Logic Programming

- Logic programming began as a collaboration between AI researchers (e.g., John McCarthy) and logicians (e.g., John Alan Robinson) to solve problems in artificial intelligence.
- Cordell Green built the first “question and answer” system using Robinson’s “unification algorithm,” demonstrating that it was practical to prove theorems automatically.
Prolog

- Alain Colmerauer and Philippe Roussel at Aix-Marseille University invented Prolog in 1972.
- They were inspired by a particular formulation of logic, called "Horn clauses," popularized by the logician Robert Kowalski.
- Horn clauses have a "procedural interpretation," meaning that they suggest a simple procedure for solving them, called "resolution."
- John Alan Robinson’s unification algorithm is an efficient algorithm for doing resolution, and this is essentially the algorithm used by Prolog.

Declarative Programming

- Declarative programming is a very different style of programming than you have seen to this point.
- Mostly, you have seen imperative programs.
- In imperative-style programming, the programmer instructs the computer how to compute the desired result.
- In declarative-style programming, the computer already knows how to compute results.
- Instead, the programmer asks the computer what to compute.

Declarative Programming

- Most of you have probably been CS majors for long enough that we have sufficiently damaged your brain such that you do not recognize the difference between these two concepts.
- In fact, imperative-style programming is a very unnatural way of communicating desires.
- Declarative: "Make me a PB&J sandwich."
- Imperative: https://youtu.be/cDA3_cg82h8

Prolog

- The goal of AI is to enable a computer to answer declarative queries.
- I.e., it already knows how to answer you.
- Prolog was an attempt to solve this problem.
- Since this was early work, the input language was somewhat primitive: predicate logic.
- As you will see, formulating queries in pure logic is not the easiest thing to do.
- However, for certain classes of logic, there are known efficient, deterministic algorithms for solving every possible query.
Horn Clause

• Horn clauses are composed of two simple pieces:
  • facts
  • rules (clauses)
• Rules are composed of facts
• Complex facts may also be composed using conjunction.
• We will explore these concepts using Prolog syntax.
• Note that Horn clauses can be "satisfied" in polynomial time.
  • In fact, Horn logic is the most expressive form of logic
    known to be satisfiable in polynomial time.

Facts (Prolog syntax)

• Here are some facts:
  rainy.
  cloudy.
  thursday.
• Facts are assumed to be true.
• Facts of this form are sometimes called "atoms", since they are indivisible.
• The meaning of these facts is up to the programmer.
• Facts can also be compound:
  rainy, cloudy.
  cloudy, thursday.
• ",," denotes "logical and".
• Note that, in Prolog, facts are always lowercase and must begin
  with a letter.

Rules (Prolog syntax)

• Here are some rules:
  sleep_deprived :- thursday.
  unhappy :- raining, cloudy.
• The interpretation of a rule X :- Y is:
  if Y is true, then X is true
• In other words, Y is the antecedent and X is the consequent.
• So, we might interpret the above as:
  "students are sleep deprived if it is Thursday"
  "I am unhappy if it is raining and cloudy."

Variables (Prolog syntax)

• Note that I just used a generalization of rules
  without definition:
  X :- Y
• Prolog explicitly allows generalizations of facts like this.
• We call these generalizations "variables", because their
  precise values (i.e., facts) may not be known to us.
• In the "execution" of a Prolog program, we seek to "instantiate"
  variables with facts.
• In Prolog, variables are always written starting with an uppercase letter.
• We will come back to variables shortly.
Complex facts (Prolog syntax)

- Prolog allows one additional form:
  - musician(mia).
  - musician(john).
  - friends_with(mia,john).
- Statements of this form are called “complex facts.”
- Again, the interpretation is up to you.
- E.g.,
  - ‘Mia is a musician’
  - ‘John is a musician’
  - ‘Mia is friends with John’
- Note that we do not automatically assume that
  - ‘John is friends with Mia’!

Queries

- Taken together, facts and rules form a “knowledge base.”
  - raining.
  - cloudy.
  - thursday.
  - sleep_deprived :- thursday.
  - unhappy :- raining,cloudy.
- A query asks the knowledge base a question. E.g.,
  - ?- sleep_deprived.
  - true
  - ?- unhappy.
  - true

Resolution

- ‘Resolution’ is the name of the procedure that Prolog uses to
  ‘satisfy’ a query.
  - raining.
  - cloudy.
  - thursday.
  - sleep_deprived :- thursday.
  - unhappy :- raining,cloudy.
- Essentially, we seek to reduce a query expression to the expression
  true by substitution.
- Remember that facts are assumed to be true.
Resolution

- Given the following knowledge base,
  1. a :- b,c.
  2. b :- d,e.
  3. b :- g,e.
  4. c :- e.
  5. d.
  6. e.
  7. f :- a,g.

- Let’s try to satisfy the following query using resolution:
  ?- a.

Proof Search

- Nonetheless, Prolog is not generally sensitive to the order of the facts in a database. How does this work?
- The answer is that resolution is actually a form of backtracking search.

Resolution with Variables

- Resolution with variables can be very computationally expensive.
- Unification allows resolution with variables to be completed in polynomial time.
- The basic insight is to “instantiate” variables “on demand” instead of enumerating all possible variable instantiations into facts.
- Hindley-Milner is essentially just unification.

Resolution

- Note that we get a slightly different outcome if the same set of facts are written in a slightly different order:
  1. a :- b,c.
  2. b :- g,e.
  3. b :- d,e.
  4. c :- e.
  5. d.
  6. e.
  7. f :- a,g.

- Again, let’s try to satisfy the following query using resolution:
  ?- a.
Resolution with Variables

- When asking a query that utilizes variables, Prolog will both search for a satisfying assignment and it will return that assignment.
- There may be more than one possible assignment.
- If so, use the `;` command to ask for another solution.
- Let's resolve the following query:
  ```prolog
  ?- friends_with(mia,Who).
  ```
- We may even ask:
  ```prolog
  ?- friends_with(Who1,Who2).
  ```

Exercise

- Construct the a knowledge base containing the following facts:
  - “Giants eat people.”
  - “Giants eat bunnies.”
  - “Bunnies eat grass.”
  - “People eat bunnies.”
  - “People eat people.”
  - “Those who are eaten by others hate those others.”
  - “Monsters love those who hate themselves.”
- Then supply a query that can answer:
  - “Who do monsters love?”