Foundations of AI

16. Natural Language Processing

Wolfram Burgard, Bernhard Nebel, and Luc De Raedt

Contents

Speech Acts

Speech acts achieve the speaker's goals:

- Inform: "There's a pit in front of you."
- Query: "Can you see the gold?"
- Command: "Pick it up."
- Promise: "I'll share the gold with you."
- Acknowledge: "OK."

Speech act planning requires knowledge of

- Situation
- Semantic and syntactic conventions
- Hearer's goals, knowledge base, and rationality

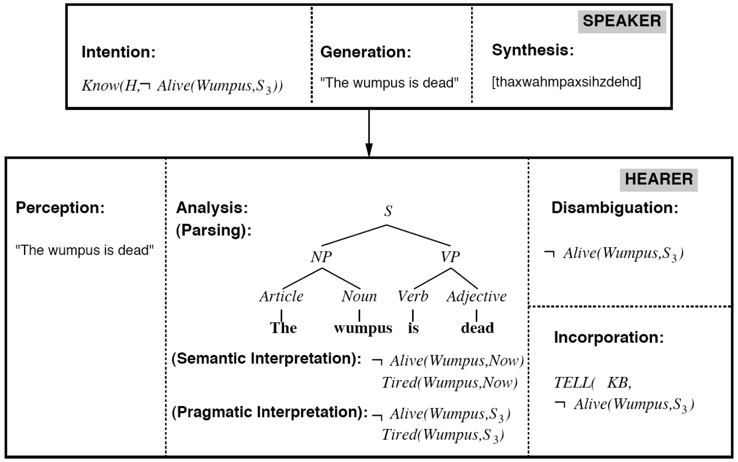
Stages in communication (informing)

- Intention: S wants to inform H that P
- Generation: S selects words W to express P in context C
- Synthesis: S utters words W
- Perception: H perceives W in context C'
- Analysis: H infers possible meanings P1, …, Pn
- Disambiguation: H infers intended meaning Pi
- Incorporation: H incorporates Pi into KB

How could this go wrong ?

- Insincerity : S doesn't believe P
- Speech wreck ignition failure
- Ambiguous utterance
- Differing understanding of current context C<>C'

Stages in communication (informing)



Speech Acts

- Animals use isolated symbols for sentences
  - restricted set of communicable propositions, no *generative capacity*
- Grammar specifies the compositional structure of complex message (speech, language, music)
- A formal language is a set of strings of terminal symbols
- Each string in the language can be analyzed/generated by the grammar
- The grammar is a set of rewrite rules,
  - e.g.,

$$S \rightarrow NP\ VP$$
$$Article \rightarrow \textbf{\textit{the}} \mid \textbf{\textit{a}} \mid \textbf{\textit{an}} \mid \ldots$$

- Here *S* is the sentence symbol, *NP* and *VP* are non-terminals, and *the, a* ... terminals

