Logic for CS

黃瀚萱
Department of Computer Science
National Chengchi University
2020 Spring

Schedule, Part I

Date	Topic
3/6	Introduction to this course
3/13	Thinking as computation
3/20	Propositional Logic
3/27	Logic Inference
4/3	Off
4/10	First Order Logic
4/17	Interpretation of FOL
4/24	Inference in FOL (Online)

Schedule, Part II

Date	Topic
5/1	Prolog Basics & KR (Online)
5/8	Midterm Exam
5/15	Prolog Programming
5/22	Prolog Programming II
5/29	**Logic Programming for NLP (Non-remote)**
6/5	Discussion of the Final Project (Non-remote)
6/12	Final Project Presentation (Non-remote)
6/19	Term Exam (Non-remote)

Logic Programming for Natural Language Understanding

Natural Language Understanding

- An area in natural language processing, or computational linguistics.
- Aimed at knowing the meaning of text written in natural languages such as English and Chinese.
- As human can communicate using a system as rich as a natural language, it requires thinking to deal with the richness.
    - The American President during the Civil War.
    - Abraham Lincoln.

Language and Thinking

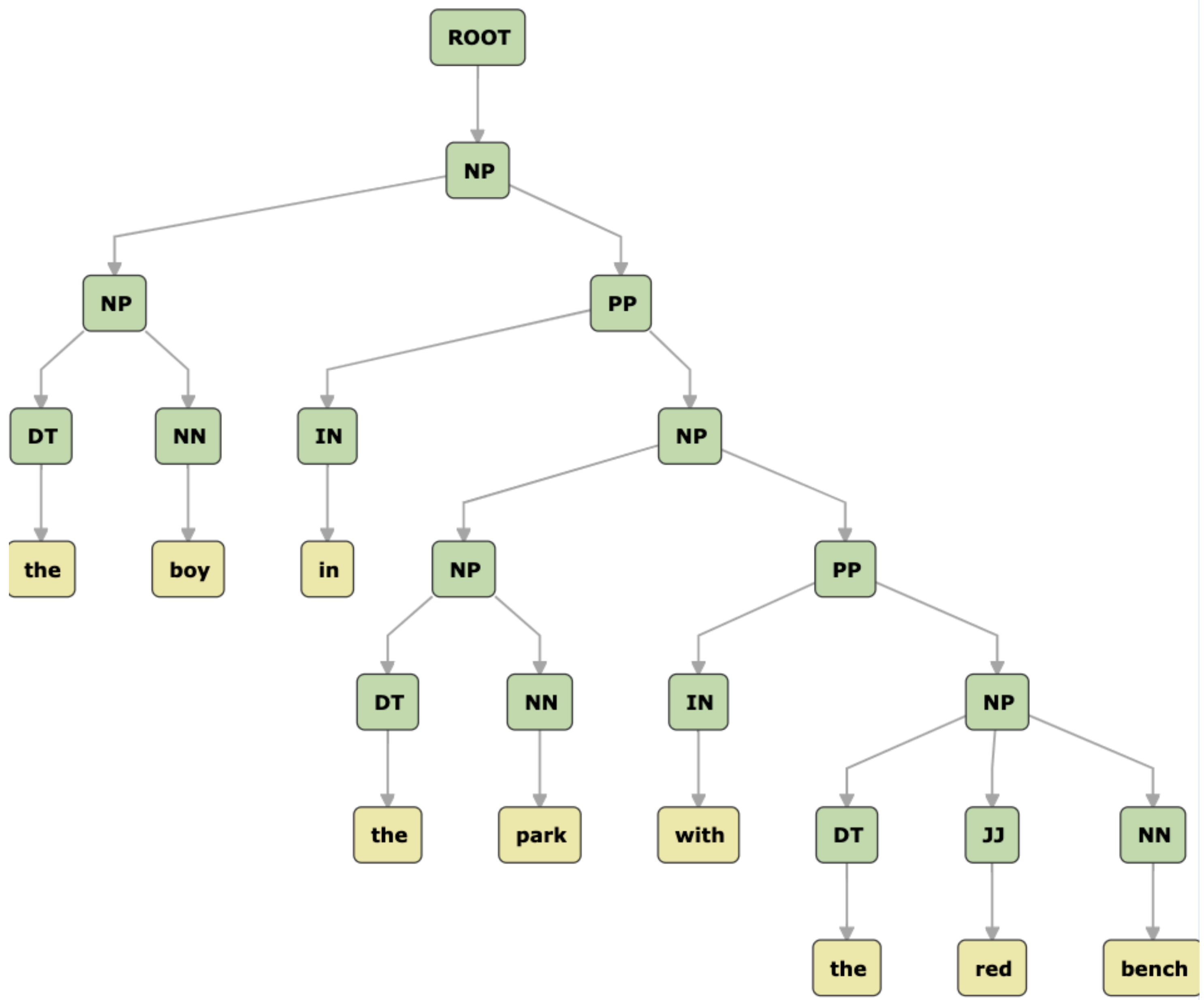
- The connection between thinking and language goes deeper.
  - Thinking supports our ability to use language.
  - Language also feeds our thinking.
- Thinking is use **what we know**.
  - Much of **what we know** is due to language.

