
An extension of the Montague semantics

2. The operator 'always in the future', \([F]\), characterized by \((o, \tau)R(o', \tau')\) if and only if \(\tau < \tau'\);

3. The operator 'always in the past', \([P]\), characterized by \((o, \tau)R(o', \tau')\) if and only if \(\tau' < \tau\).

In this context, the set \(W\) is assumed to have the structure of a Cartesian product, \(W = \Omega \times T\); the elements \(o\) of \(\Omega\) are interpreted as the possible worlds (in a restricted sense), while the elements \(\tau\) of \(T\) are interpreted as the time instants. The symbol < represents a total order relation over the set \(T\).

In the proposed extension, the modal operators \([M]\) and \(<M>\) can act on the logical expressions of any Boolean type. (The classical theory restricts itself to the type \(t\).) More precisely, if \(\alpha\) is an expression of type \(a\), assumed to be Boolean, then we consider \([M]\alpha\) and \(<M>\alpha\) as well-formed expressions of type \(a\). For example, if \(\alpha\) represents a common noun, such as 'president', then the expression \(<P>\alpha\) could represent the phrase 'former president'. In the next section, it will be seen how such an extension of the modal operators can actually be used in the formal translation of natural language into Montague's intensional language.

Let us mention a particularly interesting application, concerning the tense modifications of the verbs. If \(\alpha\) is a constant, of type \(<s, e>, t,>\), representing an intransitive verb, like 'sing', then the modal expression \(<F>\alpha\) can be used to represent the future form 'will sing'. As explained at the end of this section, such a representation is satisfactory in that the analysis '(Mary) ((in the future) sings)' is equivalent to the analysis '(in the future) (Mary sings)'.

For any expression \(\alpha\) (of Boolean type), the semantic values of the modal expressions \([M]\alpha\) and \(<M>\alpha\) are defined as follows:

\[
\llbracket [M] \alpha \rrbracket^{R, w} = \bigcap_{w' \in R(w)} \llbracket \alpha \rrbracket^{R, w'},
\]
\[
\llbracket <M> \alpha \rrbracket^{R, w} = \bigcup_{w' \in R(w)} \llbracket \alpha \rrbracket^{R, w'},
\]

where \(R(w)\) is the subset of \(W\) containing the possible worlds that are accessible from \(w\), that is

\[R(w) = \{w' \in W : wRw'\}\]

These definitions are natural extensions of the classical concepts (concerning the type \(t\)) to the case of an arbitrary Boolean type.
Let us now examine (in a similar manner as in the preceding subsection) the question of semantic equivalence for the compound expressions that involve modalities. We will see that, under a suitable assumption, the following equivalence is satisfied:

\[(\text{M}er)(P) \approx \text{M}(er:(P)),\]

with \(\alpha \in E_{b,a}\) and \(\beta \in E_b\), for a given Boolean type \(a\). (Here, \([M]\) represents any universal modal operator. The case of the existential modal operator \(\langle M \rangle\) is exactly the same, owing to the duality property \(\langle M \rangle \alpha \approx -[M](\neg \alpha)\), resulting from de Morgan's law.) For the reason explained below, the semantic equivalence written above is not satisfied in full generality. We will be led to assume that \(b\) is an intensional type, i.e. a type of the form \(b = \langle s, c \rangle\) for a certain type \(c\). Under this assumption (which is quite natural in this context), the equivalence we are interested in is always satisfied, for any modal operator, whatever the expressions \(\alpha\) and \(\beta\).

By use of the second Boolean identity given in Sec. 3.1 we see that the desired semantic equivalence amounts to the equality

\[\text{IV}' \text{eR(IV) IV}' \text{eR(IV)}\]

Assume that \(b\) is an intensional type, which means that the expression \(\beta\) denotes an intension. In this case, the semantic value of \(\beta\) does not depend on the choice of the possible world (see the first part of Sec. 3.1). As a consequence, the desired equality is obviously satisfied.

