Introduction to Formal Semantics and Compositionality

Barbara H. Partee, University of Massachusetts, Amherst
Vladimir Borschev, VINITI, Russian Academy of Sciences, and Univ. of Massachusetts, Amherst
partee@linguist.umass.edu, borschev@linguist.umass.edu; http://people.umass.edu/partee/
Universidad Rovira i Virgili, Tarragona, April 13, 2005

1. Compositional Semantics

1.1. The Principle of Compositionality.

A basic starting point of generative grammar: there are infinitely many sentences in any natural language, and the brain is finite, so linguistic competence must involve some finitely describable means for specifying an infinite class of sentences. That is a central task of syntax.

Semantics: A speaker of a language knows the meanings of those infinitely many sentences, is able to understand a sentence he/she has never heard before or to express a meaning he/she has never expressed before. So for semantics also there must be a finite way to specify the meanings of the infinite set of sentences of any natural language.

A central principle of formal semantics is that the relation between syntax and semantics is compositional.

The Principle of Compositionality: The meaning of an expression is a function of the meanings of its parts and of the way they are syntactically combined.

Each of the key terms in the principle of compositionality is a “theory-dependent” term, and there are as many different versions of the principle as there are ways of specifying those terms. (function, meaning, parts (syntax))

Some of the different kinds of things meanings could be in a compositional framework:

(a) Representations in a “language of thought” or “conceptual representation” (Jackendoff, Jerry Fodor): if semantics is treated in terms of representations, then semantic composition becomes a matter of compositional translation from a syntactic representation to a semantic representation.

(b) The logic tradition: Frege, Tarski, Carnap, Montague. The basic meaning of a sentence is its truth-conditions: to know the meaning of a sentence is to know what the world must be like if the sentence is true. Knowing the meaning of a sentence does not require knowing whether the sentence is in fact true; it only requires being able to discriminate between situations in which the sentence is true and situations in which the sentence is false.

Starting from the idea that the meaning of a sentence consists of its truth-conditions, meanings of other kinds of expressions are analyzed in terms of their contribution to the truth-conditions of the sentences in which they occur.


In formal semantics, truth-conditions are expressed in terms of truth relative to various parameters — a formula may be true at a given time, in a given possible world, relative to a certain context that fixes speaker, addressee, etc., and relative to a certain assignment of meanings to its atomic “lexical” expressions and of particular values to its variables. For simple formal languages, all of the relevant variation except for assignment of values to variables is incorporated in the notion of truth relative to a model. Semantics which is based on truth-conditions is called model-theoretic.

Compositionality in the Montague Grammar tradition:

The task of a semantics for language L is to provide truth conditions for every well-formed sentence of L, and to do so in a compositional way. This task requires providing appropriate model-theoretic interpretations for the parts of the sentence, including the lexical items. The task of a syntax for language L is (a) to specify the set of well-formed expressions of L (of every category, not only sentences), and (b) to do so in a way which supports a compositional semantics. The syntactic part-whole structure must provide a basis for semantic rules that specify the meaning of a whole as a function of the meanings of its parts.

Basic structure in classic Montague grammar:

(1) Syntactic categories and semantic “types”: For each syntactic category there must be a uniform semantic type. For example, one could hypothesize that sentences express propositions, nouns and adjectives express properties of entities, verbs express properties of events.

(2) Basic (lexical) expressions and their interpretation. Some syntactic categories include basic expressions; for each such expression, the semantics must assign an interpretation of the appropriate type. Within the tradition of formal semantics, most lexical meanings are left unanalyzed and treated as if primitive; Montague regarded most aspects of the analysis of lexical meaning as an empirical rather than formal matter; formal semantics is concerned with the types of lexical meanings and with certain aspects of lexical meaning that interact directly with compositional semantics, such as verbal aspect.

(3) Syntactic and semantic rules. Syntactic and semantic rules come in pairs: «Syntactic Rule n, Semantic Rule n»: in this sense compositional semantics concerns “the semantics of syntax”.

