LING 581: Advanced Computational Linguistics

Lecture Notes

April 17th
Two Topics

• Homework 9
• Grammar and Logic
Homework 9

- Use `cosines.py` or `bfs4.perl` (or some nltk similarity measure mentioned in a previous lecture) or a mix of the two to come up with a method of matching words to (simple) definitions for quizzes A and B (shown on the next slides)
- State your results: how many your method got right.
- State your heuristics.
- State how you consistently handled multiword definitions, e.g.

  ```python
  >>> [tuple[0] for tuple in nltk.pos_tag(nltk.word_tokenize(df)) if re.match('NN|JJ', tuple[1])]
  ['art', 'beauty']       # 'relating to or appreciating art or beauty'
  ['warn', 'express', 'disapproval'] # 'warn or express disapproval'
  ```
- Due next Monday night (by midnight)
Homework 9: Quiz A

1. adroit  
2. adumbrate  
3. adulation  
4. admonish  
5. alchemy  
6. alacrity  
7. adulterate  
8. aggrandize  
9. aesthetic

a. foreshadow  
b. adept  
c. warn or express disapproval  
d. increase in power  
e. relating to or appreciating art or beauty  
f. excessive praise  
g. enthusiastic willingness  
h. reduce purity of something  
i. magical transformation
Homework 9: Quiz B

1. episodic
2. epithet
3. ephemeral
4. errant
5. epitome
6. erudite
7. equivocate
8. epicure
9. equanimity

a. one devoted to sensual pleasure
b. embodiment
c. traveling
d. disparaging word
e. scholarly
f. fleeting
g. self-possession
h. loosely connected
i. use ambiguous language
Homework 9

• Use `cosines.py` or `bfs4.perl` (or some nltk similarity measure mentioned in a previous lecture) or a mix of the two to come up with a method of matching words to (simple) definitions for quizzes A and B (shown on the next slides)

• State your results: how many your method got right.

• State your heuristics.

• State how you consistently handled multiword definitions, e.g.

```python
>>> [tuple[0] for tuple in nltk.pos_tag(nltk.word_tokenize(df)) if re.match('NN|JJ',tuple[1])]
['art', 'beauty']
'relating to or appreciating art or beauty'
['warn', 'express', 'disapproval']
'warn or express disapproval'
```

• Due next Monday night (by midnight)
Homework 9: Quiz A

1. adroit
2. adumbrate
3. adulation
4. admonish
5. alchemy
6. alacrity
7. adulterate
8. aggrandize
9. aesthetic

a. foreshadow
b. adept
c. warn or express disapproval
d. increase in power
e. relating to or appreciating art or beauty
f. excessive praise
g. enthusiastic willingness
h. reduce purity of something
i. magical transformation
Homework 9: Quiz B

1. episodic
2. epithet
3. ephemeral
4. errant
5. epitome
6. erudite
7. equivocate
8. epicure
9. equanimity

   a. one devoted to sensual pleasure
   b. embodiment
   c. traveling
   d. disparaging word
   e. scholarly
   f. fleeting
   g. self-possession
   h. loosely connected
   i. use ambiguous language
Meaning

• What is a meaning and how do we represent it?
  • difficult to pin down precisely for computers
  • even difficult for humans sometimes...

• Example: word *dog*
  • by reference to other words
    • Merriam-Webster: a highly variable domestic mammal (Canis familiaris) closely related to the gray wolf
  • translation
    • 犬 (INU, Japanese) = 狗 (GOU, Chinese) = “dog” (English)

• Computer:
  • meaning ➔ formal concept (or thought or idea)
  • “dog” maps to DOG
  • <word> maps to <concept>
  • need to provide a concept for every meaningful piece of language?
Understanding

• Suppose we write a computer program to compute the meaning of sentences
• Question: does it understand sentences?
• How do you know?
• Ask questions?
• Turing test:
  • converse with a human, convince human the computer is a human
• Searle’s Chinese room experiment (adapted)
  • suppose we have a Perl/Python/Prolog program capable of processing Chinese, and we “run” the program manually
  • i.e. we carry out the instructions of the program
  • do we understand Chinese?
• Weak AI / Strong AI
Truth Conditions and Values