The equivalence that we have just established does not directly apply to the above-mentioned question about tense modifications of the verbs. Let us now examine how to treat this question. Consider an expression \(\beta\) such that

\[\beta \approx \lambda x[\alpha(y)],\]

where \(\gamma\) is an expression of type \(c\), assumed to be intensional, and where \(x\) is a variable of type \(b = \langle s, c, a \rangle\), for a certain Boolean type \(a\). Assume that \(\gamma\) contains no free occurrence of \(x\). Let \(\alpha\) be an expression of type \(\langle c, a \rangle\). Then, for any modal universal operator \([M]\), we have the semantic equivalences

\[\beta([M]\alpha)) \approx ([M]\alpha)(\gamma) \approx [M](\alpha(\gamma)) \approx [M](\beta(\alpha)).\]
Let us finally mention the 'commutativity' between the lambda operator and any modal operator. If x is a variable (of an arbitrary type), and if α is an expression of Boolean type, then we have the semantic equivalence

\[ \lambda x[[M]α] \approx [M](\lambda x[α]). \]

### 3.2.3. Quantifiers

A similar approach can be used for the quantifiers ∀y and ∃y (whatever the type of the variable y). Let a be any Boolean type. If α is an expression of type a, then we consider ∀yα and ∃yα as two well-formed expressions of type a (in the extended intensional logic that we are introducing). This allows us to quantify some expressions that are not formulae. Before giving the precise semantics of quantifiers in the general case, let us examine an example stated in a 'natural language' style. (It should be pointed out, however, that the extension introduced in this third paragraph does not seem to have direct applications in the framework of the translation of natural language into intensional logic language, as this translation is conceived by Montague and his continuators.)

Consider a two-place predicate, \( A(t_1, t_2) \), where the arguments \( t_1 \) and \( t_2 \) are of type \( C_1 \) and \( C_2 \), respectively. The symbol \( A \) can be viewed as a constant of type \( (C_2, \langle C_1', t \rangle, \ldots, \text{via the equivalence} \)

\[ A(t_2)(t_1) \approx A(t_1, t_2). \]

Assume that the constant \( A \) represents a transitive verb, e.g. 'admire', and that the predicate \( A(t_1, t_2) \) represents a sentence such as '\( t_1 \) admires \( t_2 \)'. If the constant \( j \), of type \( C_1 \), represents the proper name 'John', then the formula \( \forall x A(j, x) \) can be paraphrased by 'John admires everybody'. The extension suggested here gives a meaning to the quantified expression \( \forall y_2 A(y_2) \), of type \( \langle c_1, t \rangle \), and ensures the semantic equivalence

\[ (\forall y_2 A(y_2))(t_1) \approx \forall y_2 A(t_1, y_2). \]

In our example, we thus allow the intransitive verb phrase 'admire everybody' to combine directly with the subject 'John' so as to produce the considered sentence.

In a similar manner, we can analyse the converse sentence 'everybody admires John' as a combination of the expression 'everybody admires' (of a somewhat unusual appearance) with the proper name 'John'. To that end we
introduce a two-place predicate constant $\bar{A}$, of type $\langle c_1, \langle c_2, t \rangle \rangle$, with the following equivalence:

$$\bar{A}(t_2, t_1) \approx A(t_1, t_2).$$

By use of the quantified expression $\forall y_1 \bar{A}(y_1)$, of type $\langle c_2, t \rangle$, we obtain the semantic equivalence

$$(\forall y_1 \bar{A}(y_1))(t_2) \approx \forall y_1 A(y_1, t_2).$$

The left-hand side can be paraphrased by the following natural language reading: 'everybody admires (John)'.

Let us go back to the general theory. For a variable $Y$ of type $c$ (arbitrary) and an expression $a$ of type $a$ (Boolean), the semantic values of the expressions $\forall Y a$ and $\exists Y a$ are defined in the following manner:

$$\{\forall Y a\}^g = \bigcap_{reD_c} \{a\}^{g_r},$$

$$\{\exists Y a\}^g = \bigcup_{reD_c} \{a\}^{g_r},$$

where $g$ is the considered assignment function (which assigns values to the variables), and $g'$ represents the $y$-variant of $g$ specified by $g(y) = r$. This is a natural extension of the classical definition (which corresponds to the case $a = t$).

We are mainly interested in the commutativity between the quantifiers and the composition operator. More precisely, we wish to have the semantic equivalence

$$(\forall Y a)(\beta) \approx \forall Y (a(\beta)),$$

for $a \in E_{\langle b, a \rangle}$ and $\beta \in E_b$, where $a$ is a Boolean type and $b$ is an arbitrary type. (The existential quantifier can be treated in the same way, in view of the duality property $\exists Y a \approx -\forall Y (-a)$.)

