- Midterm #3 next week — more details posted on web site (yes, it's like the last two)

"Contrariwise,' continued Tweedledee, 'if it was so, it might be; and if it were so, it would be; but as it isn't, it ain't. That's logic."

Lewis Carroll, Through the Looking-Glass

Since the midterm...

Done:

- Syntax and semantics of propositional definite clauses
- Model a simple domain using propositional definite clauses
- Bottom-up proof procedure computes a consequence set using modus ponens.
- Top-down proof procedure answers a query using resolution.
- The box model provides a way to procedurally understand the top-down proof procedure with depth-first search.
- Prolog Syntax: Predicate symbols, constants, variables, function symbols.
- Prolog Semantics: Interpretations, variable assignments, models, logical consequence.
- Functions applied to arguments refer to individuals.
  Individuals are described using clauses.
  (Prolog's function symbols are like Haskell constructors.)
  Special syntax for lists; internally a binary function `'[|]'`.

Trees (bstree.pl)

A binary search tree can be used as a representation for dictionaries.

- A binary search tree is either
  - *empty* or
  - $bnode(Key, Val, T0, T1)$ where Key has value Val and T0 is the tree of keys less than Key and T1 is the tree of keys greater than Key
- Define $val(K, V, T)$ is true if key K has value V in tree T
- Define $insert(K, V, T0, T1)$ true if T1 is the result of inserting $K = V$ into tree T0

Trees (bstreec.pl)

- In Prolog, when $X < Y$ is called, both X and Y must be ground (variable free) numbers
- There are constraint solvers that let Prolog act more logically. `X #< Y` specifies the constraint that $X < Y$.
- Eg, consider the query

```
val(K,V,bnode(2,22, bnode(1,57,empty,empty),
                    bnode(5,105,empty,empty))).
```

- $<$ is much faster as it can be evaluated immediately.
- `#<` requires more sophisticated reasoning.

```
?- val(K,V,bnode(2,22, bnode(1,57,empty,empty),
                   bnode(5,105,empty,empty))), V #< 99.
?- V #< 99, val(K,V,bnode(2,22,
                         bnode(1,57,empty,empty),
                         bnode(5,105,empty,empty))).
```

Clicker Question

What is the answer to query

```
?- append([a,b,c],R,L), append([1,2,3],S,R).
```

- A There are no proofs
- B `R = [1, 2, 3|S], L = [1, 2, 3, a, b, c|S].`
- C `R = [1, 2, 3|S], L = R.`
- D `R = [1, 2, 3|S], L = [a, b, c, 1, 2, 3|S].`
- E `R = L, L = [a, b, c, 1, 2, 3|S].`

Natural Language Understanding

- We want to communicate with computers using natural language (spoken and written).
  - unstructured natural language — allow any statements, but make mistakes or failure.
  - controlled natural language — only allow unambiguous statements with fixed vocabulary (e.g., in supermarkets or for doctors).
- There is a vast amount of information in natural language.
- Understanding language to answer questions is more difficult than extracting gestalt properties such as topic, or choosing a web page.

Syntax, Semantics, Pragmatics

- Syntax describes the form of language (using a grammar).
- Semantics provides the meaning of language.
- Pragmatics explains the purpose or the use of language (how utterances relate to the world).

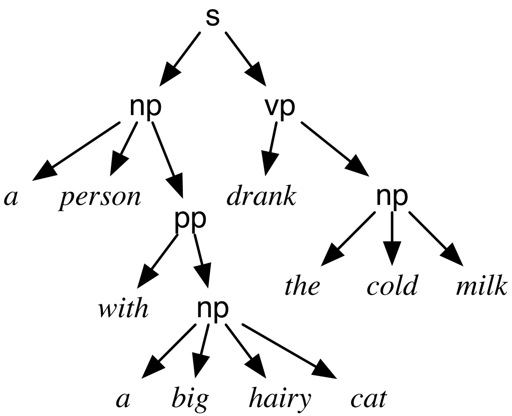
Examples:

- *This lecture is about natural language.*
- *The green frogs sleep soundly.*
- *Colorless green ideas sleep furiously.*
- *Furiously sleep ideas green colorless.*

Parsing Language

- *A person with a big hairy cat drank the cold milk.*
- Who or what drank the milk?

