Semantic Analysis in Prolog

Roberto Basili,

Department of Computer Science, System and Production
University of Roma, Tor Vergata
Via Della Ricerca Scientifica, 00133, Roma, ITALY
e-mail: basili@info.uniroma2.it
1. Semantics in NLP and Predicate Calculus
   - Compositionality in NL

2. Compositionality in Prolog
   - Lambda-Calculus & NL
   - $\beta$-reduction
   - $\beta$-reduction and Prolog
   - Compositionality and Verbs

3. Lexicons, Semantics and Compositionality
   - Semantics of Prepositional Modifiers
   - Semantics of Prepositional Verb Arguments
   - Semantics of Lexical Modifiers
   - Introduction to Semantics of Quantification in NLs
Outline

- The semantic level: Interpretation and compositionality
- A simple compositional semantic model for NL in $\lambda$-calculus
- DCG Formalism and compositionality
- Roles, Thematic structures and Quantification
Predicate Calculus in NLP: Objectives

- Define a semantic representation for NL
- Determine a procedural semantics for the interpretation
- Automate all inferences allowed by sentences under such a representation
How to use FOL: first approximation

\[ \text{Gianni corre} \rightarrow \text{corre}(g) \]
\[ \text{Gianni vede Michele} \rightarrow \text{vede}(g, m) \]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Gianni</td>
<td>g</td>
</tr>
<tr>
<td>Michele</td>
<td>m</td>
</tr>
<tr>
<td>corre</td>
<td>{ x : corre(x) }</td>
</tr>
<tr>
<td>vede</td>
<td>{ &lt;x,y&gt; : vede(x,y) }</td>
</tr>
</tbody>
</table>

- It represents a syntax for the semantic level
- How to compute it?
Compositionality

The meaning of an expression is some function of the meaning of its components and of the operators used to combine the latter ones (i.e. syntactic dependencies)

- the meaning of *Michele vede Gianni* is a function of *Michele* and *vede Gianni*
- the meaning of *vede Gianni* is a function of the meanings of *vede* and *Gianni*
- Compositional interpretation proceeds *recursively* with respect to the syntactic operators
Compositionality

```
S
  saw(s,k)

NP
  s
  Sam
  s

VP
  { x : saw(x,k) }

  V
  { <x,y> : saw(x,y) }

  NP
  k
  Kim
  k
```
FOL has a compositional semantics so that the mapping from linguistic expressions to FOL must be compositional too.

This must be systematic: the meaning of complex expressions must systematically correspond to the meaning of the *simpler constituent components*.

We need:
- a mapping for the basic expressions
- a semantic interpretation for each syntactic rule
The compositionality principle for NL expressions

- Every syntactic rule can be seen as a function from combinations (i.e. sequences) of morphems (or grammatical categories) results in an output expression (e.g. a partial tree)
- Every syntactic rule $R$ applied to $\alpha_1, \alpha_2, \ldots, \alpha_n$ results in the expression $\xi$ as:

$$\xi = R(\alpha_1, \ldots, \alpha_n)$$
The compositionality principle for NL expressions

- Every syntactic rule can be seen as a function from combinations (i.e. sequences) of morphems (or grammatical categories) results in an output expression (e.g. a partial tree).
- Every syntactic rule $R$ applied to $\alpha_1, \alpha_2, \ldots, \alpha_n$ results in the expression $\xi$ as:

$$\xi = R(\alpha_1, \ldots, \alpha_n)$$

- It is reasonable to assume that every atomic element $\alpha$ (e.g. nouns) corresponds to a real-world entity, property or relation as well, $sem(\alpha)$ (e.g. a proper noun maps to an individual).
- Every $R$ corresponds to a semantic counterpart $R'$ such that: if $\xi = R(\alpha_1, \ldots, \alpha_n)$ then

$$sem(\xi) = R'(sem(\alpha_1), \ldots, sem(\alpha_n))$$
Compositionality in Prolog

<table>
<thead>
<tr>
<th>Kim</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam</td>
<td>s</td>
</tr>
<tr>
<td>Kim left</td>
<td>left(k)</td>
</tr>
<tr>
<td>Sam saw Kim</td>
<td>saw(sam,kim)</td>
</tr>
</tbody>
</table>

pn(k) --> [kim].

pn(s) --> [sam].

np(Sem) --> pn(Sem).

vp(Sem) --> iv(Sem).

iv(leave(X)) --> [left].
Compositionality in Prolog

How to interpret the non terminal $S$, in $S \rightarrow NP \ VP$?

$s(\text{SSem}) \rightarrow np(\text{NPSem}), \ vp(\text{VPSem})$.

How to deal with transitive verbs?

$vp \rightarrow tv, np$.
$tv(\text{see}(X,Y)) \rightarrow [\text{saw}]$. 
Compositionality in Prolog

vp --> tv, np.
tv(see(X,Y)) --> [saw].

How to unify \(k\) with \(Y\) (rather than with \(X\))?  

Sol1. vp(V(_,NP)) -->
    v(V(_,NP)),
    np(NP).