The commutativity in question is easily seen to be satisfied under the following restrictive assumption: the expression $\beta$ contains no free occurrence of the quantified variable $y$. The argument is basically the same as in the two preceding paragraphs (about connectives and modalities); it also uses the second Boolean identity given in Sec. 3.1. The equivalence we are interested
An extension of the Montague semantics

in can be written as follows:

\[ \bigcap_{r \in D_e} \llbracket \alpha \rrbracket^e(\llbracket \beta \rrbracket^e) = \bigcap_{r \in D_e} \llbracket \alpha \rrbracket^e(\llbracket \beta \rrbracket^e). \]

Under our assumption concerning \( \beta \), this equality is obviously satisfied, since we have \( \llbracket \beta \rrbracket^e = \llbracket \beta \rrbracket^e \) for every value \( r \) belonging to the set \( D_e \).

Analogously, the extension operator can be shown to commute with the quantifiers. If \( \alpha \) is an expression of type \( \langle s, a \rangle \), then we have the semantic equivalence

\[ \forall y (\alpha) \approx \forall y (\neg (\forall x) \alpha). \]

Let us finally mention the commutativity properties concerning the lambda operator and the intension operator. For any expression \( \alpha \) of Boolean type, we have the following equivalences:

\[ \lambda x [\forall y \alpha] \approx \forall y (\lambda x [\alpha]), \]
\[ \neg (\forall y \alpha) \approx \forall y (\neg \alpha). \]

4. Application to natural language representation

In the preceding section we introduced three ideas of enrichment of Montague's intensional logic; they are concerned with the way of using the logical connectives, the modal operators, and the quantifiers. This section aims at demonstrating the effect of the first two ideas upon the formalization of natural language. In particular, we will compare the representation of a fragment of natural language by means of the extended intensional logic with the representation based on the classical intensional logic. To make this comparison we will follow the same pattern as in the preceding section; we will successively examine the use of the logical connectives and of the modal operators in the formalization of natural language.

4.1. Logical connectives

In natural language, the connectives 'not', 'and', 'or', 'if ... then', 'if and only if' are informal translations of the logical connectives \( \neg, \land, \lor, \Rightarrow, \equiv \). Now, in classical intensional logic, these connectives can only affect expressions of type \( t \), i.e. formulae. Remembering that the sentences of natural language
are precisely those expressions which are translated into logical formulae, we come to the conclusion that using the logical connectives as the translations of the natural connectives allows us only to translate logical combinations of sentences (and of no other kind of natural expressions) in the logic language. In more precise terms, if we use the symbol $\Rightarrow$ to represent the translation operation (from natural language to formal language), we can write the following schema:

$$(\text{sentence})_1 \Rightarrow (\text{formula})_1$$
$$(\text{sentence})_2 \Rightarrow (\text{formula})_2$$
$$(\text{sentence})_1 \text{ and } (\text{sentence})_2 \Rightarrow (\text{formula})_1 \land (\text{formula})_2$$
$$(\text{sentence})_1 \text{ or } (\text{sentence})_2 \Rightarrow (\text{formula})_1 \lor (\text{formula})_2$$

Natural language constructions not only use logical combinations of sentences, but also logical combinations of expressions belonging to various other syntactic categories. Let us for example mention the combinations ‘John and Paul’, ‘work and talk’, ‘rich and happy’. It would be interesting to be able to translate all these logical combinations of natural expressions by means of a translation schema that would be identical (or nearly identical) with the simple schema indicated above for the translation of sentences. Now, for the reason that we mentioned, we cannot use this schema by merely replacing ‘sentence’ by ‘natural expression’ and ‘formula’ by ‘logical expression’. However, Montague’s intensional logic allows one to represent logical combinations of certain natural expressions, but this requires a different way of translating the natural connectives ‘and’, ‘or’ into logic. According to the Montague formulation, the logic translation of the natural connectives depends on the syntactic category of the expressions involved in the combinations. In the initial form of Montague’s grammar, such a translation is defined for the syntactic categories of terms ($T$) and of intransitive verbs ($IV$). The definition was recently extended by Hendriks, first to the syntactic category of transitive verbs ($TV$) and then to any syntactic category. The principle of this method is given hereunder.

