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

Outline

- The semantic level: Interpretation and compositionality
- A simple compositional semantic model for NL in λ-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

Gianni corre $\rightarrow$ *corre*(g)
Gianni vede Michele $\rightarrow$ *vede*(g, m)

Gianni	`g`
Michele	`m`
corre	`{ x : corre(x) }`
vede	`{ <x,y> : vede(x,y) }`

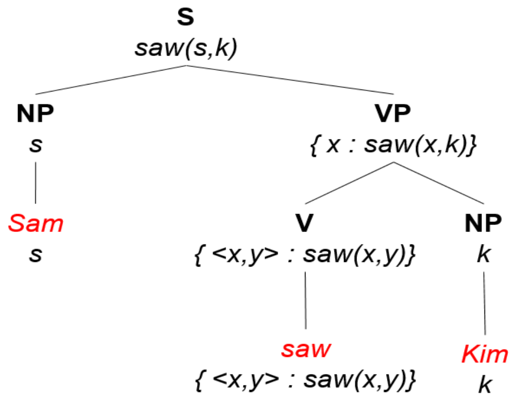
- It represent a syntax for the *semantic* level
- how to compute it?

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



FOL for Compositionality in NL semantics

- 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 $\alpha_1, \alpha_2, ..., \alpha_n$ results in the expression ξ as:

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

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 $\alpha_1, \alpha_2, ..., \alpha_n$ results in the expression ξ as:

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

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

$$sem(\xi) = R'(sem(\alpha_1), ..., sem(\alpha_n))$$

Compositionality in Prolog

Kim	k
Sam	s
Kim left	`left(k)`
Sam saw Kim	`saw(sam,kim)`

```
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 -> NP VP`?

```
s(SSem) --> np(NPSem), vp(VPSem).
```

How to deal with transitive verbs?

```
vp --> tv, np.
tv(see(X,Y)) -->
            [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. vp(V(_,NP)) -->
                      v(V(_,NP)),
                     np(NP).