Syntactic Rule n: If α is an expression of category A and β is an expression of category B, then F(α, β) is an expression of category C. [where F is some syntactic operation on expressions]

Semantic Rule n: If α is interpreted as α’ and β is interpreted as β’, then F(α, β) is interpreted as G_ι(α’, β’). [where G_ι is some semantic operation on semantic interpretations]

Illustration: See syntax and semantics of predicate calculus in Section 3.
2. Linguistic Examples.

2.1. The structure of NPs with restrictive relative clauses.

Consider NPs such as “the boy who loves Mary”, “every student who dances”, “the doctor who treated Mary”, “no computer which uses Windows”. Each of these NPs has 3 parts: a determiner (DET), a common noun (CN), and a relative clause (RC). The question is: Are there semantic reasons for choosing among three different possible syntactic structures for these NPs?

a. Flat structure:

```
NP
DET  CN  RC
```

b. “NP - RC” structure: The relative clause combines with a complete NP to form a new NP.

```
NP
DET  CN  RC
```

c. “CNP - RC” structure: (CNP: common noun phrase: common noun plus modifiers)

```
NP
DET  CNP
|     |      |
| CN  |      |
|     |      |
| CN  |      |
|     |      |
| CN  |      |
|      |      |
```

Argument (Partee 1973): we can argue that compositionality requires the third structure: that “boy who loves Mary” forms a semantic constituent with which the meaning of the DET combines. The first structure does not allow for recursivity, and the second structure cannot be interpreted compositionally. (The second structure is a good structure to provide a basis for a compositional interpretation for non-restrictive relative clauses (Rodman 1976).)

2.2. Phrasal and sentential conjunction.

Consider the following equivalent and non-equivalent pairs, where the first sentence has phrasal conjunction (VP-conjunction, in particular) and the second has sentential conjunction (S-conjunction). The puzzle is to explain why some examples are semantically equivalent and some are not, although in each case the surface syntactic relation is the same.

```
John sings and dances    =    John sings and John dances
One boy sings and dances ≠ One boy sings and one boy dances
Every boy sings and dances = Every boy sings and every boy dances
No boy sings and dances   ≠ No boy sings and no boy dances
```

We will need two parts to solve this puzzle: (i) the syntax and semantics of sentential and phrasal conjunction, particularly the question of how they are related; and (ii) the semantics of the Determiners one, every, no (and others), as well as of simple NPs like John. In Lecture 3 we will present the solution of Partee and Rooth (1983), which builds on Montague’s treatment of the semantics of Determiners and generalizes his treatment of conjunction along lines suggested by Gazdar (1980) and by Keenan and Faltz (1978, 1985).

3. Formal Semantics in Logic and Linguistics

3.1. English as a Formal Language.

R. Montague 1970, “English as a Formal Language” argued that the syntax and semantics of natural languages could be treated by the same kinds of techniques used by logicians to specify the syntax and model theoretic semantics of formal languages such as the predicate calculus.

3.2. Example. Syntax and semantics of the predicate calculus (PC).

Predicate Calculus is the most well known and in a sense the prototypical example of a formal language. We assume that it is familiar to this audience, and mention some of its relevant properties here only to demonstrate features of formal languages which are most important for us: the notions of model and model-theoretic semantics, and the Principle of Compositionality. We limit ourselves here to some examples and remarks. More exact definitions are given in Appendix 1.

Formulas and other expressions of PC are built from individual constants (or simply “constants”), (individual) variables, predicate constants (or predicate symbols), logical connectives and quantifiers. Each expression belongs to a certain type. The type structure of PC is very simple: individuals, relations of different arities (unary, binary, etc.), and truth-values.

