CPSC 312 — Functional and Logic Programming

- Midterm #3 on Monday! — more details on web site (yes, it's like the others, only different questions)
- Solution to assignment 5 available.

But what is truly distinctive and valuable about human natural language is its semantic or representational capacities — the features of language responsible for how words carry meaning, and how words can be combined into sentences to make an indefinite number of distinct, meaningful assertions about the world.

Kevin deLaplante "All the Formal Logic You Need to Know for Critical Thinking"
`https://criticalthinkeracademy.com/courses/2514/lectures/751606`

Since the midterm...

- Syntax and semantics of 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 `'[|]'`.
- Today: algebra, calculus, natural language queries...

Writing a Prolog program

To write a Prolog program:

- Have a clear intended interpretation – what all predicates, functions and constants mean
- Don't tell lies.
  Make sure all clauses are true given your meaning for the constants, functions, predicates.
- Make sure that the clauses cover all of the cases when a predicate is true.
- Avoid cycles.
- Design top-down, build bottom-up.
- Debug all predicates as you write them.
- To solve a complex problem break it into simpler problems.

Computer Algebra (algebra.pl)

- Because Prolog does not evaluate expressions, an algebraic expression can be manipulated. E.g., as in Assignment 5.
- Algebraic variables can be treated as Prolog constants. (Remember Prolog variables mean "for all").
- Derivatives can be defined using
  $deriv(E, X, DE)$ is true if DE is the derivative of E with respect to X
- Expressions can be simplified: To simplify $A * B$: first simplify A and B, then check for multiplication by 0 or 1 or when both simplify to numbers.
- More sophisticated simplification is possible (but difficult).
- Multivariate differentiation just works.
- Integration is more difficult (finding when to apply rules is more complicated)

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, _).
```