- Example:
  - The square is bigger than the circle
  - The circle is smaller than the square

- We can draw pictures of scenarios for which the statement is true and the statement is false (truth-conditions)
  - set of possible worlds (aka situations)

- **Note:** truth-conditions different from truth-value (True/False)
  - Are they synonymous?
  - Are they contradictory?
  - Is there an entailment relationship?
  - Are they tautologies?
More examples

• 1. Does *sleep* entail *snore*?
   A. *He is sleeping* entails *He is snoring*
   B. *He is snoring* entails *He is sleeping*

• 2. Does *snore* presuppose *sleep*?

<table>
<thead>
<tr>
<th>Verb</th>
<th>Wordnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <strong>S:</strong> <em>(v)</em> snore, saw wood, saw logs <em>(breathe noisily during one's sleep)</em> “she complained that her husband snores”</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>direct hypernym</em> / <em>inherited hypernym</em> / <em>sister term</em></td>
</tr>
<tr>
<td></td>
<td><strong>entailment</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>S:</strong> <em>(v)</em> sleep, kip, slumber, log Z's, catch some Z's <em>(be asleep)</em></td>
</tr>
<tr>
<td></td>
<td><em>derivationally related form</em></td>
</tr>
<tr>
<td></td>
<td><em>sentence frame</em></td>
</tr>
</tbody>
</table>
More examples

• 3. What does “when did you stop beating your wife?” presuppose?

• 4. Given the statement “All crows are black”, give an example of a sentence expressing a tautology involving this statement?
  • Answer: Stmt or negation of Stmt
Propositional Logic

• Recall the distinction between truth conditions and truth values ...

• Possible world or situation:
  • we can create a possible world in Prolog by asserting (positive) facts into its database
  • Prolog use the closed world assumption
    • i.e. things not explicitly stated to be true are assumed to be false
Propositional Logic

Cheat sheet

• Starting SWI Prolog from Terminal/Shell:
  • `swipl` (if in PATH)
  • `/Applications/SWI-Prolog.app/Contents/MacOS/swipl` (default install location on my mac)
  • ^D (control-D) or `halt` to quit
Propositional Logic

Cheat sheet

• Viewing the database:
  • listing.

• Assert (and delete) facts at the command line directly using predicates:
  • assert(\textit{fact}).
  • retract(\textit{fact}).

• Put facts into a file and load file
  • $[^\textit{filename}]$. \hspace{1cm} (assumed to have extension .pl)
  • (or via pull-down menu in Windows)

• Propositions:
  • named beginning with a lower case letter (not number, not starting with capital letter or underscore: variable – no variables in propositional logic), examples:
    • assert(p). \hspace{1cm} (makes p true in this situation)
    • p. \hspace{1cm} (asks Prolog if p true in this situation)
    • dynamic q. \hspace{1cm} (registers proposition q, prevents error message)
Propositional Logic

• Example:

Note: meta-level predicates like `dynamic` and `assert` evaluate to true if they succeed
Propositional Logic

• Propositions can be combined using logical connectives and operators
  • Conjunction \( p \land q \)
  • Disjunction \( p \lor q \)
  • Negation \( \neg p \)

• Not directly implemented in Prolog
  • Implication \( p \rightarrow q \)

Use parentheses ( ) to restrict/clarify scope

can’t add \( p, q \) to the database

needs both \( p \) and \( q \) to be true, see next slide
Propositional Logic

• Help:
  • ?- help(->).
  • true.

  takes a very long time for this window to pop up ... it uses the X11 Window system, which may or may not exist on your system

IF -> THEN ; ELSE is a programming construct
Propositional Logic

• Also not implemented
  • Logical equality  \( p = q \).

```prolog
?- p = q.
false.

?- p = p.
true.

?- q = q.
true.

?- 
```

= doesn't mean assignment to a variable!
It means **unifiable**
Propositional Logic

• Help:

```prolog
?- help(=).
true.

?- (p,q) = (p,q).
true.

?- (p;(q,r)) = (p;(q,r)).
true.

?- 
```

= is unifiability in Prolog
Propositional Logic

• Prolog exercise:
  • evaluate formula below for different truth values of A and B

\[
(\neg A) \lor (A \land \neg B) \lor (\neg B).
\]

From wikipedia

?- dynamic a.
true.

