

Sviluppo di un Parser in Prolog secondo la DCG per l'analisi di frasi in lingua italiana

Corso di Intelligenza Artificiale
A.A. 2013/2024

Alessio Capriotti, Davide De Grazia, Enrico Piergallini

Natural Language Processing (NLP)

è un ramo dell'AI che si occupa di sviluppare modelli computazionali allo scopo di conferire ai computer la capacità di comprendere il testo e le parole pronunciate in modo simile a quanto possono fare gli esseri umani.



LINGUAGGIO DI PROGRAMMAZIONE

(PYTHON, C++, PROLOG, ...)

- Rappresenta un numero finito di operazioni matematiche
- E' altamente preciso e strutturato
- E' deterministico
- E' univoco

LINGUAGGIO NATURALE

(ITALIANO, INGLESE, FRANCESE, ...)

- E' utilizzato dagli esseri umani per comunicare, per esprimere idee, sentimenti, concetti vaghi
- Flessibilità nella comprensione
- E' indeterminato
- E' ambiguo

SISTEMI DI REGOLE FORMALI

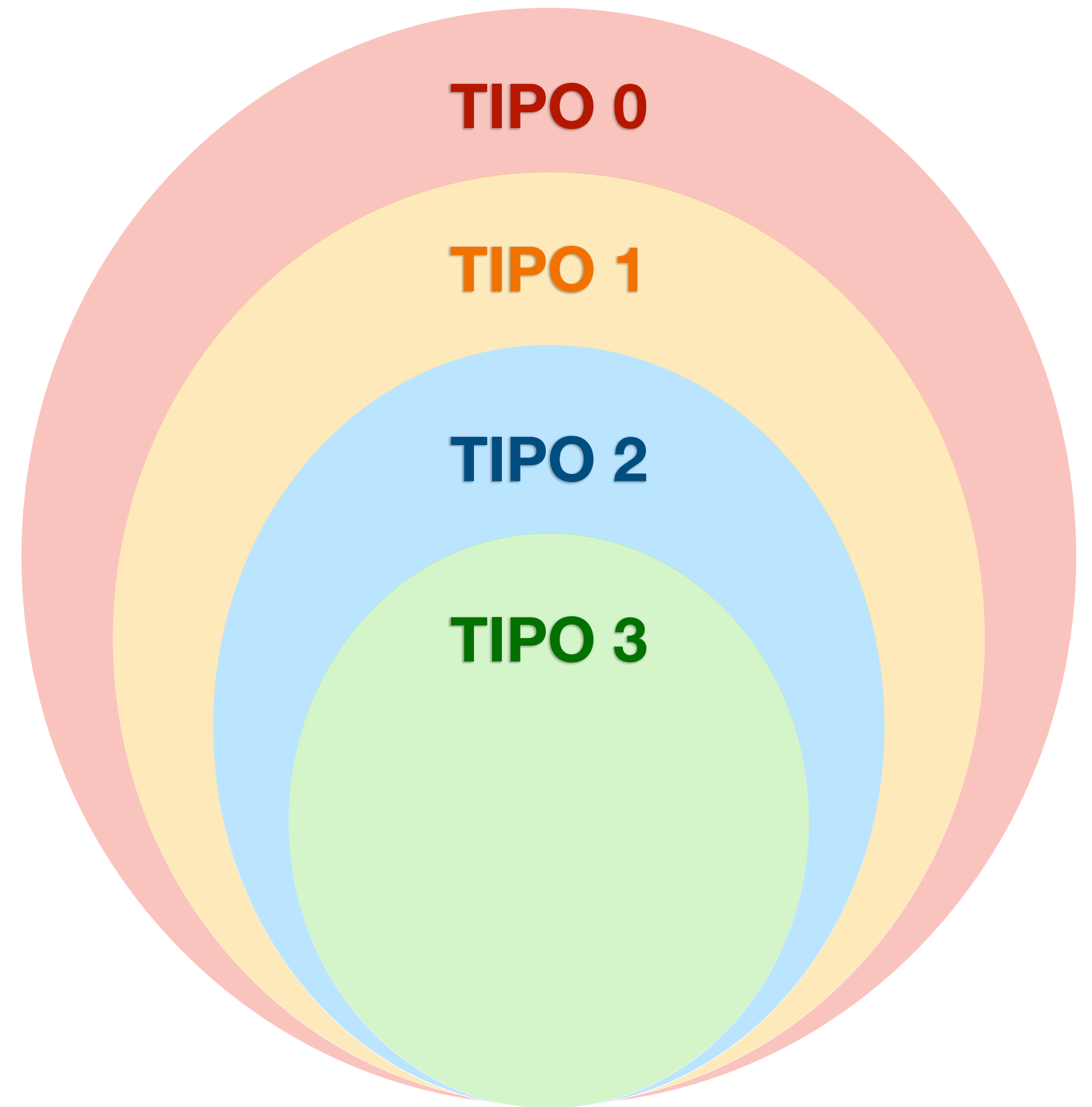
GRAMMATICA FORMALE

E' una notazione matematica che consente di esprimere in modo rigoroso la sintassi di un linguaggio naturale.

GERARCHIA DI CHOMSKY

E' un insieme di classi di grammatiche formali che generano linguaggi formali.

- GRAMMATICHE REGOLARI
(TIPO 3)
- GRAMMATICHE LIBERE DAL CONTESTO
(TIPO 2)
- GRAMMATICHE SENSIBILI AL CONTESTO
(TIPO 1)
- GRAMMATICHE NON LIMITATE
(TIPO 0)



GRAMMATICA LIBERA DAL CONTESTO

(CONTEXT-FREE GRAMMAR)

$$G = \langle C, T, R, \Sigma \rangle$$

SIMBOLI
NON TERMINALI

SIMBOLI
TERMINALI

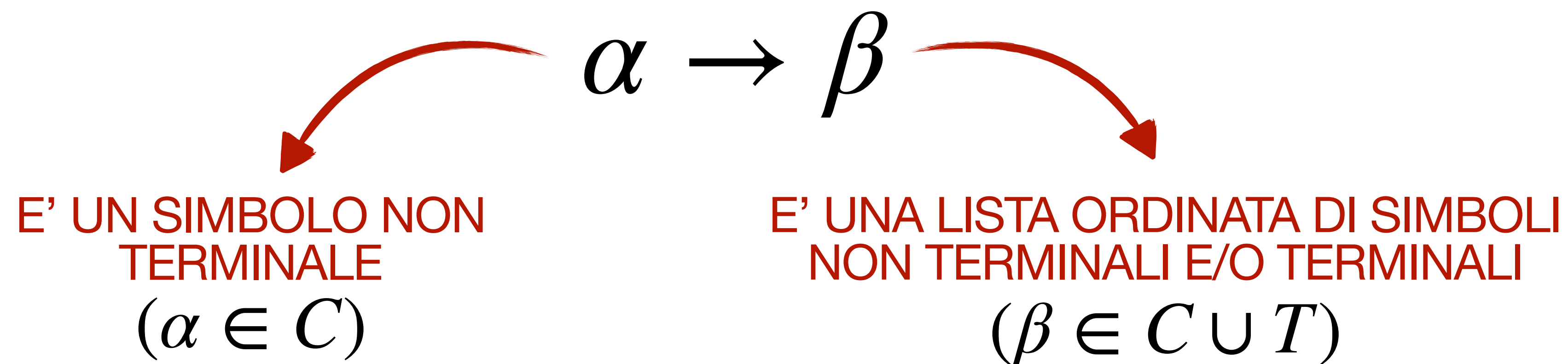
REGOLE DI
PRODUZIONE
(O DERIVAZIONE)

SIMBOLO
DI PARTENZA

GRAMMATICA LIBERA DAL CONTESTO

(CONTEXT-FREE GRAMMAR)

