Analisi Semantica del Linguaggio Naturale - Introduzione alla analisi grammaticale e semantica in Prolog

Roberto Basili

Department of Enterprise Engineering
University of Roma, *Tor Vergata*
Via Della Ricerca Scientifica, 00133, Roma, ITALY
e-mail: `basili@info.uniroma2.it`

a.a. 2024-25

Semantica delle Lingue e Logica

Outline

1. *Introduzione*
2. *Analisi Sintattica*
3. *Semantica composizionale in Prolog*
   - Digressione: DCG in Prolog
   - Trattamento dei Sintagmi preposizionali
   - Esercizi
4. *Estensioni*
5. *Riferimenti*

Uso del Calcolo dei Predicati (o FOL) in NLP ed AI

Obbiettivi della FOL in NLP

- Definire una **formalismo per rappresentare la semantica delle frasi di una lingua**: forme logiche de
- Definire una **sintassi per le regole di riscrittura** delle frasi di una lingua in forme logiche
- Supportare il **processo di generazione della forma logica** (*parsing*)
- Automatizzare l'uso della forma logica per le inferenze derivabili da essa: **inferenza come deduzione**

Il ruolo del FOL

Usare il FOL per esprimere i significati delle espr. linguistiche

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) }`

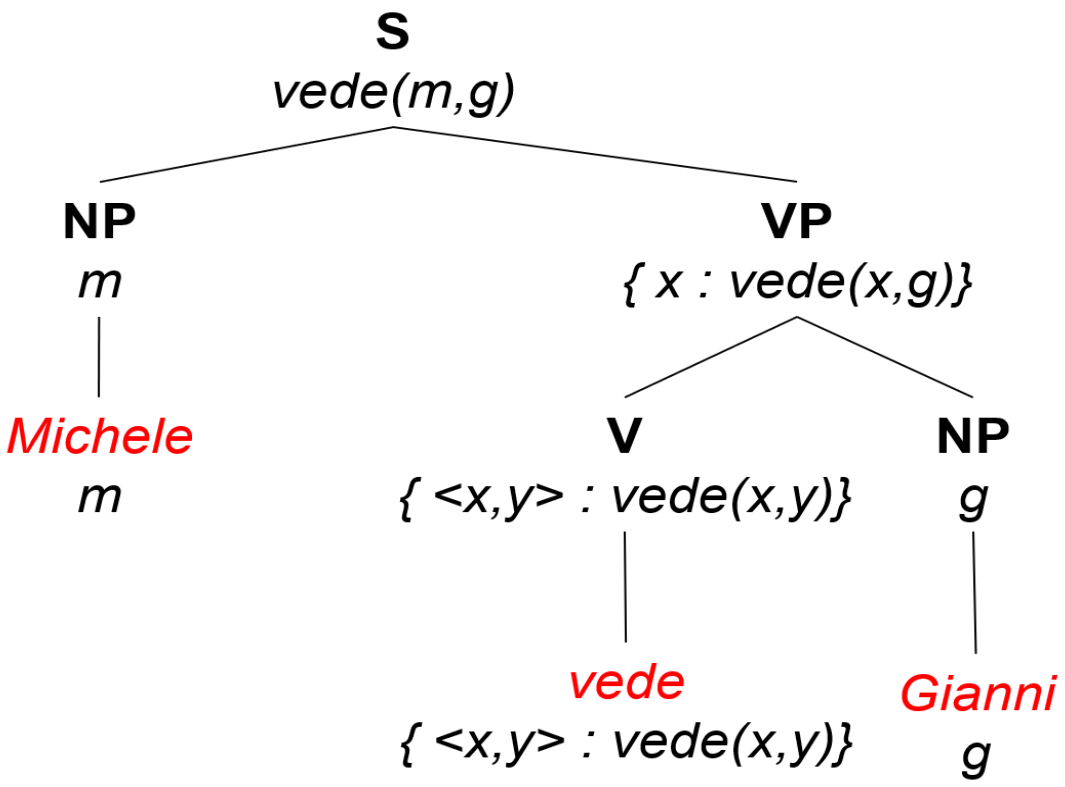
- Risultato: Fornire un sintassi per la *semantica*
- Qual'e' il processo di traduzione?

Composizionalita'

Il significato di una espressione e' una qualche funzione del significato delle sue parti e delle operazioni di combinazione (cioe' delle dipendenze/relazioni sintattiche). Ne segue che:

- Il significato di "*Michele vede Gianni*" e' una funzione del significato di "*Michele*" e di "*vede Gianni*"
- Il significato di "*vede Gianni*" e' una funzione del significato di "*vede*" e di "*Gianni*"
- L'interpretazione composizionale e' *ricorsiva* (rispetto alla sintassi)

Composizionalita'



Uso di FOL nella semantica composizionale

- La FOL ha una semantica composizionale: quindi va garantita una traduzione *verso* la FOL che sia composizionale
- Inoltre, il significato delle espressioni linguistiche deve essere messo in corrispondenza sistematica con il significato delle *parti* componenti
- Serve:
  - una traduzione semantica per ogni espressione di base o complessa
  - una operazione semantica per ogni regola sintattica di composizione

Il principio di composizionalita'

- Ogni regola sintattica puo' essere vista come una funzione che dati dei (o delle combinazioni di) morfemi produce una espressione in uscita
- Ogni regola sintattica R applicata ai morfemi $\alpha_1, \alpha_2, ..., \alpha_n$ produce una espressione in uscita ξ del tipo