?- dynamic b.
true.

?- (a \_ b) ; \+_a ; \_+ b.
Propositional Logic

• How to demonstrate a propositional formula is a tautology?

  • **One answer:** exhaustively enumerate a truth table

```
<table>
<thead>
<tr>
<th></th>
<th>Logical Conjunction</th>
<th>Logical Disjunction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( p \wedge q )</td>
<td>( p \lor q )</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Logical Negation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>( \neg p )</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
```

http://en.wikipedia.org/wiki/Truth_table
### Propositional Logic

- **Example:**
\[(A \land B) \lor (\neg A) \lor (\neg B)\]

<table>
<thead>
<tr>
<th>(A, B)</th>
<th>(\lor A)</th>
<th>(\lor B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T, T</td>
<td>T, T</td>
<td>T, T</td>
</tr>
<tr>
<td>T, F</td>
<td>T, T</td>
<td>T, T</td>
</tr>
<tr>
<td>F, F</td>
<td>T, T</td>
<td>T, T</td>
</tr>
<tr>
<td>F, F</td>
<td>T, T</td>
<td>T, T</td>
</tr>
</tbody>
</table>

Table has \(2^n\) rows, where \(n\) is the number of propositional elements.

**Complexity:** exponential in \(n\)
Propositional Logic

• Other connectives (are non-primitive)
Propositional Logic

• Other connectives (are non-primitive)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>q</td>
<td>p → q</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

aka

p ↔ q

• From 1\textsuperscript{st} and 4\textsuperscript{th} line of truth table, we can easily deduce how to simulate \( p \leftrightarrow q \) in Prolog using , ; and \(+\)
Propositional Logic

There are infinitely many tautologies. Examples include:

- \((A \lor \neg A)\) ("A or not A"), the law of the excluded middle. This formula has only one propositional variable, \(A\). Any valuation for this formula must, by definition, assign \(A\) one of the truth values true or false, and assign \(\neg A\) the other truth value.
- \((A \rightarrow B) \iff (\neg B \rightarrow \neg A)\) ("if \(A\) implies \(B\) then not-\(B\) implies not-\(A\)", and vice versa), which expresses the law of contraposition.
- \(((\neg A \rightarrow B) \land (\neg A \rightarrow \neg B)) \rightarrow A\) ("if not-\(A\) implies both \(B\) and its negation not-\(B\), then not-\(A\) must be false, then \(A\) must be true"), which is the principle known as reductio ad absurdum.
- \((\neg A \land B) \iff (\neg A \lor \neg B)\) ("if not both \(A\) and \(B\), then either not-\(A\) or not-\(B\)", and vice versa), which is known as de Morgan's law.
- \(((A \rightarrow B) \land (B \rightarrow C)) \rightarrow (A \rightarrow C)\) ("if \(A\) implies \(B\) and \(B\) implies \(C\), then \(A\) implies \(C\)"), which is the principle known as syllogism.
- \(((A \lor B) \land (A \rightarrow C) \land (B \rightarrow C)) \rightarrow C\) (if at least one of \(A\) or \(B\) is true, and each implies \(C\), then \(C\) must be true as well), which is the principle known as proof by cases.

A minimal tautology is a tautology that is not the instance of a shorter tautology.

- \((A \lor B) \rightarrow (A \lor B)\) is a tautology, but not a minimal one, because it is an instantiation of \(C \rightarrow C\).
Propositional Logic

• Prove both sides of De Morgan’s Laws:

\[ \neg (P \lor Q) \iff (\neg P) \land (\neg Q) \]
\[ \neg (P \land Q) \iff (\neg P) \lor (\neg Q) \]

**Note:** De Morgan’s laws tell us we can do without one of conjunction or disjunction. Why?
Propositional Logic

- It’s easy to write a short program in Prolog to automate all this ...