Sol2. vp(Sem) -->
    v(Sem),
    np(NP),
    \{Sem=V(_,NP)\}. 
Compositionality in Prolog - Problems

Sol1. \( \text{vp}(V(\_, \text{NP})) \rightarrow \)
\( v(V(\_, \text{NP})) , \)
\( \text{np}(\text{NP}) . \)

Sol2. \( \text{vp}(\text{Sem}) \rightarrow \)
\( v(\text{Sem}) , \)
\( \text{np}(\text{NP}) , \)
\( \{\text{Sem}=V(\_, \text{NP})\} . \)

Problems:

- A variable \( v \) stands for a predicate (bad use of Prolog);
- It is not flexible, e.g. how to deal with \( \text{give}(X, Y, Z) \)
A formal language for NL semantics: \(\lambda\)-Calculus

- \textit{Giuseppe corre} should produce: 
  \[\text{corre}(\text{Giuseppe})\]

- \textit{Every student writes a program:}
  \[\forall x \ \text{student}(x) \Rightarrow (\exists p) (\text{program}(p) \& \text{write}(p, x))\]
A formal language for NL semantics: $\lambda$-Calculus

- *Giuseppe corre* should produce:
  $$\text{corre}(\text{Giuseppe})$$

- *Every student writes a program*:
  $$\forall x \quad \text{student}(x) \ \Rightarrow \ (\exists p)(\text{program}(p) \& \text{write}(p, x))$$

- Main consequences:
  - VP map to *predicative symbols*
A formal language for NL semantics: $\lambda$-Calculus

- **Giuseppe corre** should produce:
  $corre(Giuseppe)$

- **Every student writes a program**:  
  $\forall x \ student(x) \Rightarrow (\exists p)(\text{program}(p) \& \text{write}(p, x))$

- Main consequences:
  - VP map to *predicative symbols*
  - Proper nouns map to *atomic (ground) symbols*
A formal language for NL semantics: $\lambda$-Calculus

- **Giuseppe corre** should produce: $\text{corre}(\text{Giuseppe})$
- **Every student writes a program**: $\forall x \; \text{student}(x) \; \Rightarrow \; (\exists p)(\text{program}(p) \& \text{write}(p, x))$

Main consequences:
- VP map to *predicative symbols*
- Proper nouns map to *atomic (ground) symbols*
- The interpretations of VPs (i.e. logical forms called VP') are *functions from entities to propositions*
- Quantification generates more complex structures
Functions in $\lambda$-Calcolo

- We define functions through slight extensions of equations:
  $$f(x) = x + 1$$

- A formalism with a better abstraction for the example function $f$ is:
  $$\lambda x.x + 1$$

- $(\lambda x.x + 1)(3) \ ((\lambda x.(x + 1))(3))$ is equivalent to $3 + 1$

- Main consequences:
  - No different names are used for different functions
  - Only operations $\Omega$ are necessary to compute $f$

- $\beta$-reduction: $(\lambda x.\Omega)(a)$ generates $[\Omega]\{x = a\}$ while,
  $$\ (\lambda x.\lambda y.\Omega)(a)(b) = \lambda y.\Omega\{x = a\}(b) = [\Omega]\{x = a, y = b\}$$
When $\phi$ is a formula and $\nu$ a variable then $\lambda \nu. \phi$ is a predicate. In general, when $\psi$ is an $n$-ary predicate and $\nu$ is a variable, then $\lambda \nu. \psi$ is an $n + 1$-ary predicate.

- $\lambda x. \text{corre}(x)$
- $\lambda x. \text{vede}(x, g)$
- $\lambda x. \text{vede}(m, x)$
- $\lambda y. \lambda x. \text{vede}(x, y)$
When $\phi$ is a formula and $v$ is a variable then $\lambda v. \phi$ is the characteristic function of the set of real-world objects that satisfy $\phi$ (i.e. they make it true).

- $\lambda x. \text{corre}(x)$
- $\lambda x. \text{vede}(x, g)$
- $\lambda x. \text{vede}(m, x)$
- $\lambda y. \lambda x. \text{vede}(x, y)$
The computation of the compositional semantics is mapped into the recursive application of functions (according to the underlying syntactic structure).
The *beta*-reduction \((\lambda x.\Omega)a\) operates by substituting contemporarily all the (free) occurrences of the variable \(x\) in \(\Omega\) with the expression \(a\).

<table>
<thead>
<tr>
<th>Operator</th>
<th>(\Lambda)-Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\beta)-reduction:</td>
<td>((\lambda x.\Omega)a)</td>
<td>([\Omega{x = a})</td>
</tr>
<tr>
<td></td>
<td>((\lambda x.\lambda y.\Omega)(a)(b))</td>
<td>(\lambda y.\Omega{x = a}(b) = [\Omega]{x = a, y = b})</td>
</tr>
</tbody>
</table>
\( \beta \)-reduction and Compositional Semantics

\[
S \\
\text{saw}(s,k) \\
\lambda x. \text{saw}(x,k)(s)
\]

\[
\text{NP} \\
\text{s} \\
\text{Sam}
\]

\[
\text{VP} \\
\lambda x. \text{saw}(x,k) \\
\lambda y. \lambda x. \text{saw}(x,y) (k)
\]

\[
\text{V} \\
\lambda y. \lambda x. \text{saw}(x,y)
\]

\[
\text{saw} \\
\lambda y. \lambda x. \text{saw}(x,y)
\]

\[
\text{NP} \\
k \\
\text{Kim}
\]
\textit{β}-reduction and Compositional Semantics