Simple parse tree:



Context-free grammar

- A terminal symbol is a string representing a word (perhaps including punctuation and composite words, such as "hot dog" or "Buenos Aires").
- A non-terminal symbol can be rewritten as a sequence of terminal and non-terminal symbols, e.g.,

$$sentence \longmapsto noun\_phrase, verb\_phrase$$

$$verb\_phrase \longmapsto verb, noun\_phrase$$

$$verb \longmapsto [''drank'']$$

- Can be written as a logic program, where a sentence is a sequence of words:

$$sentence(S)\ {:-}\ noun\_phrase(N), verb\_phrase(V), append(N, V, S).$$

$$verb\_phrase(P)\ {:-}\ verb(V), noun\_phrase(N), append(V, N, P).$$

To say word "drank" is a verb:

$$verb([''drank'']).$$

Difference Lists

- Non-terminal symbol s becomes a predicate with two arguments, $s(T_1, T_2)$, meaning:
  - T_2 is an ending of the list T_1
  - all of the words in T_1 before T_2 form a sequence of words of the category s.
- Lists T_1 and T_2 together form a difference list.
- "the student" is a noun phrase:

$$noun\_phrase(["the", "student", "passed", "the", "course"], \quad ["passed", "the", "course"])$$

- The words "drank" and "passed" are verbs:

$$verb(["drank" \mid W], W).$$

$$verb(["passed" \mid W], W).$$

Definite clause grammar

The grammar rule

$$sentence \longmapsto noun\_phrase, verb\_phrase$$

represented as: there is a sentence between T_0 and T_2 if there is a noun phrase between T_0 and T_1 and a verb phrase between T_1 and T_2:

$$\begin{aligned} &sentence(T_0, T_2) \ \texttt{:-} \\ &\qquad noun\_phrase(T_0, T_1), \\ &\qquad verb\_phrase(T_1, T_2). \end{aligned}$$

$$\overbrace{\underbrace{T_0 \qquad\qquad\qquad\qquad}_{noun\_phrase}\underbrace{T_1 \qquad\qquad\qquad\qquad}_{verb\_phrase}}^{sentence} T_2$$

Definite clause grammar rules

The rewriting rule

$$h \longmapsto b_1, b_2, \ldots, b_n$$

says that h is b_1 followed by b_2, ..., followed by b_n:

$$\begin{aligned}
&h(T_0, T_n) \text{ :- } \\
&\qquad b_1(T_0, T_1), \\
&\qquad b_2(T_1, T_2), \\
&\qquad \vdots \\
&\qquad b_n(T_{n-1}, T_n).
\end{aligned}$$

using the interpretation

$$\overbrace{\underbrace{T_0 \qquad\quad}_{b_1}\underbrace{T_1 \qquad\quad}_{b_2} T_2 \cdots \underbrace{T_{n-1} \qquad\quad}_{b_n}}^{h} T_n$$

Terminal Symbols

Non-terminal h gets mapped to the terminal symbols, $t_1, ..., t_n$:

$$h([t_1, \cdots, t_n \mid T], T)$$

using the interpretation

$$\overbrace{t_1, \cdots, t_n}^{h} \, T$$

Thus, $h(T_1, T_2)$ is true if $T_1 = [t_1, ..., t_n \mid T_2]$.