Program: plogic.pl

```prolog
try(Vars,Formula) :-
    toggleL(Vars,Xs), eval(Formula,Xs),
    fail.
try(_,_).

% toggles list of propositional variables as true and false
% tries every combination
toggleL([],[]).
toggleL([V|Vs],[X|Xs]) :- toggle(V,X), toggleL(Vs,Xs).
toggle(V,X) :- assert(V, X =\+ V) ; retract(V), X = \+ V.
eval(F,Xs) :- call(F) -> format('~w true-n', [Xs]) ; format('~w false-n', [Xs]).
```
Propositional Logic

• Example using `try/2`:

\[(A \land B) \lor (\neg A) \lor (\neg B)\]

So it's a tautology! i.e. true under all possible conditions

Careful with Prolog syntax:

\[\text{\texttt{\textbackslash + ((p,q)) NO}}\]
\[\text{\texttt{\textbackslash +(p,q) NO}}\]
\[\text{\texttt{\textbackslash +((p,q)) OK}}\]
Propositional Logic

• We can get a bit fancier, support $\rightarrow$ and $\leftrightarrow$

```
1 :- op(1200, xfx, '↔').
2 try(Vars, Formula) :-
3     convert(Formula, PrologFormula),
4     toggleL(Vars, Xs), eval(PrologFormula, Xs),
5     fail.
6 try(_, _).
7
8 % handles $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence)
9 convert((\+ X), (\+ A)) :- !, convert(X, A).
10 convert((X, Y), (A, B)) :- !, convert(X, A), convert(Y, B).
11 convert((X; Y), (A; B)) :- !, convert(X, A), convert(Y, B).
12 convert((X→Y), (\+A; B)) :- !, convert(X, A), convert(Y, B).
13 convert((X↔Y), ((A, B); (\+A, \+B))) :- !, convert(X, A), convert(Y, B).
14 convert(X, X).
15
16 % toggles list of propositional variables as true and false
17 % tries every combination
18 toggleL([], []).
19 toggleL([V|Vs], [X|Xs]) :- toggle(V, X), toggleL(Vs, Xs).
20
toggle(V, X) :- assert(V), X = V ; retract(V), X = ~V.
21 eval(F, Xs) :- call(F) -> format('¬w true¬n', [Xs]) ; format('¬w false¬n', [Xs]).
```
Propositional Logic

- We can get even fancier; eliminate having to supply the propositional variables

```prolog
1 :- op(1200,xfX, '<->').
2 try(Formula) :-
3     convert(Formula, PrologFormula, [], Vars),
4     reverse(Vars, Vars2),
5     toggle(Vars2, Xs), eval(PrologFormula, Xs),
6     fail.
7 try(_).
8 % handles -> (implication) and <-> (equivalence)
9 convert(\(\(\rightarrow\) X\)), \(\(\rightarrow\) A\), Vs1, Vs2) :- !, convert(X, A, Vs1, Vs2).
10 convert(X, Y, [A, B], Vs1, Vs3) :- !, convert(X, A, Vs1, Vs2), convert(Y, B, Vs2, Vs3).
11 convert(X, Y, [A, B], Vs1, Vs3) :- !, convert(X, A, Vs1, Vs2), convert(Y, B, Vs2, Vs3).
12 convert(X, Y, [\(\neg\) A, B], Vs1, Vs3) :- !, convert(X, A, Vs1, Vs2), convert(Y, B, Vs2, Vs3).
13 convert(X, Y, [\(\neg\) A, B], Vs1, Vs3) :- !, convert(X, A, Vs1, Vs2), convert(Y, B, Vs2, Vs3).
14 convert(X, Y, [\(\neg\) A, \(\neg\) B], Vs1, Vs3) :- !, convert(X, A, Vs1, Vs2), convert(Y, B, Vs2, Vs3).
15 convert(X, Y, [\(\neg\) A, \(\neg\) B, \(\neg\) (\(\neg\) A, \(\neg\) B)], Vs1, Vs3) :- !, convert(X, A, Vs1, Vs2), convert(Y, B, Vs2, Vs3).
16 convert(X, X, Vs1, Vs2) :- member(X, Vs1) -> Vs2 = Vs1 ; Vs2 = [X| Vs1].
17 % toggles list of propositional variables as true and false
18 % tries every combination
19 toggle([[], []].
20 toggle([\(\neg\) Vs], [X|Vs]) :- toggle(V, X), toggle(Vs, Xs).
21 toggle(V, X) :- assert(V), X \(\neg\) V ; retract(V), X \(\neg\) V.
22 eval(F, Xs) :- call(F) -> format('\(\neg\) w true-n', [Xs]); format('\(\neg\) w false-n', [Xs]).
```
Truth table enumeration