- \textit{Giuseppe corre: corre(giuseppe)}

\[ S \rightarrow NP \ VP \]

- Semantic Rule1 (\textit{intransitive verbs}):

  \textit{IF the Logic Form (FL) of NP is NP' and the FL of VP is VP': THEN the FL of S' is given by VP'(NP')}  

- Consequences:
  - a good candidate as a VP' for the verb \textit{corre} is: \( \lambda x.\textit{corre}(x) \)
  - a standard mapping of proper nouns (e.g.) \textit{Giuseppe} into domain constants (e.g. \textit{giuseppe}) is adopted.

\[ S' = VP'(NP') = (\lambda x.\textit{corre}(x))(\textit{giuseppe}) = \textit{corre}(\textit{giuseppe}) \]
\(\beta\text{-reduction and Compositional Semantics (2)}\)

- **Giuseppe usa Prolog**: \(usa(\text{giuseppe}, \text{prolog})\)
  \[ VP \rightarrow V \ NP \]

- **Semantic Rule 2 (transitive verbs)**:
  
  \[ IF \text{ the FL of NP is NP'} and \text{ the FL of V is V'} : \]
  
  \[ THEN \text{ the FL of VP'} is given by V'(NP') \]

- **Consequences (in modelling V')**:
  
  \[ usa: \lambda x. \lambda y. usa(y, x) \]

\[ S' = VP'(NP'_0) = \]
\[ = V'(NP'_1)(NP'_0) = (\lambda x. \lambda y. usa(y, x))(prolog)(\text{giuseppe}) = \]
\[ = usa(\text{giuseppe}, \text{prolog}) \]
First, syntactic rules $S \rightarrow NPVP$ are modeled in a standard way:

They have a standard DCG Form as:

$$s(SP) \rightarrow np(NP), \ vp(VP).$$
Compositional semantics in Prolog

First, syntactic rules $S \rightarrow NPVP$ are modeled in a standard way:

They have a standard DCG Form as:

$$s(SP) \rightarrow np(NP), \ vp(VP).$$

The DCG format corresponds to the following list manipulation operation in the following standard syntax:

$$s(SP, InputList, OutputList) :- np(NP, InputList, TmpList),
\quad vp(VP, TmpList, OutputList).$$

A sentence is recognized as a legal $SP$ iff

$$?\-s(SP, SentenceList, []) \text{ is true.}$$
Compositional semantics in Prolog

- Given a syntactic rule in a standard DCG Form as:
  \[ s(SP) \rightarrow np(NP), \ vp(VP). \]

- In semantic terms, \( SP \) must be derived compositionally from \( NP \) and \( VP \).

  **HOW:** \( VP \) is applied to \( NP \) !!!!
Compositional semantics in Prolog

Given a syntactic rule in a standard DCG Form as:
\[
s(SP) \rightarrow np(NP), \ vp(VP).
\]

In semantic terms, \( SP \) must be derived compositionally from \( NP \) and \( VP \).

**HOW:** \( VP \) *is applied to* \( NP \) !!!!

\[
s(S) \rightarrow np(NP), \ vp(VP), \ \{\text{betareduce}(VP,NP,S)\}.
\]
\[
\text{betareduce}(\text{Arg}^\text{Expr}, \text{Arg}, \text{Expr}).
\]
...

\[
vp(X^\text{corre}(X)) \rightarrow [\text{corre}]. \ // \text{lexical rule for } "\text{corre}"\]
\[
\text{np}(\text{giuseppe}) \rightarrow [\text{giuseppe}]. \ //\text{lexical rule for } "\text{Giuseppe}"
\]
...

?\(-s(S,[\text{giuseppe,corre}],[]).
\]
\[
S = \text{corre}(\text{giuseppe})
\]
Yes
Compositional semantics in Prolog

\[ S \rightarrow NP \ VP \]

DCG Form: \( s(SP) \rightarrow np(NP), vp(VP). \)
Compositional semantics in Prolog

\[ S \rightarrow NP \ VP \]

DCG Form: \[ s(SP) \rightarrow np(NP), \ vp(VP). \]

- \( SP \) must be derived compositionally from \( NP \) and \( VP \).

HOW: \( VP \) is applied to \( NP \) !!!!
Compositional semantics in Prolog

\[ S \rightarrow NP \ VP \]

DCG Form: \( s(SP) \rightarrow np(NP), \ vp(VP). \)

- \( SP \) must be derived compositionally from \( NP \) and \( VP \).

**HOW:** \( VP \) *is applied to* \( NP \) !!!!

\( s(S) \rightarrow np(NP), \ vp(VP), \{ \text{betareduce}(VP,NP,S) \}. \)

betareduce(Arg^Expr, Arg, Expr).

\ldots

\( vp(X^\text{corre}(X)) \rightarrow [\text{corre}]. // \text{lexical rule for "corre" (runs)} \)

\( np(giuseppe) \rightarrow [\text{giuseppe}]. // \text{lexical rule for "Giuseppe"} \)

\ldots

?-s(S,[giuseppe,corre],[[]]).