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

- E' ragionevole assumere che ogni elemento atomico del linguaggio α (ad es. nomi) sia in corrispondenza, nel mondo, con una precisa entita' o proprieta' o relazione $sem(\alpha)$ (es. i nomi propri mappano verso individui/entita' nel mondo)
- Ne segue che ogni relazione R ammette una controparte semantica R' tale che:
se $\xi = R(\alpha_1, ..., \alpha_n)$ allora $sem(\xi) = R'(sem(\alpha_1), ..., sem(\alpha_n))$

Regole context-free e Logica

- Un Linguaggio è definito dall'insieme di regole di riscrittura che ne caratterizzano le strutture a partire da uno specifico non terminale S che costituisce la rappresentazione corrispondente. Ad esempio

$$S \rightarrow NP\ VP$$

  è una possibile descrizione delle frasi dichiarative soggetto-predicato che sono molto comuni in Italiano
- Esempio sono frasi come:
  "Gianni mangia", "Maria dorme di notte", La zia donò l'orologio a Mattia lo scorso anno, ..

Regole context-free e Logica (2)

- Le regole sintattiche possono essere formalizzate sia come regole di riscrittura (come da sempre i corsi di algoritmica le descrivono) ma anche, da un punto di vista logico, come assiomatizzazioni di una certa teoria sul linguaggio che si intende descrivere.
- Nell'esempio
  $S \rightarrow NP\ VP$
  può essere visto come un assioma del tipo

$$\forall x(\exists y, z \quad NP(y) \wedge VP(z) \wedge concat(y, z, x)) \rightarrow S(x) \qquad (1)$$

- La regola 1 descrive una coppia di sottofrasi y e z della x che riscritte nei non terminali NP e VP rispettivamente qualificano la x come frase (S) del linguaggio.

Parsing come deduzione in Logica

Le regole sintattiche come la (1) possono essere facilmente espresse dunque in Prolog, dove il formalismo dichiarativo proprio della logica coincide in modo diretto con le regole di riscrittura di una grammatica. Infatti la regola descritta nella Eq. (1)[1], corrisponde al seguente frammento Prolog

- `s(X) :- append(Y,Z,X), np(Y), vp(Z)`
  dove
  - `append(Y,Z,X)` consente di enumerare tutte le coppie di sottosequenze `Y` e `Z` che concatenate ricoprono `X`, cioè la frase in input
  - le proprietà `np(Y)` e `vp(Z)` ricorsivamente garantiscono le `Y` e `Z` svolgano il ruolo di frase Soggetto e frase Verbale corrispondentemente.

[1] $\forall x(\exists y,z \quad NP(y) \wedge VP(z) \wedge concat(y,z,x)) \rightarrow S(x)$

Parsing come deduzione in Logica (2)

Le regole sintattiche come quella corrispondente al seguente frammento Prolog

```
s(X) :- append(Y,Z,X), np(Y), vp(Z)
```

danno luogo ad una processo di parsing grazie all'interprete Prolog. Questo proverà prima a decomporre la frase in ingresso[2] `X` in due sottofrasi `Y` e `Z`, poi applicherà ricorsivamente ad esse i criteri della riscrittura nei due non terminali `NP` e `VP` rispettivamente.

[2] Vedi predicato `append` a pagina successiva.

Digressione: il predicato `append/3`

Il seguente predicato Prolog

```
append([],X,X).
append([HY|TailY],Z, [HY|X]) :-
    append(TailY,Z,X).
```

può essere usato sia per calcolare la concatenazione delle due liste in ingresso, cioè

`?-append([a,b], [c], X).` con `X = [a,b,c]`

che per enumerare le due sottoliste di una lista data, come ad esempio in

`?-append(Y, Z, [a,c]).`

con `Y=[], Z=[a,c]; Y=[a], Z=[c]; Y=[a,c], Z=[]`

Parsing di un esempio: "Giovanni dorme'

Il seguente frammento di grammatica:

```
s(X) :- append(Y,Z,X), np(Y), vp(Z)
np(X) :- pn(X)
vp(X) :- v(X)
```

dovrebbe essere sufficiente per la frase

Giovanni dorme

L'unico requisito è descrivere come in dizionario, gli assiomi per rappresentare la categoria grammaticale delle due parole qui coinvolte di *''Giovanni'* e *'dorme'*. Esse sono un *nome proprio* ed un *verbo*, rispettivamente e quindi sono sufficienti i seguenti fatti Prolog:

```
pn(['Giovanni']).
v(['dorme']).
```

Parsing e riscrittura di un esempio: "Giovanni dorme'

Per **riscrivere** l'output della interpretazione della lista in ingresso in una struttura che rende conto della interpretazione dei diversi elementi della sintassi, è possibile estendere le regole della grammatica aggiungendo un argomento nell'ouput di ogni regola, come segue:

```
s(X, s(NP,VP)) :-
   append(Y,Z,X), np(Y, NP), vp(Z, VP).
np(X, pn(X)) :- pn(X).
vp(X, v(X)) :- v(X).
```

Con questa estensione il parsing della frase *Giovanni dorme* si esegue nel modo seguente:

`?-s(['Giovanni', 'dorme'], Tree).`

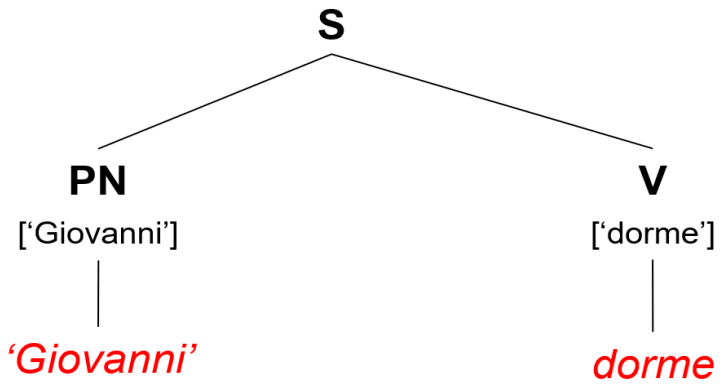
con `Tree=s(pn(['Giovanni']), v(['dorme']))`.

che enfatizza la struttura ad albero della interpretazione.

