Logic Programming

- Based on symbolic logic
- Sometimes called declarative programming
- Concerned with specifying the properties of the results, rather than how to compute them
- We will focus on Prolog

Predicate Calculus Review

We will need the following concepts from predicate calculus.

- Predicates: a relation consisting of the name and an ordered list of parameters. It is either true or false. e.g. parent(john, bill)
- Logical connectives: negation, conjunction, disjunction, equivalence, implication
- Quantifiers: universal and existential
- In Prolog, names of relations and individual objects are in lowercase, variables (quantified) starts with uppercase or underscore.

Clauses

- A proposition in clausal form looks like:

$$A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee \ldots \vee B_m$$

  It means if $A_1, \ldots, A_n$ are all true, then (at least) one of $B_1, \ldots, B_m$ is true.

- Variables in A_i and B_j are assumed to be universally quantified.
- All propositions in predicate logic can be converted to this form.
- Note: $(A \rightarrow B) \equiv (\neg A \vee B)$

Theorem Proving

- Given a set of facts and rules, as well as a goal, an automatic theorem attempts to prove that the goal follows logically from the facts and rules.
- Inference rules are needed to derive conclusions from given facts.

Inference Rules

- Modus Ponens is one of the most basic inference rules.
- From $P \to Q$ and P, we can derive Q.
- There are other inference rules.
- How many different inference rules do we need? i.e. Can anything that follows logically be proven with the given rules?

Resolution

- Given $P \vee Q$ and $\neg Q \vee R$, we can derive $P \vee R$.
- P, Q, and R can be arbitrarily complicated propositions.
- This is basically the rule that combines $\neg P \rightarrow Q$ and $Q \rightarrow R$.
- Repeated literals are removed.

Resolution Refutation Proof

- A set of propositions is inconsistent if there is no truth value that can be assigned to make each proposition true at the same time.
- Resolution Refutation Proof:
  1. Choose any two propositions from the set that has the same term but in opposite form, apply resolution and simplify result.
  2. If result is empty, we have a contradiction. The set of propositions is inconsistent.
  3. If a term appears in both positive and negative form, ignore the result.
  4. Add result to the set of propositions.
  5. Repeat until the empty proposition is obtained, or if we do not have any new results generated from resolution.

Resolution Refutation Proof

- This process will terminate (finite number of possible results).
- Theorem: resolution is refutation complete.
- That is, if a set of propositions is inconsistent, this process will always generate the empty proposition at some point.

Unification

- If there are variables in the terms, appropriate values of the variables would have to be determined.
- The process of assigning values to variables is called unification.
- With unification, two propositions may resolve in different ways. All possibilities need to be considered.

$$p(x) \vee q(x), \neg p(a) \vee \neg q(b)$$

Theorem Proving

- Instead of proving that a certain goal statement is a theorem (i.e. follows from given facts and rules)...
- We ask whether the facts and rules, together with the negation of the goal, is inconsistent.
- If this set is consistent, then the truth assignment to the various propositions is a counterexample.
- If this set is inconsistent, there is no counterexample.
- If there are variables, different values would have to be tried through backtracking.

Example

$$\{\neg A \vee \neg B \vee \neg D, \neg B \vee D, \neg A \vee B, A\}$$

Horn Clauses

- Horn Clauses are disjunction of terms, such that at most one atom is not negated.
- The three possibilities correspond to rules, facts and goals.
- Slightly more restrictive than all propositions, but allows resolution to be more efficient

Logic Programming in Prolog

- A term is a constant, variable, or a structure.
- Constants: an atom (symbolic name) or an integer
- Variables: name starts with uppercase letter or underscore
- Structure: predicate

Facts, Rules, and Goals

- Facts are simply listed as a predicate. They are true.
- Rules indicates: `B :- A1, A2, ... An`
- This means if $A_1, \ldots, A_n$ are true, then B is true.
- Goal statements are similar to facts, but variables in goal statements means that we are asking Prolog to perform unification to find values that make the goal statement a theorem.