\( S = \text{corre}(giuseppe) \)

Yes
Compositional semantics in Prolog

Given \(?- s(S, [giuseppe, corre], []).\)

\begin{align*}
& \text{CALL( } s(S, [giuseppe, corre], []) ) \\
& \text{CALL( np(NP, [giuseppe, corre], L1) ) } \\
& \text{EXIT( np(giuseppe, [giuseppe, corre], [corre]). } \%\text{consu} \\
& \text{CALL( vp(VP, [corre], []) )}, \\
& \text{EXIT( vp(X\^corre(X), [corre], []). } //\text{consu} \\
& \text{CALL( betareduce(X\^corre(X), giuseppe, corre(X)).} \\
& \quad \%\text{unifica Arg con giuseppe} \\
& \text{EXIT( betareduce(giuseppe\^corre(giuseppe), giuseppe, corre(giuseppe)).} \\
& \text{EXIT( s(corre(giuseppe), [giuseppe, corre], []) )} \\
\end{align*}
Compositional semantics in Prolog (2)

- Transitive verbs have a different lexical form.
s(SP) --> np(NP), vp(VP).

vp(VP) --> tv(NP), np(NP).

- **Transitive verbs have a different lexical form.**

  vp(VP) --> iv(VP).
  vp(VP) --> tv(V), np(NP), {betareduce(V,NP,VP)}.
  s(S) --> np(NP), vp(VP), {betareduce(VP,NP,S)}.
  betareduce(Arg^Expr, Arg, Expr).

  ...
  tv(X^Y^usa(Y,X)) --> [usa].
  np(giuseppe) --> [giuseppe].
  np(prolog) --> [prolog].
  ...
  vp(Y^usa(Y,prolog)) --> tv(X^Y^usa(Y,X)), np(prolog),
  {betareduce(X^Y^usa(Y,X), prolog, Y^usa(Y,prolog) )}
Interpretation of verbs (tr/intr)

Every verbal phrase for transitive and intransitive verbs obeys to:

- A **DCG grammar** \( \text{vp}(\text{VP}) \rightarrow \text{tv}(\text{NP}), \text{np}(\text{NP}). \)
- Some **mechanisms for implementing compositionality**
  \[ \text{s}(\text{S}) \rightarrow \text{np}(\text{NP}), \text{vp}(\text{VP}), \{\text{betareduce}(\text{VP},\text{NP},\text{S})\}. \]
  \[ \text{betareduce}(\text{Arg}^\text{Expr}, \text{Arg}, \text{Expr}) . \]
  or more syntetically
  \[ \text{s}(\text{S}) \rightarrow \text{np}(\text{Arg}), \text{vp}(\text{Arg}^\text{S}). \]
- A **Lexicon** expressing the different simple lexical entries
  \[ \text{tv}(\text{X}^\text{Y}^\text{usa}(\text{Y},\text{X})) \rightarrow [\text{usa}]. \]
Compositionality and Verbs

Observations

- Compositional semantics is *strongly lexicalized* (verbs and nouns)
- The number of arguments varies *across verbs* and ...
- ... across *verb senses* (i.e. *operate a patient* vs. *operate in a market*)
- The lexicon also include *preference rules* for ambiguous phenomena (per es. PP dependencies that are wildly ambiguous)
- Knowledge of the *domain* is crucial for implementing and optimizing these mechanisms
Semantic analysis has the objective of generating a truth-conditional representation of the meaning of NL sentences.
Semantic analysis has the objective of generating a truth-conditional representation of the meaning of NL sentences.

Compositional semantics is mapped into a recursive process applied to the syntactic material produced during parsing.
Semantic analysis has the objective of generating a truth-conditional representation of the meaning of NL sentences.

Compositional semantics is mapped into a recursive process applied to the syntactic material produced during parsing.

Functional programming maps the semantic analysis task to a recursive process combining lexical and grammatical functions.
Semantic analysis has the objective of generating a truth-conditional representation of the meaning of NL sentences.

Compositional semantics is mapped into a recursive process applied to the syntactic material produced during parsing.

Functional programming maps the semantic analysis task to a recursive process combining lexical and grammatical functions.

We presented simple models for the semantic interpretation of major lexical classes: common nouns, proper nouns, transitive and intransitive verbs.
We implemented in the DCG Prolog formalism a model for the semantic analysis process based on:

- Unification (in the \textit{beta}-reduction operator)
- A \textit{depth-first} strategy (used by the Prolog interpreter)
- A declarative model of the lexicon
References

- *An Introduction to Unification-based Approaches to Grammar*, S. Shieber, Chapter 1, 2, 7, 8, CSLI Lecture Notes, n. 4, 1986.
Some of the linguistic phenomena have not been discussed yet

- Verb Arguments expressed by propositional phrases
- Thematic Roles
- Quantification

The above phenomena are crucially dependent on the lexicon and on the domain model, i.e. an ontology
Prepositional phrases (syntagms)

- Prepositional phrases have very different roles in the semantic description. They can be
  - Verb Arguments introduced by prepositions
    \[ Mario \text{ da' } a \text{ Gianni una penna } \]
  - Accidental (i.e. non-core) Modifiers
    \[ Mario \text{ da' la penna a Gianni in affitto — con affetto } \]
    \[ Mario \text{ da’ la penna a Gianni in cucina } \]
  - Empty Arguments
    \[ John relies on Fido \rightarrow rely\_on(j,f) \]
Treatment of Empty prepositional modifiers