As a translation of a natural conjunction or disjunction, Hendriks proposes a certain lambda expression, the form of which is determined by the type of the logical expression it acts on. Assume that the connectives are used to combine natural expressions whose logical translations are expressions of type $\langle a_1, a_2, \ldots, a_n, t \rangle$ (which we simply write as $\langle a, t \rangle$).

Let $x_{a_i}$ be a variable of type $a_i$, for $i = 1, \ldots, n$. The translation of the
conjunction and of the disjunction, involving natural expressions $A$ and $B$ whose logical translations $A'$ and $B'$ are of type $\langle d, t \rangle$, is defined as follows:

$$(A \text{ and } B) \Rightarrow \lambda x_a.\left[\ldots \left[\lambda x_{a_n}.\left[A'(x_{a_1}) \ldots (x_{a_n}) \land B'(x_{a_1}) \ldots (x_{a_n})\right]\right]\ldots \right]$$

$$(A \text{ or } B) \Rightarrow \lambda x_a.\left[\ldots \left[\lambda x_{a_n}.\left[A'(x_{a_1}) \ldots (x_{a_n}) \lor B'(x_{a_1}) \ldots (x_{a_n})\right]\right]\ldots \right]$$

In this kind of translation, the expressions $A'$ and $B'$, of type $\langle d, t \rangle$, are temporarily transformed into expressions $A'(x_{a_1}) \ldots (x_{a_n})$ and $B'(x_{a_1}) \ldots (x_{a_n})$, of type $t$; since these expressions are formulae, it is possible to apply the logical connectives $\land$ and $\lor$ to them so as to produce some new formulae. Then, the formulae thus obtained are transformed, by means of lambda operators, into expressions of type $\langle d, t \rangle$ which are the intensional logic translations of the phrases 'A and B' and 'A or B'. In summary, the Hendriks method amounts to first transforming arbitrary logical expressions (of Boolean type) into formulae, which can be combined by means of the logical connectives, and then transforming the resulting formula into a logical expression of the initial type. This is seen to require introducing a rather complex mechanism for the translation of the natural language connectives into formal logic. Furthermore, the form of the translation depends on the syntactic category of the expressions affected by the connectives.

The language of the extended intensional logic, introduced in Sec. 3.2, often allows us to preserve the simplest translation for the natural language connectives: 'and' and 'or' are directly translated into the logical connectives $\land$ and $\lor$, whatever the syntactic category of the expressions involved in the combination. This way of solving the problem is near to the method devised by Keenan and Faltz in the framework of a formal language which is strictly propositional (and, therefore, significantly less expressive than Montague's intensional language).

Let us illustrate and substantiate the statements above by way of three examples. The first of them aims at demonstrating the translation of a combination of terms (proper names or noun phrases) by means of natural connectives. The formalization of this example within the intensional language is based on the semantic equivalence

$$(\alpha_1 \land \alpha_2)(\beta) \approx \alpha_1(\beta) \land \alpha_2(\beta)$$

established in Sec. 3.2. When applied to our example, this equivalence allows us to combine logically two proper names, and then to combine functionally the result with an intransitive verb so as to produce a sentence. We will analyse
this example (and the other ones) by means of the Montague–Hendriks translation schema, and by means of the translation schema associated with the extended intensional logic.

We will use the symbol ⇒ to represent the meta-expression ‘is translated into’. When it is necessary to distinguish between the rules of the strict intensional logic and those of the extended intensional logic, we use the translation symbol ⇒ in the first case (translation based on Montague’s rules) and the translation symbol ⇒ in the second case (translation based on the rules proper to the extended intensional language).

Example 1: John and Mary talk

Translation of the vocabulary words:

John ⇒ \( \lambda P[\neg P(j)] \)
Mary ⇒ \( \lambda Q[\neg Q(m)] \)
Talk \( ⇒ talk' \)

Schema of Montague’s intensional logic:

\[ (A \text{ and } B); A, B \in T \quad \Rightarrow \lambda x[A'(x) \land B'(x)] \]

\[ (\text{John and Mary}) \quad \Rightarrow \lambda x[\lambda P[\neg P(j)](x) \land \lambda Q[\neg Q(m)](x)] \]

\[ (\text{John and Mary talk}) \quad \Rightarrow (\lambda x[\lambda P[\neg P(j)](x) \land \lambda Q[\neg Q(m)](x)])(\neg \text{'talk'}) \]

\[ \approx \lambda P[\neg P(j)](\neg \text{'talk'}) \land \lambda Q[\neg Q(m)](\neg \text{'talk'}) \]

\[ \approx \text{'talk'(j) \land talk'(m)} \]