Levels of Natural Language Processing

- Morphology: What are the roots of words
  - ran = run + PAST
  - children = child + PLURAL
- Syntax: How do the words group together
  - Mary kicked the boy in the knee.
  - Mary kicked the boy in the first row.
- Semantics: What do the words mean?
  - The astronomer spotted a **star** [star on the sky]
  - The astronomer married a **star** [celebrity]
- Pragmatics: What are the words being used for?
  - Can you **juggle**?
  - Can you **pass the salt**? (Get over it and move on)

Syntax Analysis

- A sentence can be converted into a parse tree according to its syntactic structure.



Complex Syntax Analysis

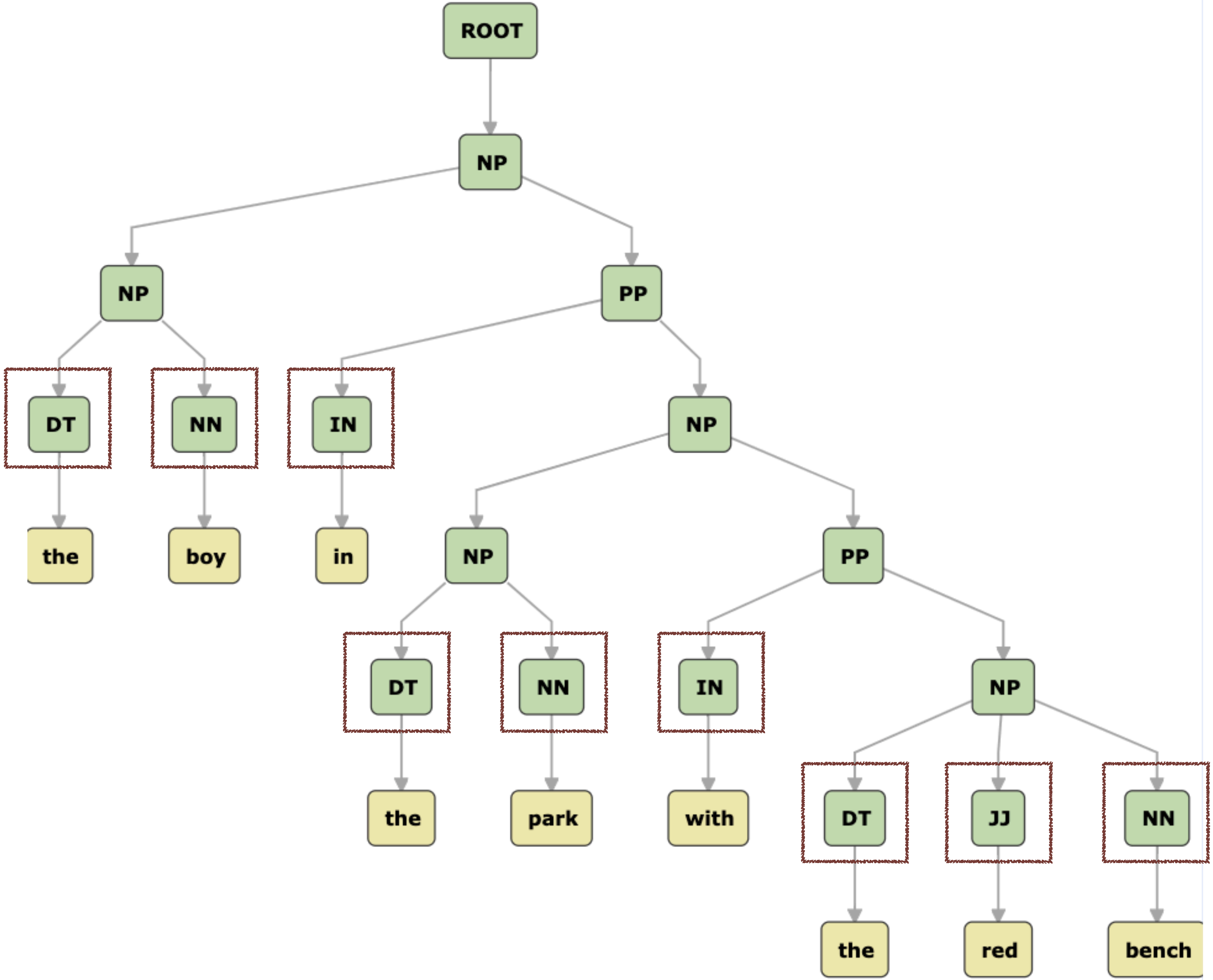
- The cat the dog the boy next door owns chases sleeps all day.
- (The cat (the dog (the boy next door owns) chases) sleeps all day.)
- Knowing how the words in a phrase or sentence should be grouped together is not the same as understanding what they mean.
    - A sentence can be grammatically correct but without meaning.
        - Colorless green ideas sleep furiously.
    - A meaningful group of words can also be syntactically odd.
        - Accident car passenger hospital.

Lexicon

- A basic unit of language is lexical.
- Categories of words (POS)
  - Articles
  - Adjectives
  - Proper nouns
  - Common nouns
  - Transitive verbs
  - Intransitive verbs
  - ...
- A word may belong to more than one word categories.

Grammar

- A grammar of a language is a specification of how the various types of words can be grouped in the language.
- Syntactic categories
  - Lexical or terminal categories: POS
  - Group or non-terminal categories: Noun phrase, verb phrase, prepositional phrase.