In our examples we use the following expressions:

```
Expressions  | Syntactic categories  | Semantic Types
---------------------------------------------------------
John, Mary   | (individual) constant | individuals
x            | variable              | individuals
happy        | unary predicate constant | unary relations
love         | binary predicate constant | binary relations
love (John, Mary) | {                         |
love(Mary, x) | }                         |
\forall x (happy(Mary, x) \rightarrow happy(x)) | {                      |
```

1 “I reject the contention that an important theoretical difference exists between formal and natural languages. ... In the present paper I shall accordingly present a precise treatment, culminating in a theory of truth, of a formal language that I believe may reasonably be regarded as a fragment of ordinary English. ... The treatment given here will be found to resemble the usual syntax and model theory (or semantics) [due to Tarski] of the predicate calculus, but leans rather heavily on the intuitive aspects of certain recent developments in intensional logic [due to Montague himself].” (Montague 1970b, p.188 in Montague 1974)
Expressions are interpreted in models. The structure common to all of the models in which a given language is interpreted (the model structure for the model-theoretic interpretation of the given language) reflects certain basic presuppositions about the “structure of the world” that are implicit in the language. For PC, any given model structure consists of the set of truth-values \{0,1\}, a domain D which is some set of objects (or entities), and some n-ary relations on this set.

A model, or interpreted model, consists of a model structure plus a (“lexical”, “basic”) interpretation function I which assigns semantic values to all constants.

\[ M = \langle D, I \rangle \]

An interpretation \( \| \cdot \|_M \) is built up recursively on the basis of the basic interpretation function I, assigns to every expression \( \alpha \) its semantic value \( \| \alpha \|_M \) in a given model M. (More precisely, \( \| \cdot \|_M \).) These semantic values must correspond to the types of the expressions. Thus, in our examples to the individual constants John and Mary are assigned certain objects, individual variables take their values in the set of objects (entities), to the predicate constant love is assigned a binary relation \( \| \text{love} \|_M \), and to the predicate constant happy, a unary relation (property) \( \| \text{happy} \|_M \). Formulas receive truth values. The formula love (John, Mary) is true in the model M if the pair of objects corresponding to the constants John and Mary belongs to the relation \( \| \text{love} \|_M \).

We state the truth conditions for the formula \( \forall x(\text{love}(x, y) \rightarrow \text{happy}(x)) \) with respect to a model M and an assignment g:

\[ \| \forall x(\text{love}(x, y) \rightarrow \text{happy}(x)) \|_M^g = 1 \text{ iff:} \]

for each d in D,
if \( \| y \|_M^g = d \), then \( \| x \|_M^g \in \| \text{happy} \|_M^g \).

For each constant \( \alpha \), \( \| \alpha \|_M^g = I(\alpha) \). And \( \| x \|_M^g = g[d/x] = \| \text{happy} \|_M^g \).

Equivalent to:
iff: for each d in D,
if \( \| I(\text{Mary}) \| \notin g \), then d \( \in \| \text{happy} \|_M^g \).

The semantics of PC illustrates the Principle of Compositionality.

As we know the infinite set of formulas of PC are built from terms (individual variables and constants) and predicate symbols by recursive syntactic rules (rules R1—R8 in Appendix 1). The semantics of these formulas – their interpretation in every given model – is defined by semantic rules S1 – S8, which correspond in a direct way to the syntactic rules. The semantics of the whole is based on the semantics of parts by means of this pairing of semantic interpretation rules with syntactic formation rules. This is a very important feature of every formal language – The Principle of Compositionality – and it is natural to think that this principle holds also for natural language.

3.3. The inadequacy of 1st-order predicate logic for natural language semantics.

What is the interpretation of “every student”? There is no appropriate syntactic category or semantic type in predicate logic. Inadequacy of 1st-order predicate logic for representing the semantic structure of natural language.

Categories of PC:

| Formula | - Sentence |
| Predicate | - Verb, Common Noun, Adjective |
| Term | - Proper Noun |
| Constant | - Pronoun (he, she, it) |
| Variable | - Determiner, Preposition, Prepositional Phrase, Adverb, … |

In the next lectures, we will see how a logic built on a richer type theory including the tools of the lambda-calculus can provide a richer formal semantics that can more adequately represent the structure of natural language semantics in a compositional way.