Schema of the extended intensional logic:

\[ (A \text{ and } B) \quad \Rightarrow \quad A' \land B' \]

\[ (\text{John and Mary}) \quad \Rightarrow \lambda P[\neg P(j)] \land \lambda Q[\neg Q(m)] \]

\[ (\text{John and Mary talk}) \quad \Rightarrow (\lambda P[\neg P(j)] \land \lambda Q[\neg Q(m)])(\neg \text{'talk'}) \]

\[ \approx \lambda P[\neg P(j)](\neg \text{'talk'}) \land \lambda Q[\neg Q(m)](\neg \text{'talk'}) \]

\[ \approx \text{'talk'(j) \land talk'(m)} \]

The second and third examples are concerned with the formalization of logical combinations of verb phrases. In the framework of the extended intensional logic, we can make use, in this case, of the semantic equivalence

\[ \beta(\neg (\alpha_1 \land \alpha_2)) \approx \beta(\neg \alpha_1) \land \beta(\neg \alpha_2) \]

also established in Sec. 3.2. As in the first example, we give first the Montague–Hendriks version and then the extended intensional logic version.
An extension of the Montague semantics

- Example 2: John talks and works
  Translation of the vocabulary words:
  
  work ⇒ work'

  Schema of Montague’s intensional logic:
  
  (A and B); A, B ∈ IV  \[ \Rightarrow \lambda x [A'(x) \land B'(x)] \]
  (talk and work) \[ \Rightarrow \lambda x [\text{talk}'(x) \land \text{work}'(x)] \]
  (John talks and works) \[ \Rightarrow \lambda P[\neg P(j)](\neg \lambda x [\text{talk}'(x) \land \text{work}'(x)]) \]
  \[ \approx \lambda x [\text{talk}'(x) \land \text{work}'(x)](j) \]
  \[ \approx \text{talk}'(j) \land \text{work}'(j) \]

  Schema of the extended intensional logic:
  
  (A and B) \[ \Rightarrow A' \land B' \]
  (talk and work) \[ \Rightarrow \text{talk}' \land \text{work}' \]
  (John talks and works) \[ \Rightarrow \lambda P[\neg P(j)](\neg (\text{talk}' \land \text{work}')) \]
  \[ \approx (\text{talk}' \land \text{work}')(j) \]
  \[ \approx \text{talk}'(j) \land \text{work}'(j) \]

- Example 3: John examines and solves a problem
  Translation of the vocabulary words and of the compound expressions:
  
  examine ⇒ exm'
  solve ⇒ solv'
  a ⇒ \lambda Z[\lambda Y[\exists x(\neg Z(x) \land Y(x))]](\neg \text{prob}')
  problem ⇒ \text{prob}'
  (a problem) ⇒ \lambda Z[\lambda Y[\exists x(\neg Z(x) \land Y(x))]](\neg \text{prob}')
  \approx \lambda Y[\exists x(\text{prob}'(x) \land Y(x))]

  Schema of Montague’s intensional logic:
  
  (A and B); A, B ∈ TV  \[ \Rightarrow \lambda \nu[\lambda w [A'(v)(w) \land B'(v)(w)]] \]
  (examine and solve) \[ \Rightarrow \lambda \nu[\lambda w [\text{exm}'(v)(w) \land \text{solv}'(v)(w)]] \]
  (examine and solve a problem) \[ \Rightarrow \lambda \nu[\lambda w [\text{exm}'(v)(w) \land \text{solv}'(v)(w)]] \]
  \[ (\neg \lambda Y[\exists x(\text{prob}'(x) \land Y(x)))] \]
  \[ \approx \lambda w [\text{exm}'(\neg \lambda Y[\exists x(\text{prob}'(x) \land Y(x))])(w) \land \text{solv}'(\neg \lambda Y[\exists x(\text{prob}'(x) \land Y(x))])(w)] \]
John examines and solves a problem

\[ \frac{1}{\lambda P[\overline{P}(j)](\overline{\lambda w}
\begin{align*}
& \text{exm'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]}) \\wedge \text{solv'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]}) \approx \lambda w[\text{exm'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]}) \\wedge \text{solv'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]})]) \\wedge \text{solv'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]})]) \approx \text{exm'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]}) \wedge \text{solv'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]})](j)
\end{align*}
\]