Ogni regola di produzione è nella forma:



esempio

REGOLE DI PRODUZIONE

$S \rightarrow A, B, C.$

$A \rightarrow [a].$

$A \rightarrow [a], A.$

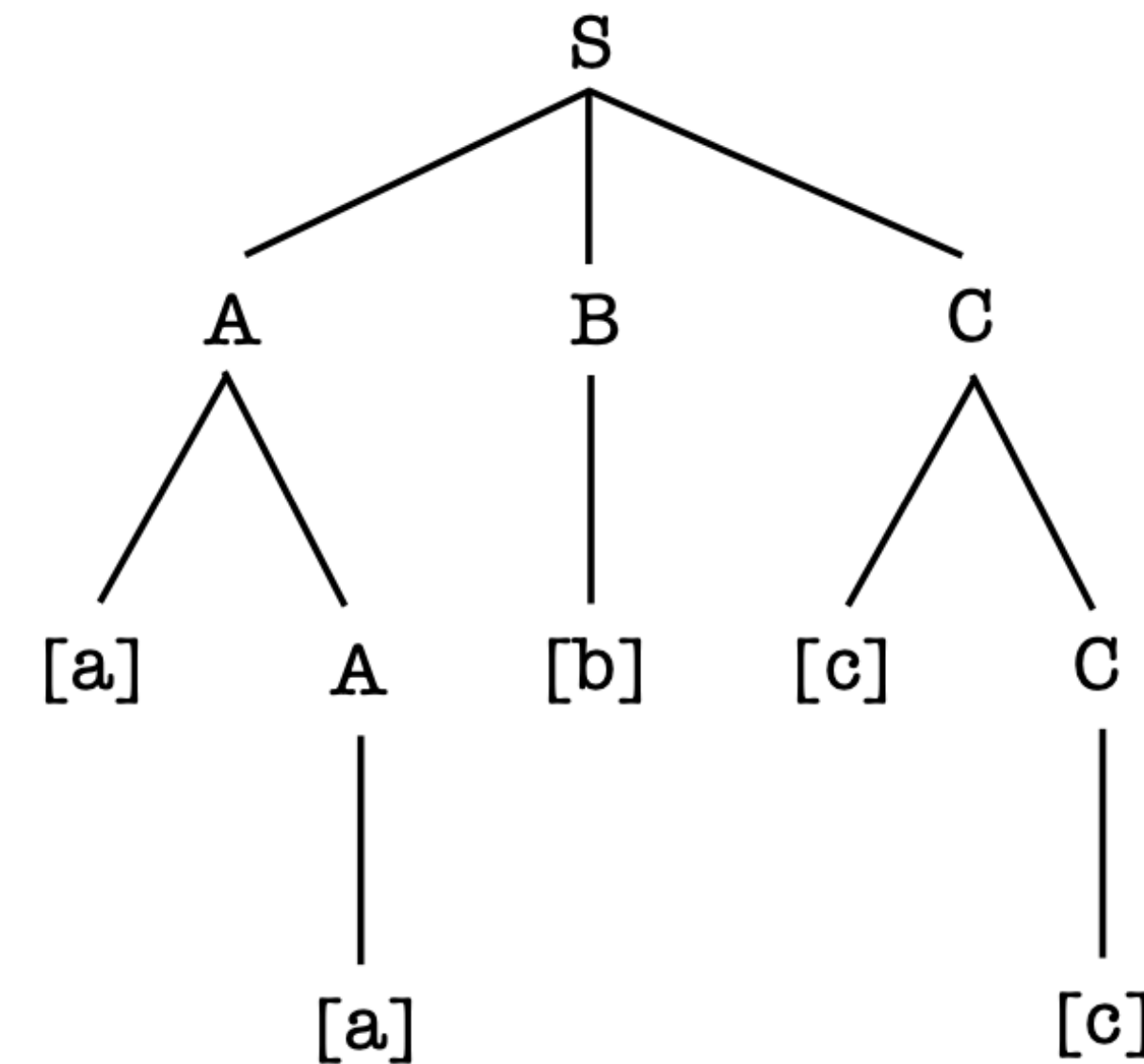
$B \rightarrow [b].$

$C \rightarrow [c].$

$C \rightarrow [c], C.$

REGOLE
RICORSIVE

ALBERO DI DERIVAZIONE



Stringhe ammesse: “aaaabc”, “aabccccc”, ...