ROOT
NP
NP
PP
DT
NN
IN
NP
the
boy
in
NP
PP
DT
NN
IN
NP
the
park
with
DT
JJ
NN
the
red
bench

Rules of a Grammar

- Usually grammars are specified by a collection of rules
  - Describing how each type of word group is formed from other word groups.
  - Similar to Prolog clauses
- VP → V, NP
- NP → DT, NN

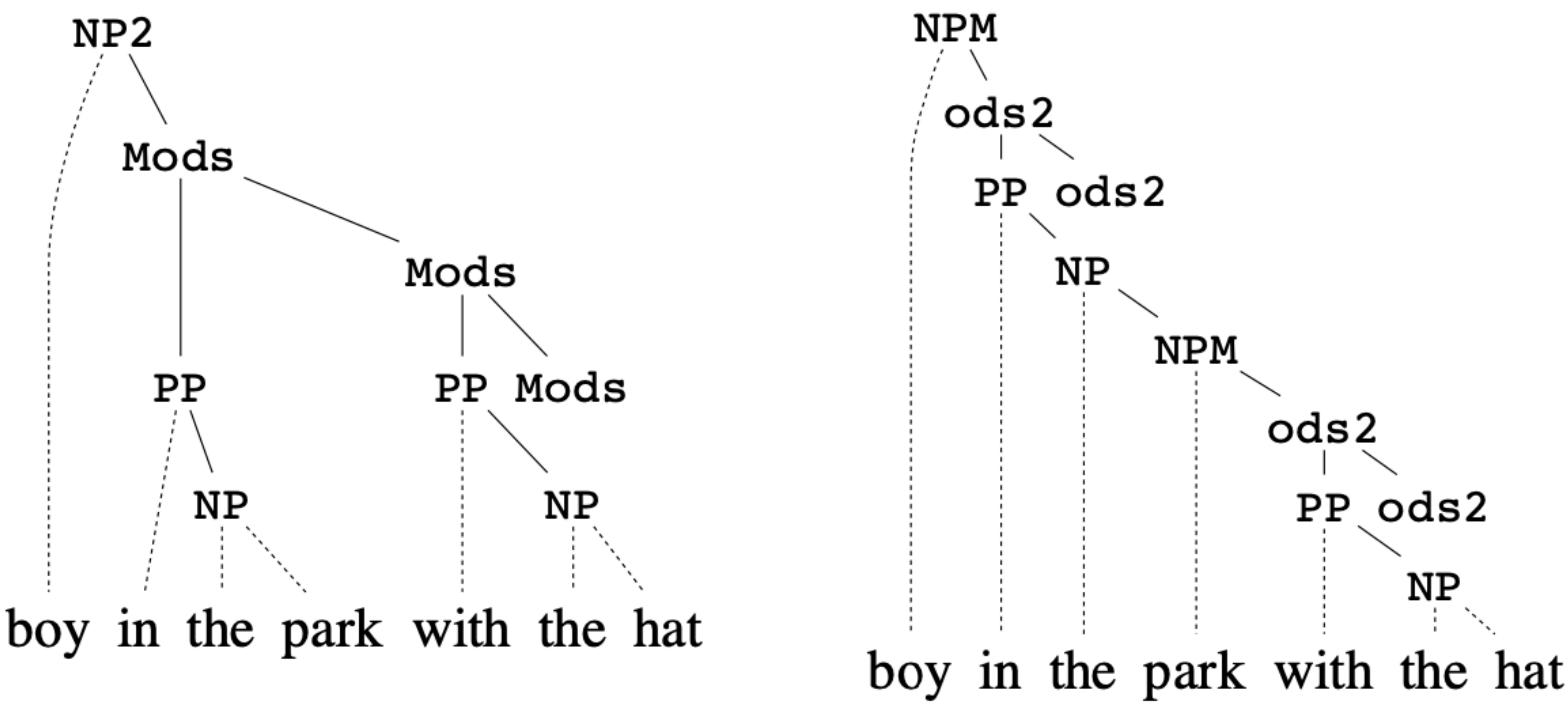
Grammar for a Language

- The grammar is recursive.
- The boy from France in the library on the phone with the gray sweater.

```
   S  →  NP VP
  VP  →  copula_verb Mods
  VP  →  transitive_verb NP Mods
  VP  →  intransitive_verb Mods
Mods  →
Mods  →  PP Mods
  PP  →  preposition NP
  NP  →  proper_noun
  NP  →  article NP2
 NP2  →  adjective NP2
 NP2  →  common_noun Mods
```

Parsing and Ambiguity

- A grammar is said to be ambiguous if there is a sequence of words with two distinct parse trees.



The issue of prepositional phrase attachment

Natural Language Processing with Prolog

- We showcase how to develop a Prolog program that does simple syntactic and semantic processing of noun phrase written in English.
  - Parsing the noun phrases according to a given grammar.
  - Determining what is being referred to by the noun phrases.

A Knowledge Base in Prolog

- World model
  - Clauses represent what is known about the relevant words.
  - What the objects are, who the people are, where they are located, what they are wearing, etc.
  - Nothing in the word model is intended to be language-specific.
- Lexicon
  - Clauses describe the English words used in the noun phrases.
  - Linking the words to their meaning in the world model.
  - Nothing in the lexicon depends on the grammar.
- Parser
  - Clauses define the grammar.

