CS 230
Programming Languages

12 / 05 / 2022

Instructor: Michael Eckmann

Today's Topics

- Questions? / Comments?
- More Prolog (Logic Programming Language)

Logical connectors

Name	Symbol	Example	Meaning
negation	$\neg$	$\neg$ a	not a
conjunction	$\cap$	a $\cap$ b	a and b
disjunction	$\cup$	a $\cup$ b	a or b
equivalence	$\equiv$	a $\equiv$ b	a is equivalent to b
implication	$\supset$ $\subset$	a $\supset$ b a $\subset$ b	a implies b b implies a

Logic Languages

- To use a variable in a proposition we need to preface it with either “for all” or “there exists”
- These are quantifiers
  - For all is represented by an upside-down A
  - There exists is represented by a backwards E

- Predicate calculus consists of the logical connectors we just saw in addition to the two quantifiers above.

Predicate Calc and proving theorems

- A restricted kind of clausal form is used when propositions are used for resolution. These special kinds of propositions are **Horn clauses**
  - Only 2 forms
  - A single atomic proposition on left side
  - Or
  - Empty left side
- Left side is called the *head*. If left side is not empty the Horn clause is called a *headed Horn clause*.

Predicate Calc and proving theorems

- An atomic propostion is a functor and a set of parameters in parentheses.
  - e.g. mother(joanne, jake)
- A headed Horn clause example:

  older(joanne, jake) $\subset$ mother(joanne, jake)
  - these state relationships among relations

- A headless Horn clauses: e.g. mother(joanne, jake)

Overview of Logic Programming

- Logical programming is nonprocedural
  - Programs do not state how a result is to be computed, but rather the form of the result
- We don't tell the computer how to get a result. Instead we state the form of the result and let the logic programming "engine" determine how to get that result.
- We provide the relevant information via the predicate calculus and the computer infers the results using resolution.

Overview of Logic Programming

- Describe the characteristics of a sorted list, not the process of rearranging a list

sort(old_list, new_list) $\subset$
permute (old_list, new_list) $\cap$ sorted (new_list)

sorted (list) $\subset$ ForAll j such that $1 \leq j < n$, list(j) $\leq$ list (j+1)

- This says that if a list is in non-decreasing order, it is sorted. Also, if the old_list is a permutation of the new_list and the new_list is sorted, then that's what it means to sort an old_list to be the new_list

Prolog

- Our text uses the Edinburgh Syntax version of Prolog
- Term: a constant, variable, or structure
- Constant: an atom or an integer
- Atom: symbolic value of Prolog
- Atom consists of either:
  - a string of letters, digits, and underscores beginning with a lowercase letter
  - a string of printable ASCII characters delimited by apostrophes

Prolog

- Variable: any string of letters, digits, and underscores beginning with an uppercase letter
- Instantiation: binding of a variable to a value
  - Lasts only as long as it takes to satisfy one complete goal
- Structure: represents atomic proposition

functor(parameter list)

Prolog

- Fact statements
  - Headless Horn clauses

```
student(jonathan).
brother(tyler, cj).
```

- Rule statements
  - Headed Horn clauses
  - Right side: antecedent (***if*** part)
    - May be single term or conjunction
  - Left side: consequent (***then*** part)
    - Must be single term
- Conjunction: multiple terms separated by logical AND operations (implied)

Prolog

```
parent(kim,kathy):- mother(kim,kathy).
```

- Can use variables (universal objects) to generalize meaning:

```
parent(X,Y):- mother(X,Y).
sibling(X,Y):- mother(M,X),
    mother(M,Y),
    father(F,X),
    father(F,Y).
```

Prolog

- GOAL STATEMENTS
- For theorem proving, a theorem is in the form of a proposition that we want the system to prove or disprove
- Same format as headless Horn

```
student(james)
```

  - this asks if james is a student, it either fails (james is not a student) or it doesn't (james is indeed a student)

- Conjunctive propositions and propositions with variables are also legal goals

```
father(X,joe)
```

  - this asks for all fathers of joe, if any, otherwise it fails.

Prolog

- INFERENCING
- Queries are called goals
- If a goal is a compound proposition, each of the facts is a subgoal
- To prove a goal is true, the system must find a chain of inference rules and/or facts. For goal Q:

```
B :- A
C :- B
...
Q :- P
```

- Process of proving a subgoal is called **matching, satisfying,** or **resolution**

Prolog

- POSSIBLE WAYS TO INFER
- Bottom-up resolution, forward chaining
  - Begin with facts and rules of database and attempt to find sequence that leads to goal
- Top-down resolution, backward chaining
  - begin with goal and attempt to find sequence that leads to set of facts in database
- Prolog implementations use **backward chaining**

Prolog

- POSSIBLE WAYS TO INFER
- When goal has more than one subgoal, can use either
  - *Depth-first search*: find a complete proof for the first subgoal before working on others
  - *Breadth-first search*: work on all subgoals in parallel
- Prolog uses **depth-first search**
  - Can be done with fewer resources so it is preferred for use in Prolog

Prolog

- Backtrack during inference
- For a goal with multiple subgoals, if fail to show truth of one of subgoals, reconsider previous subgoal to find an alternative solution: backtracking
- Begin search where previous search left off
- In worst case, can take lots of time and space because may find all possible proofs to every subgoal

Prolog

- Recap: When trying to prove a goal, Prolog uses
  - backward chaining (starting at goal) to try to find facts that satisfy it
  - depth first search when satisfying a chain of subgoals
  - backtracking during inference to possibly get an alternative solution to an earlier subgoal

Prolog

- Prolog also supports integer variables and integer arithmetic
- ***is*** operator: takes an arithmetic expression as right operand and variable as left operand
- A is B / 10 + C
- Not the same as an assignment statement
- B & C must have been instantiated and A must not have been instantiated for that expression to have A take on the value of the expression.
- But if A is instantiated prior to this expression or B or C are not instantiated, then the clause is not satisfied and no instantiation can take place.
- Example: Sum is Sum + Number. Is this possible?

Prolog

```
speed(ford,100).
speed(chevy,105).
speed(dodge,95).
time(ford,20).
time(chevy,21).
time(dodge,24).
distance(X,Y) :- speed(X,Speed),
                 time(X,Time),
                 Y is Speed * Time.
```

- Here, the facts are the first six statements and the last is a rule stating the relationship between time, speed and distance.
- Example query: distance(chevy, Chevy_Distance)
- Chevy_Distance is instantiated with the value 2205

Prolog

- TRACE
- Tracing model of execution - four events:
  - Call (beginning of attempt to satisfy goal)
  - Exit (when a goal has been satisfied)
  - Redo (when backtrack occurs)
  - Fail (when goal fails)

Prolog

```
likes(jake,chocolate).
likes(jake,apricots).
likes(darcie,licorice).
likes(darcie,apricots).
```

```
trace.
likes(jake,X),
likes(darcie,X).
```

This is asking, what is X such that both jake and darcie like X

Let's trace it on the board.