Making Inferences

- Forward chaining: start from the facts and rules, and keep deriving new propositions until the goal is reached.
- Backward chaining: start from the goal and work backwards
- Prolog uses backward chaining:
  - start with the goal
  - find a corresponding fact (possibly with unification)
  - otherwise, find a rule and try to satisfy subgoals
- This process can be thought of as traversing a tree. Depth-first or breadth-first? Prolog uses depth-first.
- Backtracking may be performed, especially with unification.

Arithmetic

- Recall that there are no "functions" in Prolog, only predicates.
- To say $f(a,b) = c$, we should make a predicate `f(a,b,c)`.
- You can write something like `A is B + C`. The LHS cannot yet be instantiated yet.
- If we have `speed(abc, 10).` and `time(abc, 5).`, we can compute the distance as:

```
distance(X, Y) :- speed(X, Speed), time(X, Time),
                  Y is Speed * Time.
```

Lists in Prolog

- Lists are similar to those in Lisp, but uses square brackets.
- Lists can be nested.
- Empty list: `[]`
- A nonempty list can be dismantled into first element and the rest: `[H|T]`.

  This can also be used to build lists.
- There are built-in predicates for lists.

Lists in Prolog

- Example: `distinct(L)` that is true if `L` contains no duplicate elements.

```
distinct([]).
distinct([H|T]) :- distinct(T), \+ member(H, T).
```

  `member` is a built-in predicate, `\+` means “not”.

- Example: `append(L1, L2, L3)` means `L3` is the result of appending `L2` to `L1`.

```
append([], L, L).
append([H|L1], L2, [H|L3]) :- append(L1, L2, L3).
```

Ordering

- Order of facts and rules are irrelevant for correctness.
- But order is important in Prolog for efficiency.
- Prolog applies the facts and rules from top to bottom.
- Within a rule, the subgoals and examined from left to right.
- Putting more restrictive rules/subgoals first makes it easier to prune the recursive backtracking search (especially for unification).
- Cuts (`!`) can be used to control pruning: a cut is a goal that is always satisfied immediately the first time, but subsequent backtracking attempts cannot satisfy the goal again.
- There is no point to try other ways to satisfy goals to the left of a cut.

Cuts

- Consider the max function:

```
max(X,Y,X) :- X >= Y.
max(X,Y,Y) :- Y > X.
```

- If this is a subgoal of some rule:

```
f(X,Y) :- max(X,Y,Z), test(Z).
```

- If we try to prove `f(3,4)`, then at some point we will unify `Z` to 4.
- If `test(4)` fails, it will then attempt to try other values of `Z`.
- But there are no other values of `Z`. So this is inefficient.

Cuts

- Another way

```
max(X,Y,Z) :- X >= Y, !, X = Z.
max(X,Y,Y).
```

- An incorrect attempt

```
max(X,Y,X) :- X >= Y, !.
max(X,Y,Y).
```

Why?

Closed-World Assumption

- Prolog has no knowledge of what is not included in the database.
- Prolog returning false means that it cannot prove the goal to be true.
- e.g. not enough evidence?
- The goal may in fact be true. Additional facts/rules may have to be added.

Negation

- In Prolog, negation is indicated by `\+`
- However, negation does not have the "usual" meaning.
- In Prolog, a goal is true if there is a proof.
- But a goal may be true but not provable because of the given facts and rules.
- Negation in Prolog means "not provable".
- e.g. if `male(john).` will return "no" because this is not provable in the database, so `\+ male(john).` will return "yes".

Negation

- Double negation does not have the usual property:
  `\+ \+ member(X,[a,b,c]).`
- Consider:

```
prime(7).
even(2).
even_composite(X) :- \+ prime(X), even(X).
```

- Asking for `even_composite` on 2 and 7 gives correct results
- What if you ask for `even_composite(X)`?
- The fact that there is at least one value that is prime means the first part cannot fail.

Other Uses

- Declarative programming is commonly used in databases. e.g. Structured Query Language (SQL)
- Declares the properties of the answer instead of how to compute it.