Logical Semantics
I. Montague Semantics
There is in my opinion no important theoretical difference between natural languages and the artificial languages of logicians; indeed, I consider it possible to comprehend the syntax and semantics of both kinds of languages within a single natural and mathematically precise theory. On this point I differ from a number of philosophers (...).

R. Montague,
Universal Grammar,
Montague’s legacy

- The notion of fragment.
- Semantics as an homomorphic image of syntax.
- Semantic interpretation through a translation into an intermediate logical form.
A direct naive interpretation

\[ S \rightarrow NP \ VP \quad [S] = [VP] [NP] \]
\[ VP \rightarrow tV \ NP \quad [VP] = [tV] [NP] \]
\[ tV \rightarrow \text{loves} \quad [tV] = \lambda y. \lambda x. \text{love} \ y \ x \]
\[ NP \rightarrow \text{John} \quad [NP] = j \]
\[ NP \rightarrow \text{Mary} \quad [NP] = m \]

where:

\[ j, m : \iota \]
\[ \text{love} : \iota \rightarrow \iota \rightarrow o \]
Quantified noun phrases

\[
\begin{align*}
S & \rightarrow NP \ VP \quad [S] = [VP] [NP] \\
VP & \rightarrow tV \ NP \quad [VP] = [tV] [NP] \\
tV & \rightarrow \text{loves} \quad [tV] = \lambda o. \lambda s. s (\lambda x. o (\lambda y. \text{love} x y)) \\
NP & \rightarrow \text{John} \quad [NP] = \lambda k. k j \\
NP & \rightarrow \text{somebody} \quad [NP] = \lambda k. \exists x. k x
\end{align*}
\]
Nouns, adjectives, and determiners

\[
S \rightarrow NP \ VP \\
VP \rightarrow tV \ NP \\
NP \rightarrow Det \ N \\
N \rightarrow Adj \ N \\
tV \rightarrow \text{loves} \\
NP \rightarrow \text{John} \\
NP \rightarrow \text{somebody} \\
N \rightarrow \text{woman} \\
N \rightarrow \text{man} \\
Adj \rightarrow \text{nice} \\
Det \rightarrow \text{every} \\
Det \rightarrow \text{a}
\]

\[
[S] = [VP] [NP] \\
[VP] = [tV] [NP] \\
[NP] = [Det] [N] \\
[N] = [Adj] [N] \\
[tV] = \lambda o. \lambda s. s (\lambda x. o (\lambda y. \text{love} x y)) \\
[NP] = \lambda k. k j \\
[NP] = \lambda k. \exists x. k x \\
[N] = \lambda x. \text{woman} x \\
[N] = \lambda x. \text{man} x \\
[Adj] = \lambda n. \lambda x. n x \land \text{nice} x \\
[Det] = \lambda n. \lambda m. \forall x. n x \supset m x \\
[Det] = \lambda n. \lambda m. \exists x. n x \land m x
\]

where:

\[
\text{woman, man, nice} : \iota \rightarrow o
\]
Relative clauses

\[ N \rightarrow N \ rC \quad [N] = [RC] [N] \]
\[ rC \rightarrow rP \ VP \quad [RC] = [RP] [VP] \]
\[ rP \rightarrow \text{who} \quad [rP] = \lambda r. \lambda n. \lambda x. n \ x \wedge r (\lambda k. k \ x) \]
The categorial syntax/semantics interface

loves : (NP \ S)/NP
John : NP
Mary : NP
somebody : NP
woman : N
man : N
nice : N/N
every : NP/N
a : NP/N
who : (N \ N)/(NP \ S)

\[ [S] = o \]
\[ [NP] = (\iota \rightarrow o) \rightarrow o \]
\[ [N] = \iota \rightarrow o \]
\[ [\alpha \backslash \beta] = [\alpha] \rightarrow [\beta] \]
\[ [\beta / \alpha] = [\alpha] \rightarrow [\beta] \]
Scope ambiguities

Every man loves a woman

\( \forall x. \text{man } x \supset (\exists y. \text{woman } y \land \text{love } x \ y) \)
\( \exists y. \text{woman } y \land (\forall x. \text{man } x \land \text{love } x \ y) \)

Subject wide scope:

\( \lambda o. \lambda s. s (\lambda x. o (\lambda y. \text{love } x \ y)) \)

Object wide scope:

\( \lambda o. \lambda s. o (\lambda y. s (\lambda x. \text{love } x \ y)) \)

Another solution:

\[
\begin{align*}
\text{every} &: (S/(NP \setminus S))/N \\
\text{a} &: ((S/NP) \setminus S)/N \\
\end{align*}
\]

with

\[
\begin{align*}
[S] &= o \\
[\text{NP}] &= \iota \\
\end{align*}
\]
Some intensional puzzles

*De re* and *de dicto*

John seeks a unicorn

Intersective and non-intersective adjectives

John is a french cook

John is an alleged cook

Partee’s puzzle

The temperature is ninety. The temperature rises.
Montague’s intentional logic

Higher-Order Classical Logic + modalities (necessity, past, and future).