Context Free Grammar Example

see
`https://artint.info/3e/resources/ch15/geography_CFG.pl`

(also load `https://artint.info/3e/resources/ch15/geography_DB.pl`)

What will the following query return?

```
noun_phrase(["a","country","that","borders","Chile"], L3).
```

How many answers does the following query have?

```
noun_phrase(["a", "Spanish", "speaking", "country",
             "that", "borders", "Chile"], L3).
```

Example

```
% a noun phrase is a determiner followed by adjectives
% followed by a noun followed by a prepositional phrase.
noun_phrase(L0,L4) :-
   det(L0,L1),
   adjectives(L1,L2),
   noun(L2,L3),
   pp(L3,L4).
% dictionary for determiners
det(L,L).
det(["a"|L],L).
det(["the"|L],L).
% adjectives is a sequence of adjectives
adjectives(L,L).
adjectives(L0,L2) :-
    adj(L0,L1),
    adjectives(L1,L2).
```

Clicker Question

If the query for the grammar rule

`noun_phrase([the,cat,on,the,mat,sat,on,the,hat], R).`

returns with substitution `R=[sat,on,the,hat]`
What is the noun-phrase it found?

A the cat

B the mat

C the cat on the mat

D sat on the hat

E either "the cat", "the mat" or "the hat", we can't tell

Clicker Question

If the query for the grammar rule

`noun_phrase([the,cat,on,the,mat,sat,on,the,hat], R).`

returns with substitution `R=[on,the,mat,sat,on,the,hat]`
What is the noun-phrase it found?

- A the cat
- B the mat
- C the cat on the mat
- D sat on the hat
- E either "the cat", "the mat" or "the hat", we can't tell

Augmenting the Grammar

Two mechanisms can make the grammar more expressive:

extra arguments to the non-terminal symbols

arbitrary conditions on the rules.

We have a Turing-complete programming language at our disposal!

Question-answering

- How can we get from natural language directly to the answer?
- Goal: map natural language to a query that is asked of a knowledge base.
- Add arguments representing the individual

  $$noun\_phrase(T_0, T_1, O)$$

  means
  - $T_0 - T_1$ is a difference list forming a noun phrase.
  - The noun phrase refers to the individual O.
- Can be implemented by the parser directly calling the knowledge base.

Example natural language to query

see
`https://artint.info/3e/resources/ch15/geography_QA.pl`

Noun Phrases

```
% A noun phrase is a determiner followed by adjectives fol
% by a noun followed by an optional modifying phrase.
% They all refer to the same individual.
noun_phrase(L0, L4, Ind) :-
    det(L0, L1, Ind),
    adjectives(L1, L2, Ind),
    noun(L2, L3, Ind),
    omp(L3, L4, Ind).
```

Adjectives provide properties

```
% adj(T0,T1,Entity) is true if T0-T1
% is an adjective that is true of Entity
adj(["large" | L], L, Ind) :- large(Ind).
adj([LangName, "speaking" | L], L, Ind) :-
    language(Ind, Lang), name(Lang, LangName).

% adjectives(T0,T1,Entity) is true if
% T0-T1 is a sequence of adjectives that true of Entity
adjectives(T0,T2,Entity) :-
    adj(T0,T1,Entity),
    adjectives(T1,T2,Entity).
adjectives(T,T,_).
```

Verbs and propositions provide relations

reln($T0, T1, Subject, Object$)

- $T0 - T1$ is a verb or preposition that provides
- a relation that true between *Subject* and *Object*

```
reln(["borders" | L], L, Sub, Obj) :- borders(Sub, Obj).
reln(["bordering" | L], L, Sub, Obj) :- borders(Sub, Obj).
reln(["next", "to" | L], L, Sub, Obj) :- borders(Sub, Obj).
reln(["the", "capital", "of" | L], L, Sub, Obj) :-
     capital(Obj, Sub).
reln(["the", "name", "of" | L], L, Sub, Obj) :-
     name(Obj, Sub).
```

Verbs and propositions provide relations

```
% A  modifying phrase / relative clause is either
% a relation (verb or preposition)
%      followed by a phrase or
% 'that' followed by a relation then a phrase
mp(L0, L2, Subject) :-
    reln(L0, L1, Subject, Object),
    aphrase(L1, L2, Object).
mp(["that" | L0], L2, Subject) :-
    reln(L0, L1, Subject, Object),
    aphrase(L1, L2, Object).

% An optional modifying phrase is either a modifying phrase
omp(L0,L1,E) :-
    mp(L0,L1,E).
omp(L, L, _).
```