3.4. The typed lambda calculus.

The full version of the typed lambda calculus fits into Montague’s intensional logic with its type theory; see Appendix 2 for a complete statement of Montague’s intensional logic. The parts we will use will be the most will be the type theory, the lambda calculus (Rule 7), and the rule of “functional application” (Rule 6). Montague’s intensional logic includes the predicate calculus as a subpart (see Rule 2), but not restricted to first-order: we can quantify over variables of any type.

Lambda-abstraction, full version.

In general: \( \lambda \)-expressions denote functions.

\( \lambda \alpha \beta \) denotes a function whose argument is represented by the variable \( \alpha \) and whose value for any given value of \( \beta \) is specified by the expression \( \alpha \).

Example: \( \lambda x[x^2 + 1] \) denotes the function \( x \rightarrow x^2 + 1 \).

Function-argument application: \( \lambda f(x) \longrightarrow f[x/5] = 52 + 1 = 26 \)

\( \lambda \)-expressions provide explicit specification of the functions they name, unlike arbitrary names like \( f, g \). (The \( \lambda \)-calculus was invented by the logician Alonzo Church. The programming language LISP, invented by John McCarthy, was modelled on the \( \lambda \)-calculus.)

Syntactic and Semantic Rule: (a restatement of Syntactic and Semantic Rules 7 of IL)

R7*: If " \( \alpha \) is an expression of any type \( \alpha \) and \( \nu \) is a variable of type \( \beta \), then \( \lambda \alpha \nu \) is an expression of type \( \beta \rightarrow \alpha \) (the type of functions from \( \beta \)-type things to \( \alpha \)-type things.)

S7*: If \( \lambda \alpha \beta \) is a function of type \( \beta \rightarrow \alpha \) such that for any object \( \gamma \) of type \( \beta \), \( f[\alpha] = \beta[\alpha] \).

Lambda-conversion: A principle concerning the application of \( \lambda \)-expressions to arguments.

Examples: \( \lambda x[x^2 + 1](5) = 26 \)

\( \lambda y[\text{run}(x)](b) = \text{run}(b) \)

\( \lambda y[\text{walk}(y) \& \text{talk}(y)](m) = \text{walk}(m) \& \text{talk}(m) \)

\( \lambda \nu[\text{love}(x, m)](y) = \text{love}(y, m) \)

Lambda-conversion Rule: \( \lambda \nu[\beta] = \alpha' \), where \( \alpha' \) is like \( \alpha \) but with every free occurrence of \( \nu \) replaced by \( \beta \).

(Note: Occurrences of \( \nu \) that are free in \( \alpha \) are bound by \( \lambda \nu \) in \( \lambda \nu[\beta] \).)

One famous example of Montague’s application of the \(\lambda\)-calculus to the semantics of English was his treatment of English noun phrases as uniformly denoting generalized quantifiers. Below we indicate his semantics of English NPs by translations of English expressions into the \(\lambda\)-calculus. \(P\) is a variable ranging over sets, i.e. a predicate variable. (In full Montague grammar with intensionality, the analysis uses variables over properties; here we use simplified extensional types but we often talk as though \(P\) were a variable over properties.)

John \(\lambda P[P(j)]\)
John walks \(\lambda P[P(j)] \; (\text{walk} = \text{walk}(j))\)
every student \(\lambda P[\forall x \; (\text{student}(x) \rightarrow P(x))]\)
every student walks \(\lambda P[\forall x \; (\text{student}(x) \rightarrow P(x))] \; (\text{walk} = \forall x(\text{student}(x) \rightarrow \text{walk}(x))\)

We use the notation \(g\) for the semantic value of an expression the king \(\lambda \exists x \; (\text{king}(x) \rightarrow y = x) \; & \; P(x)\)

More on the semantics of NPs and their types, and type-shifting, in the next two lectures.

APPENDIX 1. Syntax and semantics of the predicate calculus (PC).

SYNTAX.

Syntactic Categories: terms (Term), 1-place predicates (Pred-1), 2-place predicates (Pred-2), ..., n-place predicates (Pred-n), formulas (Form).