Grammar Types

Regular: $nonterminal \rightarrow \textbf{\textit{terminal}}[nonterminal]$

$$S \rightarrow \boldsymbol{a}S$$
$$S \rightarrow \Lambda$$

Context-free: $nonterminal \rightarrow anything$

$$S \rightarrow \boldsymbol{a}S\boldsymbol{b}$$

Context-sensitive: more nonterminals on right-hand side

$$ASB \rightarrow AA\boldsymbol{a}BB$$

Recursively enumerable: no constraints

Natural language probably context-free

The Wumpus Lexicon

$$\begin{aligned}
Noun &\rightarrow \textbf{\textit{stench}} \mid \textbf{\textit{breeze}} \mid \textbf{\textit{glitter}} \mid \textbf{\textit{nothing}} \\
&\quad \mid \textbf{\textit{wumpus}} \mid \textbf{\textit{pit}} \mid \textbf{\textit{pits}} \mid \textbf{\textit{gold}} \mid \textbf{\textit{east}} \mid \ldots \\
Verb &\rightarrow \textbf{\textit{is}} \mid \textbf{\textit{see}} \mid \textbf{\textit{smell}} \mid \textbf{\textit{shoot}} \mid \textbf{\textit{feel}} \mid \textbf{\textit{stinks}} \\
&\quad \mid \textbf{\textit{go}} \mid \textbf{\textit{grab}} \mid \textbf{\textit{carry}} \mid \textbf{\textit{kill}} \mid \textbf{\textit{turn}} \mid \ldots \\
Adjective &\rightarrow \textbf{\textit{right}} \mid \textbf{\textit{left}} \mid \textbf{\textit{east}} \mid \textbf{\textit{south}} \mid \textbf{\textit{back}} \mid \textbf{\textit{smelly}} \mid \ldots \\
Adverb &\rightarrow \textbf{\textit{here}} \mid \textbf{\textit{there}} \mid \textbf{\textit{nearby}} \mid \textbf{\textit{ahead}} \\
&\quad \mid \textbf{\textit{right}} \mid \textbf{\textit{left}} \mid \textbf{\textit{east}} \mid \textbf{\textit{south}} \mid \textbf{\textit{back}} \mid \ldots \\
Pronoun &\rightarrow \textbf{\textit{me}} \mid \textbf{\textit{you}} \mid \textbf{\textit{I}} \mid \textbf{\textit{it}} \mid \textbf{\textit{S/HE}} \mid \textbf{\textit{Y'ALL}} \ldots \\
Name &\rightarrow \textbf{\textit{John}} \mid \textbf{\textit{Mary}} \mid \textbf{\textit{Boston}} \mid \textbf{\textit{UCB}} \mid \textbf{\textit{PAJC}} \mid \ldots \\
Article &\rightarrow \textbf{\textit{the}} \mid \textbf{\textit{a}} \mid \textbf{\textit{an}} \mid \ldots \\
Preposition &\rightarrow \textbf{\textit{to}} \mid \textbf{\textit{in}} \mid \textbf{\textit{on}} \mid \textbf{\textit{near}} \mid \ldots \\
Conjunction &\rightarrow \textbf{\textit{and}} \mid \textbf{\textit{or}} \mid \textbf{\textit{but}} \mid \ldots \\
Digit &\rightarrow \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9}
\end{aligned}$$

Divided into open and closed classes

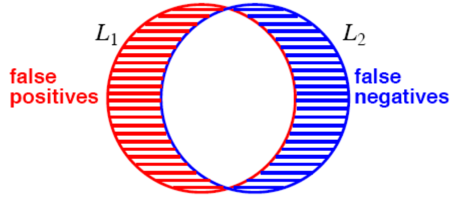
The Wumpus Grammar

S	$\rightarrow$	$NP\ VP$	I + feel a breeze
	$\mid$	$S\ Conjunction\ S$	I feel a breeze + and + I smell a v
NP	$\rightarrow$	$Pronoun$	I
	$\mid$	$Noun$	pits
	$\mid$	$Article\ Noun$	the + wumpus
	$\mid$	$Digit\ Digit$	3 4
	$\mid$	$NP\ PP$	the wumpus + to the east
	$\mid$	$NP\ RelClause$	the wumpus + that is smelly
VP	$\rightarrow$	$Verb$	stinks
	$\mid$	$VP\ NP$	feel + a breeze
	$\mid$	$VP\ Adjective$	is + smelly
	$\mid$	$VP\ PP$	turn + to the east
	$\mid$	$VP\ Adverb$	go + ahead
PP	$\rightarrow$	$Preposition\ NP$	to + the east
$RelClause$	$\rightarrow$	$\textbf{\textit{that}}\ VP$	that + is smelly

Grammatical Judgments

Formal language L_1 may well differ from natural language L_2

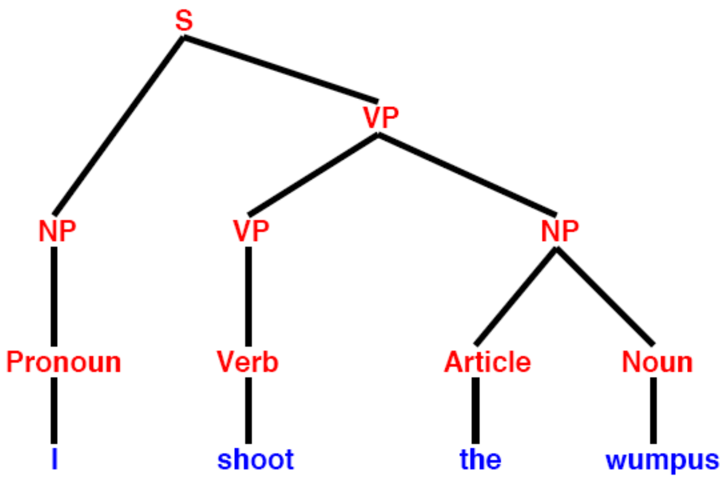
Adjusting is a learning problem



Real grammars typically 10-500 pages, insufficient for proper English

Parse Trees

Exhibit the grammatical structure of a sentence



Syntax in NLP

Most view syntactic structure as an essential step towards meaning

“Mary hit John” <> “John hit Mary”

“And since I was not informed---as a matter of fact, since I did not know that there were excess funds until we, ourselves, in that checkup after the whole thing blew up, and that was, if you'll remember, that was the incident in which the attorney general came to me and told me that he had seen a memo that indicated that there were no more funds.”