Sol2. vp(Sem) -->
              v(Sem),
             np(NP),
              {Sem=V(_,NP)}.
```

Problems:

- A variable `V` stands for a predicate (bad use of Prolog);
- It is not flexible, e.ghow to deal with . `give(X,Y,Z)`

A formal language for NL semantics: λ-Calculus

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

A formal language for NL semantics: λ-Calculus

- *Giuseppe corre* should produce:
  $corre(Giuseppe)$
- *Every student writes a program* :
  $\forall x \quad student(x) \quad \Rightarrow \quad (\exists p)(program(p) \& write(p, x))$
- Main consequences:
  - VP map to *predicative symbols*

A formal language for NL semantics: λ-Calculus

- *Giuseppe corre* should produce:
  $corre(Giuseppe)$
- *Every student writes a program* :
  $\forall x \quad student(x) \quad \Rightarrow \quad (\exists p)(program(p) \& write(p, x))$
- Main consequences:
  - VP map to *predicative symbols*
  - Proper nouns map to *atomic (ground) symbols*

A formal language for NL semantics: λ-Calculus

- *Giuseppe corre* should produce:
  $corre(Giuseppe)$
- *Every student writes a program* :
  $\forall x \quad student(x) \quad \Rightarrow \quad (\exists p)(program(p) \& write(p, x))$
- 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 λ-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 Ω are necessary to compute f
- β-reduction: $(\lambda x.\Omega)(a)$ generates $[\Omega]\{x = a\}$ while,

$$(\lambda x.\lambda y.\Omega)(a)(b) = \lambda y.\Omega\{x = a\}(b) = [\Omega]\{x = a, y = b\}$$

λ-Calculus: Sintax

When ϕ is a formula and v a variable then $\lambda v.\phi$ is a predicate. In general, when ψ is an n-ary predicate and v is a variable, then $\lambda v.\psi$ is an $n+1$-ary predicate.

- $\lambda x.corre(x)$
- $\lambda x.vede(x,g)$
- $\lambda x.vede(m,x)$
- $\lambda y.\lambda x.vede(x,y)$

λ-Calculus: Semantics

When ϕ 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** ϕ (i.e. they make it true).

- $\lambda x.corre(x)$
- $\lambda x.vede(x, g)$
- $\lambda x.vede(m, x)$
- $\lambda y.\lambda x.vede(x, y)$

β-reduction and Compositional Semantics

Equivalent expressions:

$(\lambda x.corre(x))$ (g)	$corre(g)$
$(\lambda x.vede(x,g))(m)$	$vede(m,g)$
$(\lambda x.vede(m,x))(g)$	$vede(m,g)$

The computation of the compositional semantics is mapped into the recursive application of functions (according to the underlying syntactic structure).

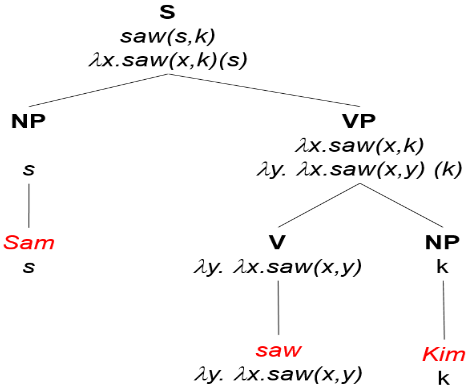
β-reduction

The *beta*-reduction $(\lambda x.\Omega)a$ operates by substituting contemporarily **all** the (free) occurrences of the variable x in Ω with the expression a.

Operator	Λ-Expression	Result
β-reduction:	$(\lambda x.\Omega)a$	$[\Omega]\{x = a\}$
	$(\lambda x.\lambda y.\Omega)(a)(b)$	$\lambda y.\Omega\{x = a\}(b) = [\Omega]\{x = a, y = b\}$

β-reduction and Compositional Semantics

:



β-reduction and Compositional Semantics

- *Giuseppe corre*: $corre(giuseppe)$

$$S \rightarrow NP\ VP$$

- Semantic Rule1 (*intransitive verbs*):

  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 *corre* is: $\lambda x.corre(x)$
  - a standard mapping of proper nouns (e.g.) *Giuseppe* into domain constants (e.g. *giuseppe*) is adopted.
- $S' = VP'(NP') = (\lambda x.corre(x))(giuseppe) = corre(giuseppe)$

β-reduction and Compositional Semantics (2)

- *Giuseppe usa Prolog*: $usa(giuseppe, prolog)$

$$VP \rightarrow V\ NP$$

- Semantic Rule2 (*transitive verbs*):

  IF the FL of NP is NP' and the FL of V is V' :
  THEN the FL of VP' is given by V'(NP')

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

- $S' = VP'(NP'_0) =$
  $= V'(NP'_1)(NP'_0) = (\lambda x.\lambda y.usa(y, x))(prolog)(giuseppe) =$
  $= usa(giuseppe, prolog)$

Compositional semantics in Prolog

- First, syntactic rules $S \rightarrow \quad NPVP$ are modeled in a standard way:
- They have a standard DCG Form as:

```
s(SP) --> 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) --> 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),
   vp(VP, TmpList, OutputList).
  ```
- A sentence is recognized as a legal `SP` **iff** `?-s(SP, SentenceList, [])` is **true**.

Compositional semantics in Prolog

- Given a syntactic rule in a standard DCG Form as:
  `s(SP) --> 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) --> np(NP), vp(VP).`
- In semantic terms, `SP` must be derived compositionally from `NP` and `VP`.

  HOW: `VP` *is applied to* `NP` *!!!!*

```
s(S) --> np(NP), vp(VP), {betareduce(VP,NP,S)}.
betareduce(Arg^Expr, Arg, Expr).
...
vp(X^corre(X)) --> [corre]. // lexical rule for "corre"
np(giuseppe) --> [giuseppe]. //lexical rule for "Giuseppe"
...
?-s(S,[giuseppe,corre],[]).
S = corre(giuseppe)
Yes
```

Compositional semantics in Prolog

$S \rightarrow \quad NP\ VP$

DCG Form: `s(SP) --> np(NP), vp(VP).`

Compositional semantics in Prolog

$S \rightarrow \quad NP\ VP$

DCG Form: `s(SP) --> np(NP), vp(VP).`

- `SP` must be derived compositionally from `NP` and `VP`.

  HOW: `VP` *is applied to* `NP` *!!!!*

Compositional semantics in Prolog

$S \rightarrow \quad NP\ VP$

DCG Form: `s(SP) --> np(NP), vp(VP).`

- `SP` must be derived compositionally from `NP` and `VP`.

  HOW: `VP` *is applied to* `NP` *!!!!*

```
s(S) --> np(NP), vp(VP), {betareduce(VP,NP,S)}.
betareduce(Arg^Expr, Arg, Expr).
...
vp(X^corre(X)) --> [corre]. // lexical rule for "corre" (run
np(giuseppe) --> [giuseppe]. //lexical rule for "Giuseppe"
...
?-s(S,[giuseppe,corre],[]).
S = corre(giuseppe)
Yes
```

Compositional semantics in Prolog

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