• Parsing the formula:

    11. \+ X converts to \+ A if (subformula) X converts to A
    12. X,Y converts to A,B if X converts to A and Y converts to B
    13. X;Y converts to A;B if X converts to A and Y converts to B
    14. X->Y converts to \+A;B if X converts to A and Y converts to B
    15. X<->Y converts to (A,B) ; (\+A,\+B) if X converts to A and Y converts to B
    16. X converts to X and add X to the list of propositional variables if it isn’t already in the list
Propositional Logic

Program: plogic3.pl

\[(A \rightarrow B) \iff (\neg B \rightarrow \neg A)\]

\[
\begin{align*}
(A \rightarrow B) & \iff (\neg B \rightarrow \neg A) \\
\text{[plogic3].} & \\
% plogic3 compiled 0.00 sec, 120 bytes true. & \\
\text{?- try((a->b)<->(\neg b \rightarrow \neg a))).} & \\
[a,b] & \text{true} \\
[a,-b] & \text{true} \\
[-a,b] & \text{true} \\
\neg (A \wedge B) & \iff (\neg A \vee \neg B) \\
\text{- - - - - - - - - -} & \\
\text{?- try((\neg(a;b))<->(\neg a;\neg b))).} & \\
[a,b] & \text{true} \\
[a,-b] & \text{true} \\
[-a,b] & \text{true} \\
[-a,-b] & \text{true} \\
\text{true.} & \\
\end{align*}
\]

\[
((A \rightarrow B) \wedge (B \rightarrow C)) \rightarrow (A \rightarrow C)
\]

\[
\text{?- try(((a->b),(b->c))\rightarrow(a->c))).} \\
[a,b,c] & \text{true} \\
[a,b,-c] & \text{true} \\
[a,-b,c] & \text{true} \\
[a,-b,-c] & \text{true} \\
[-a,b,c] & \text{true} \\
[-a,b,-c] & \text{true} \\
[-a,-b,c] & \text{true} \\
[-a,-b,-c] & \text{true} \\
\text{true.} \\
\]

\[
((A \vee B) \wedge (A \rightarrow C) \wedge (B \rightarrow C)) \rightarrow C
\]

\[
\text{?- try(((a;b),(a->c),(b->c))\rightarrow(c))).} \\
[a,b,c] & \text{true} \\
[a,b,-c] & \text{true} \\
[a,-b,c] & \text{true} \\
[a,-b,-c] & \text{true} \\
[-a,b,c] & \text{true} \\
[-a,b,-c] & \text{true} \\
[-a,-b,c] & \text{true} \\
[-a,-b,-c] & \text{true} \\
\text{true.} \\
\]
Semantic Grammars

• Use slides from course
  • LING 324 – Introduction to Semantics
    • Simon Fraser University, Prof. F.J. Pelletier
    • http://www.sfu.ca/~jeffpell/Ling324/fjpSlides4.pdf

• Difference is we’re computational linguists...
  so we’re going to implement the slides

• We’ll do the syntax part this lecture, and the semantics next time
Syntax

Syntax of a Fragment of English (F1)