The World Model

- World model has nothing to do with English.
- A collection of clauses about some world of interest.
- Representing the background knowledge that will be used in thinking about expressions in English.
- All clauses are atomic.

```
person(john). person(george). person(mary). person(linda).
park(kew_beach). park(queens_park).
tree(tree01). tree(tree02).  tree(tree03).
hat(hat01).   hat(hat02).  hat(hat03).  hat(hat04).

sex(john,male).    sex(george,male).
sex(mary,female).  sex(linda,female).

color(hat01,red).   color(hat02,blue).
color(hat03,red).   color(hat04,blue).

in(john,kew_beach).     in(george,kew_beach).
in(linda,queens_park).  in(mary,queens_park).
in(tree01,queens_park). in(tree02,queens_park).
in(tree03,kew_beach).

beside(mary,linda). beside(linda,mary).

on(hat01,john). on(hat02,mary). on(hat03,linda). on(hat04,george).

size(john,small).    size(george,big).
size(mary,small).    size(linda,small).
size(hat01,small).   size(hat02,small).
size(hat03,big).     size(hat04,big).
size(tree01,big).    size(tree02,small).  size(tree03,small).
```

The Lexicon

- The lexicon needs to describe all the English words to be used and how they relate to the **predicates** and **constraints** in the world model.

- Type of words
  - What are common nouns?
  - What are adjectives?
  - What are prepositions?
  - ...

Writing a Lexicon

- POS
    - article(W): for the word W which is an article.
- Type of Objects
    - common_noun(man, X) :- person(X), **sex**(X, male).
    - If X is a person whose sex is male, then X can be referred to when using the common noun *man*.
- Modifier
    - adjective(red, X) :- **color**(X, red)
    - If X has the color red, then X can be referred to when using the adjective *red*.

Writing a Lexicon

- Preposition
  - preposition(with, X, Y) :- **on**(Y, X).
  - If Y is on X, then that relation can be described when using the preposition *with*.
- Proper Nouns
  - proper_noun(W, X) holds when W is a name for the object X in the world model.

```prolog
article(a).  article(the).

common_noun(park,X) :- park(X).
common_noun(tree,X) :- tree(X).
common_noun(hat,X) :- hat(X).
common_noun(man,X) :- person(X), sex(X,male).
common_noun(woman,X) :- person(X), sex(X,female).

adjective(big,X) :- size(X,big).
adjective(small,X) :- size(X,small).
adjective(red,X) :- color(X,red).
adjective(blue,X) :- color(X,blue).

preposition(on,X,Y) :- on(X,Y).
preposition(in,X,Y) :- in(X,Y).
preposition(beside,X,Y) :- beside(X,Y).
% The preposition 'with' is flexible in how it is used.
preposition(with,X,Y) :- on(Y,X).        % Y can be on X
preposition(with,X,Y) :- in(Y,X).        % Y can be in X
preposition(with,X,Y) :- beside(Y,X).    % Y can be beside X

% Any word that is not in one of the four categories above.
proper_noun(X,X) :- \+ article(X), \+ adjective(X,_),
                    \+ common_noun(X,_), \+ preposition(X,_,_).
```

Any word that does not belong to the other categories is a proper noun.

Lexicon vs World Model

- Some words in the lexicon are the same as the predicates and constants in the world model.

```
common_noun(hat, X) :- hat(X).
adjective(small, X) :- size(X, small).
preposition(in, X, Y) :- in(X,Y).
```

- But the lexicon and the world model have different purposes.
  - Lexicon describes the words being used.
  - World model describes the facts in the world.

```
person(john). person(george). person(mary). person(linda).
park(kew_beach). park(queens_park).
tree(tree01). tree(tree02).  tree(tree03).
hat(hat01).   hat(hat02).  hat(hat03).  hat(hat04).

sex(john,male).    sex(george,male).
sex(mary,female).  sex(linda,female).

color(hat01,red).   color(hat02,blue).
color(hat03,red).   color(hat04,blue).

in(john,kew_beach).      in(george,kew_beach).
in(linda,queens_park).   in(mary,queens_park).
in(tree01,queens_park).  in(tree02,queens_park).
in(tree03,kew_beach).

beside(mary,linda). beside(linda,mary).

on(hat01,john). on(hat02,mary). on(hat03,linda). on(hat04,george).

size(john,small).     size(george,big).
size(mary,small).     size(linda,small).
size(hat01,small).    size(hat02,small).
size(hat03,big).      size(hat04,big).
size(tree01,big).     size(tree02,small).  size(tree03,small).
```

```
article(a).  article(the).

common_noun(park,X) :- park(X).
common_noun(tree,X) :- tree(X).
common_noun(hat,X) :- hat(X).
common_noun(man,X) :- person(X), sex(X,male).
common_noun(woman,X) :- person(X), sex(X,female).

adjective(big,X) :- size(X,big).
adjective(small,X) :- size(X,small).
adjective(red,X) :- color(X,red).
adjective(blue,X) :- color(X,blue).

preposition(on,X,Y) :- on(X,Y).
preposition(in,X,Y) :- in(X,Y).
preposition(beside,X,Y) :- beside(X,Y).
% The preposition 'with' is flexible in how it is used.
preposition(with,X,Y) :- on(Y,X).        % Y can be on X
preposition(with,X,Y) :- in(Y,X).        % Y can be in X
preposition(with,X,Y) :- beside(Y,X).    % Y can be beside X

% Any word that is not in one of the four categories above.
proper_noun(X,X) :- \+ article(X), \+ adjective(X,_),
                    \+ common_noun(X,_), \+ preposition(X,_,_).
```

Any word that does not belong to the other categories is a proper noun.

Lexicon vs World Model

- The world model is to be language-neutral.
  - All the terms such as hat and small are expressions of concepts but not real English words.
  - The used of English terms are just for convenience.
- The lexicon is about the actual words.
  - It is important to use constants that corresponding exactly to the target language.