Esempio: albero di "Giovanni dorme"'

```
?-s(['Giovanni', 'dorme'], Tree).
```

con `Tree=s(pn(['Giovanni']), v(['dorme']))`



Esercizio: completa la grammatica su frasi esempio

Estendere la grammatica e le regole lessicali sin qui descritte per elaborare con successo le seguenti frasi

```
sent(0,['Mario',mangia]).
sent(1,['Giovanni',scrisse,un,poema]).
sent(2,[il,libro,fu,scritto,da,due,autori]).
sent(3,[le,prime,sezioni,erano,noiose]).
sent(4,[le,ore,insonni,scorrevano,nella,notte,a,'Roma']).
sent(5,['Giovanna',di,notte,discorreva,del,
                                primo,capitolo,del,libro]).
```

Si aggiungano le regole per frasi verbali che includano il complemento oggetto (Frase 2) e gli altri complementi (Frasi 4 e 5), e si aggiungano le parole (verbi, nomi e aggettivi) al dizionario[3].

[3] Vedi implementazione della interfaccia utente nel file "Esercizio Parsing e Logica" distribuito con queste slide

Composizionalita' logica: dalla sintassi alla semantica delle lingue

Semantica delle espressioni linguistiche

La costruzione di una forma logica richiede la espressione completa non già della sola struttura sintattica di una frase ma del predicato che ad essa corrisponde. Per le frasi dichiarative come

Kim saw Sam

il predicato risultante dovrà essere vero ogni volta che si crede alla frase, in modo che la deduzione di altre formule dalla forma logica ottenuta possa essere coerente con i significati tratti dai parlanti nella loro comunicazione. Una buona forma logica candidata è dunque:

saw(k,s)

dove *k* ed *s* sono costanti del calcolo e *saw* è un predicato.

Composizionalita' logica e semantica delle lingue

Le espressioni linguistiche S dunque si mappano in formule logiche $FL(S)$ con il principio "empirico" che chi crede nella frase S si impegna a riconoscere come vera la formula corrispondente $FL(S)$. Conseguentemente nomi propri e verbi sono mappati in una rappresentazione che si conformi ai vincoli della loro interpretazione logica. Entrano quindi in gioco in modo diverso la rappresentazione in costanti e predicati rispettivamente per nomi propri (individui) e verbi (azioni o stati).

Semantica delle espressioni

Kim	k
Sam	s
Kim left	`leave(k)`
Sam saw Kim	`saw(s,k)`

Composizionalita' in Prolog

Questa scelta incide sulla codifica in Prolog del lessico e delle regole grammaticali.

Semantica delle espressioni e termini Prolog

Kim	k
Sam	s
Kim left	`leave(k)`
Sam saw Kim	`saw(s,k)`

In Prolog diviene:

`pn(k, ['Kim']).` I Nomi propri sono costanti, kim > k

```
pn(s,['Sam']).
np(NPSem, NP) :- pn(NPSem, NP).
vp(VPSem, VP) :- iv(VPSem, VP).
```

`iv(leave(X), [left]).` verbi in predicazioni, es. leave/1

Composizionalita' in Prolog

Come gestire la semantica del non terminale `S`, in `S -> NP VP`? Com si può cioè esibire una riscrittura di un elemento semantico in uscita dalla soddisfazione di una regola? Abbiamo dunque bisogno di argomenti semantici espliciti come segue

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

Come gestire in questo senso i verbi transitivi che hanno due argomenti?

```
vp(VPSem, VP) :- tv(VSem, V), np(NPSem, NP).
```

ed una possibilità può essere:

```
tv(see(X,Y), [saw]).
```

Composizionalita' in Prolog: mappare gli argomenti dei verbi transitivi

Dato il modello proposto:
`vp(VPSem, VP) :- tv(VSem, V), np(NPSem, NP).`
`tv(see(X,Y), [saw]).` come possiamo unificare costanti, come `k`, con `Y` (invece che con `X`)?

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

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

Composizionalita' in Prolog - Problemi

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

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

- Una variable `V` come predicato (cattivo uso del Prolog);
- scarsa flessibilita', e.g. `give(X,Y,Z)`

Dobbiamo estendere il formalismo!

Un linguaggio formale per la semantica: λ-Calcolo

- *Giuseppe corre* produrrebbe: $corre(Giuseppe)$
- *Tutti gli studenti scrivono programmi* :
  $\forall x \quad studente(x) \quad \Rightarrow \quad (\exists p)(program(p) \& scrive(p,x))$
- Conseguenze principali:
  - I SV (VP) mappano in *simboli predicativi*
  - I nomi propri mappano in simboli atomici (instanziati)
  - la quantificazione genera strutture piu' complesse
- le forme logiche dei VP (denotate con VP') sono *funzioni* da entita' a proposizioni

Funzioni espresse nel λ-Calcolo

- $f(x) = x + 1$
- Una migliore astrazione della natura di f puo' essere ottenuta nel seguente modo

$$\lambda x.x + 1$$

  - $(\lambda x.x+1)(3)$ $((\lambda x.(x+1))(3)$) e' equivalente a $3+1$
- Conseguenze principali:
  Non ho nomi per ogni funzione
  Basta descrivere le operazioni Ω necessarie a calcolare f
- β-reduction: $(\lambda x.\Omega)a$ genera $[\Omega]\{x = a\}$ mentre,
  $(\lambda x.\lambda y.\Omega)(a)(b) = \lambda y.\Omega\{x = a\}(b) = [\Omega]\{x = a, y = b\}$

λ-Calcolo: Sintassi

Se ϕ e' una formula e v una variabile allora $\lambda v.\phi$ e' un predicato. Piu' in generale se ψ e' un predicato n-ario e v e' una variabile, allora $\lambda v.\psi$ e' un predicato $n+1$-ario.