Example: *John relies on Fido* → rely_on(j,f)

\[ pp(\text{Form}, \text{Sem}) \rightarrow \]
\[ p(\text{Form}), \]
\[ np(\text{Sem}). \]

\[ p(\text{to}) \rightarrow [\text{to}]. \]
\[ p(\text{from}) \rightarrow [\text{from}]. \]
\[ p(\text{of}) \rightarrow [\text{of}]. \]
\[ p(\text{on}) \rightarrow [\text{on}]. \]

\% rely on Fido, i.e. prepositional objects
\[ vp(2/P\text{form}, \text{Sem}) \rightarrow \]
\[ v(2/P\text{form}, Y^\text{Sem}), \]
\[ pp(P\text{form}, Y). \]

\[ v(2/on, Y^X^\text{rely}\_\text{on}(X,Y)) \rightarrow [\text{relies}]. \]
Manage *empty prepositions*, i.e.

\[
p(\text{on}) \rightarrow [\text{on}]
\]

\[
\ldots
\]

\[
\text{pp}(\text{Form}) \rightarrow \text{p}(\text{Form}), \text{np}(\text{Sem}).
\]

\[
\text{vp}(2/\text{Pform}, \text{Sem}) \rightarrow
\]

\[
\quad \text{v}(2/\text{Pform}, Y^{\text{Sem}}),
\]

\[
\quad \text{pp}(\text{Pform}, Y).
\]

in coherence with other constructions, e.g.

\[
\text{s}(S) \rightarrow \text{np}(\text{Arg}), \text{vp}(\text{Arg}^{\text{S}}).
\]

**Idea:**

\[
\text{pp}(\text{Form}, \text{Sem}) \rightarrow \text{p}(\text{Form}, X^{\text{Sem}}), \text{np}(\text{Sem}).
\]
Treatment of the verb prepositional arguments

- Gianni da’ il libro a Michele → dare(g,l,m)
- Gianni parla del libro a Michele → parlare(g,l,m)
- Gianni compra il libro da Michele → comprare(g,l,m)
Some (English) verbs are called "ditransitive", as they exhibit two direct objects playing the role of arguments. They correspond to *triadic predicates*, with specific syntax-to-semantic mappings.

\[
vp(3/Pform, \text{Sem}) \rightarrow \\
v(3/Pform, Z^Y^Sem), \\
np(Y), \\
pp(Pform, Z).
\]

\[
v(3/a, Z^Y^X^\text{dare}(X,Y,Z)) \rightarrow \\
[diede].
\]

\[
v(3/da, Z^Y^X^\text{comprare}(X,Y,Z)) \rightarrow \\
[comprava].
\]
**Assignment**

Try to write a grammar fragment able to recognize other ditransitive forms such as:

*Gianni parla del libro a Michele* → parlare(g,l,m)

by exploiting suitable definitions for $vp()$ and $pp()$

Try to generalize the solutions to account for the movement of modifiers, as in:

*Gianni parla del libro a Michele* → parlare(g,l,m)

*Gianni parla a Michele del libro* → parlare(g,l,m)
Treatment of ditransitive verbs

- *John gave the book to Mary* → give(j,b,m)
- *John gave Mary the book* → give(j,b,m)

Notice how the logic form *FL* should be the invariant with respect to the two grammatical structures. It corresponds to specific roles:

\[ \text{give} (\text{Giver}, \text{Gift}, \text{Recipient}) \]
Treatment of ditransitive verbs

We need two rules for the same verb that express the two structures:

\[
\text{NP VP NP1 to NP2 NP VP NP2 NP1}
\]

\[v(3/\text{to}, Z^Y^X^\text{give}(X,Y,Z) ) \rightarrow \text{[gave]}.\]

\[v(4, Z^Y^X^\text{give}(X,Y,Z) ) \rightarrow \text{[gave]}.\]

Here we have equivalent semantics for two different syntactic forms.
Treatment of ditransitive verbs

NP VP NP2 NP1
NP VP NP1 to NP2

vp(3/Pform, Sem) --> % give NP2 to NP1:

v(3/Pform, Z^Y^Sem),
np(Y),
pp(Pform, Z).

vp(4, Sem) --> % give NP1 NP2:

v(4, Z^Y^Sem),
np(Z),
np(Y).

Observation: The assumption about roles is a core property of the predicate and it is static (i.e. sentence and syntax independent). It basically corresponds to a verb sense.
The design of the representation formalism depends on a linguistic theory and it is not unique.

For example we could rely on explicit naming of roles and produce a list, e.g.

*John gave the book to Mary* → [give:target, agent:j, theme:b, goal:m]

or even make the arguments’ roles explicit within a predicative structure, e.g.