One additional base type $s$, with the following associated terms, typing rules, and reduction rules:

\[
\begin{align*}
\Gamma \vdash t : \alpha & \quad \Gamma \vdash t : s \rightarrow \alpha \\
\Gamma \vdash \neg t : s \rightarrow \alpha & \quad \Gamma \vdash \neg t : \alpha \\
\neg \neg t \rightarrow t
\end{align*}
\]
More about time and tense

Reichenbach’s three points of time: point of speech (S), point of event (E), and point of reference (R).

<table>
<thead>
<tr>
<th>tense</th>
<th>example</th>
<th>structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>pluperfect</td>
<td>I had seen</td>
<td>E-R-S</td>
</tr>
<tr>
<td>simple past</td>
<td>I saw</td>
<td>E,R-S</td>
</tr>
<tr>
<td>future in the past</td>
<td>I would see</td>
<td>R-E-S / R-E,S / R-S-E</td>
</tr>
<tr>
<td>present perfect</td>
<td>I have seen</td>
<td>E-S,R</td>
</tr>
<tr>
<td>present</td>
<td>I see</td>
<td>E,S,R</td>
</tr>
</tbody>
</table>

Remember hybrid logic:

\[
P(\downarrow r. @_r P\phi) \quad P(\downarrow r. @_r F\phi) \quad P\phi \quad \phi
\]
II. Continuations
Invented to provide programming languages control operators (exit, goto,...) with a compositional semantics.
Continuation passing style (call by value)

1. $\overline{c} = \lambda k. k c$
2. $\overline{x} = \lambda k. k x$
3. $\overline{\lambda x. M} = \lambda k. k (\lambda x. \overline{M})$
4. $\overline{M N} = \lambda k. \overline{M} (\lambda m. \overline{N} (\lambda n. m n k))$

$\overline{\alpha} = \neg \neg \alpha^*$, where:

1. $\bot^* = \bot$
2. $a^* = a$, for $a$ atomic
3. $(\alpha \rightarrow \beta)^* = \alpha^* \rightarrow \overline{\beta}$
III. Abstract Categorial Grammars
Types, signatures and $\lambda$-terms:

$\mathcal{T}(A)$ is the set of linear implicative types built on the set of atomic types $A$:

$$\mathcal{T}(A) ::= A \mid (\mathcal{T}(A) \rightarrow \mathcal{T}(A))$$

A higher-order linear signature is a triple $\Sigma = \langle A, C, \tau \rangle$, where:

- $A$ is a finite set of atomic types;
- $C$ is a finite set of constants;
- $\tau : C \rightarrow \mathcal{T}(A)$ is a function that assigns each constant in $C$ with a linear implicative type built on $A$.

$\Lambda(\Sigma)$ denotes the set of linear $\lambda$-terms built upon a higher-order linear signature $\Sigma$. 
Vocabularies and Lexicons:

A vocabulary is simply defined to be a higher-order linear signature.

Given two vocabularies $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$, a lexicon $\mathcal{L} = \langle F, G \rangle$ from $\Sigma_1$ to $\Sigma_2$ is made of two functions:

$$F : A_1 \rightarrow \mathcal{T}(A_2),$$

$$G : C_1 \rightarrow \Lambda(\Sigma_2),$$

such that

$$\vdash_{\Sigma_2} G(c) : \hat{F}(\tau_1(c)).$$
Definition:

An abstract categorial grammar is a quadruple

\[ G = \langle \Sigma_1, \Sigma_2, \mathcal{L}, s \rangle \]

where:

\[ \Sigma_1 = \langle A_1, C_1, \tau_1 \rangle \] and \[ \Sigma_2 = \langle A_2, C_2, \tau_2 \rangle \] are two higher-order linear signatures; \( \Sigma_1 \) is called the abstract vocabulary and \( \Sigma_2 \) is called the object vocabulary;

\[ \mathcal{L} : \Sigma_1 \rightarrow \Sigma_2 \] is a lexicon from the abstract vocabulary to the object vocabulary;

\[ s \in T(A_1) \] is a type of the abstract vocabulary; it is called the distinguished type of the grammar.
Languages generated by an ACG:

The abstract language generated by $G$ ($A(G)$) is defined as follows:

$$A(G) = \{ t \in \Lambda(\Sigma_1) | \Gamma_{\Sigma_1} t : s \text{ is derivable} \}$$

The object language generated by $G$ ($O(G)$) is defined to be the image of the abstract language by the term homomorphism induced by the lexicon $L$:

$$O(G) = \{ t \in \Lambda(\Sigma_2) | \exists u \in A(G). t = L(u) \}$$
Strings as linear $\lambda$-terms

There is a canonical way of representing strings as linear $\lambda$-terms. It consists of representing strings as function composition:

\['abbac' \equiv \lambda x. a (b (b (a (c x))))\]

In this setting:

\[\epsilon \overset{\triangle}{=} \lambda x. x\]
\[\alpha + \beta \overset{\triangle}{=} \lambda \alpha. \lambda \beta. \lambda x. \alpha (\beta x)\]
Example

Pierre lit un article que Marie a écrit

\[ \Sigma_0: \quad N, NP, S : \text{type}; \]
\[ P, M : NP \]
\[ A : N \]
\[ L, AE : NP \leadsto (NP \leadsto S) \]
\[ U : N \leadsto NP \]
\[ Q : (NP \leadsto S) \leadsto (N \leadsto N) \]

\[ \Sigma_1: \quad /Pierre/, /Marie/, /article/, /lit/, /a écrit/, /un/, /que/ : STRING \]
\[ \Sigma_2:\]

\begin{align*}
\iota, o & : \text{type;} \\
p, m & : \iota \\
\text{article} & : \iota \to o \\
\text{read, wrote} & : \iota \to (\iota \to o) \\
\land & : o \to (o \to o) \\
\exists & : (\iota \to o) \to o
\end{align*}
\[ \mathcal{L}_1 : \Sigma_0 \rightarrow \Sigma_1 \]

\[
N, NP, S \ := \ STRING; \\
\]

\[
P := /Pierre/ : NP \\
M := /Marie/ : NP \\
A := /article/ : N \\
L := \lambda x. \lambda y. y + /lit/ + x : NP \to (NP \to S) \\
AE := \lambda x. \lambda y. y + /a \ \acute{e}crit/ + x : NP \to (NP \to S) \\
U := \lambda x. /un/ + x : N \to NP \\
Q := \lambda x. \lambda y. y + /que/ + x \in : (NP \to S) \to (N \to N)
\]

Parsing

\[
/Pierre/ + /lit/ + /un/ + /article/ + /que/ + /Marie/ + /a \ \acute{e}crit/
\]
yields the following \(\lambda\)-term of type \(S\):

\[
L (U (Q (\lambda x. AE x M) A)) P
\]
\( \mathcal{L}_2 : \Sigma_0 \rightarrow \Sigma_2 \)

\[
\begin{align*}
S &:= o; \\
N &:= \iota \circ o; \\
NP &:= (\iota \circ o) \circ o;
\end{align*}
\]

\[
\begin{align*}
P &:= \lambda k. k \ p & \text{: NP} \\
M &:= \lambda k. k \ m & \text{: NP} \\
A &:= \lambda x. \text{article} \ x & \text{: } N \\
L &:= \lambda p. \lambda q. p (\lambda x. q (\lambda y. \text{read} \ y \ x)) & \text{: NP} \circ (NP \circ S) \\
AE &:= \lambda p. \lambda q. p (\lambda x. q (\lambda y. \text{wrote} \ y \ x)) & \text{: NP} \circ (NP \circ S) \\
U &:= \lambda p. \lambda q. \exists x. (p \ x) \land (q \ x) & \text{: } N \circ NP \\
Q &:= \lambda r. \lambda p. \lambda x. (p \ x) \land (r (\lambda k. k \ x)) & \text{: } (NP \circ P) \circ (N \circ N)
\end{align*}
\]

Applying \( \mathcal{L}_2 \) to

\[
L \ (U \ (Q \ (\lambda x. AE \ x \ M) \ A)) \ P
\]

yields a term that \( \beta \)-reduces to:

\[
\exists x. (\text{article} \ x) \land (\text{wrote} \ m \ x) \land (\text{read} \ p \ x)
\]