```
CALL( s(S,[giuseppe,corre],[]))
CALL( np(NP,[giuseppe,corre],L1) )
EXIT( np(giuseppe,[giuseppe,corre],[corre]). %consuma NP
CALL( vp(VP,[corre],[]) ),
EXIT( vp(X^corre(X),[corre],[]). //consuma VP
CALL( betareduce(X^corre(X), giuseppe, corre(X)).
                              %unifica Arg con giuseppe
EXIT( betareduce(giuseppe^corre(giuseppe), giuseppe,
corre(giuseppe)).
EXIT( s(corre(giuseppe), [giuseppe,corre],[]))
```

Compositional semantics in Prolog (2)

```
s(SP) --> np(NP), vp(VP).
vp(VP) --> tv(NP), np(NP).
```

- Transitive verbs have a different lexical form.

Compositional semantics in Prolog (2)

```
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** `vp(VP) --> tv(NP), np(NP).`
- Some **mechanisms for implementing compositionality**

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

  or more syntetically

  ```
  s(S) --> np(Arg), vp(Arg^S).
  ```

- A **Lexicon** expressing the different simple lexical entries

  ```
  tv(X^Y^usa(Y,X)) --> [usa].
  ```

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 imlpementing and optimizing these mechanisms

Outline (1.1)

- Semantic analysis has the objective of generating a truth-conditional representation of the meaning of NL sentences

Outline (1.1)

- 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

Outline (1.1)

- 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

Outline (1.1)

- 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: coomon nouns, proper nouns, transitive and intransitive verbs

Outline (1.2)

- We implemented in the DCG Prolog formalism a model for the semantic analysis process based on
  - Unification (in the *beta*-reduction operator)
  - A *depth-first* strategy (used by the Prolog interpreter)
  - A declarative model of the lexicon

References

- Chapter 4. "*Further Topics in NL Analysis*", in Fernando C.N. Pereira and Stuart M. Shieber. "*Prolog and Natural-Language Analysis*", volume 10 of CSLI Lecture Notes. Chicago University Press, Stanford, 1987.
  (see also http://www.mtome.com/Publications/PNLA/prolog-digital.pdf).
- *Intelligenza Artificiale*, S. J. Russel, P. Norvig, Prentice Hall Int., Chapter 22.3-22.8, 23, 1998.
- *NLP In Prolog*, G. Gazdar, C. Mellish, Chapter 7, 8, 1998.
- *An Introduction to Unification-based Approaches to Grammar*, S. Shieber, Chapter 1, 2, 7, 8, CSLI Lecture Notes, n. 4, 1986.

Contents (2nd Part)

- Some of the linguistic phenomena have not been discussed yet
  - **Verb Aguments** 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 da' **a Gianni** una penna*
  - Accidental (i.e. non-core) Modifiers
    *Mario da' la penna a Gianni **in affitto** — **con affetto***
    *Mario da' la penna a Gianni **in cucina***
  - *Empty* Arguments
    *John relies **on** Fido* → `rely_on(j,f)`

Treatment of Empty prepositional modifiers

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

```
pp(Form,Sem) -->
  p(Form),
  np(Sem).

p(to) --> [to].
p(from) --> [from].
p(of) --> [of].
p(on) --> [on].

% rely on Fido, i.e. prepositional objects
vp(2/Pform, Sem) -->
  v(2/Pform,Y^Sem),
  pp(Pform,Y).

v(2/on, Y^X^rely_on(X,Y) ) --> [relies].
```

Treatment of Empty prepositional Modifiers

Manage *empty prepositions*, i.e.

```
p(on) --> [on]
...
pp(Form) --> p(Form), np(Sem).
vp(2/Pform, Sem) -->
          v(2/Pform,Y^Sem),
          pp(Pform,Y).
```

in coherence with other constructions, e.g.

```
s(S) --> np(Arg), vp(Arg^S).
```

Idea:

```
pp(Form, Sem) --> p(Form,X^Sem), np(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)`

Treatment of the verb prepositional arguments

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

v(3/a, Z^Y^X^dare(X,Y,Z) ) -->
                              [diede].
v(3/da, Z^Y^X^comprare(X,Y,Z) ) -->
                              [comprava].
```

Treatment of the verb prepositional arguments

Assignment
Try to write a grammar fragment able to recognize other ditransitive forms such as:
Gianni parla del libro a Michele $\rightarrow$ `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 $\rightarrow$ `parlare(g,l,m)`
Gianni parla a Michele del libro $\rightarrow$ `parlare(g,l,m)`