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

λ-Calcolo: Semantica

Se ϕ e' una formula e v una variabile allora $\lambda v.\phi$ e' la funzione caratteristica dell'insieme degli oggetti che **soddisfano** ϕ (cioe' lo rendono vero).

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

β-reduction e Semantica Composizionale

Le seguenti espressioni sono equivalenti:

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

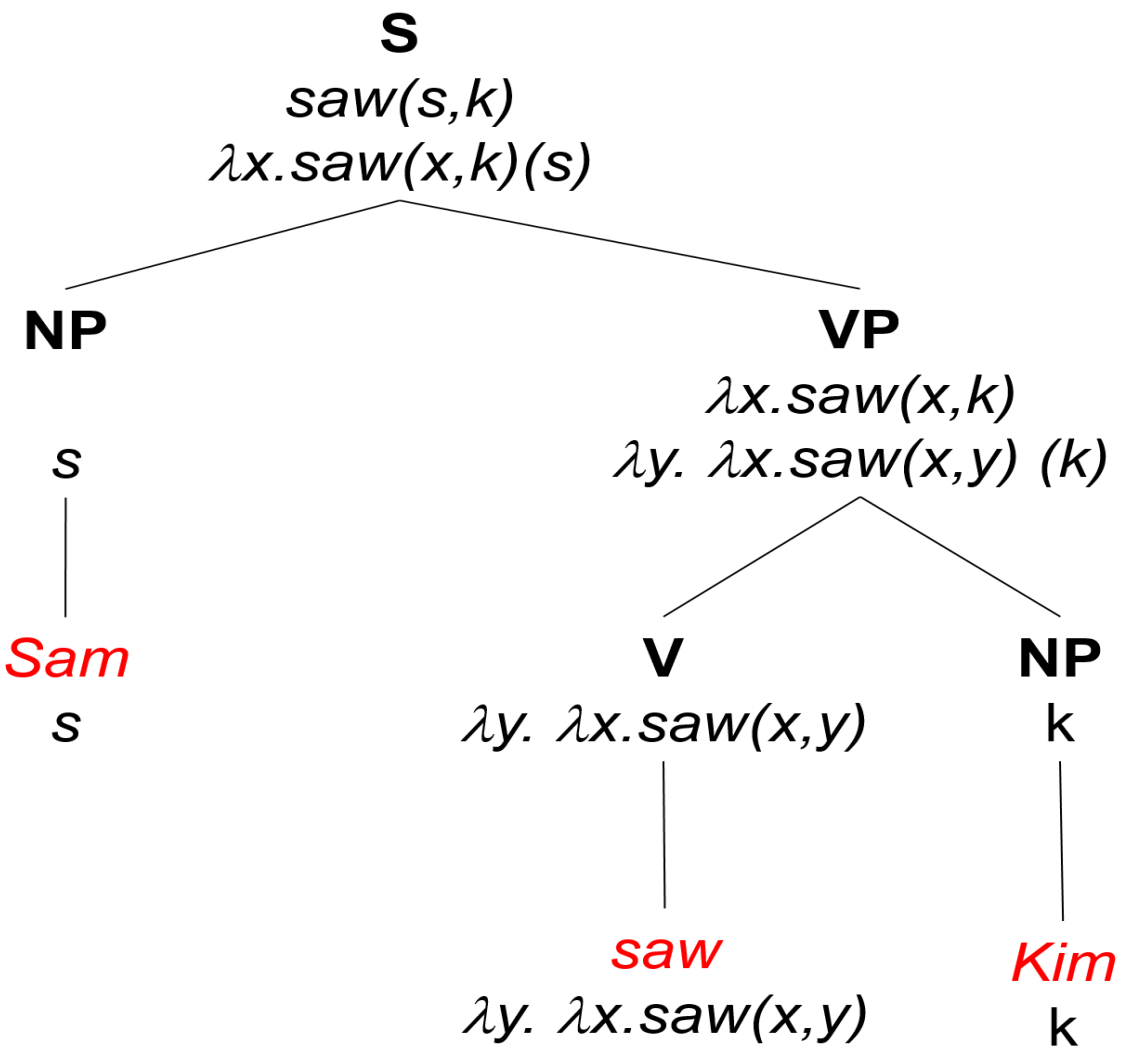
Possiamo ridurre il calcolo della semantica composizionale ad processo ricorsivo di applicazione di funzioni.

β-reduction

La *beta*-riduzione $(\lambda x.\Omega)a$ opera sostituendo contemporaneamente **tutte** le occorrenze (libere) della variabile x in Ω con la espressione a.