*John saw Mary* →
some(E,[seeing(E),agent(E,j),theme(E,m),before(E,now)])
**Alternative Semantic Representations**

*John gave the book to Mary* → [give, agent:j, theme:b, goal:m]

\[ v(1, X^\text{[die, agent: X]} ) \rightarrow \text{[died]}. \]

\[ v(2, Y^X^\text{[love,agent:X,theme:Y]} ) \rightarrow \text{[loved]}. \]

\[ v(3/to, Z^Y^X^\text{[give,agent:X, theme:Y, goal:Z]} ) \rightarrow \text{[gave]}. \]

\[ v(3/from, Z^Y^X^\text{[buy, agent:X, theme:Y, source:Z]} ) \rightarrow \text{[bought]}. \]

\[ v(5, Z^Y^X^\text{[give, agent:X, theme:Z, goal:Y]} ) \rightarrow \text{[gave]}. \]
A variety of semantic phenomena depends on the individual words, as these constraints the underlying/intended interpretation of syntactic structures

- Semantics of Argumental Prepositional Modifiers
  - *l’uomo* bevve *birra* tutta la notte
  - *la macchina* beveva troppo *gasolio*

- Arity and Roles in the Logic Form:
  - beve(*uomo*, *birra*)...
  - bere(*macchina*, *gasolio*) vs. consumare(*macchina*, *gasolio*)
A variety of semantic phenomena depends on the individual words, as these constraints the underlying/intended interpretation of syntactic structures

- Semantics of Argumental Prepositional Modifiers
  
  *lo zio di Mario*
  
  *il libro di Mario*

- Arity and Roles in the Logic Form:
  
  *parente(zio,'Mario')*
  
  *possessore(libro,'Mario')*
The above cases suggest that we need to express the different interpretation at the lexical level, i.e. through specific lexical constraints

- **Sense** distinctions ($bere_{ingerire}$ vs. $bere_{consumare}$)
- Constraints on the use of modifiers, also called (selectional restrictions)
  - $trattare$ di storia, $dare$ a qualcuno,
  - $il$ libro di Mario, ... di storia, ... di sogni, ... di marmo
  - residente a Roma, ... a Gennaio, ... a motore, ... ad acqua
- Relational Models of modifier interpretation (Syntax-semantics interface)
  - $parente(zio,'Mario')$
  - $possessore(libro,'Mario')$
An example: nominal postmodifiers

*il libro di Mario, ... di storia, ... di sogni, ... d’acqua*

*residente a Roma, ... a Gennaio, ... a motore, ... ad acqua*

\[
\text{np(Sem & Mod)} \rightarrow \\
\quad \text{npk(Sem),} \\
\quad \text{pp(np/Sem, Mod).}
\]

\[
\text{pp(np/PPHead_Sem, PPSem) } \rightarrow \\
\quad \text{p(np,Arg^PPHead_Sem^Expr),} \\
\quad \text{np(Arg),} \\
\quad \{pp\_interpretation(Arg^PPHead_Sem^Expr, PPSem)}.
\]

\[
\text{p(np,X^Y^di(Y,X,PPSem)) } \rightarrow \\
\quad [di].
\]
An example: nominal postmodifiers

%----------------------------------------------------------------
...
pp_interpretation( Arg^Head^Expr, SemForm) :-
call(Expr),
Expr =.. [Prep, Head, Arg, SemForm].

....
%regole PostModificatori Nominail (predicati diadici)
di(Head,ModNP,possessor(Head,ModNP)) :-
tc_isa(Head,oggetto),
tc_isa(ModNP,persona).
di(Head,ModNP,parente(Head,ModNP)) :-
tc_isa(Head,parente),
tc_isa(ModNP,persona).
Managing Quantification

Given a sentence such as: "Ogni ingegnere studia" expressed by a syntax like:

\[ s(SP) \rightarrow np(NP), \ vp(VP). \]

it is obvious that the noun phrase "Ogni ingegnere" expresses a quantification.

A logic form that is coherent with intuition is thus:

\[ \forall x \  ingegnere(x) \Rightarrow studia(x) \]
Quantifiers and $\lambda$-calculus

- Quantification in noun phrases can be expressed in the lexicon through the following $\lambda$-abstraction corresponding to the phrase "Ogni ingegnere":

\[
\lambda q. (\forall x) \text{ ingegnere}(x) \Rightarrow q(x)
\]

However in the above DCG rule
\[
s(SP) \rightarrow np(NP), \ vp(VP)
\]
it is the noun phrase semantics $NP'$ (originated by $NP$) that applies to verb phrase semantics $VP'$ ($VP$), that is $NP'(VP')$ is the proper modeling, and not vice versa as we assumed so far.
Quantifiers and $\lambda$-calculus

- Quantification in noun phrases can be expressed in the lexicon through the following $\lambda$-abstraction corresponding to the phrase "Ogni ingegnere":

$$\lambda q. (\forall x) \text{ingegnere}(x) \Rightarrow q(x)$$

However in the above DCG rule

s(SP) --> np(NP), vp(VP) it is the noun phrase semantics $NP'$ (originated by $NP$) that applies to verb phrase semantics $VP'$ ($VP$), that is $NP'(VP')$ is the proper modeling, and not vice versa as we assumed so far.

- In fact, with $VP' = \lambda y. \text{studia}(y)$ then $NP'(VP')$) corresponds to:

$$(\lambda q. (\forall x) \text{ingegnere}(x) \Rightarrow q(x))(\lambda y. \text{studia}(y)) =
((\forall x) \text{ingegnere}(x) \Rightarrow (\lambda y. \text{studia}(y))(x)) =
(\forall x) \text{ingegnere}(x) \Rightarrow \text{studia}(x)$$
Quantifiers and $\lambda$-calculus (2)

The noun phrase "Ogni ingegnere" is grammatically described by

$$\text{np}(\text{NP}) \rightarrow \text{det}(\text{DT}), \text{n}(\text{N}), \ldots$$

where DT is the determiner.