Parsing CFGs

Many different parsing algorithms

- Top-down
- Bottom-up
- Chart-parsing (aka CYK algorithm)
  - Dynamic programming
  - $O(n^3)$

[S: [NP :[Pronoun:I]]
[VP :[VP:[Verb:shoot]]
[NP:[Article:the][Noun:wumpus]]]]

Top-down parsing

- Initial state : [S: ?]
- Successor function:
  - Select rules for leftmost node in tree with unknown children and apply
  - [S:[S:?][Conjunction:?][S:?]]
  - [S:[NP:?][VP:?]]
- Goal Test:
  - Check whether leaves of the parse tree correspond to (complete) input

Problem for Top-down: left-recursive rules S -> S Conjunction S
infinite loops

Top-down parsing

Successful top-down parse

[S: ?]
[S:[NP:?][VP:?]]
[S:[NP:[Art:?][Noun:?]][VP:?]]
[S:[NP:[Art:The][Noun:wumpus]][VP:?]]
[S:[NP:[Art:The][Noun:wumpus]][VP:[Verb:?]]]
[S:[NP:[Art:The][Noun:wumpus]][VP:[Verb:stinks]]]

Alternative notation

S	[The, wumpus, stinks]
NP VP	[The, wumpus, stinks]
Art Noun VP	[The, wumpus, stinks]
Noun VP	[wumpus, stinks]
VP	[stinks]
Verb	[stinks]
[]	[]

Bottom-up parsing

- Initial state
  - [the, wumpus, is, dead]
- Successor function
  - If subsequence at pos i matches right-hand of rule then replace by left hand
  - [[Art:the],wumpus, is, dead]
- Goal state :
  - A single state with root S

Problem for Bottom up:
Art -> []

Bottom-up parsing

- [the, wumpus, is, dead]
- [[Art:the],wumpus,is,dead]
- [[Art:the][Noun:wumpus],is,dead]
- [[NP:[Art:the][Noun:wumpus]],is,dead]
- [[NP:[Art:the][Noun:wumpus]],[Verb:is],dead]
- [[NP:[Art:the][Noun:wumpus]],[VP:[Verb:is]],dead]
- [[NP:[Art:the][Noun:wumpus]],[VP:[Verb:is]],[Adj:dead]]
- [[NP:[Art:the][Noun:wumpus]],[VP:[VP:[Verb:is]],[Adj:dead]]]
- [S:[[NP:[Art:the][Noun:wumpus]],[VP:[VP:[Verb:is]],[Adj:dead]]]]

Bottom-up parsing

step	*list of nodes*	*subsequence*	*rule*
INIT	**the wumpus is dead**	**the**	Article → **the**
2	*Article* **wumpus is dead**	**wumpus**	Noun → **wumpus**
3	*Article Noun* **is dead**	*Article Noun*	*NP* → *Article Noun*
4	*NP* **is dead**	**is**	*Verb* → **is**
5	*NP Verb* **dead**	**dead**	*Adjective* → **dead**
6	*NP Verb Adjective*	*Verb*	*VP* → *Verb*
7	*NP VP Adjective*	*VP Adjective*	*VP* → *VP Adjective*
8	*NP VP*	*NP VP*	*S* → *NP VP*
GOAL	*S*		

Chart parsing

- Have the students in B.Sc. Informatik take the exam of AI.
- Have the students in B.Sc. Informatik taken the exam of AI?
- Double work
  - Dynamic programming - combine bottom-up and top-down
- Chart :
  - N+1 vertices
  - Labeled edges, e.g., [0,2 S-> NP *VP]
    - Denotes that from 0 to 2 we have a NP, and if we find a VP, from 2 to k then we have an S from 0 to k

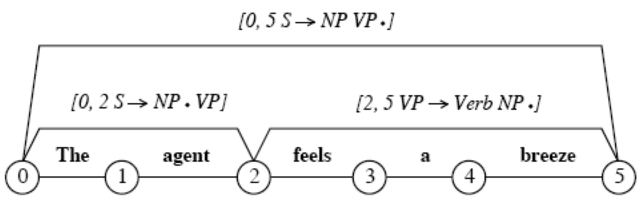


Chart parsing

- Initialization
  - Add [0,0 S' -> * S]
- Add edge [i,j A -> B * c]
  - if c is empty then call extender
  - else call predictor
- Predictor [i,j A ->b * C e] - top-down
  - With (all) rules C -> d
  - Add edge [i,j A->b * d e]
- Extender [j,k B -> c *] - bottom-up
  - With (all) edges [i,j A -> e * B f]
  - Add edge [i,k A -> e c * f]
- Scanner [j,k A -> c * D e] - bottom-up
  - Word of type D at position k
  - Add edge [j,k+1 A -> c D * e]