Operazione	Λ-Espressione	Risultato
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 e Semantica Composizionale



β-reduction e Semantica Composizionale

- *Giuseppe corre* $corre(giuseppe)$
- $S \quad \rightarrow \quad NP\ VP$
- Sem Rule1 (*verbi intransitivi*):
  SE la Forma Logica (FL) di NP e' NP' e la FL di VP e' VP' :
  ALLORA la FL di S' = VP'(NP')
- Conseguenze:
  la semantica di *corre*, $Sem(''corre'')$, è $\lambda x.corre(x)$
  la semantica $Sem(''Giuseppe'')$ è la costante *giuseppe*
- Da cui:
  $S' = VP'(NP') = (\lambda x.corre(x))(giuseppe) = corre(giuseppe)$

β-reduction e Semantica Composizionale (2)

- *Giuseppe usa Prolog* $usa(giuseppe, prolog)$
- $VP \quad \rightarrow \quad V\ NP$
- Sem Rule2 (*verbi transitivi*):
  SE la FL di NP e' NP' e la FL di V e' V' :
  ALLORA la FL di VP' = V'(NP')
- Conseguenze (nella modellazione di V'):
  La semantica di *usa* è $\lambda x.\lambda y.usa(y,x)$
- Da cui: $S' = VP'(NP'_0) =$
  $= V'(NP'_1)(NP'_0) = (\lambda x.\lambda y.usa(y,x))(prolog)(giuseppe) =$
  $= usa(giuseppe, prolog)$

A first example: a little grammar of English

Sarebbe ideale un formalismo come:

```
 S   =>  NP VP

NP   =>  Art N
VP   =>  VSemplice, NP
VP   =>  VSemplice

Art =>  "il"
Art  =>  "un"
N  =>  "gatto"
N  =>  "topo"
VSemplice =>  "mangia"
```

Usiamo le liste Prolog ...

```
 S   =>  NP, VP.

NP   =>  Art, N.
VP   =>  VSemplice, NP
VP   =>  VSemplice.

Art =>  [il]
Art  =>  [un].
N  =>  [gatto].
N  =>  [topo].
VSemplice =>  [mangia].
```

Cosa Manca? ...

... consumare i terminali

Data una lista che rappresenta una frase `ListaF` in ingresso, una regola come

```
S => NP, VP
```

ESPRIME una predicazione logica del tipo:

```
s(ListaF, []) :- np(ListaF, X), vp(X,[])
```

che significa:
affinchè `ListaF` rappresenti una lista di tipo `S`, deve esistere una lista `X` (cioè il resto della frase dopo che la frase nominale `NP` e' stata consumata) che corrisponde completamente ad una frase del tipo `VP`.
Esempio: data(`N =>[topo]`), se `ListaF` = `[topo]` allora `X` = `[]`.
OSS: questo processo per cui ogni non terminale consuma una lista `ListaF` − `X` puo' essere generalizzato.

Il formalismo DCG

Poichè il comportamento di regole del tipo `S => NP, VP` è del tutto generale, introduciamo una notazione esplicita per la **riscrittura degli elementi della lista** che implicitamente applica i predicati nel corpo di una regola come invocazioni ricorsive su liste parziali. In questo modo la scrittura :

```
s(ListaF, []) :- np(ListaF, X), vp(X,[])
```

corrisponde alla espressione equivalente in DCG seguente:

```
s -> np, vp
```

dove i due argomenti, lista in ingresso e lista in uscita di ogni simbolo non terminale vengono omessi. Le regole per i simboli terminali divengono dunque:

```
n -> [topo]
```

dove l'argomento lista viene esplicitato (e la lista vuota omessa).

Il formalismo DCG: riscrittura

La scrittura :

```
s(ListaF, []) :- np(ListaF, X), vp(X,[])
```

può ulteriormente essere estesa a contenere l'esito della interpretazione sintattica. Immaginiamo di volere restituire una struttura arborea della sintassi, in un argomento separato. Potremmo dunque sfruttare l'unificazione Prolog come segue :

```
s( s(NP,VP) ) -> np(NP), vp(VP)
```

qui i due non terminali `NP, VP` producono in uscita due riscritture `NP e VP` e questi possono trovare posto nei due argomenti distinti in uscita dalla soddisfazione della regola `S`.
OSS: i due argomento impliciti (lista in ingresso e lista in ucita) rimangono ma vengono estesi con l'esito della riscrittura.

Il formalismo DCG: esempio di riscrittura

Il frammento di grammatica:

```
s( s(np(NP), vp(VP)) ) --> np(NP), vp(VP).
np(NPr) --> nproprio(NPr).
vp(VP) --> vSemplice(VP).

nproprio(npro(gianni)) --> ['Gianni'].
vSemplice(v(corre)) --> ['corre'].
```

genera a fronte della query:

```
?-s(O, ['Gianni', 'corre'], []).
```

la riscrittura `O` della frase in ingresso nella struttura ricorsiva dell'albero sintattico:

```
O = s( np(npro(gianni)), vp(v(corre))).
```

Il formalismo DCG: riscrittura e semantica della frase

La generazione di strutture sintattiche come riscrittura realizzato in:

```
?-s(O, ['Gianni', `corre'], []).
  O = s( np(npro(gianni)), vp(v(corre))).
```

ci aiuta a definire una riscrittura analoga anche per la forma logica, cioè l'espressione logica che vogliamo corrisponda al significato delle frasi. Nel nostro caso vorremmo:

```
?-s( Sem, ['Gianni', `corre'], []).
  Sem = corre(gianni).
```

Semantica Composizionale in Prolog

$S \rightarrow \quad NP\ VP$

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

`SP` deve essere prodotto composizionalmente da `NP` e `VP`. COME:

Applico `VP` a `NP` *!!!!*

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

Semantica Composizionale in Prolog

L'elaborazione dell'esempio

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

è la seguente:

```
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],[]))
```

Semantica Composizionale in Prolog (2)

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

I verbi transitivi hanno una forma lessicale diversa.

```
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) )}
```

Interpretazione dei verbi (tr/intr)