We need a compositional semantic account for the NP derivable through $\beta$-reduction from the suitable lexical forms for "Ogni" (DT) and "ingegnere" (N).
The noun phrase "Ogni ingegnere" is grammatically described by
\[ np(NP) \rightarrow \text{det}(DT), n(N), \ldots \] % where DT is the determiner

We need a compositional semantic account for the NP derivable through \( \beta \)-reduction from the suitable lexical forms for "Ogni" (DT) and "ingegnere" (N)

"Ogni ingegnere" can thus be fully described by the following DCG rule:
\[ np(NPSem) \rightarrow \text{det}(DTSem), n(NSem), \text{betareduce}(DTSem, NSem, NPSem) \]

whereas we can find the following definitions in the lexicon for DT and N, respectively:
The noun phrase "Ogni ingegnere" is grammatically described by
np(NP) --> det(DT), n(N), ... %where DT is the determiner

We need a compositional semantic account for the NP derivable through $\beta$-reduction from the suitable lexical forms for "Ogni" (DT) and "ingegnere" (N)

"Ogni ingegnere" can thus be fully described by the following DCG rule:
np(NPSem) --> det(DTSem), n(NSem),
betareduce(DTSem,NSem,NPSem)

whereas we can find the following definitions in the lexicon for DT and N, respectively:
DT: $\lambda p. \lambda q. (\forall x) p(x) \Rightarrow q(x)$
N: $\lambda y. \text{ingegnere}(y)$

It follows that nouns such as "ingegnere" correspond to properties that are unary predicates, in a strict analogy with (intransitive) verbs.
Quantifiers and \( \lambda \)-calculus (2)

- The noun phrase "Ogni ingegnere" is grammatically described by
  \[
  np(NP) \rightarrow \operatorname{det}(DT), n(N), \ldots \%
  \]
  where \( DT \) is the determiner.

We need a compositional semantic account for the \( NP \) derivable through \( \beta \)-reduction from the suitable lexical forms for "Ogni" (\( DT \)) and "ingegnere" (\( N \)).

- "Ogni ingegnere" can thus be fully described by the following DCG rule:
  \[
  np(NPSem) \rightarrow \operatorname{det}(DTSem), n(NSem), \operatorname{betareduce}(DTSem, NSem, NPSem)
  \]
  whereas we can find the following definitions in the lexicon for \( DT \) and \( N \), respectively:
  \[
  DT: \lambda p. \lambda q. \left( \forall x \right) p(x) \Rightarrow q(x)
  \]
  \[
  N: \lambda y. \text{ingegnere}(y)
  \]
  It follows that nouns such as "ingegnere" corresponds to properties that are unary predicates, in a strict analogy with (intransitive) verbs.
The sentence "Ogni ingegnere studia", described by the grammar as
\begin{align*}
s(S) &\rightarrow np(NP), \ vp(VP), \ betareduce(NP,VP,S) \\
np(NP) &\rightarrow det(DT), \ n(N), \ betareduce(DT,N,NP)
\end{align*}
triggers the following chain of \(\beta\)-reductions:
The sentence "Ogni ingegnere studia", described by the
grammar as
s(S) → np(NP), vp(VP), betareduce(NP,VP,S)
np(NP) → det(DT), n(N), betareduce(DT,N,NP)
triggers the following chain of β-reductions:
NP=DT(N):
(\lambda p. \lambda q. (\forall x) p(x) \Rightarrow q(x))(\lambda y. ingegnere(y)) =
Quantifiers and \(\lambda\)-calculus (3)

The sentence "Ogni ingegnere studia", described by the grammar as

\[
\begin{align*}
s(S) & \rightarrow \ np(NP), \ \ vp(VP), \ \ betareduce(NP,VP,S) \\
np(NP) & \rightarrow \ det(DT), \ \ n(N), \ \ betareduce(DT,N,NP)
\end{align*}
\]

triggers the following chain of \(\beta\)-reductions:

\[
\begin{align*}
NP=\ & DT(N): \\
\quad \quad & (\lambda p.\lambda q.(\forall x)p(x) \Rightarrow q(x))(\lambda y.\text{ingegnere}(y)) = \\
\quad \quad = \ & (\lambda p.\lambda q.(\forall x)(\lambda y.\text{ingegnere}(y))(x) \Rightarrow q(x))(\lambda y.\text{ingegnere}(y)) =
\end{align*}
\]
The sentence "Ogni ingegnere studia", described by the grammar as
\[
\begin{align*}
s(S) & \rightarrow \text{np}(NP), \text{vp}(VP), \text{betareduce}(NP, VP, S) \\
\text{np}(NP) & \rightarrow \text{det}(DT), \text{n}(N), \text{betareduce}(DT, N, NP)
\end{align*}
\]
triggers the following chain of \(\beta\)-reductions:

\[
\begin{align*}
\text{NP}=\text{DT}(N): \\
(\lambda p. \lambda q. (\forall x) p(x) \Rightarrow q(x))(\lambda y. \text{ingegnere}(y)) = \\
= (\lambda p. \lambda q. (\forall x)(\lambda y. \text{ingegnere}(y))(x) \Rightarrow q(x))(\lambda y. \text{ingegnere}(y)) = \\
= \lambda q. (\forall x) \text{ingegnere}(x) \Rightarrow q(x)
\end{align*}
\]
The sentence "Ogni ingegnere studia", described by the grammar as

\[\begin{align*}
\text{s(S)} & \rightarrow \text{np(NP)}, \text{vp(VP)}, \text{betareduce(NP,VP,S)} \\
\text{np(NP)} & \rightarrow \text{det(DT)}, \text{n(N)}, \text{betareduce(DT,N,NP)}
\end{align*}\]

triggers the following chain of $\beta$-reductions:

\[\begin{align*}
\text{NP} = \text{DT}(N) : \\
(\lambda p. \lambda q. (\forall x)p(x) \Rightarrow q(x))(\lambda y. \text{ingegnere}(y)) &= \\
= (\lambda p. \lambda q. (\forall x)(\lambda y. \text{ingegnere}(y))(x) \Rightarrow q(x))(\lambda y. \text{ingegnere}(y)) &= \\
= \lambda q. (\forall x)\text{ingegnere}(x) \Rightarrow q(x)
\end{align*}\]

and similarly, \( S = \text{NP}(\text{VP}) : \)

\[\begin{align*}
(\lambda q. (\forall x)\text{ingegnere}(x) \Rightarrow q(x))(\lambda y. \text{studia}(y)) &= \\
= ((\forall x)\text{ingegnere}(x) \Rightarrow (\lambda y. \text{studia}(y))(x)) &= \\
= (\forall x)\text{ingegnere}(x) \Rightarrow \text{studia}(x)
\end{align*}\]
In order to manipulate quantifiers in Prolog we need to model the following expressions:
\[ \forall x \ P(x) \quad \text{and} \quad \exists x \ P(x) \]

This is carried out by introducing two special purpose predicates `all/2` and `exist/2`, and by exploiting constraints imposed by unification.

A possible definition in Prolog could be:
\[ \forall x \ P(x): \ all(X, p(X)) \]
\[ \exists x \ P(x): \ exist(X, p(X)) \]
It must be noticed that $P$ in 
$\forall x \quad P(x) \; \text{ed} \; \exists x \quad P(x)$
can be complex, as in we observed in the semantic description of 
the determiner "ogni".

Also here, Prolog structures can offer a useful syntactic support 
as follows:

$\forall x \quad P(x) \Rightarrow Q(x): \text{all}(X, \ p(X) \Rightarrow q(X))$
$\exists x \quad P(x) \Rightarrow Q(x): \text{exist}(X, \ p(X) \Rightarrow q(X))$
given a suitable definition of $\Rightarrow$ as a binary infix operator through 
the following Prolog declaration:

```prolog
:-op(500, xfy, =>).
```
By using the two predicates above and the $\beta$-reduction we can define the lexical structures able to characterize the quantification. The following lessical forms:

\begin{align*}
\text{ingegnere}: & \quad \lambda y.\text{ingegnere}(y) \\
\text{studia}: & \quad \lambda y.\text{studia}(y) \\
\text{ogni}: & \quad \lambda p.\lambda q.(\forall x)p(x) \Rightarrow q(x)
\end{align*}

can be thus defined in Prolog as:

\begin{align*}
n( X^\text{ingegnere}(X)) & \rightarrow [\text{ingegnere}]. \\
iv( X^\text{studia}(X)) & \rightarrow [\text{studia}]. \\
det( (X^P)(X^Q)^\text{all}(X,(P \Rightarrow Q)) ) & \rightarrow [\text{ogni}].
\end{align*}
Finally, non-lexical DCG rules change in:

\[ np(\ NP) \rightarrow \ det(DT), \ n(N), \ \{ \ betareduce(DT,N,NP) \} \].
\[ s(S) \rightarrow \ np(NP), \ vp(VP), \ \{ \ betareduce(NP,VP,S) \} \].
\[ vp(VP) \rightarrow \ iv(VP). \]

or, more syntactically, by exploiting to the unification contraints:

\[ np(NP) \rightarrow \ det(N^NP), \ n(N). \]
\[ s(S) \rightarrow \ np(VP^S), \ vp(VP). \]
Scrivere un modello lessicale per l’aggettivo *tutti*.
Scrivere un modello lessicale per gli aggettivi dimostrativi *questo*, *quello*, *questi*. Scrivere un modello lessicale per alcuni determiner quali *un*, *uno*, *il*.
Scrivere un modello semantico per frasi nominali quali:

- *il libro giallo, il libro di Mario, il libro di Storia*
- *I libri di Mario*
- *L’abito a scacchi*
In the Prolog DCG formalism an implementation of the semantic analysis process based on the interpreter resolution strategy has been defined.

Several linguistic phenomena have been discussed:

- Empty Prepositional Modifiers
- Argumental Prepositional Modifiers within *n*-ary predicates
- Semantic equivalence of distinct syntactic argument structures (i.e. ditransitive verbs)
- Lexical dependencies within the semantic interpretation process
- (*) Quantifiers
Suggested Bibliography