Convention :
non-terminal - upper case
sequence - lower case

```
function CHART-PARSE(words, grammar) returns chart
  chart ← array[0... LENGTH(words)] of empty lists
  ADD-EDGE([0, 0, S' → • S])
  for i ← from 0 to LENGTH(words) do
     SCANNER(i, words[i])
  return chart

procedure ADD-EDGE(edge)
   /* Add edge to chart, and see if it extends or predicts another edge. */
   if edge not in chart[END(edge)] then
      append edge to chart[END(edge)]
      if edge has nothing after the dot then EXTENDER(edge)
      else PREDICTOR(edge)

procedure SCANNER(j, word)
   /* For each edge expecting a word of this category here, extend the edge. */
   for each [i, j, A → α • B β] in chart[j] do
      if word is of category B then
         ADD-EDGE([i, j+1, A → α B • β])

procedure PREDICTOR([i, j, A → α • B β])
   /* Add to chart any rules for B that could help extend this edge */
   for each (B → γ) in REWRITES-FOR(B, grammar) do
      ADD-EDGE([j, j, B → • γ])

procedure EXTENDER([j, k, B → γ •])
   /* See what edges can be extended by this edge */
   e_B ← the edge that is the input to this procedure
   for each [i, j, A → α • B' β] in chart[j] do
      if B = B' then
         ADD-EDGE([i, k, A → α e_B • β])
```

Figure 22.9

A completed chart

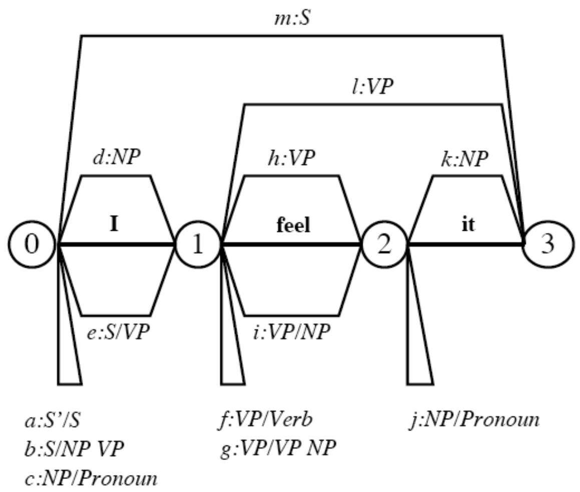


Chart Parse Trace

Edge	Procedure	Derivation
a	INITIALIZER	$[0,0,\ S' \rightarrow \bullet S]$
b	PREDICTOR(a)	$[0,0,\ S \rightarrow \bullet NP\ VP]$
c	PREDICTOR(b)	$[0,0,\ NP \rightarrow \bullet Pronoun]$
d	SCANNER(c)	$[0,1,\ NP \rightarrow Pronoun \bullet]$
e	EXTENDER(b,d)	$[0,1,\ S \rightarrow NP \bullet VP]$
f	PREDICTOR(e)	$[1,1,\ VP \rightarrow \bullet Verb]$
g	PREDICTOR(e)	$[1,1,\ VP \rightarrow \bullet VP\ NP]$
h	SCANNER(f)	$[1,2,\ VP \rightarrow Verb \bullet]$
i	EXTENDER(g,h)	$[1,2,\ VP \rightarrow VP \bullet NP]$
j	PREDICTOR(g)	$[2,2,\ NP \rightarrow \bullet Pronoun]$
k	SCANNER(j)	$[2,3,\ NP \rightarrow Pronoun \bullet]$
l	EXTENDER(i,k)	$[1,3,\ VP \rightarrow VP\ NP \bullet]$
m	EXTENDER(e,l)	$[0,3,\ S \rightarrow NP\ VP \bullet]$

Figure 22.9 Trace of a parse of "$_0$ I $_1$ feel $_2$ it $_3$." For each edge a-m, we show the procedure used to derive the edge from other edges already in the chart. Some edges were omitted for brevity.

