- Project #2 - get started!

"In Prolog, as in most halfway decent programming languages, there is no tension between writing a beautiful program an writing an efficient program. If your Prolog code is ugly, the chances are that you either don't understand your problem or don't understand your programming language, and in neither case does your code stand much chance of being efficient. In order to ensure your program is efficient, you need to know what it is doing, and if your code is ugly, you will find it hard to analyse."

Richard A. O'Keefe, "The Craft of Prolog", 1990.

Plan

Last time

- difference lists
- definite clause grammars
- computer algebra and calculus

To do

- natural language interfaces to databases
- Semantic web
- negation as failure
- pragmatic choices of Prolog
- proofs with variables and complex terms

Definite Clause Grammars

- A sentence consists of a noun phrase followed by a verb phrase.
- *sentence*(L, E) is true if (L, E) forms a difference list that is a sentence
- *noun_phrase*(L, E) is true if (L, E) forms a difference list that is a noun phrase
- *verb_phrase*(L, E) is true if (L, E) forms a difference list that is a verb phrase

```
sentence(L_0,L_2) :-
    noun_phrase(L_0,L_1),
    verb_phrase(L_1,L_2).
```

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://www.cs.ubc.ca/~poole/cs312/2024/prolog/geography_QA.pl`

Noun Phrases

```
% A noun phrase is a determiner followed by adjectives foll
% by a noun followed by an optional modifying phrase:
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 noun_phrase or
% 'that' followed by a relation then a noun_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, _).
```

Clicker Question

What if `geography_QA` contained this as the definition of `adjectives` (where `adjectives` and `adj` are in a different order in the body):

```
adjectives(L0, L2, Ind) :-
    adjectives (L0, L1, Ind),
    adj(L1, L2, Ind).
adjectives(L, L, _).
```

A it would work the same

B it might give incorrect results when the original doesn't

C it might fail in cases when the original doesn't

D it might not halt in cases when the original halts

Real-world queries

- Want a tokenizer: mapping from strings to sequence of words. `split_string` or `readln` provides a simple ones.
- What should the system do with ungrammatical sentences?
- What should the system do with new words?
- What about pronoun references?

  The student took many courses. Two computer science courses and one mathematics course were particularly difficult. The mathematics course...
  Who was the captain of the Titanic?
  Was she tall?

- And other tricky and subtle aspects of English?
  — program them
  — learn them

Example natural language to query

- see
  `http://www.cs.ubc.ca/~poole/cs312/2024/prolog/geography_QA.pl`
- What does it mean if it answers *false*?
  We can't tell whether it couldn't parse the question or there were no answers to the question.
- Almost impossible to debug.
- Idea: parse the sentence first, building a query that is then asked of the database.

`http://www.cs.ubc.ca/~poole/cs312/2024/prolog/geography_QA_query.pl`

Building a list of constraints on the entity (geography_QA_query.pl)

noun_phrase(L0,L4,Entity,C0,C4) is true if

- L0 and L4 are list of words, such that
  - $L4$ is an ending of $L0$
  - the words in $L0$ before $L4$ (written $L0 - L4$) form a noun phrase
- Entity is an individual that the noun phrase is referring to
- C0 is a list such that $C4$ is an ending of $C0$ and $C0 - C4$ contains the constraints imposed by the noun phrase

```
noun_phrase(L0,L4,Entity,C0,C4) :-
    det(L0,L1,Entity,C0,C1),
    adjectives(L1,L2,Entity,C1,C2),
    noun(L2,L3,Entity,C2,C3),
    mp(L3,L4,Entity,C3,C4).
```

Building a list of constraints on the entity (geography_QA_query.pl)

- Nouns and adjectives provide constraints:

```
adj([large | L],L,Entity, [large(Entity)|C],C).
adj([Lang,speaking | L],L,Entity,
        [speaks(Entity,Lang)|C],C).

noun([country | L],L,Entity, [country(Entity)|C],C).
noun([city | L],L,Entity, [city(Entity)|C],C).
```

- Verbs and propositions provide relations
  $reln(T0, T1, Subject, Object, C0, C1)$
  - $T0 - T1$ is a verb or preposition that provides relations in $C0 - C1$ that is true between individuals *Subject* and *Object*

```
reln([borders | L],L,O1,O2,[borders(O1,O2)|C],C).
reln([the,capital,of | L],L,O1,O2,
                          [capital(O2,O1)|C],C).
reln([next,to | L],L,O1,O2, [borders(O1,O2)|C],C).
```

Clicker Question

If the query for the grammar rule

```
noun_phrase(["the","cat","on","the","mat","sat","on","the",
                    R, Ent, C0, C1).
```

returns with substitution `R=["sat","on","the","hat"]`
What do we hope the value of $C0$ and $C1$ is:

A $C0 = [\text{"}the\text{"}, \text{"}cat\text{"}, \text{"}on\text{"}, \text{"}the\text{"}, \text{"}mat\text{"} | C1]$

B $C0 = [cat(Ent), on(Ent, A), mat(A) | C1]$

C $C0 = felix, C1 = fluffy$

D $C0 = C1$

E we would hope this would fail

Clicker Question

Which specifies that sat is a verb, than indicates the relation sat:

A
```
reln([sat | L], L, Sub, Obj, [sat(Sub,Obj) | C], C).
```

B
```
reln([sat | L], L, Sub, Obj, [sat | C], C).
```

C
```
reln([sat | L], L, Sub, Obj, sat(Sub,C), C).
```

D
```
reln([sat | L], L, Sub, Obj, sat(Sub,Obj), C).
```

E It can't be done

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!

NLP requires Understanding (Winograd Schemas)

- The city councilmen refused the demonstrators a permit because they feared violence. Who feared violence?
- The city councilmen refused the demonstrators a permit because they advocated violence. Who advocated violence?
- Steve follows Fred's example in everything. He [admires/influences] him hugely. Who [admires/influences] whom?
- The table won't fit through the doorway because it is too [wide/narrow]. What is too [wide/narrow]?
- Grace was happy to trade me her sweater for my jacket. She thinks it looks [great/dowdy] on her. What looks [great/dowdy] on Grace?
- Bill thinks that calling attention to himself was rude [to/of] Bert. Who called attention to himself?
- In a recent competition, the best algorithm got 58% correct!

`https://cs.nyu.edu/faculty/davise/papers/WinogradSchemas/WS.html`