Basic Expressions:

Basic Term(s):  (i) (individual) variables: \(x, y, z, x_1, y_1, z_1, x_2, ...\)

(ii) (individual) constants: \(a, b, a_1, \text{John}, \text{Mary}, ...

Basic Pred-1:  \(\text{run, walk, happy, calm, ...}\)

Basic Pred-2:  \(\text{love, kiss, like, see, ...}\)

Basic Form(ulas):  —  (none)

Syntactic Rules:

R1:  If \(P\) \in Pred-1 and \(T\) \in Term, then \(P(T)\) \in Form.
R2:  If \(R\) \in Pred-2 and \(T_1, T_2\) \in Term, then \(R(T_1, T_2)\) \in Form.

More general rule:  If \(R\) \in Pred-n and \(T_1, ..., T_n\) \in Term, then \(R(T_1, ..., T_n)\) \in Form.

R3:  If \(\varphi\) \in Form, then \(\neg\varphi\) \in Form.
R4:  If \(\varphi\) \in Form and \(\psi\) \in Form, then \(\varphi \; & \; \psi\) \in Form.
R5:  If \(\varphi\) \in Form and \(\psi\) \in Form, then \(\varphi \; \lor \; \psi\) \in Form.
R6:  If \(\varphi\) \in Form and \(\psi\) \in Form, then \(\varphi \; \rightarrow \; \psi\) \in Form.
R7:  If \(\varphi\) is a variable and \(\varphi\) \in Form, then \(\exists \varphi\) \in Form.
R8:  If \(\varphi\) is a variable and \(\varphi\) \in Form, then \(\forall \varphi\) \in Form.

SEMANTICS.

Model structure:

Domain \(D\) of entities (individuals)
Truth values \{True, False\} or \{0,1\}

I: Interpretation function which assigns semantic values to all constants (in Term and in Pred-1, Pred-2, ..., Pred-n)
\(M = \langle D, 1 \rangle\)
Set \(G\) of assignment functions \(g\), functions from variables to \(D\).

Semantic Types assigned to Syntactic Categories:

Term:  entities, individuals. The semantic values of this type are the members of \(D\).
Pred-1:  sets of entities. Semantic values of this type are members of \(\varphi(D)\).
Pred-2:  relations between entities (sets of pairs). Values: members of \(\varphi(Dx \times D)\).
Pred-n:  n-place relations; sets of n-tuples of entities. Values: members of \(\varphi(Dx \times \ldots \times D)\).

Form:  Truth values. Values: members of \{0,1\}.

Semantic interpretation relative to \(M, g\):

We use the notation \(\langle \varphi \rangle_M^g\) for the semantic value of an expression \(\varphi\) relative to \(M, g\).

Basic Expressions ("lexical semantics"):

A. If \(\alpha\) is a variable, then \(\langle \alpha \rangle_M^g = g(\alpha)\).
B. If \(\alpha\) is a constant, then \(\langle \alpha \rangle_M^g = 1(\alpha)\).

Semantic Rules ("semantics of syntax"):

S1:  If \(P\) \in Pred-1 and \(T\) \in Term, then \(\langle P(T) \rangle_M^g = 1\) iff \(\langle T \rangle_M^g \in \langle P \rangle_M^g\).
S2:  More general rule:  If \(R\) \in Pred-n and \(T_1, ..., T_n\) \in Term, then \(\langle R(T_1, ..., T_n) \rangle_M^g = 1\) iff \(<\langle T_1 \rangle_M^g, ..., \langle T_n \rangle_M^g> \in \langle R \rangle_M^g\).
S3:  If \(\varphi\) \in Form, then \(-\langle \varphi \rangle_M^g = 1\) iff \(\langle \varphi \rangle_M^g = 0\).
S4:  If \(\varphi, \psi\) \in Form, then \(\langle \varphi \; & \; \psi \rangle_M^g = 1\) iff \(\langle \varphi \rangle_M^g = 1\) and \(\langle \psi \rangle_M^g = 1\).
S5:  If \(\varphi, \psi\) \in Form, then \(\langle \varphi \; \lor \; \psi \rangle_M^g = 1\) iff \(\langle \varphi \rangle_M^g = 1\) or \(\langle \psi \rangle_M^g = 1\).
S6:  If \(\varphi, \psi\) \in Form, then \(\langle \varphi \; \rightarrow \; \psi \rangle_M^g = 1\) iff \(\langle \varphi \rangle_M^g = 0\) or \(\langle \psi \rangle_M^g = 1\).
S7:  If \(\varphi\) is a variable and \(\varphi\) \in Form, then \(\langle \exists \varphi \rangle_M^g = 1\) iff for all \(d \in D\), \(\langle \varphi \rangle_M^{d(d)} = 1\).
S8:  If \(\varphi\) is a variable and \(\varphi\) \in Form, then \(\langle \forall \varphi \rangle_M^g = 1\) iff there is a \(d \in D\) such that \(\langle \varphi \rangle_M^{d(d)} = 1\).