Distinguished the Lexicon from the World Model

- It generally good to distinguished between the world model and the lexicon.
- Reason 1: Different words may be used for the same concept.

```
common_noun(cap, X) :- hat(X).
common_noun(bonnet, X) :- hat(X).
```

```
common_noun(帽子, X) :- hat(X).
common_noun(chapeau, X) :- hat(X).
```

Distinguished the Lexicon from the World Model

- It generally good to distinguished between the world model and the lexicon.

- Reason 2: The same word may be used for different concepts.

```
common_noun(cap, X) :- hat(X).
common_noun(cap, X) :- bottle_top(X).
common_noun(cap, X) :- regulated_maximum(X).
```

Relations between Words and Concepts

- A parser will identifies word categories and locate objects in the world model.

```
?- color(X, red), hat(X).
```

```
person(john). person(george). person(mary). person(linda).
park(kew_beach). park(queens_park).
tree(tree01). tree(tree02).  tree(tree03).
hat(hat01).   hat(hat02).  hat(hat03).  hat(hat04).
sex(john,male).     sex(george,male).
sex(mary,female).   sex(linda,female).
color(hat01,red).   color(hat02,blue).
color(hat03,red).   color(hat04,blue).
```

```
X = hat01
X = hat03
```

Relations between Words and Concepts

- A parser will identifies word categories and locate objects in the world model.

A red hat on a man

```
?- color(X, red), hat(X), on(X, Y), person(Y), sex(Y, male)
```

```
person(john). person(george). person(mary). person(linda).
park(kew_beach). park(queens_park).
tree(tree01). tree(tree02).  tree(tree03).
hat(hat01).   hat(hat02).  hat(hat03).  hat(hat04).
sex(john,male).    sex(george,male).
sex(mary,female).  sex(linda,female).
color(hat01,red).  color(hat02,blue).
color(hat03,red).  color(hat04,blue).
```

```
X = hat01
X = hat03
```

```
?- color(X, red), hat(X), on(X, Y), person(Y), sex(Y, male)
```

```
person(john). person(george). person(mary). person(linda).
park(kew_beach). park(queens_park).
tree(tree01). tree(tree02).  tree(tree03).
hat(hat01).   hat(hat02).  hat(hat03).  hat(hat04).

sex(john,male).     sex(george,male).
sex(mary,female).   sex(linda,female).

color(hat01,red).    color(hat02,blue).
color(hat03,red).    color(hat04,blue).

in(john,kew_beach).      in(george,kew_beach).
in(linda,queens_park).   in(mary,queens_park).
in(tree01,queens_park).  in(tree02,queens_park).
in(tree03,kew_beach).

beside(mary,linda). beside(linda,mary).

on(hat01,john). on(hat02,mary). on(hat03,linda). on(hat04,george).

size(john,small).     size(george,big).
size(mary,small).     size(linda,small).
size(hat01,small).    size(hat02,small).
size(hat03,big).      size(hat04,big).
size(tree01,big).     size(tree02,small).   size(tree03,small).
```

X = hat01
Y = john

The Parser

- Each non-terminal category in the grammar will have its own predicate in the parser.
- Each predicate takes two arguments:
  - A sequence of words to be parsed.
  - An object in the world model.
- A predicate will hold if the sequence of words is both of **the category stated by the predicate** and can be used to refer to the object according to **the facts in the world model**.

```
np([Name],X) :- proper_noun(Name,X).
np([Art|Rest],X) :- article(Art), np2(Rest,X).

np2([Adj|Rest],X) :- adjective(Adj,X), np2(Rest,X).
np2([Noun|Rest],X) :- common_noun(Noun,X), mods(Rest,X).
```

Syntactic Predicates

- Sequence of words is the essential form of the data in natural language.
- Each predicate uses list notation to extract the first word in the sequence and the remaining words.

```
np2([Adj|Rest], X) :- adjective(Adj, X), np2(Rest, X).
```

np2 → Adjective np2

Syntactic Predicates

- Sequence of words is the essential form of the data in natural language.
- Each predicate uses list notation to extract the first word in the sequence and the remaining words.

```
pp([Prep | Rest], X) :- preposition(Prep, X, Y), np(Rest, Y).
```

pp → Preposition np

X: The whole prepositional phrase
Y: The remaining NP

Syntactic Predicates

- Some predicates do not follow **the first and the rest** pattern.
- Breaking the sequence into two parts
  - The first part should be a pp
  - The second part should be another mods.

```
mods(Words, X) :- append(Start, End, Words),
                  pp(Start, X),
                  mods(End, X).
```

mods → pp mods

[In the park] [with a red hat]

```
np([Name],X) :- proper_noun(Name,X).
np([Art|Rest],X) :- article(Art), np2(Rest,X).

np2([Adj|Rest],X) :- adjective(Adj,X), np2(Rest,X).
np2([Noun|Rest],X) :- common_noun(Noun,X), mods(Rest,X).

mods([],_).
mods(Words,X) :-
   append(Start,End,Words),   % Break the words into two pieces.
   pp(Start,X),               % The first part is a PP.
   mods(End,X).               % The last part is a Mods again.

pp([Prep|Rest],X) :- preposition(Prep,X,Y), np(Rest,Y).
```

Perform Parsing

- With the world model, the lexicon, and the parser are ready, Prolog can perform parsing and interpret noun phrases.

- query the predicate **np** with a list of words as the first argument and a variable as the second argument.

