

Natural Language Processing in Prolog

Part I: From Lists to Grammars

CSCI 315: Artificial Intelligence

Simon D. Levy

Fall 2007

Recall Wumpus Grammar

Noun → *stench* | *breeze* | *glitter* | *nothing*
| *wumpus* | *pit* | *pits* | *gold* | *east* | ...

Verb → *is* | *see* | *smell* | *shoot* | *feel* | *stinks*
| *go* | *grab* | *carry* | *kill* | *turn* | ...

Adjective → *right* | *left* | *east* | *south* | *back* | *smelly* | ...

Adverb → *here* | *there* | *nearby* | *ahead*
| *right* | *left* | *east* | *south* | *back* | ...

Pronoun → *me* | *you* | *I* | *it* | *S/HE* | *Y'ALL*...

Name → *John* | *Mary* | *Boston* | *UCB* | *PAJC* | ...

Article → *the* | *a* | *an* | ...

Preposition → *to* | *in* | *on* | *near* | ...

Conjunction → *and* | *or* | *but* | ...

Digit → *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9*

Recall Wumpus Grammar

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

Grammars in Prolog

- Prolog will allow us to write grammars that look very much like this
- First we need to understand how lists work

Lists in Prolog

- Syntactically, look like lists in Python:

```
[foo, bar, baz, moo]
```

- Special notation for head, tail:

```
[H | T ]
```

- E.g.,

```
| ?- [foo, bar, baz] = [foo | T].
```

```
T = [bar, baz]
```

```
yes
```

- Tail of one-item list is empty list:

$$[_ | T] = [foo]$$
$$T = []$$

yes

- Multiple items before tail:

$$[X, Y | T] = [a, b, c, d, e]$$
$$X = a$$
$$Y = b$$
$$T = [c, d, e]$$

yes

- A common mistake:

```
[X, Y, T] = [a, b, c, d, e]
```

```
no
```

- **append** predicate concatenates lists:

```
| ?- import append/3 from basics.
```

```
yes
```

```
| ?- append([a,b,c], [d,e], X).
```

```
X = [a,b,c,d,e]
```

```
yes
```

Sentences as Lists

```
sentence(S) :-  
    append(NP, VP, S),  
    nounphrase(NP),  
    verbphrase(VP).
```

```
nounphrase(NP) :-  
    append(Art, N, NP),  
    article(Art),  
    noun(N).
```

```
verbphrase(VP) :-  
    append(V, NP, VP),  
    verb(V),  
    nounphrase(NP).
```

Lexical Rules

```
article([a]).  
article([the]).
```

Why brackets ?

```
noun([stench]).  
noun([breeze]).  
noun([wumpus]).
```

```
verb([sees]).  
verb([smells]).  
verb([feels]).
```

Accepting Sentences

```
| ?- nounphrase( [the,wumpus] ).
```

```
yes
```

```
| ?- sentence( [the,wumpus,feels,a,breeze] ).
```

```
yes
```

```
| ?- sentence( [wumpus,feels,a,breeze] ).
```

```
no
```

Generating Sentences

```
| ?- sentence(S).
```

```
S = [a, stench, sees, a, stench];
```

```
S = [a, stench, sees, a, breeze];
```

```
S = [a, stench, sees, a, wumpus];
```

```
S = [a, stench, sees, the, stench];
```

```
S = [a, stench, sees, the, breeze];
```

```
S = [a, stench, sees, the, wumpus]
```

Difference Lists

- Using `append` this way will work but is inefficient - we have to try every possible grouping of words for each predicate.
- Better strategy: take items from front of list, and pass remainder to next predicate
- This is the method of *Difference Lists*

Difference Lists

```
sentence (S) :-  
    nounphrase (S-S1),  
    verbphrase (S1-[]).
```

```
nounphrase (NP-X) :-  
    article (NP-NP1),  
    noun (NP1-X).
```

```
verbphrase (VP-X) :-  
    verb (VP-VP1),  
    nounphrase (VP1 - X).
```

Difference Lists

```
| ?- consult(grammar2).  
[Compiling ./grammar2]  
++Warning[XSB]: [Compiler] ./grammar2: A partially  
instantiated call to article/1 will fail!  
++Warning[XSB]: [Compiler] ./grammar2: A partially  
instantiated call to noun/1 will fail!  
++Warning[XSB]: [Compiler] ./grammar2: A partially  
instantiated call to verb/1 will fail!  
[grammar2 compiled, cpu time used: 0.0100 seconds]  
[grammar2 loaded]
```

yes

```
| ?- nounphrase([the,wumpus]).
```

no

```
| ?-
```

Lexical Difference Lists

```
noun ([stench | X] - X) . % WTF???  
noun ([breeze | X] - X) .  
noun ([wumpus | X] - X) .  
  
verb ([sees | X] - X) .  
verb ([smells | X] - X) .  
verb ([feels | X] - X) .  
  
article ([the | X] - X) .  
article ([a | X] - X) .
```

Lexical Difference Lists

```
| ?- noun([wumpus, smells, the, breeze] - X) .
```

```
X = [smells, the, breeze]
```

```
yes
```

```
| ?- verb([wumpus, smells, the, breeze] - X) .
```

```
no
```

Gimme Some Sugar: Definite Clause Grammars

sentence --> nounphrase, verbphrase.

nounphrase --> article, noun.

verbphrase --> verb, nounphrase.

Lexical DCG Rules

```
article --> [a].  
article --> [the].
```

```
noun --> [stench].  
noun --> [breeze].  
noun --> [wumpus].
```

```
verb --> [sees].  
verb --> [smells].  
verb --> [feels].
```

DCG

```
?- sentence([the,wumpus,smells,a,breeze]).  
++Error[XSB/Runtime/P]: [Existence (No  
procedure usermod : sentence / 1 exists)] []
```

DCG

```
sentence --> nounphrase, verbphrase.
```

```
sentence(S1, S2) :-  
    nounphrase(S1, S3),  
    verbphrase(S3, S2).
```

DCG

```
?- sentence([the,wumpus,smells,a,breeze], []).
```

```
yes
```

DCG

?- sentence ([the, wumpus, smells, a, breeze], []).

yes



DCG, Part I: Conclusions

- Write helper functions:

```
accept("the wumpus feels a breeze").
```

```
accept(String) :-
```

```
    string2list(String, List),
```

```
    sentence(List, []).
```

- Should accepting be the goal of NLP?