\begin{align*}
\text{exm'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]})
\end{align*}

\begin{align*}
\text{solv'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]})
\end{align*}

\begin{align*}
\text{exm'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]}) \wedge \text{solv'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]})]
\end{align*}

\begin{align*}
\text{solv'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]})](j)
\end{align*}

\begin{align*}
\text{solv'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]})](j)
\end{align*}

\begin{align*}
\text{solv'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]})]
\end{align*}

\begin{align*}
\text{solv'}(\overline{\lambda Y[\exists x(pro'b'(x) \land \overline{Y}(x))]})](j)
\end{align*}

The case of negation in natural language is the subject of another interesting application. In the framework of the extended intensional language, the negation connective \((-\cdot\)) can act on any logical expression of Boolean type (not only on a formula as in the classical Montague language). Consider, for example, the negated forms of the verbs. The sentence ‘John does not talk’ can be analysed syntactically as ‘(John) (does not) (talk)’, which prompts us to translate the phrase ‘does not’ into the connective \((\cdot\cdot\)) and to apply this directly to the translation of the verb ‘talk’. The detailed formalization can be presented as follows.

• Example 4: John does not talk

\begin{align*}
\text{John} & \Rightarrow \lambda P[\overline{P}(j)]
\end{align*}

\begin{align*}
\text{does not} & \Rightarrow \overline{\cdot}
\end{align*}

\begin{align*}
\text{talk} & \Rightarrow \text{talk'}
\end{align*}

\begin{align*}
\text{does not talk} & \Rightarrow \overline{\cdot} \text{talk'}
\end{align*}

\begin{align*}
\text{John does not talk} & \Rightarrow \lambda P[\overline{P}(j)](\overline{\cdot} \text{talk'}(j))
\end{align*}

\begin{align*}
\approx (\overline{\cdot} \text{talk'}(j))
\end{align*}

\begin{align*}
\approx \overline{\cdot} \text{talk'}(j)
\end{align*}
An extension of the Montague semantics

4.2. Modal operators

In usual speech, we often talk about possibilities, hypothetical events, goals to be achieved, forecasts regarding the future, and so on. In natural language, the modes ‘necessary’, ‘possible’ and ‘permitted’ are expressed through auxiliary verbs such as ‘must’ and ‘may’. The temporal modalities are expressed by means of the conjugation tenses, such as future, perfect, imperfect and future perfect. The possibility, the necessity, the future, the past, the belief, the knowledge are modalities that can affect the sentences of natural language (and, more generally, the expressions of natural language) so as to modify their meaning. The sentences ‘John works’, ‘John must work’, ‘John will work’, ‘John will have to work’ have different meanings, since their verb ‘work’ is affected, in four different ways, by the modalities of the necessity (‘must’, ‘have to’) and of the future (‘will’).

The researchers working in the field of classical logic (propositional logic and first-order predicate logic) observed the inability of this formalism to represent the sentences and the expressions of natural language that are affected by modalities. The study of this limitation of the classical logic formalism led to the development of some other logic systems, which are generally referred to as non-classical logics. Modal logics belong to this family; their name originates from the fact that these logic systems comprise modal operators that can affect the formulae so as to modify their interpretation. For example, in statements such as ‘It is possible that $F$’, ‘It will probably be true in the future that $F$’, ‘John thinks that $F$’, ‘It is often true that $F$’, the expressions written in front of the logical formula $F$ are formalized by means of modal operators. The truth value of such statements depends not only on the truth value of the given formula $F$; it depends also on the time instant when this formula is stated (temporal logics), on the person who thinks or believes that $F$ (logics of belief), or on the necessity, possibility or randomness of an event (logics of possibility). In the framework of the extended intensional logic introduced in Sec. 3, the modalities can affect not only a formula $F$, but also any well-formed expression $a$ (of Boolean type).

Let us now examine the question of the translation of natural modalities into the extended intensional logic. A modality of natural language is treated as an element of the syntactic category $\mathcal{C}/\mathcal{C}$ if it combines with an element of the syntactic category $\mathcal{C}$ to produce a new element of the same syntactic category $\mathcal{C}$.