- Tutte le frasi del tipo transitivo ed intransitivo semplice sono soggette quindi:
  - Ad una grammatica DCG
    ```
    vp(VP) -> tv(TVP), np(NP).
    ```
  - A meccanismi per la gestione della composizionalita'
    ```
    s(S) --> np(NP), vp(VP), {betareduce(VP,NP,S)}.
      betareduce(Arg^Expr, Arg, Expr).
    ```
    o piu' sinteticamente
    ```
    s(S) --> np(Arg), vp(Arg^S).
    ```
  - Ad un lessico in grado di coprire le varie unita' semplici
    ```
    tv(X^Y^usa(Y,X)) --> [usa].
    ```

Alcune riflessioni

- La semantica composizionale dipende fortemente dal *lessico* (verbi e nomi):
- Il numero degli argomenti e' *variabile da verbo a verbo* e ...
- ... da *senso verbale* a *senso verbale* (es. *operare in clinica* vs. *operare in un mercato*)
- E' possibile codificare nel lessico anche alcune *regole di preferenza* per i fenomeni ambigui (per es. le associazioni con i sintagmi preposizionali, cioè in *dare qualcosa* **a qualcuno** ...)
- Qual'e' il ruolo del *dominio/ambiente* in questi meccanismi? Posso sfruttare le aspettative del dominio per ridurre la ambiguità?

Sommario (1.1)

- La analisi semantica ha come obbiettivo la generazione di una rappresentazione vero-condizionale del significato delle frasi in LN
- La semantica composizionale richiede un processo ricorsivo basato sulla sintassi
- Lo strumento offerto dal paradigma di elaborazione funzionale riduce il processo ricorsivo alla applicazione di funzioni di carattere grammaticale e lessicale
- Sono stati presentati i modelli di rappresentazione semantica degli elementi principali del lessico: nomi comuni, nomi propri, verbi transitivi e intransitivi

Sommario (1.2)

- Nel formalismo DCG del Prolog e' stata implementata la analisi semantica basata su
  - Unificazione (nelle operazioni di β-reduction)
  - L'analisi *depth-first* dell'interprete
  - Modello dichiarativo del lessico

Bibliografia

- Prolog Implementation: SWI-Prolog, URL: `https://www.swi-prolog.org/`
- Prolog Basics. `https://lpn.swi-prolog.org/lpnpage.php?pageid=online`
- 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.
- *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.

Contenuti (2)

- Alcuni fenomeni linguistici non sono stati ancora trattati
  - Argomenti verbali espressi da preposizioni
  - Ruoli tematici
  - Quantificazione
- Il ruolo del lessico
- Il ruolo del dominio

I sintagmi preposizionali

- I sintagmi preposizionali possono avere diversi ruoli nella descrizione semantica
  - Argomenti verbali espressi da preposizioni
    Mario da' a Gianni una penna
  - Modificatori accidentali
    Mario da' la penna a Gianni in affitto | *con affetto*
    Mario da' la penna a Gianni in cucina
  - Argomenti *vuoti*
    John parlava di matematica → `parlare_di(j,m)`

Trattamento dei modificatori preposizionali senza semantica

- *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) )      -->
        [relied].
```

Trattamento dei modificatori preposizionali senza semantica

Esercizio. Gestire la semantica *vuota* delle preposizioni i.e.

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

in modo coerente con le altre costruzioni, e.g.

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

Idea:

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

Trattamento dei modificatori preposizionali argomentali

- *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)`

Trattamento dei modificatori preposizionali argomentali

Sono (in Inglese) verbi cosiddetti "*ditransitivi*",
con oggetti preposizionali nucleici,
che generano *predicati triadici*.

```
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) ) -->
        [da'].
v(3/da, Z^Y^X^comprare(X,Y,Z) )  -->
        [compra].
```

Trattamento dei modificatori preposizionali argomentali

Esercizio

Scrivere un frammento della grammatica in grado di interpretare il caso di
Gianni parla del libro a Michele → `parlare(g,l,m)`
a partire dalle definizioni di `vp()` e `pp()` opportune

Generalizzare la soluzione per la gestione del movimento dei modificatori, i.e.:

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

Trattamento dei verbi ditransitivi

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

La *FL* e' invariante rispetto alle due forme

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

Trattamento dei verbi ditransitivi

Abbiamo bisogno di due regole distinte per lo stesso verbo che tengano conto delle due strutture

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

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

v(4, Z^Y^X^give(X,Y,Z) ) -->
        [gave].
```

Qui la semantica e' unica per le due forme.

Trattamento dei verbi ditransitivi

```
 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).
```

Osservazione: Sussiste una assunzione fondamentale sui "ruoli" svolti dagli argomenti del predicato

Altre rappresentazioni Semantiche

La rappresentazione scelta per le espressioni linguistiche non e' unica. Ad esempio, potremmo esplicitare il ruolo dei singoli argomenti facendo uso di una lista di coppie, e.g.

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

o ancora esplicitare il ruolo dei singoli argomenti mediante una struttura predicativa, e.g.