Treatment of ditransitive verbs

- *John gave the book to Mary* $\rightarrow$ `give(j,b,m)`
- *John gave Mary the book* $\rightarrow$ `give(j,b,m)`

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

$$give(Giver, Gift, Recipient)$$

Treatment of ditransitive verbs

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

```
NP VP NP1 to NP2  NP VP NP2 NP1
```

```
v(3/to, Z^Y^X^give(X,Y,Z) ) -->
        [gave].

v(4, Z^Y^X^give(X,Y,Z) ) -->
        [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**.

Alternative Semantic Representations

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 $\rightarrow$ `[give:target, agent:j, theme:b, goal:m]`

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

John saw Mary $\rightarrow$
`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^[die, agent: X] ) -->
                   [died].

v(2, Y^X^[love,agent:X,theme:Y] ) -->
                  [loved].

v(3/to, Z^Y^X^[give,agent:X, theme:Y, goal:Z ] ) -->
                  [gave].

v(3/from, Z^Y^X^[buy, agent:X, theme:Y, source:Z ] ) -->
                  [bought].

v(5, Z^Y^X^[give, agent:X, theme:Z, goal:Y ] ) -->
                  [gave].
```

Lexical Phenomena

A variety of semantic phenomena depends on the individual words, as these constraints the underlying/intended interpretation of syntactic structures

- Semantics of Argumtnal 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. $consumare(macchina, gasolio)$

Lexical Phenomena (2)

A variety of semantic phenomena depends on the individual words, as these constraints the underlying/intended interpretation of syntactic structures

- Semantics of Argumtnal Prepositional Modifiers

  lo zio di Mario

  il libro di Mario

- Arity and Roles in the Logic Form:

  $parente(zio, 'Mario')$

  $possessore(libro, 'Mario')$

The Role of Lexicon in NL interpretation

The above cases suggest that we need to express the different interpretation at the lexical level, i.e. through specifici *lexical constraints*