[The notation \(g[d/x]\) means:  The variable assignment which is identical to \(g\) except for the (possible) difference that \(g[d/x]\) assigns the individual \(d\) to the variable \(x\).]

Truth:  Some formulas are true independent of the choice of assignment; those can be called true relative to just \(M\), i.e. simply true on the given interpretation.

If \(\varphi\) \in Form, then:  \(\langle \varphi \rangle_M^g = 1\) iff for all assignments \(g\), \(\langle \varphi \rangle_M^{g} = 1\).
otherwise \(\langle \varphi \rangle_M^g = 0\).

Tarragona_05_Lec1.doc Page 7
Appendix 2: Montague’s intensional logic, with lambdas and types.

A.1 Introduction
Tools like Montague’s Intensional Logic are important in making a more satisfactory compositional analysis of natural language semantics possible. What are the differences between Montague’s IL and PC? Here are some of the most important:

(i) The rich type structure of IL.
(ii) The central role played by function-denoting expressions. All of the types except the basic types $e$ and $t$ are functional types, and all of the expressions of IL except those of types $e$ and $t$ are expressions which denote functions. Functions may serve as the arguments and as the values of other functions. In particular, all relations are also represented as functions.
(iii) The inclusion of the operation of “functional application” or “function-argument application”, the application of a function to its argument.
(iv) The use of lambda-expressions. The lambda-operator is the basic tool for building expressions which denote functions.
(v) In place of the one “world” of PC (where there is in effect no distinction between a “world” and a model), the models of IL include a set of possible worlds. Possible worlds are crucially connected with intension/extension distinction and with intensional types. Possible worlds, in particular, underlie the interpretation of modal operators and referential opacity.
(vi) The models of IL also include, in one way or another, a structure of time, used among other things in the interpretation of tense operators like PAST in the fragment below.

A.2. Intensional Logic (IL).
A.2.1. Types and model structures.
A.2.1.1. Types
Montague’s IL is a typed intensional language; unlike the predicate calculus, which has variables of only one type (the type of entities or individuals), and expressions only of the types of individuals, truth-values, and n-ary relations over individuals, IL has a rich system of types which makes it much easier to achieve a (relatively) close fit between expressions of various categories of a natural language and expressions of IL. The types serve as syntactic categories for the expressions of IL; because of the role of IL as an intermediate language in the semantic interpretation of natural language, the same types are referred to as semantic types for expressions of natural language.

The types of Montague’s IL are as follows:

**Basic types:** $e$ (entities), $t$ (truth values)

**Functional types:** If $a,b$ are types, then $<a,b>$ is a type (the type of functions from $a$-type things to $b$-type things.) Note: We use interchangeably the two notations $<a,b>$ and $a \to b$, both of which are common in the literature.

**Intensional types:** If $a$ is a type, then $<s,a>$ is a type (the type of functions from possible worlds to things (extensions) of type $a$.)