Stringhe non ammesse: “abbbc”, “aab”, ...

CATEGORIE SINTATTICHE

CATEGORIE GRAMMATICALI

- sostantivi (N)
- determinanti (DET)
- verbi (V)
- aggettivi (ADJ)
- avverbi (ADV)
- preposizioni (PREP)
- pronomi (PRON)

STRUTTURE COSTITUENTI

- frase (S)
- sintagma nominale (NP)
- sintagma verbale (VP)
- sintagma preposizionale (PP)
- sintagma avverbiale (ADVP)

STRUTTURE COSTITUENTI

Le strutture costituenti sono formate a partire da categorie grammaticali di base.
Seguendo il formalismo della grammatica libera dal contesto, possiamo rappresentarle nel seguente modo:

$S \rightarrow NP, VP.$	(Una frase S è composta da un sintagma nominale NP seguito da un sintagma verbale VP)
$NP \rightarrow DET, N.$	(Un sintagma nominale NP è composto da un articolo DET seguito da un sostantivo N)
$VP \rightarrow V, NP.$	(Un sintagma verbale VP è composto da un verbo V seguito da un sintagma nominale NP)

Le combinazioni sono numerosissime 😱

IL PROBLEMA DEL PARSING IN PROLOG

- Data una grammatica formale, una frase è detta **BEN FORMATA** se esiste almeno un set o una sequenza di regole che permette una completa descrizione strutturale della frase, in modo da determinare la sua correttezza secondo la grammatica definita.
- Il **PROBLEMA DI PARSING** è il problema di definire se una frase è ben formata.
- La descrizione strutturale della frase può essere anche rappresentata sotto forma di una struttura ad albero chiamata **ALBERO DI PARSING** (o **PARSE TREE**).
- Un **PARSER** è un programma che si occupa di fare il parsing di una frase data in input.

Per la sua natura dichiarativa, il linguaggio di programmazione Prolog si rivela particolarmente adatto a questo scopo.



IMPLEMENTAZIONE DEL PARSER

Sviluppiamo la nostra analisi prendendo ad esempio la seguente frase rappresentata in Prolog come una lista di atomi nel seguente modo:

[il, cane, mangia, un, biscotto]

definiamo una grammatica

```
graph TD; A([definiamo una grammatica]) --> B[Enunciato logico]; A --> C[Regole di produzione];
```

Enunciato logico

$S = NP \wedge VP$

$NP = DET \wedge N$

$VP = V \vee (V \wedge NP)$

Regole di produzione

$S \rightarrow NP, VP.$

$NP \rightarrow DET, N.$

$VP \rightarrow V.$

$VP \rightarrow V, NP.$

1. APPROCCIO CON USO DELL'APPEND

Secondo la prima regola della grammatica ($S \rightarrow NP, VP$), il compito si scompone nel trovare una noun_phrase all'inizio della sequenza e poi una verb_phrase in ciò che rimane. Alla fine del processo dovremmo aver esaurito esattamente le parole della sequenza.

```
sentence(X) :- append(Y, Z, X), noun_phrase(Y), verb_phrase(Z).
```

Stesso ragionamento per le altre regole...

```
noun_phrase(X) :- append(Y, Z, X), determiner(Y), noun(Z).
```

```
verb_phrase(X) :- append(Y, Z, X), verb(Y), noun_phrase(Z).
```

```
verb_phrase(X) :- verb(X).
```

... e per la definizione delle parole.

```
determiner([il]). determiner([un]).
```

```
noun([cane]). noun([biscotto]).
```

```
verb([mangia]).
```


1. APPROCCIO CON USO DELL'APPEND

Problema!