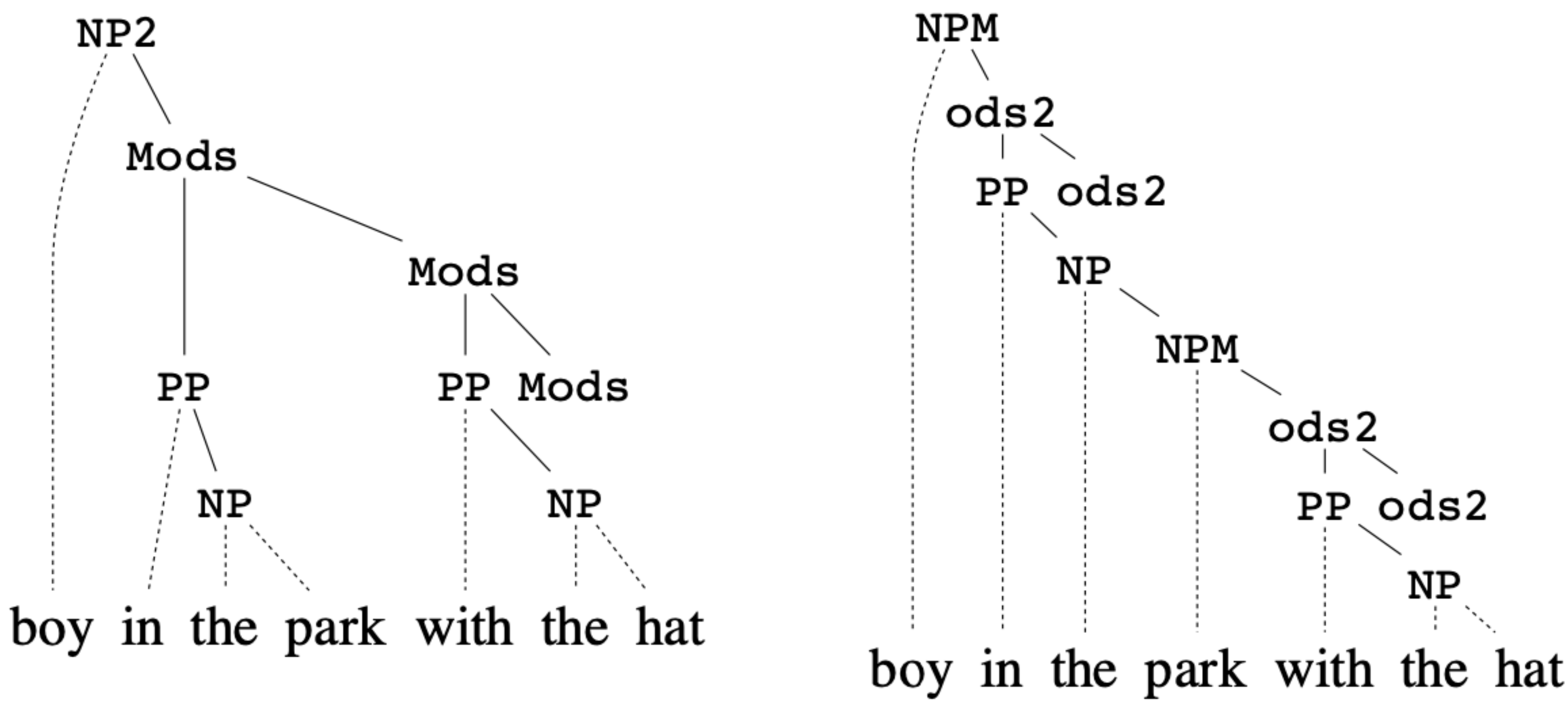
```
?- np([a, big, tree], X).
```

- Prolog will try to locate an object from the world model described by the noun phrase.

```
[trace]  ?- np([a, big, tree], X).
   Call: (8) np([a, big, tree], _3312) ? creep
   Call: (9) article(a) ? creep
   Exit: (9) article(a) ? creep
   Call: (9) np2([big, tree], _3312) ? creep
   Call: (10) adjective(big, _3312) ? creep
   Call: (11) size(_3312, big) ? creep
   Exit: (11) size(hat03, big) ? creep
   Exit: (10) adjective(big, hat03) ? creep
   Call: (10) np2([tree], hat03) ? creep
   Call: (11) adjective(tree, hat03) ? creep
   Fail: (11) adjective(tree, hat03) ? creep
   Redo: (10) np2([tree], hat03) ? creep
   Call: (11) common_noun(tree, hat03) ? creep
   Call: (12) tree(hat03) ? creep
   Fail: (12) tree(hat03) ? creep
   Fail: (11) common_noun(tree, hat03) ? creep
   Fail: (10) np2([tree], hat03) ? creep
   Redo: (11) size(_3312, big) ? creep
   Exit: (11) size(tree01, big) ? creep
   Exit: (10) adjective(big, tree01) ? creep
   Call: (10) np2([tree], tree01) ? creep
   Call: (11) adjective(tree, tree01) ? creep
   Fail: (11) adjective(tree, tree01) ? creep
   Redo: (10) np2([tree], tree01) ? creep
   Call: (11) common_noun(tree, tree01) ? creep
   Call: (12) tree(tree01) ? creep
   Exit: (12) tree(tree01) ? creep
   Exit: (11) common_noun(tree, tree01) ? creep
   Call: (11) mods([], tree01) ? creep
   Exit: (11) mods([], tree01) ? creep
   Exit: (10) np2([tree], tree01) ? creep
   Exit: (9) np2([big, tree], tree01) ? creep
   Exit: (8) np([a, big, tree], tree01) ? creep
X = tree01
```

Parsing and Ambiguity

- A grammar is said to be ambiguous if there is a sequence of words with two distinct parse trees.