(2)  a. $S \rightarrow N \ VP$
    b. $S \rightarrow S \ conj \ S$
    c. $S \rightarrow \ neg \ S$
    d. $VP \rightarrow V_t \ N$
    e. $VP \rightarrow V_i$
    f. $N \rightarrow \text{Jack, Sophia, James}$
    g. $V_i \rightarrow \text{is boring, is hungry, is cute}$
    h. $V_t \rightarrow \text{likes}$
    i. $\text{conj} \rightarrow \text{and, or}$
    j. $\text{neg} \rightarrow \text{it is not the case that}$
Syntax

• We already know how to build Prolog grammars
• See
  for the executive summary

4.12 DCG Grammar rules

Grammar rules form a comfortable interface to difference-lists. They are designed both to support writing parsers that build a parse tree from a list of characters or tokens as for generating a flat list from a term.

Grammar rules look like ordinary clauses using $-->/2$ for separating the head and body rather than $:~/2$. Expanding grammar rules is done by `expand_term/2`, which adds two additional argument to each term for representing the difference list.

The body of a grammar rule can contain three types of terms. A callable term is interpreted as a reference to a grammar-rule. Code between `{...}` is interpreted as plain Prolog code and finally, a list is interpreted as a sequence of literals. The Prolog control-constructs (`\+/1, ~-/2, /\-2, /\+2 and \+0`) can be used in grammar rules.
Syntax

• **Step 1:** let’s build the simplest possible Prolog grammar for this

\[
\begin{align*}
(2) & \quad a. \quad S \rightarrow N \ VP \\
& \quad b. \quad S \rightarrow S \ conj \ S \\
& \quad c. \quad S \rightarrow \ neg \ S \\
& \quad d. \quad VP \rightarrow V_t \ N \\
& \quad e. \quad VP \rightarrow V_i \\
& \quad f. \quad N \rightarrow \ Jack, \ Sophia, \ James \\
& \quad g. \quad V_i \rightarrow \ is \ boring, \ is \ hungry, \ is \ cute \\
& \quad h. \quad V_t \rightarrow \ likes \\
& \quad i. \quad conj \rightarrow \ and, \ or \\
& \quad j. \quad neg \rightarrow \ it \ is \ not \ the \ case \ that \\
\end{align*}
\]

(3) Jack is hungry.
(4) Sophia likes James.
(5) It is not the case that James is cute.
(6) Jack is hungry, and it is not the case that James likes Jack.
(7) It is not the case that Jack is hungry or Sophia is boring.

**fjpSlides4.pdf**

Slide 4
Simplest possible grammar

```plaintext
psg1.pl

1 s --> n, vp.
2 s --> neg, s.
3 vp --> vt, n.
4 vp --> vi.
5 n --> [jack].
6 n --> [sophia].
7 n --> [james].
8 vi --> [is,boring].
9 vi --> [is,hungry].
10 vi --> [is,cute].
11 vt --> [likes].
12 conj --> [and].
13 conj --> [or].
14 neg --> [it,is,not,the,case,that].
```

Excluding (2b) for the time being
Simplest possible grammar

Examples (3), (4) and (5) from two slides back
Syntax

• **Step 2:** let’s add the parse tree component to our grammar …

(2) a. \( S \rightarrow N \ VP \)
b. \( S \rightarrow S \ conj \ S \)
c. \( S \rightarrow \text{neg} \ S \)
d. \( VP \rightarrow V_t \ N \)
e. \( VP \rightarrow V_i \)
f. \( N \rightarrow \text{Jack, Sophia, James} \)
g. \( V_i \rightarrow \text{is boring, is hungry, is cute} \)
h. \( V_t \rightarrow \text{likes} \)
i. \( \text{conj} \rightarrow \text{and, or} \)
j. \( \text{neg} \rightarrow \text{it is not the case that} \)

Recall: grammar rules can have extra arguments
(1) Parse tree
(2) Implement agreement etc.
Syntax