Prendiamo la prima clausola

```
sentence(X) :- append(Y, Z, X), noun_phrase(Y), verb_phrase(Z).
```

e il goal:

```
?- sentence([il, cane, mangia, un, biscotto]).
```

Inizialmente X viene istanziato, ma Y e Z no. Per cui il goal genererà (con backtracking) tutte le possibili combinazioni di Y e Z tali che quando Z viene aggiunto a Y il risultato è X.

```
Y = [], Z = [il, cane, mangia, un, biscotto]
```

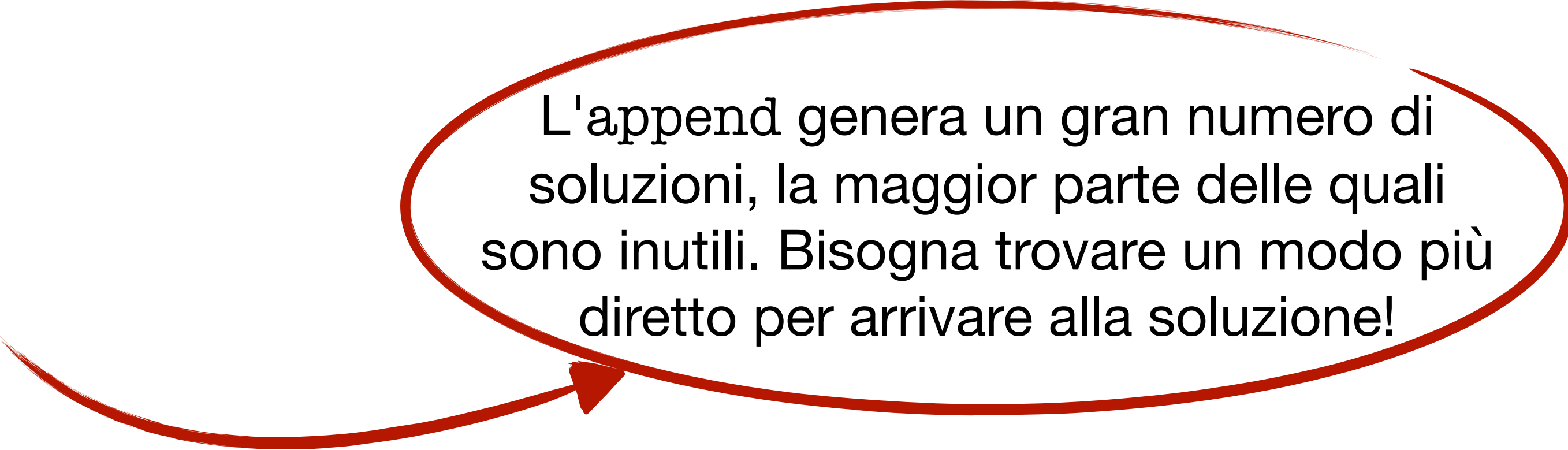
```
Y = [il], Z = [cane, mangia, un, biscotto]
```

```
Y = [il, cane], Z = [mangia, un, biscotto]
```

```
Y = [il, cane, mangia], Z = [un, biscotto]
```

```
Y = [il, cane, mangia, un], Z = [biscotto]
```

```
Y = [il, cane, mangia, un, biscotto], Z = []
```



L'append genera un gran numero di soluzioni, la maggior parte delle quali sono inutili. Bisogna trovare un modo più diretto per arrivare alla soluzione!

