Reminder:
• Homework 4 on recursion on grammar due Wednesday by midnight

• Homework 3 review today
Homework 3 review

• **Question 1:** what does this do?

```prolog
?- Term = np(nn(boy), dt(the)), arg(N, Term, Argument).
```

• **Query is:** for what values of variables N and Argument is the Nth argument of Term true?

• **Answers:**
  1. N=1, Argument = nn(boy)
  2. N=2, Argument = dt(the)
Homework 2 review

• **Question 2:** what does this do?

  \[ ?- \text{Term} = \text{np}(\text{np}(\text{nn}(\text{boy}),\text{dt}(\text{the})),\text{pp}(\text{in}(\text{with}),\text{np}(\text{dt}(a),\text{nn}(\text{telescope}))))), \text{arg}(2,\text{Term},\text{Arg}1), \text{arg}(2,\text{Arg}1,\text{Arg}2). \]

• **Query is:** for what values of variables Arg1 and Arg2 is \( \text{arg}(2,\text{Term},\text{Arg}1), \text{arg}(2,\text{Arg}1,\text{Arg}2) \) true?

• **Answer is:**
  
  — Arg1 = \( \text{pp}(\text{in}(\text{with}),\text{np}(\text{dt}(a),\text{nn}(\text{telescope})))) \),
  
  — Arg2 = \( \text{np}(\text{dt}(a),\text{nn}(\text{telescope})) \)

*In plain English:* it picks out the 2nd sub-constituent of the 2nd sub-constituent of Term

\[ ?- \text{T} = \text{np}(\text{np}(\text{nn}(\text{boy}),\text{dt}(\text{the})),\text{pp}(\text{in}(\text{with}),\text{np}(\text{dt}(a),\text{nn}(\text{telescope}))))), \text{arg}(2,\text{T},\text{A}1),\text{arg}(2,\text{A}1,\text{A}2). \]

\( \text{T} = \text{np}(\text{np}(\text{nn}(\text{boy}), \text{dt}(\text{the})), \text{pp}(\text{in}(\text{with}), \text{np}(\text{dt}(a), \text{nn}(\text{telescope}))))), \)

\( \text{A}1 = \text{pp}(\text{in}(\text{with}), \text{np}(\text{dt}(a), \text{nn}(\text{telescope})))) \),

\( \text{A}2 = \text{np}(\text{dt}(a), \text{nn}(\text{telescope})). \)
Homework 2 review

- **Question 3:** give a Prolog query that picks out the word *telescope* from the compound term

\[
\text{Term} = \text{np(np(nn(boy), dt(the)), pp(in(with), np(dt(a), nn(telescope))))}
\]

- **Query is:**

\[
\text{?- T = np(np(nn(boy), dt(the)), pp(in(with), np(dt(a), nn(telescope))))}, \text{ arg(2,T,A1), arg(2,A1,A2), arg(2,A2,A3), arg(1,A3,A4).}
\]

\[
\begin{array}{l}
A1 = \text{pp(in(with), np(dt(a), nn(telescope))}, \\
A2 = \text{np(dt(a), nn(telescope)),} \\
A3 = \text{nn(telescope),} \\
A4 = \text{telescope.}
\end{array}
\]

**Idea:**
1. take the 2\textsuperscript{nd} argument of Term, that’s the PP constituent
2. take the 2\textsuperscript{nd} argument of PP, that’s the NP object of PP
3. take the 2\textsuperscript{nd} argument of NP, that’s the NN head of the NP
4. Take the 1\textsuperscript{st} argument of NN, that’s the word *telescope*
Homework 2 review

• Question 4: Given

?- List1 = [dt, the], List2 = [nn, telescope], List3 = [in, with]
devise a query using =.. that produces the complex
term NP: \( NP = np(dt(\text{the}), \text{nn(telescope)}) \)

Answer is:

?- L1 = [dt, the], L2 = [nn, telescope], L3 = [in, with], DT =.. L1, NN =.. L2, NP =.. [np, DT, NN].
L1 = [dt, the],
L2 = [nn, telescope],
L3 = [in, with],
DT = dt(the),
NN = nn(telescope),
NP = np(dt(the), nn(telescope)).