To satisfy the compositionality principle in the interpretation of the phrase ‘Mod $A$’ (where $A$ represents a natural expression and Mod represents a natural modality) we have to apply the translation of Mod to a logical expression which denotes the intension of the translation $A'$ of $A$ (we cannot apply it to
The translation of $\text{Mod}$ is expressed in terms of the modal operator $[M]$ (‘universal’ case) or of its dual $\langle M \rangle$ (‘existential’ case). In particular, the notation $\Box$ and $\Diamond$ is used for the necessity and possibility operators.

Let $p$ be a variable of intensional type, i.e. of type $\langle s, a \rangle$ for a suitable type $a$. The translation rule alluded to above is the following:

<table>
<thead>
<tr>
<th>Natural expression</th>
<th>Formal expression</th>
<th>Associated type</th>
</tr>
</thead>
<tbody>
<tr>
<td>universal modality</td>
<td>$\lambda p[[M] \neg p]$</td>
<td>$\langle \langle s, a \rangle, a \rangle$</td>
</tr>
<tr>
<td>existential modality</td>
<td>$\lambda p[\langle M \rangle \neg p]$</td>
<td>$\langle \langle s, a \rangle, a \rangle$</td>
</tr>
</tbody>
</table>

Note that these translations generalize the translation given by Montague in the special case where the type $a$ is $t$ and where the modality $[M]$ is the necessity $\Box$:

Necessarily $\Rightarrow \lambda p[\Box \neg p] \quad \langle \langle s, t \rangle, t \rangle$

It is then possible to combine functionally a modality with an expression $x$, of type $a$, prefixed by the intension operator; this functional combination is easily seen to satisfy the following semantic equivalences:

\[
\lambda p[[M] \neg p](\neg x) \approx [M]x, \\
\lambda p[\langle M \rangle \neg p](\neg x) \approx \langle M \rangle x.
\]

As a first example, we examine the sentence ‘John will talk’. Here, we have a temporal modality, represented by $\langle F \rangle$, the meaning of which is ‘at least once in the future’. Since the modality is applied to the intransitive verb ‘talk’ (syntactic category $IV$), the type of the corresponding variable $p$ is $\langle s, \langle \langle s, e \rangle, t \rangle \rangle$.

- Example 5: John will talk

  John $\Rightarrow \lambda Q[\neg Q(j)]$
  will $\Rightarrow \lambda p[\langle F \rangle \neg p]$
  talk $\Rightarrow \langle \langle F \rangle \neg p \rangle(talk')$
  (will talk) $\approx \langle \langle F \rangle \langle F \rangle(talk') \rangle$
  (John will talk) $\Rightarrow \lambda Q[\neg Q(j)](\langle \langle F \rangle(talk') \rangle(j))$
  $\approx \langle\langle F \rangle(talk'(j))\rangle$
This example demonstrates the usefulness of one of the generalizations proposed in the framework of the extended intensional logic. Roughly speaking, the fact that a modal operator can affect any logical expression (here, the translation of an intransitive verb) allows us to 'conjugate' the logic translation of a verb. According to Montague's classical analysis, prior to any translation, a sentence in the future tense such as 'John will talk' first has to be transformed into the statement 'In the future John talks'. In the translation of this version, the modal operator \( \langle F \rangle \) acts on the logical formula that represents the sentence 'John talks'. If we use the extended intensional logic, then we can make the formal representation of the future modality act directly on the translation of the verb. Note that the last expression occurring in the example above, i.e. the expression \( \langle F \rangle(talk'(j)) \), is obtained from the preceding expression via the semantic equivalence

\[
\langle M \rangle x(\beta) \approx \langle M \rangle (x(\beta))
\]

established in Sec. 3.2. This property ensures the equivalence between the formulae \( \langle F \rangle(talk')(j) \) and \( \langle F \rangle(talk'(j)) \), which are the respective translations of the two formulations '(In the future) (John talks)' and 'John ((in the future) (talks))' of our sentence 'John will talk'. For the result to be correct, it is necessary that the constant \( j \) be of the type \( (s, e) \) (intensional) and not of the type \( e \); thus, we have to use the Montague version of the translation function and not the Bennett version.