2. APPROCCIO CON LISTE DI DIFFERENZA

Ogni simbolo non terminale viene rappresentato come un predicato con due argomenti dove il primo rappresenta l'**input iniziale**, ovvero l'intera sequenza di parole da analizzare, e il secondo rappresenta la **sequenza rimanente** dopo l'applicazione del predicato.

```
sentence(Input, Rest) :- noun_phrase(Input, Part), verb_phrase(Part, Rest).  
noun_phrase(Input, Rest) :- determiner(Input, Part), noun(Part, Rest).  
verb_phrase(Input, Rest) :- verb(Input, Rest).  
verb_phrase(Input, Rest) :- verb(Input, Part), noun_phrase(Part, Rest).
```

determiner([il|Rest], Rest).

determiner([un|Rest], Rest).

noun([cane|Rest], Rest).

noun([biscotto|Rest], Rest).

verb([mangia|Rest], Rest).

esempio

Prendiamo la seguente clausola:

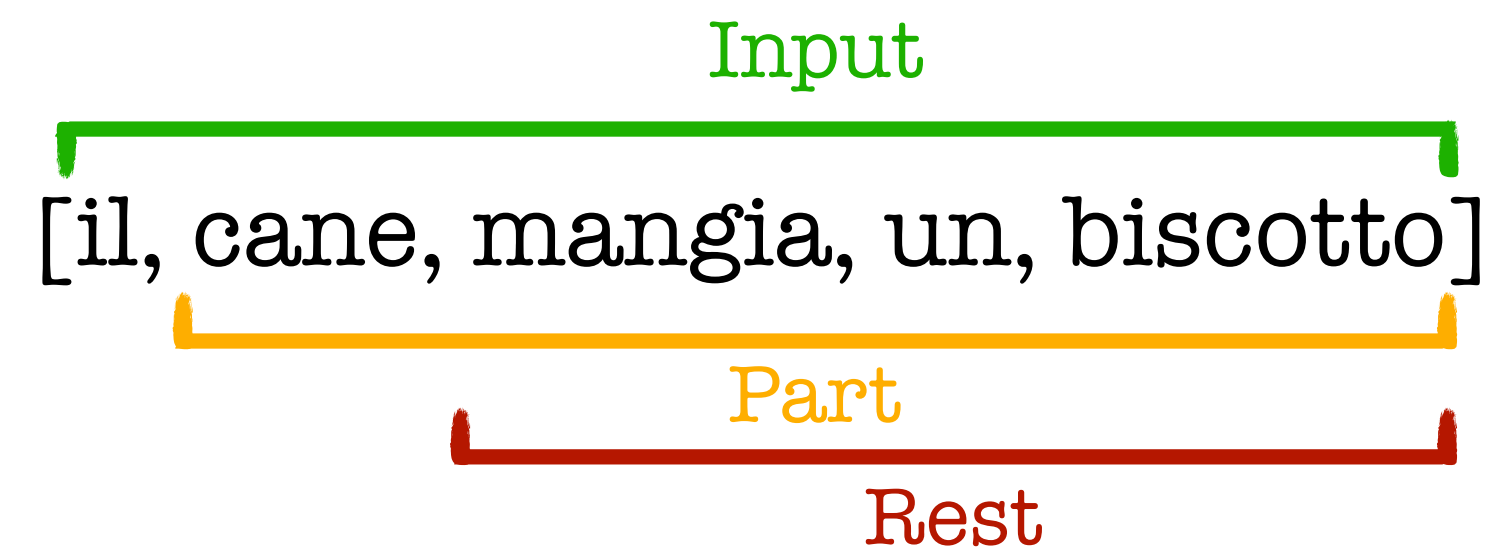
```
noun_phrase(Input, Rest) :- determiner(Input, Part), noun(Part, Rest).
```

caso 1

Input = [il, cane, mangia, un, biscotto]

Part = [cane, mangia, un, biscotto]

Rest = [mangia, un, biscotto]