Idea:
1. Build term dt(the) using =.. applied to L1
2. Build term nn(telescope) using =.. applied to L2
3. Form the list [np,dt(the),nn(telescope)]
4. Build term np(dt(the),nn(telescope)) using =..
Homework 2 review

• **Question 5**: devise a query using `=..` that produces the complex term `PP`:

```
PP = pp(in(with), np(dt(the), nn(telescope))).
```

?- List1 = [dt, the], List2 = [nn, telescope], List3 = [in, with]

• **Answer is**:

```
?- L1 = [dt, the], L2 = [nn, telescope], L3 = [in, with], DT =.. L1, NN =.. L2, NP =.. [np, DT, NN], P =.. L3, PP =.. [pp, P, NP].
L1 = [dt, the],
L2 = [nn, telescope],
L3 = [in, with],
DT = dt(the),
NN = nn(telescope),
NP = np(dt(the), nn(telescope)),
P = in(with),
PP = pp(in(with), np(dt(the), nn(telescope)).
```

**Idea:**

1. Build term `dt(the)` using `=..` applied to `L1`
2. Build term `nn(telescope)` using `=..` applied to `L2`
3. Form the list `[np, dt(the), nn(telescope)]`
4. Build term `np(dt(the), nn(telescope))` using `=..`
5. Build term `in(with)` using `=..` applied to `L3`
6. Form the list `[pp, in(with), np(dt(the), nn(telescope))]`
7. Build term `PP` using `=..`
Today’s Topic

• Dealing with left recursion
Right Recursive Rules

• Example (right-recursive):
  - nn --> a, nn.
  - nn --> [bus].
• Example: big red shiny bus
• Derivation (top-down, left-to-right):
  1. nn
  2. a nn
  3. big nn
  4. big a nn
  5. big red nn
  6. big red a nn
  7. big red shiny nn and so on...
Left Recursion

• Given:
  – *I saw the boy with a telescope* (2 parses)

• Strategy:
  – PP with a telescope adjoins to NP or VP:
    – np(np(NP, PP)) --> np(NP), pp(PP).
    – vp(vp(VP, PP)) --> vp(VP), pp(PP).

Unfortunately, this is left recursive, and Prolog will crash..
Left Recursion

• Instead of:
  – np(np(DT,NN)) --> dt(DT), nn(NN).
  – np(np(NP,PP)) --> np(NP), pp(PP).

• Quick fix:
  – np(np(DT,NN)) --> dt(DT), nn(NN).
  – np(np(DT,NN,PP)) --> dt(DT), nn(NN), pp(PP).

• Can opt to simulate a left recursive parse (as seen in a previous lecture):
  – np(np(DT,NN)) --> dt(DT), nn(NN).
  – np(np(np(DT,NN,PP))) --> dt(DT), nn(NN), pp(PP).
Left Recursion

• Instead of:
  – \text{vp(vp(VBD,NP))} \rightarrow \text{vbd(VBD), np(NP)}.
  – \text{vp(vp(VP,PP))} \rightarrow \text{vp(VP), pp(PP)}.

• Quick fix:
  – \text{vp(vp(VBD,NP))} \rightarrow \text{vbd(VBD), np(NP)}.
  – \text{vp(vp(VBD,NP,PP))} \rightarrow \text{vbd(VBD), np(NP), pp(PP)}.

• Can also opt to return a left recursive parse:
  – \text{vp(vp(VBD,NP))} \rightarrow \text{vbd(VBD), np(NP)}.
  – \text{vp(vp(vp(VBD,NP),PP))} \rightarrow \text{vbd(VBD), np(NP), pp(PP)}. 
Transform Left into Right Recursion

• Transform left recursive into right recursive rules

• An abstract example:
  – $x \rightarrow x, [y]$.  
  – $x \rightarrow [z]$.

• Language generated by this grammar:
  – $z$
  – $zy$
  – $zyy$
  – $zyyy$ and so on...

• Parses:
  – $[x \ z]$
  – $[x [\ x \ z] \ y]$
  – $[x [\ x [\ x \ z] \ y] \ y]$
  – $[x [\ x [\ x \ z] \ y] \ y] \ y]$
  – and so on...

• Transformed example:
  – $x \rightarrow [z], v.$
  – $v \rightarrow [y], v.$
  – $v \rightarrow [y]$.

• Language:
  – $z$
  – $zy$
  – $zyy$
  – $zyyy$ ...

• Parses:
  – $[x \ z]$
  – $[x z [v \ y]]$
  – $[x z [v y [v y]]]$
  – $[x z [v y [v y [v y]]]]$
  – and so on...

**rewrite as unchanged**

Same language!
Transform Left into Right Recursion

• right recursive version:
  – x --> [z], v.
  – v --> [y], v.
  – v --> [y].
  – x --> [z].

• add a parse:
  – x(x(z,V)) --> [z], v(V).
  – v(v(y,V)) --> [y], v(V).
  – v(v(y)) --> [y].
  – x(x(z)) --> [z].
Transform Left into Right Recursion

- We can still get a left recursive parse if we add a 2\textsuperscript{nd} argument to nonterminal v:

1. $x(X) \rightarrow [z], v(X,x(z))$
2. $v(V,X) \rightarrow [y], v(V,x(X,y))$
3. $v(x(X,y), X) \rightarrow [y]$
4. $x(x(z)) \rightarrow [z]$

Compare with:
- $x(x(z,V)) \rightarrow [z], v(V)$
- $v(v(y,V)) \rightarrow [y], v(V)$
- $v(v(v)) \rightarrow [y]$
- $x(x(z)) \rightarrow [z]$