(In some systems, the basic type $t$ is taken as intensional, interpreted as the type of propositions rather than of truth-values. In general, we will mostly ignore intensionality in these lectures, working most of the time with extensional versions of our fragments and mentioning intensionality only where directly relevant. But that is only for simplicity of exposition; in general, a thoroughly intensional semantics is presupposed.)
The first rule is a rule for atomic expressions, and the first semantic rule is its interpretation:

**Syntactic Rule 1:** Every constant and variable of type $a$ is in $ME_a$.

**Semantic Rule 1:**
(a) If $\alpha$ is a constant, then $[\alpha]_{M,w,g} = I(\alpha)(w)$.
(b) If $\alpha$ is a variable, then $[\alpha]_{M,w,g} = g(\alpha)$.

**Note:** The recursive semantic rules give extensions relative to model, world, and assignment. Read $[\alpha]_{M,w,g}$ as "the semantic value (extension) of alpha relative to $M, w,$ and $g$.”

The interpretation function $I$ assigns to each constant an intension, i.e. a function from possible worlds to extensions; applying that function to a given world $w$ gives the extension.

**Syntactic Rule 2:** (logical connectives and operators that apply to formulas, mostly from propositional and predicate logic, plus some modal and tense operators.) If $\phi, \psi \in ME_b$ and $u$ is a variable of any type, then $\neg \phi, \phi \land \psi, \phi \lor \psi, \phi \rightarrow \psi$ (also written as $\phi \psi, \phi \land \psi, \phi \lor \psi$), $\exists \alpha, \forall \alpha$.

**Syntactic Rule 3:**
(a) $\phi, \psi \in ME_b$, then $\phi \land \psi \in ME_b$.
(b) $\alpha \in ME_a$, then $\alpha \in ME_b$.
(c) $\psi \in ME_b$, then $\alpha \psi \in ME_b$.

**Semantic Rule 2:**
(a) $\phi, \psi \in ME_b$, then $[\phi \land \psi](w) = [\phi](w) \cdot [\psi](w)$.
(b) $\alpha \in ME_a$, then $[\alpha](w) = [\alpha](w)$.
(c) $\psi \in ME_b$, then $[\alpha \psi](w) = [\alpha](w)[\psi](w)$.

**Semantic Rule 3:**
(a) $\phi, \psi \in ME_b$, then $[\phi \lor \psi](w) = [\phi](w) + [\psi](w)$.
(b) $\alpha \in ME_a$, then $[\alpha \lor \psi](w) = [\alpha](w) + [\psi](w)$.
(c) $\psi \in ME_b$, then $[\alpha \rightarrow \psi](w) = [\alpha](w)[\psi](w)$.

**Semantic Rule 4:**
(a) $\neg \phi \in ME_b$, then $[\neg \phi](w) = 1 - [\phi](w)$.
(b) $\phi \land \psi \in ME_b$, then $[\phi \land \psi](w) = [\phi](w) \cdot [\psi](w)$.
(c) $\alpha \psi \in ME_b$, then $[\alpha \psi](w) = [\alpha](w)[\psi](w)$.

The next two pairs of rules, function-argument application and lambda-abstraction, are among the most important devices of IL, and we will make repeated use of them.

**Function-argument application:**

**Syntactic Rule 6:** If $\alpha \in ME_{a,b}$ and $\beta \in ME_a$, then $\alpha(\beta) \in ME_b$.

**Semantic Rule 6:** $[\alpha(\beta)]_{M,w,g} = [\alpha](\beta)_{M,w,g}$.

**Lambda-abstraction:**

**Syntactic Rule 7:** If $\alpha \in ME_0$ and $u$ is a variable of type $b$, then $\lambda u \alpha \in ME_{b,a}$.

**Semantic Rule 7:** $[\lambda u \alpha](w)_{M,w,g} = [\alpha](w)_{M,w,g}(u)$.

**REFERENCES.**


Borschev, V. B., and Knorina, L. V. 1990. Tipy realij i ix jazykovoe vosprijatie [Types of entities and their perception in language]. In Language of Logic and Logic of Language, ed. V.V. Ivanov, 106-134.

Moscow: Akademija Nauk SSSR, Nauchnyj Sovet po Kompleksnoj Probleme "Kibernetika".