The schema that we have used to analyse the preceding example can also be applied to any other modality acting on a verb. For instance, the sentence 'John must talk' can be translated as follows:

- **Example 6**: John must talk

  \[
  \text{must} \implies \lambda p[\square \neg p] \\
  (\text{John must talk}) \implies (\square(talk'))(j) \\
  \approx \square(talk'(j))
  \]

In the last two examples above, the auxiliary verbs 'will' and 'must' modify the meaning of the verb ('talk') that follows them in the two sentences; these auxiliary verbs act as modifiers of the meaning. In fact, the modifiers can act not only on verbs, but also on expressions belonging to various other syntactic categories. The adjectives, acting on common nouns, are typical examples of modifiers. It is worth emphasizing a distinction between adjectives such as 'former' or 'alleged' (in the noun phrases 'the former president' or 'an alleged proof', for instance), which deeply alter the meaning of the noun they refer
to, and adjectives such as 'big' or 'nice' (in the noun phrases 'a big house' or 'a nice picture', for instance), which express a mere quality of the noun. The formalism of the extended intensional language, which admits modalities acting on logical expressions of any type, enables us to translate the modifiers in a syncategorematic manner. The example below is concerned with the analysis of a sentence in which the adjective 'difficult' modifies the common noun 'problem' that follows it.

- **Example 7:** John will examine a difficult problem

\[
\text{difficult} \\ \Rightarrow \lambda q[\langle D \rangle \sim q] \\
(\text{difficult problem}) \Rightarrow \lambda q[\langle D \rangle \sim q](\sim \text{prob'}) \\
\approx \langle D \rangle \text{prob'} \\
(\text{a difficult problem}) \Rightarrow \lambda \lambda y[\exists x(\sim Z(x) \land \sim Y(x))](\langle D \rangle \text{prob'}) \\
\approx \lambda y[\exists x(\langle D \rangle \text{prob'}(x) \land Y(x))] \\
\approx \lambda y[x] \quad (\text{abbreviation})
\]

(examine
a difficult problem) \Rightarrow exm'(\lambda y[x])
(will examine
a difficult problem) \Rightarrow \lambda p[\langle F \rangle \sim p](\sim \text{exm'}(\lambda y[x])) \\
\approx \langle F \rangle \text{exm'}(\lambda y[x])
(John will examine
a difficult problem) \Rightarrow \lambda q[\sim Q(j)](\langle F \rangle \text{exm'}(\lambda y[x])) \\
\approx (\langle F \rangle \text{exm'}(\lambda y[x])(j)) \\
\approx \langle F \rangle (\text{exm'}(\lambda y[x])(j))$

Finally, let us give the formal analysis of a sentence that contains both a negation connective and a modality.

- **Example 8:** John will not talk

\[
\text{John} \Rightarrow \lambda q[\sim Q(j)] \\
\text{will} \Rightarrow \lambda p[\langle F \rangle \sim p] \\
\text{not} \Rightarrow \sim \\
\text{talk} \Rightarrow \text{talk'} \\
(\text{not talk}) \Rightarrow \sim \text{talk'} \\
(\text{will not talk}) \Rightarrow \lambda p[\langle F \rangle \sim p](\sim \text{talk'}) \\
\approx \langle F \rangle \sim \text{talk'} \\
(\text{John will not talk}) \Rightarrow \lambda q[\sim Q(j)](\langle F \rangle \sim \text{talk'}) \\
\approx (\langle F \rangle \sim \text{talk'})(j) \\
\approx \langle F \rangle \sim \text{talk'}(j)
\]
An extension of the Montague semantics

REFERENCES

4) H. Hendriks, Flexible Montague grammar, First European Summer School on Natural Language Processing, Logic and Knowledge Representation, Groningen, 1989.

Authors

P. Delcsarte: Ir. degree (electrical engineering and applied mathematics), University of Louvain, 1965 and 1966; Dr. degree (applied sciences), University of Louvain, 1973; Philips Research Laboratory Brussels, 1966– . His research interests include algebraic coding theory, combinatorial mathematics, the theory and applications of orthogonal polynomials and Toeplitz matrices, and formal semantics for natural language.

A. Thayse: Ir. degree (electrical engineering and applied mathematics), University of Louvain, 1964 and 1965; Dr. degree (applied sciences), Ecole Polytechnique Fédérale de Lausanne, Switzerland, 1980; Philips Research Laboratory Brussels, 1967– ; Visiting Professor at the Ecole Polytechnique Fédérale de Lausanne, 1983; Visiting Professor at the University of Louvain, 1986–1990. His research interests include logic and its application to natural language representation, speech recognition and artificial intelligence.