**Note:** on handling left recursion in Prolog grammar rules

- techniques:
  1. use a bottom-up parser
  2. rewrite grammar (left recursive -> right recursive)
  3. or use lookahead (today’s lecture)

```
b. \[S \rightarrow S \text{ conj } S\]
```

```prolog
lookahead is a dummy nonterminal that does not contribute to the parse, it is a “guard” that prevents rule from firing unless appropriate

lookahead succeeds if it can find a conjunction in the input and marks it (so it can’t find it twice)
```

```
conj(conj(and)) --> [and1].
conj(conj(or)) --> [or1].

conj(and, and1).
conj(or, or1).

lookahead(Words1, Words2) :- conj(C, C1), append(Left, [C1|Right], Words1), !, append(Left, [C1|Right], Words2).
```
Grammar: version 2

psg2.pl

1 \( s(s(N, VP)) \rightarrow n(N), vp(VP). \)
2 \( s(s(S1, Conj, S2)) \rightarrow \text{lookahead}, s(S1), \text{conj}(Conj), s(S2). \)
3 \( s(s(Neg, S)) \rightarrow \text{neg}(Neg), s(S). \)
4 \( vp(vp(VT, N)) \rightarrow vt(VT), n(N). \)
5 \( vp(vp(VI)) \rightarrow vi(VI). \)
6 \( n(n(jack)) \rightarrow [jack]. \)
7 \( n(n(sophia)) \rightarrow [sophia]. \)
8 \( n(n(james)) \rightarrow [james]. \)
9 \( vi(v(v(is), ap(boring))) \rightarrow [is, boring]. \)
10 \( vi(v(v(is), ap(hungry))) \rightarrow [is, hungry]. \)
11 \( vi(v(v(is), ap(cute))) \rightarrow [is, cute]. \)
12 \( vt(v(likes)) \rightarrow [likes]. \)
13 \( \text{conj}(\text{conj}(and)) \rightarrow [and1]. \)
14 \( \text{conj}(\text{conj}(or)) \rightarrow [or1]. \)
15 \( \text{neg}(\text{neg}(not)) \rightarrow [it, is, not, the, case, that]. \)
Grammar: version 2

16 % lookahead/2 succeeds when it can find some conjunct in the input
17 % it marks the conjunct and returns the modified list
18 lookahead(L1,L2) :-
19     append(Left,[ConjRight],L1),
20     conj(Conj,MarkedConj),
21     !, % commit
22     append(Left,[MarkedConjRight],L2).
23
24 conj(and,and1).
25 conj(or,or1).
Grammar: version 2

?- [g2].
Warning: /Users/sandiway/Desktop/g2.pl:25:
    Redefined static procedure conj/2
% g2 compiled 0.00 sec, 8,024 bytes
ture.

?- s(X,[jack,is,hungry],[]).
X = s(n(jack), vp(v(v(is), ap(hungry)))) ;
false.

?- s(X,[sophia,likes,james],[]).
X = s(n(sophia), vp(v(likes), n(james))) ;
false.

?- s(X,[it,is,not,the,case,that,james,is,cute],[]).
X = s(neg(not), s(n(james), vp(v(v(is), ap(cute)))))) ;
false.

Examples (3), (4) and (5) again from slide 9
Grammar: version 2

Examples (6) and (7) from slide 9

?- s(X, [jack, is, hungry, and, it, is, not, the, case, that, james, likes, jack], []).
X = s(s(n(jack), vp(v(is), ap(hungry)))), conj(and), s(neg(not), s(n(james), vp(v(likes), n(jack)))), false.

?- s(X, [it, is, not, the, case, that, jack, is, hungry, or, sophia, is, boring], []).
X = s(neg(not), s(n(jack), vp(v(is), ap(hungry)))), conj(or), s(n(sophia), vp(v(is), ap(boring))), false.
X = s(neg(not), s(s(n(jack), vp(v(is), ap(hungry)))), conj(or), s(n(sophia), vp(v(is), ap(boring)))), false.

?-

Just syntax, but we'll need to produce a semantic parse (instead)
Semantics

- We want to obtain a semantic parse for our sentences that we can “run” (i.e. evaluate) against the Prolog database (i.e. situation or possible world).
- So the semantic parse should be valid Prolog code (that we can call)