Definite Clause Grammars

- A form of unification based grammar
- Non-terminals become atomic expressions
  - s(Num) --> np(Num), vp(Num)
  - Numb is a variable
  - Binds to singular and plural
- Employ unification during parsing.
- Directly executable in Prolog
- Rules directly translate to (definite) clause logic
- Next slides employ Prolog notation
  - Running examples with, e.g., YAP Prolog or SWI Prolog

Non-terminals with arguments

```
sentence            --> noun_phrase(N),verb_phrase(N).
noun_phrase         --> article(N),noun(N).
verb_phrase(N)      --> intransitive_verb(N).
article(singular)  --> [a].
article(singular)  --> [the].
article(plural)    --> [the].
noun(singular)     --> [turtle].
noun(plural)       --> [turtles].
intransitive_verb(singular) --> [sleeps].
intransitive_verb(plural)   --> [sleep].
```

Agreement

Case Marking

```
pronoun(singular,nominative)--> [he];[she]
pronoun(singular,accusative)--> [him];[her]
pronoun(plural,nominative)   --> [they]
pronoun(plural,accusative)   --> [them]
sentence    --> np(Number,nominative), vp(Number)
vp(Number) --> v(Number), np(X,accusative)
np(Number,Case)   --> pronoun(Number,Case)
np(Number,X)      --> det, n(Number)
```

He sees her. She sees him. They see her.

But not Them see he.

Top-down parsing with DCG

sentence	[He,sees,her]
np(Num,nom), vp(Num)	[He,sees,her]
pronoun(Num,nom), vp(Num)	[He,sees,her]
Num = sing !!! pronoun(sing,nom) --> [he]	
vp(sing)	[sees,her]
v(sing), np(sing,accusative)	[sees,her]
np(sing,accusative)	[her]
pronoun(sing,accusative]	[her]
[]	[]

Sub-categorization

```
vp    --> v(1).               v(1) --> [sleep]
vp    --> v(2), np.           v(2) --> [chase]
vp    --> v(3), np, np.       …
vp    --> v(4), s.
```

Verb	Complement	Example
Sleep	None	The cat slept
Chase	One NP	The cat chased the dog
Give	Two NP	John gave Bill the book
Say	sentence	John said he loved Mary.

Constructing parse trees

```
sentence(s(NP,VP))           --> noun_phrase(NP),verb_phrase(VP).
noun_phrase(np(N))           --> proper_noun(N).
noun_phrase(np(Art,Adj,N))   --> article(Art),adjective(Adj),
                                   noun(N).
noun_phrase(np(Art,N))       --> article(Art),noun(N).
verb_phrase(vp(IV))          --> intransitive_verb(IV).
verb_phrase(vp(TV,NP))       --> transitive_verb(TV),
                                  noun_phrase(NP).
article(art(the))            --> [the].
adjective(adj(lazy))         --> [lazy].
adjective(adj(rapid))        --> [rapid].
proper_noun(pn(achilles))    --> [achilles].
noun(n(turtle))              --> [turtle].
intransitive_verb(iv(sleeps)) --> [sleeps].
transitive_verb(tv(beats)) --> [beats].
```

```
?-phrase(sentence(T),[achilles,beats,the,lazy,turtle])
  T = s(np(pn(achilles)),
         vp(tv(beats),
            np(art(the),
               adj(lazy),
               n(turtle))))
```

Semantic Interpretation

```
np(fido)   --> [fido].
np(felix)  --> [felix].

v(X^slept(X))          --> [slept].
v(Y^(X^chased(X,Y))) --> [chased].

s(Pred) --> np(Subj), vp(Subj^Pred).

vp(Subj^Pred) --> v(Subj^Pred).
vp(Subj^Pred) --> v(Obj^(Subj^Pred)), np(Obj).
```

```
?-prhase(s(X),[fido, chased, felix]).
      X = chased(fido,felix)
```


Lambda Expressions

`X^slept(X)` stands for `λX. slept(X)`
`Y^(X^chased(X,Y))` stands for `λY λX. chased(X,Y)`

These lambda-expressions denote relationships
They can be instantiated by unification

With unification
`X^slept(X)    with X=fido yields fido^slept(fido)`
`unify fido^slept(fido)  with X^R`
`Yields R=slept(fido)`

In lambda-calculus
`(λX. slept(X)) fido gives slept(fido)`

Natural Language Processing

- Complex process
  - Requiring knowledge, reasoning and AI in general
- Grammars to represent natural languages
- Parsing techniques for syntactic analysis
- DCG add expressive power to CFGs
  - Convenient in practice
  - Unification based grammars