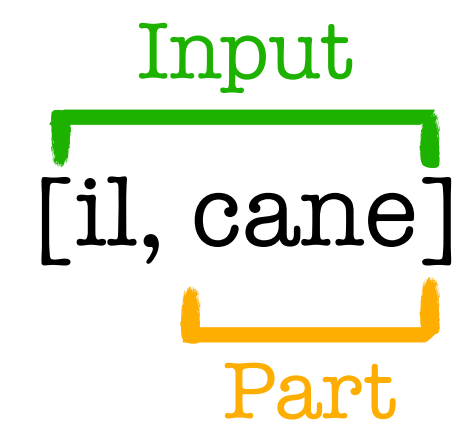


caso 2

Input = [il, cane]

Part = [cane]

Rest = []



Perciò, riprendendo la prima clausola:

```
sentence(Input, [ ]) :- noun_phrase(Input, Part), verb_phrase(Part, [ ]).
```

e **Rest** è una lista vuota, si ha che sentence può essere decomposta per trovare una noun_phrase all'inizio della sequenza e una verb_phrase in ciò che resta, senza che nessun'altra parola “avanzi”. Come nel caso precedente ma senza l'uso dell'append.

```
?- sentence([il, cane, mangia, un, biscotto], []).
```

```
true.
```

```
?- sentence([il, cane, mangia], []).
```

```
true.
```

```
?- sentence([il, cane, mangia, un], []).
```

```
false.
```

```
?- sentence(X, []).
```

```
X = [il, cane, mangia] ;
```

```
X = [il, cane, mangia, il, cane] ;
```

```
X = [il, cane, mangia, il, biscotto] ;
```

```
X = [il, cane, mangia, un, cane] ;
```

```
X = [il, cane, mangia, un, biscotto] ;
```

```
etc ...
```

3. APPROCCIO CON DCG

Le “Grammatiche a Clausole Definite” (DCG) si riferiscono a una notazione utilizzata in Prolog per esprimere regole grammaticali in modo più conciso, migliorando la leggibilità del codice.

```
sentence --> noun_phrase, verb_phrase.  
noun_phrase --> determiner, noun.  
verb_phrase --> verb.  
verb_phrase --> verb, noun_phrase.
```

```
determiner --> [il].  
determiner --> [un].  
noun --> [cane].  
noun --> [biscotto].  
verb --> [mangia].
```

Il risultato finale è totalmente equivalente all'approccio con liste di differenza. !

L'unica cosa che cambia è il goal:

?- phrase(Grammar, InputList).

esempio

?- phrase(sentence, [il, cane, mangia, un, biscotto]).
true.



?- phrase(sentence, [il, cane, mangia, un]).
false.

?- phrase(sentence, X).

X = [il, cane, mangia] ;

X = [il, cane, mangia, il, cane] ;

X = [il, cane, mangia, il, biscotto] ;

X = [il, cane, mangia, un, cane] ;

etc ...

Si potrebbe pensare di sfruttare ulteriori argomenti dei predicati per risolvere il problema della concordanza e per generare il parse tree! 🤔

Le regole che gestiscono la coerenza tra le parti variabili del discorso quando sono collegate sintatticamente tra di loro.

numero (singolare o plurale)

genere (maschile o femminile)

~~persona (1a, 2a, 3a)~~

~~concordanza verbale~~

IMPLEMENTAZIONE DEL PARSER IN UN CASO REALISTICO

Abbiamo pensato di caricare il codice completo su GitHub, il link per la repository è il seguente:

<https://github.com/davidedg11/prologparsing/>

```
prologparsing / parsingtest.pl
davidedg11 Update parsingtest.pl
Code Blame 107 lines (75 loc) · 4.87 KB Code 55% faster with GitHub Copilot
1 :- use_module(library(apply)).
2 :- use_module(library(gv)).
3 :- use_module(library(yall)).
4 :- use_module(library(term_ext)).
5
6
7 sentence(Number, Gender, sentence(NP, VP)) --> noun_phrase(Number, Gender, NP), verb_phrase(Number, Gender, VP).
8 sentence(Number, Gender, sentence(NP, PP, VP)) --> noun_phrase(Number, Gender, NP), prepositional_phrase(_, _, PP), verb_phrase(Number, Gender, VP).
9 sentence(Number, Gender, sentence(NP, VP, PP)) --> noun_phrase(Number, Gender, NP), verb_phrase(Number, Gender, VP), prepositional_phrase(_, _, PP).
10
11 proper_noun_phrase(Number, Gender, noun_phrase(PN)) --> proper_noun(Number, Gender, PN).
12 noun_phrase(Number, Gender, noun_phrase(DET, N)) --> determiner(Number, Gender, DET), noun(Number, Gender, N).
13 noun_phrase(Number, Gender, noun_phrase(DET, N, REL)) --> determiner(Number, Gender, DET), noun(Number, Gender, N), relative_clause(Number, Gender, N, REL).
14
15 noun_phrase(Number, Gender, noun_phrase(DET, N, ADJ)) --> determiner(Number, Gender, DET), noun(Number, Gender, N), adjective(Number, Gender, N, ADJ).
16 noun_phrase(Number, Gender, noun_phrase(DET, N, ADJ, REL)) --> determiner(Number, Gender, DET), noun(Number, Gender, N), adjective(Number, Gender, N, ADJ), relative_clause(Number, Gender, N, REL).
17
18 relative_clause(Number, _, relative_clause(RP, VP)) --> relative_pronoun(_, _, RP), verb_phrase(Number, _, VP).
19 prepositional_phrase(Number, Gender, prepositional_phrase(PREP, NP)) --> preposition(_, _, PREP), noun_phrase(Number, Gender, NP).
20
21 verb_phrase(Number, _, verb_phrase(V)) --> transitive_verb(Number, V).
22 verb_phrase(Number, _, verb_phrase(V, NP)) --> transitive_verb(Number, V), noun_phrase(_, _, NP).
23 verb_phrase(Number, _, verb_phrase(V)) --> intransitive_verb(Number, V).
24
25 verb_phrase(Number, _, verb_phrase(V, ADV, NP)) --> transitive_verb(Number, V), adverb(_, _, ADV), noun_phrase(_, _, NP).
26 verb_phrase(Number, _, verb_phrase(V, ADV)) --> transitive_verb(Number, V), adverb(_, _, ADV).
27 verb_phrase(Number, _, verb_phrase(V, ADV)) --> intransitive_verb(Number, V), adverb(_, _, ADV).
28
29 determiner(singular, ma, determiner(il)) --> [il].
30 determiner(singular, fem, determiner(la)) --> [la].
31 determiner(plural, ma, determiner(i)) --> [i].
32 determiner(plural, fem, determiner(le)) --> [le].
33
34 noun(singular, ma, noun(ragazzo)) --> [ragazzo].
35 noun(singular, fem, noun(ragazza)) --> [ragazza].
36 noun(plural, ma, noun(ragazzi)) --> [ragazzi].
37 noun(plural, fem, noun(ragazze)) --> [ragazze].
38
39 noun(singular, fem, noun(giraffa)) --> [giraffa].
40 noun(plural, fem, noun(giraffe)) --> [giraffe].
41
42 noun(singular, ma, noun(leone)) --> [leone].
43 noun(plural, ma, noun(leoni)) --> [leoni].
44
45 noun(singular, ma, noun(binocolo)) --> [binocolo].
46
47 proper_noun(singular, _, proper_noun('Jenny')) --> ['Jenny'].
48
49 adjective(singular, ma, adjective(biondo)) --> [biondo].
50 adjective(singular, fem, adjective(bionda)) --> [bionda].
51 adjective(plural, ma, adjective(biondi)) --> [biondi].
52 adjective(plural, fem, adjective(bionde)) --> [bionde].
53
54 adjective(singular, ma, adjective(alto)) --> [alto].
55 adjective(singular, fem, adjective(alta)) --> [alta].
56 adjective(plural, ma, adjective(alti)) --> [alti].
57 adjective(plural, fem, adjective(alte)) --> [alte].
58
```