John saw Mary →

```
some(E,[seeing(E),agent(E,j),theme(E,m),before(E,now)])
```

Altre rappresentazioni Semantiche

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].
```

Fenomeni Lessicali

- Modificatori Preposizionali Argomentali

  l'uomo bevve birra tutta la notte

  la macchina beveva troppo gasolio
- Forma logica:

  $beve(uomo, birra)\ldots$

  $bere(macchina, gasolio)$????

Fenomeni Lessicali

- Modificatori Preposizionali di sintagmi nominali

  lo zio di Mario

  il libro di Mario

- Forma logica:

  $parente(zio,'Mario')$

  $possessore(libro,'Mario')$

Ruolo del Lessico

In entrambi i casi e' necessario codificare le relative interpretazione secondo *regole lessicali*

- Distinzione tra i **sensi** delle parole ($bere_{ingerire}$ vs. $bere_{consumare}$)
- Regole di restrizione all'uso (*selectional restrictions*$^{(*)}$) dei modificatori

  il libro di Mario, ... di storia, ... di sogni , ... d'acqua residente a Roma, ... a Gennaio, ... a motore, ... ad acqua
- Regole di interpretazione semantica (interfaccia sintassi-semantica)
  $parente(zio, 'Mario')$ vs. $possessore(libro, 'Mario')$

(*) Chomsky, N. 1965. Aspects of the Theory of Syntax, MIT Press, Cambridge, MA

Interpretazione Semantica dei modificatori preposizionali

Per interpretare correttamente un sintagma preposizionale, ad es. residente *a Roma* vs. ... *a Gennaio*, oppure motore *a poppa* vs. ... *a benzina*, sono **necessarie** (almeno) le seguenti informazioni:

1) I diversi **sensi** di una preposizione

2) La **semantica del modificatore** coinvolto: *poppa* vs. *benzina* oppure *Roma* vs. *Gennaio*.
Cio' corrisponde ad una descrizione del significato (senso) delle parole, ma anche della descrizione di classi generali di parole, ad es. i tempi (*Gennaio* vs. .... *Dicembre*).
Tali classi costituiscono categorie di rappresentazione del mondo (modello del mondo/dominio)

3) Regole per **comporre** le informazioni in 1) con quelle in 2)

Un esempio: postmodificatori nominali

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].
```

Interpretazione dei postmodificatori nominali

```
%-----------------------------------------------------------------
...
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).
```

I predicati `tc_isa(X,Y)` stabiliscono il ragionamento (sussunzione) lungo la gerarchia delle categorie di rappresentazione del mondo, cioè lungo modello del dominio.

Esempio: mini modello del mondo

```
% Toy Ontology for the Mini Semantic Interpreter

% Mostly General Concepts
top(luogo).
top(tempo).
top(animate).
top(abstraction).

% IS_A Hierarchy
is_a(luogo_naturale, luogo).
is_a(luogo_artificiale,luogo).
is_a(oggetto,luogo_artificiale).
is_a(manufatto,oggetto).
is_a(campagna,luogo_naturale).
is_a(lattina,manufatto).

is_a(giorno,tempo).
is_a(mese,tempo).
is_a(anno,tempo).

is_a(persona,animate).
is_a(animale,animate).
is_a(parente,persona).
is_a(zio,parente).

is_a(discipline, abstraction).
```

Esempio: dalle parole alle categorie

```
% Proper Noun Classes
is_a(person_np, persona).
is_a(artifact_np, manufatto).
is_a(location_np, luogo_naturale).
is_a(city_np, luogo_naturale).
is_a(discipline_np, discipline).
is_a(year_np,anno).
is_a(day_np,giorno).
is_a(month_np,mese).


% Instances/Individuals in the Ontology
%Person Names
np_is('Mario',person_np).
np_is(giuseppe,person_np).
np_is('Giuseppe',person_np).
np_is('Maria',person_np).
np_is('Fabio',person_np).

%Products/Artifacts
np_is(prolog,artifact_np).
np_is('C',artifact_np).
np_is(libro,artifact_np).
np_is(matematica,discipline_np).
```

Esempio: navigare tra le categorie

```
% Transitive closure of the basic IS_A relation

%Treatment of Proper Nouns
tc_isa(X,Y) :-
   np_is(X, Classe_NP),
   !,
   tc_isa(Classe_NP,Y).

%General Case
tc_isa(X,Y) :-
   is_a(X,Y),
   !.
tc_isa(X,Y) :-
   is_a(X,Z),
   !,
   tc_isa(Z,Y).
```

Esempio: analisi semantica e disambiguazione del senso

```
?- s(L, ['Maria',ama,il,libro,di,'Mario'],[]).
L = amare('Maria', libro)&possessor(libro, 'Mario') ;
false.

?- s(L, ['Maria',ama,lo,zio,di,'Mario'],[]).
L = amare('Maria', zio)&parente(zio, 'Mario') ;
false.
%-----------------------------------------
?- s(L, ['Giuseppe',corre,in,campagna],[]).
L = corre('Giuseppe')&luogo(campagna) .

?- s(L, ['Giuseppe',corre,in,'Gennaio'],[]).
L = corre('Giuseppe')&tempo('Gennaio') .
```

Esempio: analisi semantica e disambiguazione del senso

```
%-----------------------------------------
?- s(L, ['Maria',corre,con,lo,zio,di,'Mario'],[]).
L = corre('Maria')&in_compagnia_di(zio)
                  &parente(zio, 'Mario') .

?- s(L, ['Maria',corre,con,lo,zio,in,'Gennaio'],[]).
L = corre('Maria')&in_compagnia_di(zio)
                  &tempo('Gennaio') .
```

Esercizi Proposti

Si scrivano le modifiche al codice ricevuto per la lezione, al fine di riconoscere le seguenti frasi e produrre la loro forma logica.

- *Giuseppe ama Maria*
- *Giuseppe ama la sorella di Maria*
- *Giuseppe parla di Maria alla sorella di Mario*
- *Il gatto mangia il topo in cucina*
- *I gatti mangiano ogni giorno.*
- *Il libro di storia parla di Picasso.*
- *Il libro di storia parla di Guernica a pagina 5.*

Discutere eventuali interpretazoni multiple.