The issue of prepositional phrase attachment

Disambiguation

- The parser deal with not only syntactic structure but also the ambiguity.

```
?- np([a, man, in, the, park, with, a, tree], X).

X = john .
```

- The parser needs to confirm the mods "in the park with a tree" describes john.
  - "in the park" and "with a tree" are two prepositional phrases attached to john.
  - "in the park with a tree" is a single prepositional phrase attached to john.
- The parser uses semantic considerations to reject the first interpretation.

```
person(john). person(george). person(mary). person(linda).
park(kew_beach). park(queens_park).
tree(tree01). tree(tree02).  tree(tree03).
hat(hat01).   hat(hat02).  hat(hat03).  hat(hat04).

sex(john,male).    sex(george,male).
sex(mary,female).  sex(linda,female).

color(hat01,red).   color(hat02,blue).
color(hat03,red).   color(hat04,blue).

in(john,kew_beach).     in(george,kew_beach).
in(linda,queens_park).  in(mary,queens_park).
in(tree01,queens_park). in(tree02,queens_park).
in(tree03,kew_beach).

beside(mary,linda). beside(linda,mary).

on(hat01,john). on(hat02,mary). on(hat03,linda). on(hat04,george).

size(john,small).    size(george,big).
size(mary,small).    size(linda,small).
size(hat01,small).   size(hat02,small).
size(hat03,big).     size(hat04,big).
size(tree01,big).    size(tree02,small).  size(tree03,small).
```

Additional Queries

```
?- np([a,man,with,a,big,hat], X).
X = george ;
false

?- np([the,hat,on,george], X).
X = hat04 ;
false

?-  np([a,man,in,a,park,with,a,big,tree], X).
false

?-  np([a,woman,in,a,park,with,a,big,tree], X).
X = mary ;
X = linda ;
false

?- np([a,woman,in,a,park,with,a,big,red,hat], X).
X = linda ;
false
```

More Complex Queries

- *A woman beside a woman with a blue hat*
  - The blue hat is attached to the first woman.
  - The blue hat is attached to the second woman.

```
?- np([a,woman,beside,a,woman,with,a,blue,hat],X).
X = mary ;  % Who wears a blue hat
X = linda ;
false

?- np([a,woman,with,a,blue,hat,beside,a,woman],X).
X = mary ;
false
```

```
?- np([a,woman,beside,a,woman,with,a,blue,hat],X).
```

```
person(john). person(george). person(mary). person(linda).
park(kew_beach). park(queens_park).
tree(tree01). tree(tree02).  tree(tree03).
hat(hat01).   hat(hat02).  hat(hat03).  hat(hat04).

sex(john,male).    sex(george,male).
sex(mary,female). sex(linda,female).

color(hat01,red).   color(hat02,blue).
color(hat03,red).   color(hat04,blue).

in(john,kew_beach).      in(george,kew_beach).
in(linda,queens_park).   in(mary,queens_park).
in(tree01,queens_park).  in(tree02,queens_park).
in(tree03,kew_beach).

beside(mary,linda). beside(linda,mary).

on(hat01,john). on(hat02,mary). on(hat03,linda). on(hat04,george).

size(john,small).     size(george,big).
size(mary,small).     size(linda,small).
size(hat01,small).    size(hat02,small).
size(hat03,big).      size(hat04,big).
size(tree01,big).     size(tree02,small).  size(tree03,small).
```

Phrase Generation

- Ask Prolog to fill a sequence with *n* words by considering both grammar and facts.

```
?- L=[_,_,_,_,_], np(L, linda), \+ member(the, L).
L = [a, small, small, small, woman] ;
L = [a, small, woman, in, queens_park] ;
L = [a, small, woman, beside, mary] ;
L = [a, small, woman, with, hat03] ;
L = [a, small, woman, with, mary] ;
L = [a, woman, in, a, park] ;
L = [a, woman, beside, a, woman] ;
L = [a, woman, with, a, hat] ;
L = [a, woman, with, a, woman] ;
false.
```

Interpreting Sentences

- With the simple parser, which deals with noun phrases, some simple forms of English sentences can now be considered.

- A sentence is also a list of words.

- Support sentence interpretation by adding additional syntactic predicates.

Parser for Yes/No Questions

- Add predicates for dealing with yes/no questions.

```
yes_no(S) :- yn(Words).


yn([Verb|Rest]) :- Verb=is,
                   append(W1,W2,Rest),
                   np(W1,Ref),
                   np_or_pp(W2,Ref).

np_or_pp(W,Ref) :- np(W,Ref).
np_or_pp(W,Ref) :- pp(W,Ref).
```

Queries with Yes/No Questions

- Returns false if grammatical error or the answer is false.

```
?- yes_no(["is", "mary", "in", "a", "park", "with",
"linda"]).
true .

?- yes_no(["is", "the", "man", "with", "the", "blue",
"hat", "john"]).
false .

?- yes_no(["is", "the", "big", "red", "hat", "on",
"the", "woman", "beside", "mary"]).
true .

?- yes_no(["is", "a", "red", "with", "a", "woman",
"hat"]). % Ungrammatical
false .
```

Updating the World Model

- In addition to the predefined world model, we can also update it on the fly.