- **Sense** distinctions ($bere_{ingerire}$ vs. $bere_{consumare}$)
- Constraints on the use of modifiers, alse 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

```
%---------------------------------------------------------------------
np(Sem & Mod) -->
   npk(Sem),
   pp(np/Sem, Mod).
....
%caso nominale - sequenze NP --> NPK PP
pp(np/PPHead_Sem, PPSem) -->
   p(np,Arg^PPHead_Sem^Expr),
   np(Arg),
   {pp_interpretation(Arg^PPHead_Sem^Expr, PPSem)}.
...
%Caso postmodificatori nominali - esempio del "di"
p(np,X^Y^di(Y,X,PPSem)) -->
     [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) --> 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 :

  $\forall x \quad ingegnere(x) \Rightarrow studia(x)$

Quantifiers and λ-calculus

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

  $\lambda q.(\forall x) \quad ingegnere(x) \Rightarrow q(x)$

  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.

Quantifiers and λ-calculus

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

  $\lambda q.(\forall x) \quad ingegnere(x) \Rightarrow q(x)$

  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 $VP' = \lambda y.studia(y)$ then $NP'(VP'))$ corresponds to:

  $(\lambda q.(\forall x)ingegnere(x) \Rightarrow q(x))(\lambda y.studia(y)) =$
  $((\forall x)ingegnere(x) \Rightarrow (\lambda y.studia(y))(x)) =$
  $(\forall x)ingegnere(x) \Rightarrow studia(x)$

Quantifiers and λ-calculus (2)

- 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 β-reduction from the suitable lexical forms for "*Ogni*" (`DT`) and "*ingegnere*" (`N`)

Quantifiers and λ-calculus (2)

- 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 β-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:

Quantifiers and λ-calculus (2)

- 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 β-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`: $\lambda y.ingegnere(y)$

Quantifiers and λ-calculus (2)

- 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 β-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`: $\lambda y.ingegnere(y)$
  It follows that nouns such as "*ingegnere*" corresponds to properties that are unary predicates, in astrict analogy with (intransitive) verbs.

Quantifiers and λ-calculus (3)

- 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:

Quantifiers and λ-calculus (3)

- 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 λ-calculus (3)

- 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):`

$$(\lambda p.\lambda q.(\forall x)p(x) \Rightarrow q(x))(\lambda y.ingegnere(y)) =$$
$$= (\lambda p.\lambda q.(\forall x)(\lambda y.ingegnere(y))(x) \Rightarrow q(x))(\lambda y.ingegnere(y)) =$$

Quantifiers and λ-calculus (3)

- 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):`

  $$
  \begin{aligned}
  & (\lambda p.\lambda q.(\forall x)p(x) \Rightarrow q(x))(\lambda y.ingegnere(y)) = \\
  = \; & (\lambda p.\lambda q.(\forall x)(\lambda y.ingegnere(y))(x) \Rightarrow q(x))(\lambda y.ingegnere(y)) = \\
  = \; & \lambda q.(\forall x)ingegnere(x) \Rightarrow q(x)
  \end{aligned}
  $$

Quantifiers and λ-calculus (3)

- 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)`:

$$\begin{aligned} & (\lambda p.\lambda q.(\forall x)p(x) \Rightarrow q(x))(\lambda y.ingegnere(y)) = \\ = \ & (\lambda p.\lambda q.(\forall x)(\lambda y.ingegnere(y))(x) \Rightarrow q(x))(\lambda y.ingegnere(y)) = \\ = \ & \lambda q.(\forall x)ingegnere(x) \Rightarrow q(x) \end{aligned}$$

and similarly, `S=NP(VP)`:

$$\begin{aligned} & (\lambda q.(\forall x)ingegnere(x) \Rightarrow q(x))(\lambda y.studia(y)) = \\ = \ & ((\forall x)ingegnere(x) \Rightarrow (\lambda y.studia(y))(x)) = \\ = \ & (\forall x)ingegnere(x) \Rightarrow studia(x) \end{aligned}$$

Management of Quantifiers in Prolog

- In order to manipulate quantifiers in Prolog we need to model the following expressions:
  $\forall x \quad P(x)$ and $\exists x \quad P(x)$
- This is carried out by introducing two special purpose predicates `all/2` and `exist/2`, and by exploiting constraints imposed by unification
- A possible defintion in Prolog could be
  $\forall x \quad P(x)$: `all(X, p(X))`
  $\exists x \quad P(x)$: `exist(X, p(X))`

Management of Quantifiers in Prolog (2)

- It must be noticed that P in
  $\forall x \quad P(x)$ 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:

  $\forall x \quad P(x) \Rightarrow Q(x)$: `all(X, p(X) => q(X))`
  $\exists x \quad P(x) \Rightarrow Q(x)$: `exist(X, p(X) => q(X)`

  given a suitable defintion of => as a binary infix operator through the folowing Prolog declaration:

```
:-op(500, xfy, =>).
```

Management of Quantifiers in Prolog (3)

- By using the two predicates above and the β-reduction we can define the lexical structures able to characterize the quantification. The following lessical forms:

ingegnere: $\lambda y.ingegnere(y)$
studia: $\lambda y.studia(y)$
ogni: $\lambda p.\lambda q.(\forall x)p(x) \Rightarrow q(x)$

can be thus defined in Prolog as:

```
n( X^ingegnere(X)) --> [ingegnere].
iv( X^studia(X)) --> [studia].
det( (X^P)(X^Q)^all(X,(P => Q)) ) --> [ogni].
```

Management of Quantifiers in Prolog (4)

Finally, non-lexical DCG rules change in :

```
np( NP) --> det(DT), n(N), { betareduce(DT,N,NP) }.
s(S) --> np(NP), vp(VP), { betareduce(NP,VP,S)}.
vp(VP) --> iv(VP).
```

o, more syntatically, by exploiting to the unification contraints:

```
np(NP) --> det(N^NP), n(N).
s(S) --> np(VP^S), vp(VP).
```

Management of Quantifiers in Prolog - Assignments

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*

Outline (2.1)

- 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 wihtin *n*-ary predicates
  - Semantic equivalence of distinct syntactic argument structures (i.e. ditransitive verbs)
  - Lexical dependencies within the semantic interpretation process
  - (*) Quantifiers

Suggested Bibliography

- Chapter 4. "*Further Topics in NL Analysis*", in Fernando C.N. Pereira and Stuart M. Shieber. "*Prolog and Natural-Language Analysis*", volume 10 of CSLI Lecture Notes. Chicago University Press, Stanford, 1987.
- Chapter 18: "*Computational Semantics*, in Jurafsky, Daniel, and James H. Martin. "*Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics*. 2nd edition. Prentice-Hall. 2009.
- *Intelligenza Artificiale*, S. J. Russel, P. Norvig, Prentice Hall Int., Chapter 22.3-22.8, 23, 1998.