Esercizi Proposti (2)

Introdurre vincoli lessicali per la gestione dell'accordo in genere e numero in modo da accettare (o scongiurare, vedi simbolo *) le seguenti frasi

- **Giuseppe amano Maria*
- *I cani amano Maria*
- **Mario parlano di Maria alla sorella di Giuseppe*
- *I gatti parlano alla sorella di Giuseppe*

Discutere eventuali interpretazoni multiple.

Gestione dei Quantificatori

- Data una frase come: "*Ogni ingegnere studia*" e rappresentabile con una sintassi del tipo:

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

  e' evidente come la frase nominale "*Ogni ingegnere*" e' soggetta ad una quantificazione.
- Una forma logica coerente con le nostre intuizioni e' quindi:

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

Quantificatori e λ-calcolo

- La quantificazione delle frasi nominali puo' essere catturata dal lessico mediante la seguente λ-astrazione associata alla frase "*Ogni ingegnere*":

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

  E' evidente pero' che in tal caso nella regola DCG `s(SP) -> np(NP), vp(VP)` e' la semantica della frase nominale NP' (dato da `NP`) che si applica alla semantica della frase verbale VP' (`VP`), cioe' $NP'(VP')$, e non il viceversa come prima suggerito dalla grammatica precedente.
- Infatti se $VP' = \lambda y.studia(y)$ allora $NP'(VP'))$ risulta:

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

Quantificatori e λ-calcolo (2)

La frase nominale "*Ogni ingegnere*" e' descritta grammaticalmente da

```
np(NP) -> det(DT), n(N), ...  %dove DT e' il determiner
```

E' chiaro che una semantica composizionale per `NP` puo' essere derivata analogamente da una β-riduzione su opportune forme lessicali associate a "*Ogni*" (`DT`) e "*ingegnere*" (`N`)

"*Ogni ingegnere*" quindi puo' essere descritta completamente dalla seguente regola DCG:

```
np(NPSem) -> det(DTSem), n(NSem),
 betareduce(DTSem,NSem,NPSem)
```

date nel lessico le seguenti definizioni per `DT` e `N` rispettivamente

`DT` $\lambda p.\lambda q.(\forall x)p(x) \Rightarrow q(x)$

`N` $\lambda y.ingegnere(y)$

Quindi i nomi come "*ingegnere*" generano <u>proprieta'</u> cioe' predicati (unari), similmente ai verbi (intransitivi)

Quantificatori e λ-calcolo (3)

La frase "*Ogni ingegnere studia*", descritta grammaticalmente da

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

Genera quindi la seguente catena di β-riduzioni:

`NP=DT(N):`

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

`S=NP(VP):`

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

Gestione dei Quantificatori in Prolog

- Per la gestione dei quantificatori in Prolog abbiamo bisogno di modellare le espressioni:
  $\forall x \quad P(x)$ e $\exists x \quad P(x)$
- Questo e' possibile introducendo due predicati speciali `all/2` e `exist/2` ed usando i vincoli determinati dalla unificazione
- Una loro espressione Prolog soddisfacente puo' quindi essere
  $\forall x \quad P(x)$: `all(X, p(X))`
  $\exists x \quad P(x)$: `exist(X, p(X))`

Gestione dei Quantificatori in Prolog (2)

- Osserviamo che P in
  $\forall x \quad P(x)$ ed $\exists x \quad P(x)$
  puo' essere complesso come nel caso osservato nella descrizione semantica di "*ogni*".
- Anche qui le strutture complesse del Prolog ci offrono un utile strumento di espressione sintattica come segue:
  $\forall x \quad P(x) \Rightarrow Q(x)$: `all(X, p(X) => q(X))`
  $\exists x \quad P(x) \Rightarrow Q(x)$: `exist(X, p(X) => q(X)`
  con l'ipotesi che => venga definito come operatore binario infisso con la seguente dichiarazione Prolog
  `:-op(500, xfy, =>).`

Gestione dei Quantificatori in Prolog (3)

Utilizzando ora i due predicati introdotti e la β-riduzione possiamo definire le strutture lessicali che possono gestire la quantificazione.
Date le seguenti forme lessicali:
ingegnere: $\lambda y.ingegnere(y)$
studia: $\lambda y.studia(y)$
ogni: $\lambda p.\lambda q.(\forall x)p(x) \Rightarrow q(x)$

Possiamo quindi in Prolog descrivere rispettivamente il lessico come:

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

Gestione dei Quantificatori in Prolog (4)

Infine le regole DCG non lessicali cambiano 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 piu' sinteticamente sruttando appieno i vincoli dati dalla unificazione:

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

Gestione dei Quantificatori in Prolog - Esercizi

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*

Sommario (2.1)

- Nel formalismo DCG del Prolog e' stata implementata la analisi semantica basata sulle properieta' dell'interprete
- Sono stati trattati alcuni fenomeni particolari:
  - I modificatori preposizionali *vuoti* (e.g. senza semantica)
  - I modificatori preposizionali argomentali mediante la proiezione in predicati *n*-ari
  - L'equivalenza semantica di forme verbali strutturalmente distinte (i.e. verbi ditransitivi)
  - Dipendenze lessicali dell'interpretazione semantica
  - (*) I quantificatori

Bibliografia

- Prolog Implementation: SWI-Prolog, URL: `https://www.swi-prolog.org/`
- Prolog Basics. `https://lpn.swi-prolog.org/lpnpage.php?pageid=online`
- 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.
- *Intelligenza Artificiale*, S. J. Russel, P. Norvig, Prentice Hall Int., Chapter 22.3-22.8, 23, 1998.