- Instead of being used to **query** the world model, a **declarative sentence** is used to **update** it.

- In other words, a declarative sentence is interpreted as providing new information that needs to be incorporated into the world model.

    - In natural language!

Dynamic Predicates

- Prolog allows the clauses associated with some of the predicates to be changed by a program itself.
- Enabling a predicate p(X, Y, Z) **dynamic**

```
:- dynamic p/3
```

- Adding a fact

```
:- assert(atom)
```

- Removing a fact

```
:- retract(atom)
```

-

Updating Facts

- Update a fact

```
get_married(X) :- retract(single(X)), assert(married(X)).
```

- Dynamic predicates can be used to keep the changing world model up-to-date.

```
?- single(john).
true .

?- get_married(john).
true .

?- single(john).
false .

?- married(john).
false .
```

Keeping the Knowledge Base Integrity

```
single(john).
:- dynamic single/1.
is_single(X) :- single(X), \+ married(X).

married(jane).
:- dynamic married/1.
is_married(X) :- married(X), \+ single(X).

get_married(X) :- assert(married(X)).
```

```
?- is_single(john).
true.

?- is_married(john).
false.

?- get_married(john).
true.

?- is_single(john).
false.

?- is_married(john).
false.
```

Adding Facts with Natural Language

- Add a new fact that is expressed in natural language

The man with the red hat **is** in the park with the big tree

```
add_for_preposition(on, X, Y) :- assert(on(X, Y)).
add_for_preposition(in, X, Y) :- assert(in(X, Y)).

add_for_preposition(beside, X, Y) :-
    assert(beside(X, Y)), assert(beside(Y, X)).

sd(Words) :-
    append(NP1, [is, Prep | NP2], Words),
    np(NP1, X),
    np(NP2, Y),
    add_for_preposition(Prep, X, Y).
```

Adding Facts with Natural Language

```
?- np([the, man, with, the, red, hat], X).
X = john .

?- np([the, man, with, the, big, tree], X).
false.

?- sd([the, man, with, the, red, hat, is, beside, the, big, tree]).
true .

?- np([the, man, beside, the, big, tree], X).
X = john .

?- np([the, man, with, the, big, tree], X).
X = john .

?- np([the, tree, beside, the, man, with, the, red, hat], X).
X = tree01 .

?- np([the, tree, with, the, man, with, the, red, hat], X).
X = tree01 .
```

Non-referential Noun Phrases

- Not all noun phrases are referential.
- Negation

  a man without a hat

- Different objects

  Mary eats an apple every day

- Ambiguity

  John wants to marry a rich lawyer

Probabilistic Logic Programming

Problog

- Assign the probability of each clause
  - And we can do probabilistic logic programming!

```
1/6::one1; 1/6::two1; 1/6::three1; 1/6::four1; 1/6::five1;
1/6::six1.
0.15::one2; 0.15::two2; 0.15::three2; 0.15::four2; 0.15::five2;
0.25::six2.

twoSix :- six1, six2.
someSix :- six1.
someSix :- six2.

query(six1).      %  0.1666666
query(six2).      %  0.25
query(twoSix).    %  0.375
query(someSix).   %  0.0416666
```

Champion Prediction

```
 1 Brazil ------+
                +-- ? --+
 2 Chile -------+       |
                        +-- ? --+
 3 Nigeria -----+       |       |
                +-- ? --+       |
 4 Denmark -----+               |
                                +-- ? --+
 5 Holland -----+               |       |
                +-- ? --+       |       |
 6 Yugoslavia -+        |       |       |
                        +-- ? --+       |
 7 Argentina --+        |               |
                +-- ? --+               |
 8 England -----+                       |
                                        +-- World Champion
 9 Italy -------+                       |
                +-- ? --+               |
10 Norway ------+       |               |
                        +-- ? --+       |
11 France ------+       |       |       |
                +-- ? --+       |       |
12 Paraguay ---+                |       |
                                +-- ? --+
13 Germany -----+               |
                +-- ? --+       |
14 Mexico ------+       |       |
                        +-- ? --+
15 Romania -----+       |
                +-- ? --+
16 Croatia -----+
```

```
0.650000::wins(brazil, chile).
0.500000::wins(brazil, nigeria).
0.600000::wins(brazil, denmark).
0.550000::wins(brazil, holland).
0.500000::wins(brazil, yugoslavia).
0.500000::wins(brazil, argentina).
0.650000::wins(brazil, england).
0.450000::wins(brazil, italy).
0.550000::wins(brazil, norway).
0.400000::wins(brazil, france).
0.550000::wins(brazil, paraguay).
0.400000::wins(brazil, germany).
0.550000::wins(brazil, mexico).
0.500000::wins(brazil, romania).
0.500000::wins(brazil, croatia).
```

```
partition(L, A, B) :- length(L, N),
                      N > 1,
                      append(A, B, L),
                      same_length(A, B).

winner([X], X).
winner(Countries, X) :- partition(Countries, Semi1, Semi2),
                        winner(Semi1, X),
                        winner(Semi2, Y),
                        wins(X, Y).

winner(Countries, Y) :- partition(Countries, Semi1, Semi2),
                        winner(Semi1, X),
                        winner(Semi2, Y),
                        wins(Y, X).

query(winner([brazil, chile], brazil)).

query(winner([brazil, chile, nigeria, denmark], brazil)).
```